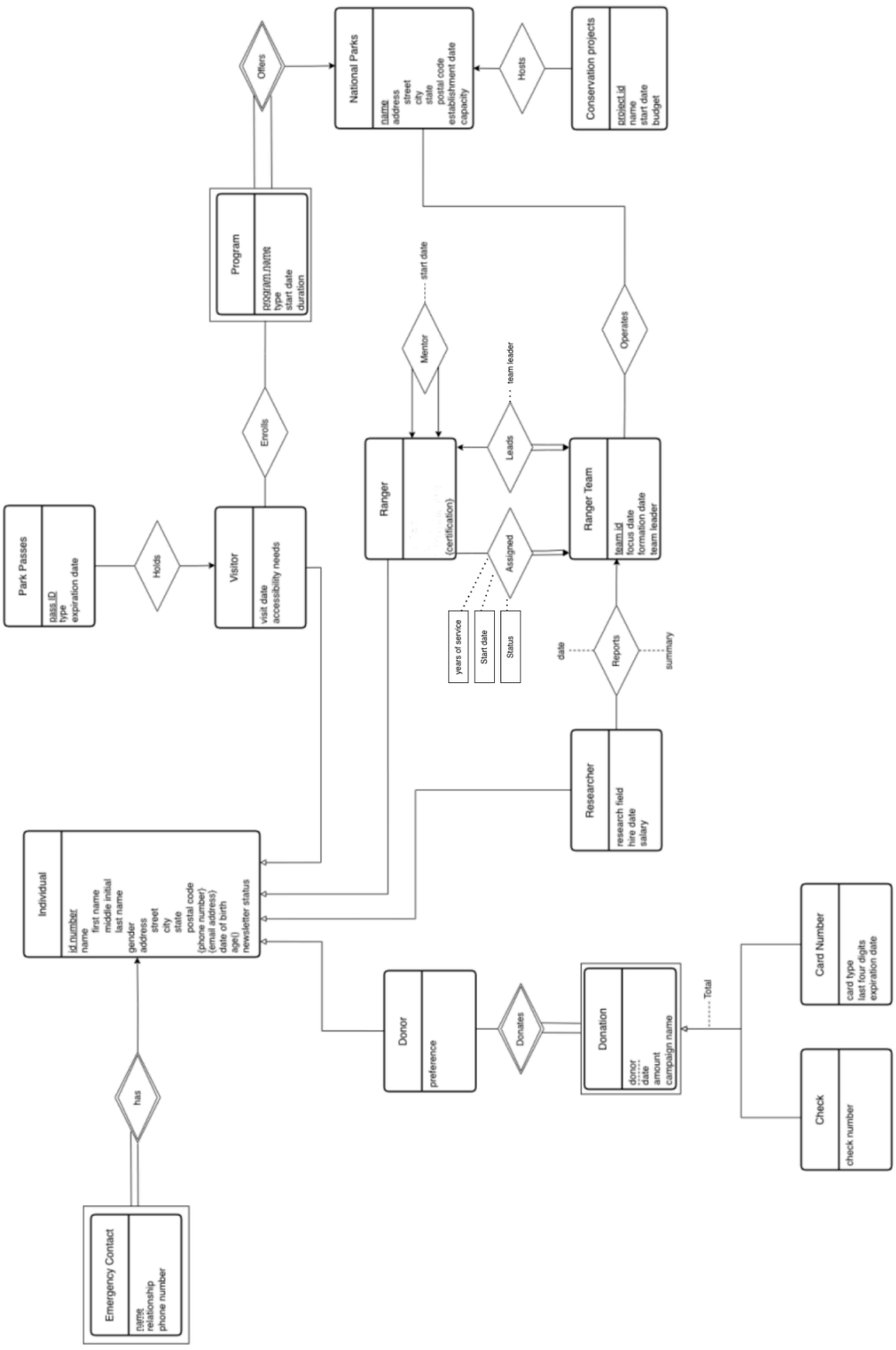
COURSE: CS/DSA-4513 – DATABASE MANAGEMENT SECTION: 001
SEMESTER: FALL 2025
INSTRUCTOR: DR. LE GRUENWALD
NAME: ASTRA NGUYEN
STUDENT ID: 11346128
EMAIL: ASTRA.K.NGUYEN-1X@OU.EDU

NATIONAL PARK SERVICE

DATABASE SYSTEM

Task Performed — Page Number

Task 1. ER Diagram



Task 2. Relational Database Schema List

2.1.0 Strong Entities

2.1.1 Individual (Superclass)

Individual(id_number, first_name, middle_initial, last_name, gender, address_street, address_city, address_state, postal_code, phone_number, email_address, date_of_birth, age, newsletter_status)
Primary key: id_number
Composite Attributes:
- name decomposed into: first_name, middle_initial, last_name
- address decomposed into: street, city, state, postal_code

Derived Attribute: age (computed from date_of_birth)
Multi-valued Attributes: phone_number, email_address

2.1.2 National Parks

National_Parks(name, address_street, city, state, postal_code, establishment_date, capacity)
Primary Key: name
Composite Attributes:
- address decomposed into: street, city, state, postal_code

2.1.3 Park Passes

Park_Passes(pass_id, type, expiration_date, visitor_id)
Primary Key: pass_id

2.1.4 Program

Program(program_name, type, start_date, duration)
Primary Key: program_name

2.1.5 Conservation_Projects

Conservation_Projects(project_id, name, start_date, budget, park_name)
Primary Key: project_id
Foreign Key: park_name REFERENCES national_parks(name)
Note: One-to-many relationship with National Parks

2.1.6 Ranger Team

Ranger_team(team_id, focus_date, formation_date, team_leader)
Primary Key: team_id

Foreign Key: team_leader REFERENCES ranger(id_number)
Team_leader is a foreign key to ranger, representing the 1:1 Leads relationship

2.2.0 Specialization Entities (ISA relationship- Inherits all attributes from Individual)

2.2.1 Donor

Donor(id_number, preference)
Primary Key: id_number
Foreign Key: id_number REFERENCES individual(id_number)

2.2.2 Researcher

Researcher(id_number, research_field, hire_date, salary)
Primary Key: id_number
Foreign Key: id_number REFERENCES individual(id_number)

2.2.3 Visitor

Visitor(id_number, visit_date, accessibility_needs)
Primary Key: id_number
Foreign Key: id_number REFERENCES individual(id_number)

2.2.4 Ranger

Ranger(id_number)

Primary Key: id_number
Foreign Key: id_number REFERENCES individual(id_number)
Multi-valued Attribute: certification (handled in separate table)
Note: start_date, status, and years_of_service are relationship attributes in the "Assigned" relationship with Ranger Team

2.3.0 Weak Entities

2.3.1 Emergency Contact (1:N relationship with Individual, depends on Individual)

Emergency_contact(id_number, name, relationship, phone_number)
Primary Key: (id_number, phone_number)
Foreign Key: id_number REFERENCES individual(id_number)
1:N relationship - each Individual may have multiple Emergency Contacts

2.3.2 Donation (Depends on donor)

Donation(donation_id, donor_id_number, date, amount, campaign_name)
Primary Key: donation_id
Foreign Key: donor_id_number REFERENCES donor(id_number)
Depends on Donor. donation_id is a surrogate key.

2.4.0 Donation Specialization (Total ISA , disjoint)

2.4.1 Check

Check(donation_id, check_number)
Primary Key: donation_id
Foreign Key: donation_id REFERENCES donation(donation_id)
Total specialization - every Donation must be either Check or Card_Number

2.4.1 Card Number

Card_number(donation_id, card_type, last_four_digits, expiration_date)

Primary Key: donation_id
Foreign Key: donation_id REFERENCES donation(donation_id)
Total specialization - every Donation must be either Check or Card_Number

2.5.0 Multi-Valued Attributes

2.5.1 Individual Phone Numbers

Individual_phone_numbers(id_number, phone_number)
Primary Key: (id_number, phone_number)
Foreign Key: id_number REFERENCES individual(id_number)
Multi-valued attribute from Individual entity

2.5.2 Individual Email Address

Individual_email_addresses(id_number, email_address)
Primary Key: (id_number, email_address)
Foreign Key: id_number REFERENCES individual(id_number)
Multi-valued attribute from Individual entity

2.5.3 Ranger Certifications

Ranger_certifications(id_number, certification)
Primary Key: (id_number, certification)
Foreign Key: id_number REFERENCES ranger(id_number)

Multi-valued attribute from Ranger entity

2.6.0 Many To Many Relationships

2.6.1 Visitor Hold Park Pass

Visitor_holds_park_passes(visitor_id_number, pass_id)
Primary Key: (visitor_id_number, pass_id)
Foreign Key: visitor_id_number REFERENCES visitor(id_number)
Foreign Key: pass_id REFERENCES park_passes(pass_id)
Many-to-many relationship between Visitor and Park Passes

2.6.2 Visitor Enroll Program

Visitor_enrolls_program(visitor_id_number, program_name)
Primary Key: (visitor_id_number, program_name)
Foreign Key: visitor_id_number REFERENCES visitor(id_number)
Foreign Key: program_name REFERENCES program(program_name)
Many-to-many relationship between Visitor and Program

2.6.3 Researcher Reports Ranger Team

Researcher_reports_ranger_team(researcher_id_number, team_id, date, summary)
Primary Key: (researcher_id_number, team_id)
Foreign Key: researcher_id_number REFERENCES researcher(id_number)
Foreign Key: team_id REFERENCES ranger_team(team_id)
Many-to-many relationship with attributes: date, summary

2.6.4 Ranger Assigned Ranger Team

Ranger_assigned_ranger_team(ranger_id_number, team_id, start_date, status, years_of_service)

Primary Key: (ranger_id_number, team_id)
Foreign Key: ranger_id_number REFERENCES ranger(id_number)
Foreign Key: team_id REFERENCES ranger_team(team_id)
Many-to-many relationship between Ranger and Ranger Team
Relationship attributes: start_date, status, years_of_service

2.6.5 Ranger Team Operates National Parks

Ranger_team_operates_national_parks(team_id, park_name)
Primary Key: (team_id, park_name)
Foreign Key: team_id REFERENCES ranger_team(team_id)
Foreign Key: park_name REFERENCES national_parks(name)

Many-to-many relationship between Ranger Team and National Parks

2.6.6 National Park Offers Program

National_parks_offers_program(park_name, program_name)
Primary Key: (park_name, program_name)
Foreign Key: park_name REFERENCES national_parks(name)
Foreign Key: program_name REFERENCES program(program_name)
Note: Many-to-many relationship between National Parks and Program

2.7.0 Recursive Relationships

Ranger_mentors_ranger(mentor_id_number, mentee_id_number, start_date)
Primary Key: (mentor_id_number, mentee_id_number)
Foreign Key: mentor_id_number REFERENCES ranger(id_number)
Foreign Key: mentee_id_number REFERENCES ranger(id_number)
Relationship Attribute: start_date
Many-to-many recursive relationship: Many Rangers can mentor many other Rangers

2.8.0 Relationships Summary

One-to-One (1:1):
- Ranger ↔ Ranger Team (Leads relationship - one ranger leads one team, one team is led by one ranger)

One-to-Many (1:N):
- Emergency Contact → Individual
- Donor → Donation (weak entity)
- National Parks → Conservation Projects (one park hosts many projects)

Many-to-Many (M:N):
- Visitor ↔ Park Passes
- Visitor ↔ Program
- Researcher ↔ Ranger Team (with attributes: date, summary)
- Ranger ↔ Ranger Team (Assigned relationship with attributes: start_date, status, years_of_service)
- Ranger ↔ Ranger (Mentor recursive relationship with attribute: start_date)
- Ranger Team ↔ National Parks (operates relationship)
- National Parks ↔ Program (offers relationship)

ISA Relationships (Specialization):
- Individual → Donor
- Individual → Researcher
- Individual → Visitor
- Individual → Ranger

Task 3. Discussion of Storage Structures For All Tables

3.1 Discussion of Storage Structures for Tables

There are many storage structures for file organization. Such as:

1. Heap File Organization — Random placement, good for random access
    a. It's best when a significant amount of data needs to be loaded into the database at once.
    b. Fetching records and retrieving them is faster in a small database than in consecutive records. Good for random access, random placement.
2. Sequential File Organization — Ordered by search key
    a. Compared to other file organization methods, sequential file organization boasts a simple design and minimal complexity.
    b. Ordered by search key and good for range queries.
    c. Efficient for small amounts of data. But not as much for large datasets.
    d. It maintains a simple order, and there's no manipulation involved that could introduce errors.
3. Clustering File Organization — Multiple relations together, good for joins
    a. Improves query performance for join operations by placing related data physically close on disk.
    b. It's cheaper to search for and retrieve distinct records from different files as they are now integrated and stored in the same cluster.
4. B +- Tree Index — Balanced tree
    a. Guaranteed time complexity of O(log n) for basic operations like insertion, deletion, and searching.
    b. Efficient storage utilization. High-concurrency and high-throughput.
    c. Good for range queries and ordered access because it's a balanced tree.
5. Hash Index — Good for point queries
    a. Fast point queries.
    b. Efficient storage utilization. High-concurrency and high-throughput.
    c. Good for point queries such as equality searches.

3.1.0 Summary of General Storage Structure Choices

For the structure choices, I've separated mine into 3 different parts for optimal databases. They are **Heap file organization, Clustering file organization, B+- Tree.**

- **Heap File:** Allows fast insertions without having to reorganize, so I used it for any tables that'll have tons of insertion queries, providing an O(1) insertion time.
- **Clustering File:** Mostly for ISA relationships, and weak entities. So I'll be using this to relate records to each other like donor, researcher, visitor , and ranger with individuals, which results in reducing I/O for joints.

- **B+- Tree:** Good and provides efficient search keys, provides balanced performances for all range queries including range search and random search.
  Note: No hash indices needed. B+- trees provide better overall performance for all queries, supporting both point queries with O(log n) and range queries efficiently. While hash induced only supports equality searches O(1) on average cases and cannot handle range queries as efficiently.

Sources:

- https://www.geeksforgeeks.org/dbms/dbms/
- CS 4513 Class Canvas Powerpoints Lecture at University of Oklahoma

Tables

Table 1: Individual

Type and Query	Search Key	Query Frequency	Selected File Organization	Justifications/Explanation
[9] Random search	Id_number(PK)	1/week	Heap + Clustering Index on Id_number	Primary key lookups are frequent, clustering index provides quick access with sequential ordering
[10] Random search	Id_number(PK)	1/month	Heap + Clustering Index on Id_number	Monthly primary key lookups are frequent, clustering index provides quick access with sequential ordering
[11] Random search	Id_number(PK)	4/year	Heap + Clustering Index on Id_number	Quarterly primary key lookups and clustering index supports efficient access
[12] Random search	Id_number(PK)	1/week	Heap + Clustering Index on Id_number	Weekly primary key lookups are frequent,

				clustering index provides quick access with sequential ordering
[13] Random search	Id_number(PK)	1/month	Heap + Clustering Index on Id_number	Monthly primary key lookups are frequent, clustering index provides quick access with sequential ordering
[14] Random search	Id_number(PK)	1/year	Heap + Clustering Index on Id_number	Annual primary key lookups and clustering index provides optimal access
[15] Random search	Id_number(PK)	4/year	Heap + Clustering Index on Id_number	Quarterly per year primary key lookups are frequent, clustering index provides quick access with sequential
[2] Range search	Date_of_birth	High	Heap + B+-Tree on Date_of_birth	Aged based queries will require range searches for quick access and necessary. B+- supports efficient range queries
[3] Random search	Last_name	Medium	Heap + B+-Tree on Last_name	Last name, or name based search queries are common, B+- supports efficient range queries
[4] Range search	(City,state)	Medium	Heap + B+-Tree on Last_name	Location search queries needs

				composite key support, and B+- supports efficient range queries
[5] Insert	—-	High	Heap	Fast insertions without reorganization overhead
[6] Delete	Id_number	Low	Heap	Easy deletions

Table 2: National_Parks

Type and Query	Search Key	Query Frequency	Selected File Organization	Justifications
[1] Random search	Name(PK)	Very high	Heap	Primary look up, easy and quick. Name searches are common.
[2] Range search	Establishment_date	Medium	Heap + B+-Tree on Establishment_date	B+-Tree supports efficient date ranges
[3] Range search	Capacity	Low	Heap + B+-Tree on Capacity	Capacity-based filtering needs range support and B+-Tree supports efficient date ranges
[1] Insertion	—	1/week	Heap	Heap handles insertion quickly

Table 3: Park_passes

Type and Query	Search Key	Query Frequency	Selected File Organization	Justifications
[1] Random search	Pass_id(PK)	Very high	Heap	Primary look up, easy and quick.

[2] Range search	Expiration_date	High	Heap + B+-Tree on Expiration_date	B+-Tree supports efficient date ranges
[3] Range search	Type	Medium	Heap + B+-Tree on Type	Type-based filtering common; B+-Tree supports equality and range queries
[5] Deletion	Pass_id	Low	Heap	Heap handles deletion quickly and infrequently

Table 4: Program

Type and Query	Search Key	Query Frequency	Selected File Organization	Justifications
[1] Random search	Program_name(PK)	Very high	Heap + Clustering Index on Program_name	Primary look up, easy and quick. Program enrollments will require fast access
[5] Range search	Type	High	Heap + B+-Tree on Type	B+-Tree supports efficient date ranges
[4] Insertion	—	Low	Heap	Heap handles insertion quickly
[5] Deletion	Program_name	Very low	Heap	Heap handles deletion quickly and infrequently

Table 5: Conservation_projects

Type and Query	Search Key	Query Frequency	Selected File Organization	Justifications

[3] Random search	Project_id (PK)	Very high	Heap + Clustering Index on Program_name	Primary look up, easy and quick. Project details can be accessed often
[2] Range search	Park_name	High	Heap + B+-Tree on Park_name	B+-Tree supports efficient date ranges
[4] Insertion	—	1/year	Heap	Heap handles insertion quickly
[5] Deletion	Project_id	Very low	Heap	Heap handles deletion quickly and infrequently

Table 6: Donor

Type and Query	Search Key	Query Frequency	Selected File Organization	Justifications
[2] Random search	Preference	Medium	Clustering + B+-Tree on Preference	B+-Tree supports efficient date ranges & filtering
[4] Insertion	—	Medium	Clustering	Clustering maintains relationship with Individual
[5] Deletion	id _number	Very low	Clustering	Clustering maintains referential integrity

Table 7: Researcher

Type and Query	Search Key	Query Frequency	Selected File Organization	Justifications

[2] Random search	Research_field	High	Clustering + B+-Tree on Research_field	B+-Tree supports efficient date ranges & filtering
[4] Insertion	—	Low	Clustering	Clustering maintains relationship
[5] Deletion	Id _number	Very low	Clustering	Clustering maintains referential integrity

Table 8: Visitor

Type and Query	**Search Key**	**Query Frequency**	**Selected File Organization**	**Justifications**
[7] Random search	Id_number(PK)	Very high	Clustering File with Individual	ISA relationship with Individual and frequent joins require clustering
[3] Insertion	—	High	Clustering	Clustering maintains relationship
[4] Deletion	Id _number	Low	Clustering	Clustering maintains referential integrity

Table 9: Ranger

Type and Query	**Search Key**	**Query Frequency**	**Selected File Organization**	**Justifications**

[7] Random search	Id_number(PK)	Very high	Clustering File with Individual	ISA relationship with Individual and frequent joins require clustering
[3] Insertion	—	Low	Clustering	Clustering maintains relationship
[4] Deletion	Id _number	Very low	Clustering	Clustering maintains referential integrity and ranger rarely are being removed

Table 10: Ranger_team

Type and Query	Search Key	Query Frequency	Selected File Organization	Justifications
[7] Random search	Team_id(PK)	Very high	Heap + Clustering Index on Team_id	Team information accessed frequently
[5] Ranger search	Team_leader	High	Heap + B+-Tree on Team_leader	Finding teams by leader common and B+-Tree supports foreign key lookups
[3] Insertion	—	Low	Heap	Heap handles insertions well
[4] Deletion	Team_id	Very low	Heap	Heap handles deletion quickly

				and infrequently

Table 11: Emergency_contact

Type and Query	Search Key	Query Frequency	Selected File Organization	Justifications
[10] Random search	Id_number	Very high	Heap + B+-Tree on Id_number	Finding all emergency contacts for an individual; B+-Tree supports efficient lookups
[3] Insertion	—	Medium	Heap	Heap handles insertions well
[4] Deletion	(id_number,phone_number)	Low	Heap	Heap handles deletion quickly and infrequently

Table 12: Donation

Type and Query	Search Key	Query Frequency	Selected File Organization	Justifications
[10] Random search	Donor_id_number	High	Heap + B+-Tree on Donor_id_number	B+-Tree supports foreign key lookups
[5] Random Search	Date	High	Heap + B+-Tree on Date	Date-based reporting needs range queries and B+-Tree supports date ranges
[4] Deletion	Donation_id	Low	Heap	Heap handles deletion quickly and infrequently

Table 13: Check_donation

Type and Query	Search Key	Query Frequency	Selected File Organization	Justifications
[7] Random search	Check_number	High	Clustering + B+-Tree on Check_number	B+-Tree supports efficient lookups
[3] Insertion	—	Medium	Clustering	Clustering maintains relationship
[4] Deletion	Donation_id	Low	Clustering	Clustering maintains integrity

Table 14: Card_number

Type and Query	Search Key	Query Frequency	Selected File Organization	Justifications
[7] Range search	Expiration_date	Medium	Clustering + B+-Tree on Expiration_date	Finding expired cards needs range queries
[3] Insertion	—	Medium	Clustering	Clustering maintains relationship
[4] Deletion	Donation_id	Low	Clustering	Clustering maintains integrity

Table 15: Individual_phone_numbers

Type and Query	Search Key	Query Frequency	Selected File Organization	Justifications

[7] Random search	Id_number	Very high	Heap + B+-Tree on Id_number	B+-Tree supports efficient lookups
[3] Insertion	—	High	Heap	Heap allows fast insertions
[4] Deletion	(Id_number,phone_number)	Low	Heap	Heap allows fast & easy deletions

Table 16: Individual_email_addresses

Type and Query	Search Key	Query Frequency	Selected File Organization	Justifications
[7] Random search	Email_address	1/week	Heap + B+-Tree on Email_address	Unique email constraint requires index And B+-Tree supports efficient lookups
[4] Random search	Email_address	1/month	Heap + B+-Tree on Email_address	Unique email constraint requires index And B+-Tree supports efficient lookups
[4] Deletion	(Id_number, Email_address)	Medium	Heap	Heap allows fast & easy deletions
[12] Random search	Id_number	Very high	Heap + B+-Tree on Id_number	B+-Tree supports efficient lookups

Table 17: Ranger_certifications

Type and Query	Search Key	Query Frequency	Selected File Organization	Justifications
[2] Deletion	(Id_number, certification)	Low	Heap	Heap allows fast & easy deletions

[3 Insertion	—	Medium	Heap	Heap allows fast insertions

Table 18: Visitor_holds_park_passes

Type and Query	Search Key	Query Frequency	Selected File Organization	Justifications
[2] Deletion	(Visitor_id_number, Pass_id)	Medium	Heap	Heap allows fast & easy deletions
[3] Insertion	—	High	Heap	Heap allows fast insertions
[7] Random search	Visitor_id_number	High	Heap + B+-Tree on Visitor_id_number	B+-Tree supports efficient lookups

Table 19: Visitor_enrolls_program

Type and Query	Search Key	Query Frequency	Selected File Organization	Justifications
[2] Deletion	(Visitor_id_number, Program_name)	Medium	Heap	Heap allows fast & easy deletions
[5] Random search	Visitor_id_number	High	Heap + B+-Tree on Visitor_id_number	B+-Tree supports efficient lookups
[7] Insertion	—	High	Heap	Heap allows fast insertions

Table 20: Researcher_reports_Ranger_team

Type and Query	Search Key	Query Frequency	Selected File Organization	Justifications
[1] Random Search	Researcher_id_number	High	Heap + B+-Tree on Researcher_id_number	B+-Tree supports efficient lookups

[5] Random search	Team_id	High	Heap + B+-Tree on team_id	B+-Tree supports efficient lookups
[7] Insertion	—	Medium	Heap	Heap allows fast insertions

Table 21: Ranger_assigned_ranger_team

Type and Query	Search Key	Query Frequency	Selected File Organization	Justifications
[1] Random Search	Ranger_id_number	Very high	Heap + B+-Tree on Ranger_id_number	B+-Tree supports efficient lookups
[5] Range search	Start_date	Medium	Heap + B+-Tree on Start_date	B+-Tree supports date ranges
[7] Range search	Years_of_service	Medium	Heap + B+-Tree on Years_of_service	B+-Tree supports range queries & efficient

Table 22: Ranger_team_operates_national_parks

Type and Query	Search Key	Query Frequency	Selected File Organization	Justifications
[5] Random Search	Team_id	High	Heap + B+-Tree on Team_id	B+-Tree supports efficient lookups
[10] Insertion	—	Low	Heap	Heap allows fast insertions
[11] Deletion	(Team_id, Park_name)	Low	Heap	Heap allows fast & easy deletions

Table 23: National_parks_offers_program

Type and Query	Search Key	Query	Selected File	Justifications

		Frequency	Organization	
[5] Random Search	Park_name	High	Heap + B+-Tree on Park_name	B+-Tree supports efficient lookups
[10] Insertion	—	Low	Heap	Heap allows fast insertions
[11] Deletion	(Park_name, Program_name)	Low	Heap	Heap allows fast & easy deletions

Table 24: Ranger_mentors_ranger

Type and Query	Search Key	Query Frequency	Selected File Organization	Justifications
[5] Range search	Start_date	Medium	Heap + B+-Tree on Start_date	B+-Tree supports date ranges & make it efficients
[10] Insertion	—	Medium	Heap	Heap allows fast insertions
[13] Deletion	(Mentor_id_number, Mentee_id_number)	Low	Heap	Heap allows fast & easy deletions
[14] Random search	Mentor_id_number	High	Heap + B+-Tree on Mentor_id_number	B+-Tree supports efficient lookups

Task 3.2 Azure SQL Database Implementation

1. All Tables: Use Clustered Index on Primary Key (automatic in Azure SQL)
- Provides optimal primary key access
- Automatic maintenance and statistics
- No reorganization needed

2. ISA Relationship Tables (Donor, Researcher, Visitor, Ranger):
- Clustered Index on Id_number (primary key)
- Non-clustered B+-Tree indices on other search keys

- Consider INCLUDE columns for covering indices

3. Weak Entity Tables (Emergency_contact, Donation, Check_donation, Card_number):
- Clustered Index on Primary Key
- Non-clustered B+-Tree index on Foreign Key
- Foreign key index supports efficient joins with parent tables

4. Junction Tables (M:N relationships):
- Clustered Index on Composite Primary Key
- Non-clustered B+-Tree indices on both foreign keys
- Supports efficient bidirectional lookups

5. Multi-valued Attribute Tables:
- Clustered Index on Composite Primary Key
- Non-clustered B+-Tree index on Id_number
- Efficient retrieval of all values for an individual

All tables will use:
- Clustered Index (B+-Tree) on primary key (automatic)
- Non-clustered B+-Tree indices on foreign keys and frequently queried attributes
- Automatic index maintenance and statistics updates
- No manual reorganization required

Task 4. SQL and Text Files Showing the Creation of Tables in Task 3

INPUT:

```
CREATE TABLE Individual (
        Id_number                       VARCHAR(50) NOT NULL,
        First_name                      VARCHAR(100) NOT NULL,
        Middle_initial                  CHAR(3) NULL,
        Last_name                       VARCHAR(100) NOT NULL,
        Gender                          CHAR(1) NOT NULL,
        Street                          VARCHAR(200) NOT NULL,
        City                            VARCHAR(100) NOT NULL,
        State                           VARCHAR(50) NOT NULL,
        Postal_code                     VARCHAR(20) NOT NULL,
        Date_of_birth                   DATE NOT NULL,
        Newsletter_status               BIT NOT NULL DEFAULT 0,
        CONSTRAINT PK_individual        PRIMARY KEY(id_number),
        CONSTRAINT CK_individual_gender CHECK (gender IN ('M', 'F', 'O'))
);
```

```
CREATE TABLE National_parks (
        Name                                    VARCHAR(200) NOT NULL,
        Street                                  VARCHAR(200) NOT NULL,
        City                                    VARCHAR(100) NOT NULL,
        State                                   VARCHAR(50) NOT NULL,
        Postal_code                             VARCHAR(20) NOT NULL,
        Establishment_date                      DATE NOT NULL,
        Capacity                                INT NOT NULL,
        CONSTRAINT PK_national_parks            PRIMARY KEY (name),
        CONSTRAINT CK_national_parks_capacity CHECK (capacity > 0)
);

CREATE TABLE Park_passes (
        Pass_id                                 VARCHAR(50) NOT NULL,
        Type                                    VARCHAR(50) NOT NULL,
        Expiration_date                         DATE NOT NULL,
        CONSTRAINT PK_park_passes                       PRIMARY KEY (pass_id)
);

CREATE TABLE Program (
        Program_name                            VARCHAR(200) NOT NULL,
        Type                                    VARCHAR(100) NOT NULL,
        Start_date                              DATE NOT NULL,
        Duration                                INT NOT NULL,
        CONSTRAINT PK_program                   PRIMARY KEY (program_name),
        CONSTRAINT CK_program_duration          CHECK (duration > 0)
);

CREATE TABLE Conservation_projects (
  Project_id                                    VARCHAR(50) NOT NULL,
  Name                                          VARCHAR(200) NOT NULL,
  Start_date                                    DATE NOT NULL,
  Budget                                        DECIMAL(18, 2) NOT NULL,
  Park_name                                     VARCHAR(200) NOT NULL,
  CONSTRAINT PK_conservation_projects           PRIMARY KEY (project_id),
  CONSTRAINT FK_conservation_projects_park
                                                FOREIGN KEY (park_name)
     REFERENCES national_parks(name)
                                                ON DELETE CASCADE ON UPDATE
                                        CASCADE,
  CONSTRAINT CK_conservation_projects_budget
                                                CHECK (budget > 0)
);
```

```
CREATE TABLE Donor (
        Id_number                               VARCHAR(50) NOT NULL,
        Preference                              VARCHAR(200) NULL,
        CONSTRAINT PK_donor                     PRIMARY KEY (id_number),
        CONSTRAINT FK_donor_individual          FOREIGN KEY (id_number)
                REFERENCES individual(id_number) ON DELETE CASCADE ON UPDATE
CASCADE
);

CREATE TABLE Researcher (
        Id_number                               VARCHAR(50) NOT NULL,
        Research_field                          VARCHAR(200) NOT NULL,
        Hire_date                               DATE NOT NULL,
        Salary                                  DECIMAL(18, 2) NOT NULL,
        CONSTRAINT PK_researcher                PRIMARY KEY (id_number),
        CONSTRAINT FK_researcher_individual     FOREIGN KEY (id_number)
                REFERENCES individual(id_number) ON DELETE CASCADE ON UPDATE
CASCADE,
        CONSTRAINT CK_researcher_salary         CHECK (salary > 0)
);

CREATE TABLE Visitor (
        Id_number                               VARCHAR(50) NOT NULL,
        Visit_date                               DATE NULL,
        Accessibility_needs                     VARCHAR(500) NULL,
        CONSTRAINT PK_visitor                   PRIMARY KEY (id_number),
        CONSTRAINT FK_visitor_individual        FOREIGN KEY (id_number)
                REFERENCES individual(id_number) ON DELETE CASCADE ON UPDATE
CASCADE
);

CREATE TABLE Ranger (
  Id_number                                     VARCHAR(50) NOT NULL,
  CONSTRAINT PK_ranger                          PRIMARY KEY (id_number),
  CONSTRAINT FK_ranger_individual               FOREIGN KEY (id_number)
     REFERENCES individual(id_number)           ON DELETE CASCADE ON UPDATE
CASCADE
);


CREATE TABLE Ranger_team (
        Team_id                                 VARCHAR(50) NOT NULL,
```

```
        Focus_date                          DATE NULL,
        Formation_date                      DATE NOT NULL,
        Team_leader                         VARCHAR(50) NULL,
        CONSTRAINT PK_ranger_team           PRIMARY KEY (team_id),
        CONSTRAINT FK_ranger_team_team_leader FOREIGN KEY (team_leader)
            REFERENCES ranger(id_number) ON DELETE NO ACTION ON UPDATE
CASCADE
);

CREATE TABLE Emergency_contact (
        Id_number                           VARCHAR(50) NOT NULL,
        Name                                VARCHAR(200) NOT NULL,
        Relationship                        VARCHAR(100) NOT NULL,
        Phone_number                        VARCHAR(20) NOT NULL,
        CONSTRAINT PK_emergency_contact     PRIMARY KEY
(id_number,phone_number),
        CONSTRAINT FK_emergency_contact_individual
                                            FOREIGN KEY (id_number)
            REFERENCES individual(id_number)
                                            ON DELETE CASCADE ON UPDATE
                                            CASCADE
);

CREATE TABLE Donation (
        Donation_id                         VARCHAR(50) NOT NULL,
        Donor_id_number                     VARCHAR(50) NOT NULL,
        Date                                DATE NOT NULL,
        Amount                              DECIMAL(18, 2) NOT NULL,
        Campaign_name                       VARCHAR(200) NULL,
        CONSTRAINT PK_donation              PRIMARY KEY (donation_id),
        CONSTRAINT FK_donation_donor        FOREIGN KEY (donor_id_number)
            REFERENCES donor(id_number)     ON DELETE CASCADE ON UPDATE
CASCADE,
        CONSTRAINT CK_donation_amount       CHECK (amount > 0)
);

CREATE TABLE Check_donation (
        Donation_id                         VARCHAR(50) NOT NULL,
        Check_number                        VARCHAR(50) NOT NULL,
        CONSTRAINT PK_check_donation        PRIMARY KEY (donation_id),
        CONSTRAINT FK_check_donation_donation
                                            FOREIGN KEY (donation_id)
            REFERENCES donation(donation_id)
```

```
                                        ON DELETE CASCADE ON UPDATE
                                    CASCADE,
        CONSTRAINT UQ_check_donation_check_number
                                        UNIQUE (check_number)
);

CREATE TABLE Card_number (
        Donation_id                     VARCHAR(50) NOT NULL,
        Card_type                       VARCHAR(50) NOT NULL,
        Last_four_digits                VARCHAR(4) NOT NULL,
        Expiration_date                 DATE NOT NULL,
        CONSTRAINT PK_card_number       PRIMARY KEY (donation_id),
        CONSTRAINT FK_card_number_donation  FOREIGN KEY (donation_id)
                REFERENCES donation(donation_id)
                                        ON DELETE CASCADE ON UPDATE
                                        CASCADE
);

CREATE TABLE Individual_phone_numbers (
        Id_number                       VARCHAR(50) NOT NULL,
        Phone_number                    VARCHAR(20) NOT NULL,
        CONSTRAINT PK_individual_phone_numbers
                                        PRIMARY KEY (id_number,
                                    phone_number),
        CONSTRAINT FK_individual_phone_numbers_individual
                                        FOREIGN KEY (id_number)
    REFERENCES individual(id_number)
                                        ON DELETE CASCADE ON UPDATE
                                    CASCADE
);

CREATE TABLE Individual_email_addresses (
        Id_number                       VARCHAR(50) NOT NULL,
        Email_address                   VARCHAR(200) NOT NULL,
        CONSTRAINT PK_individual_email_addresses
                                        PRIMARY KEY (id_number,
                                    email_address),
        CONSTRAINT FK_individual_email_addresses_individual
                                        FOREIGN KEY (id_number)
                REFERENCES individual(id_number) ON DELETE CASCADE ON UPDATE
CASCADE,
        CONSTRAINT UQ_individual_email_addresses_email
                                        UNIQUE (email_address)  -- emails should
                                    be unique across all individuals
);
```

```
CREATE TABLE Ranger_certifications (
        Id_number                                   VARCHAR(50) NOT NULL,
        Certification                               VARCHAR(200) NOT NULL,
        CONSTRAINT PK_ranger_certifications     PRIMARY KEY (id_number, certification),
        CONSTRAINT FK_ranger_certifications_ranger
                                                    FOREIGN KEY (id_number)
    REFERENCES ranger(id_number)
                                                    ON DELETE CASCADE ON UPDATE
                                                CASCADE
);

CREATE TABLE Visitor_holds_park_passes (
        Visitor_id_number                           VARCHAR(50) NOT NULL,
        Pass_id                                     VARCHAR(50) NOT NULL,
        CONSTRAINT PK_visitor_holds_park_passes
                                                    PRIMARY KEY (visitor_id_number,
                                                pass_id),
        CONSTRAINT FK_visitor_holds_park_passes_visitor
                                                    FOREIGN KEY (visitor_id_number)
            REFERENCES visitor(id_number)
                                                    ON DELETE CASCADE ON UPDATE
                                                CASCADE,
        CONSTRAINT FK_visitor_holds_park_passes_pass
                                                    FOREIGN KEY (pass_id)
            REFERENCES park_passes(pass_id)
                                                    ON DELETE CASCADE ON UPDATE
                                                CASCADE
);

CREATE TABLE Visitor_enrolls_program (
        Visitor_id_number                           VARCHAR(50) NOT NULL,
        Program_name                                VARCHAR(200) NOT NULL,
        CONSTRAINT PK_visitor_enrolls_program   PRIMARY KEY (visitor_id_number,
program_name),
        CONSTRAINT FK_visitor_enrolls_program_visitor
                                                    FOREIGN KEY (visitor_id_number)
            REFERENCES visitor(id_number)
                                                    ON DELETE CASCADE ON UPDATE
                                                CASCADE,
        CONSTRAINT FK_visitor_enrolls_program_program
                                                    FOREIGN KEY (program_name)
            REFERENCES program(program_name)
```

```
                                        ON DELETE CASCADE ON UPDATE
                                    CASCADE
);

CREATE TABLE Researcher_reports_ranger_team (
        Researcher_id_number            VARCHAR(50) NOT NULL,
        Team_id                         VARCHAR(50) NOT NULL,
        Date                            DATE NOT NULL,
        Summary                         VARCHAR(MAX) NULL,
        CONSTRAINT PK_researcher_reports_ranger_team
                                        PRIMARY KEY (researcher_id_number,
                                    team_id),
        CONSTRAINT FK_researcher_reports_ranger_team_researcher
                                        FOREIGN KEY (researcher_id_number)
                REFERENCES researcher(id_number)
                                        ON DELETE CASCADE ON UPDATE
                                    CASCADE,
        CONSTRAINT FK_researcher_reports_ranger_team_team
                                        FOREIGN KEY (team_id)
                REFERENCES ranger_team(team_id)
                                        ON DELETE CASCADE ON UPDATE
                                    CASCADE
);

CREATE TABLE Ranger_assigned_ranger_team (
        Ranger_id_number                VARCHAR(50) NOT NULL,
        Team_id                         VARCHAR(50) NOT NULL,
        Start_date                      DATE NOT NULL,
        Status                          VARCHAR(50) NOT NULL,
        Years_of_service                AS DATEDIFF(YEAR, start_date,
GETDATE()) PERSISTED,
        CONSTRAINT PK_ranger_assigned_ranger_team
                                        PRIMARY KEY (ranger_id_number,
                                    team_id),
        CONSTRAINT FK_ranger_assigned_ranger_team_ranger
                                        FOREIGN KEY (ranger_id_number)
                REFERENCES ranger(id_number)
                                        ON DELETE CASCADE ON UPDATE
                                    CASCADE,
        CONSTRAINT FK_ranger_assigned_ranger_team_team
                                        FOREIGN KEY (team_id)
                REFERENCES ranger_team(team_id)
                                        ON DELETE CASCADE ON UPDATE
                                    CASCADE,
```

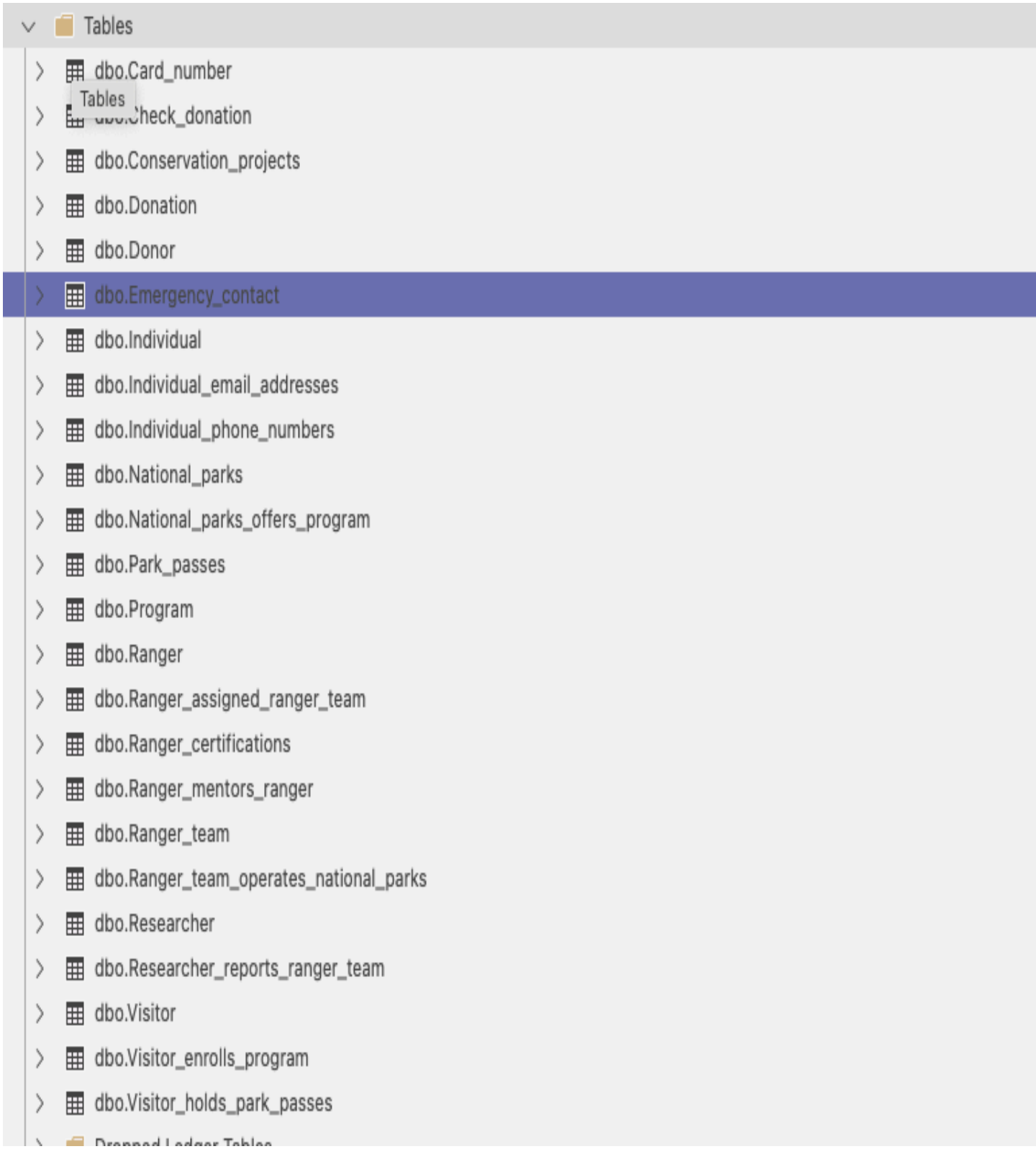
```
        CONSTRAINT CK_ranger_assigned_ranger_team_status
                                                CHECK (status IN ('active', 'inactive',
'on_leave', 'terminated'))
);


CREATE TABLE Ranger_team_operates_national_parks (
        Team_id                                 VARCHAR(50) NOT NULL,
        Park_name                               VARCHAR(200) NOT NULL,
        CONSTRAINT PK_ranger_team_operates_national_parks
                                                PRIMARY KEY (team_id, park_name),
        CONSTRAINT FK_ranger_team_operates_national_parks_team
                                                FOREIGN KEY (team_id)
                REFERENCES ranger_team(team_id)
                                                ON DELETE CASCADE ON UPDATE
                                        CASCADE,
        CONSTRAINT FK_ranger_team_operates_national_parks_park
                                                FOREIGN KEY (park_name)
                REFERENCES national_parks(name)
                                                ON DELETE CASCADE ON UPDATE
                                        CASCADE
);

CREATE TABLE National_parks_offers_program (
        Park_name                               VARCHAR(200) NOT NULL,
        Program_name                            VARCHAR(200) NOT NULL,
        CONSTRAINT PK_national_parks_offers_program
                                                PRIMARY KEY (park_name,
                                        program_name),
        CONSTRAINT FK_national_parks_offers_program_park
                                                FOREIGN KEY (park_name)
                REFERENCES national_parks(name)
                                                ON DELETE CASCADE ON UPDATE
                                        CASCADE,
        CONSTRAINT FK_national_parks_offers_program_program
                                                FOREIGN KEY (program_name)
                REFERENCES program(program_name)
                                                ON DELETE CASCADE ON UPDATE
                                        CASCADE
);

CREATE TABLE Ranger_mentors_ranger (
        Mentor_id_number                        VARCHAR(50) NOT NULL,
        Mentee_id_number                        VARCHAR(50) NOT NULL,
```

```
    Start_date                                  DATE NOT NULL,
    CONSTRAINT PK_ranger_mentors_ranger
                                                PRIMARY KEY (mentor_id_number,
                                            mentee_id_number),
    CONSTRAINT FK_ranger_mentors_ranger_mentor
                                                FOREIGN KEY (mentor_id_number)
        REFERENCES ranger(id_number)
                                                ON DELETE NO ACTION ON UPDATE
                                            CASCADE,
    CONSTRAINT FK_ranger_mentors_ranger_mentee
                                                FOREIGN KEY (mentee_id_number)
        REFERENCES ranger(id_number)
                                                ON DELETE CASCADE ON UPDATE
                                            CASCADE,
    CONSTRAINT CK_ranger_mentors_ranger_different
                                                CHECK (mentor_id_number <>
                                            mentee_id_number)
);

-- Index to help find data quickly
-- Individual Table indexes
CREATE INDEX IX_individual_date_of_birth ON individual(date_of_birth);
CREATE INDEX IX_individual_last_name ON individual(last_name);
CREATE INDEX IX_individual_city_state ON individual(city, state);

-- Donor indexes
CREATE INDEX IX_donor_preference ON donor(preference);

-- Researcher indexes
CREATE INDEX IX_researcher_research_field ON researcher(research_field);
CREATE INDEX IX_researcher_hire_date ON researcher(hire_date);

-- Visitor indexes
CREATE INDEX IX_visitor_visit_date ON visitor(visit_date);

-- Ranger Team indexes
CREATE INDEX IX_ranger_team_team_leader ON ranger_team(team_leader);
CREATE INDEX IX_ranger_team_formation_date ON ranger_team(formation_date);
CREATE INDEX IX_ranger_team_focus_date ON ranger_team(focus_date);

-- Donation indexes
CREATE INDEX IX_donation_donor_id_number ON donation(donor_id_number);
CREATE INDEX IX_donation_date ON donation(date);
CREATE INDEX IX_donation_amount ON donation(amount);
```

```
-- Check Donation indexes
CREATE INDEX IX_check_donation_check_number ON check_donation(check_number);

-- Card Number indexes
CREATE INDEX IX_card_number_expiration_date ON card_number(expiration_date);

-- National Parks indexes
CREATE INDEX IX_national_parks_establishment_date ON
national_parks(establishment_date);
CREATE INDEX IX_national_parks_capacity ON national_parks(capacity);

-- Program indexes
CREATE INDEX IX_program_type ON program(type);
CREATE INDEX IX_program_start_date ON program(start_date);

-- Conservation Projects indexes
CREATE INDEX IX_conservation_projects_park_name ON conservation_projects(park_name);
CREATE INDEX IX_conservation_projects_start_date ON conservation_projects(start_date);
CREATE INDEX IX_conservation_projects_budget ON conservation_projects(budget);

-- Park Passes indexes
CREATE INDEX IX_park_passes_expiration_date ON park_passes(expiration_date);
CREATE INDEX IX_park_passes_type ON park_passes(type);

-- Indexes on junction tables - these help with reverse lookups
CREATE INDEX IX_visitor_holds_park_passes_pass_id ON
visitor_holds_park_passes(pass_id);
CREATE INDEX IX_visitor_holds_park_passes_visitor_id_number ON
visitor_holds_park_passes(visitor_id_number);

CREATE INDEX IX_visitor_enrolls_program_visitor_id_number ON
visitor_enrolls_program(visitor_id_number);
CREATE INDEX IX_visitor_enrolls_program_program_name ON
visitor_enrolls_program(program_name);

CREATE INDEX IX_researcher_reports_ranger_team_researcher_id_number ON
researcher_reports_ranger_team(researcher_id_number);
CREATE INDEX IX_researcher_reports_ranger_team_team_id ON
researcher_reports_ranger_team(team_id);
CREATE INDEX IX_researcher_reports_ranger_team_date ON
researcher_reports_ranger_team(date);
```

```
CREATE INDEX IX_ranger_assigned_ranger_team_ranger_id_number ON
ranger_assigned_ranger_team(ranger_id_number);
CREATE INDEX IX_ranger_assigned_ranger_team_team_id ON
ranger_assigned_ranger_team(team_id);
CREATE INDEX IX_ranger_assigned_ranger_team_start_date ON
ranger_assigned_ranger_team(start_date);
CREATE INDEX IX_ranger_assigned_ranger_team_status ON
ranger_assigned_ranger_team(status);
CREATE INDEX IX_ranger_assigned_ranger_team_years_of_service ON
ranger_assigned_ranger_team(years_of_service);

CREATE INDEX IX_ranger_team_operates_national_parks_team_id ON
ranger_team_operates_national_parks(team_id);
CREATE INDEX IX_ranger_team_operates_national_parks_park_name ON
ranger_team_operates_national_parks(park_name);

CREATE INDEX IX_national_parks_offers_program_park_name ON
national_parks_offers_program(park_name);
CREATE INDEX IX_national_parks_offers_program_program_name ON
national_parks_offers_program(program_name);

CREATE INDEX IX_ranger_mentors_ranger_mentor_id_number ON
ranger_mentors_ranger(mentor_id_number);
CREATE INDEX IX_ranger_mentors_ranger_mentee_id_number ON
ranger_mentors_ranger(mentee_id_number);
CREATE INDEX IX_ranger_mentors_ranger_start_date ON ranger_mentors_ranger(start_date);

-- Indexes on multi-valued attribute tables
CREATE INDEX IX_individual_phone_numbers_id_number ON
individual_phone_numbers(id_number);

CREATE INDEX IX_individual_email_addresses_id_number ON
individual_email_addresses(id_number);

CREATE INDEX IX_emergency_contact_id_number ON emergency_contact(id_number);

CREATE INDEX IX_ranger_certifications_id_number ON ranger_certifications(id_number);
```

OUTPUT:


Tables
dbo.Card_number
Tables
dbo.Check_donation
dbo.Conservation_projects
dbo.Donation
dbo.Donor
dbo.Emergency_contact
dbo.Individual
dbo.Individual_email_addresses
dbo.Individual_phone_numbers
dbo.National_parks
dbo.National_parks_offers_program
dbo.Park_passes
dbo.Program
dbo.Ranger
dbo.Ranger_assigned_ranger_team
dbo.Ranger_certifications
dbo.Ranger_mentors_ranger
dbo.Ranger_team
dbo.Ranger_team_operates_national_parks
dbo.Researcher
dbo.Researcher_reports_ranger_team
dbo.Visitor
dbo.Visitor_enrolls_program
dbo.Visitor_holds_park_passes

Results Messages

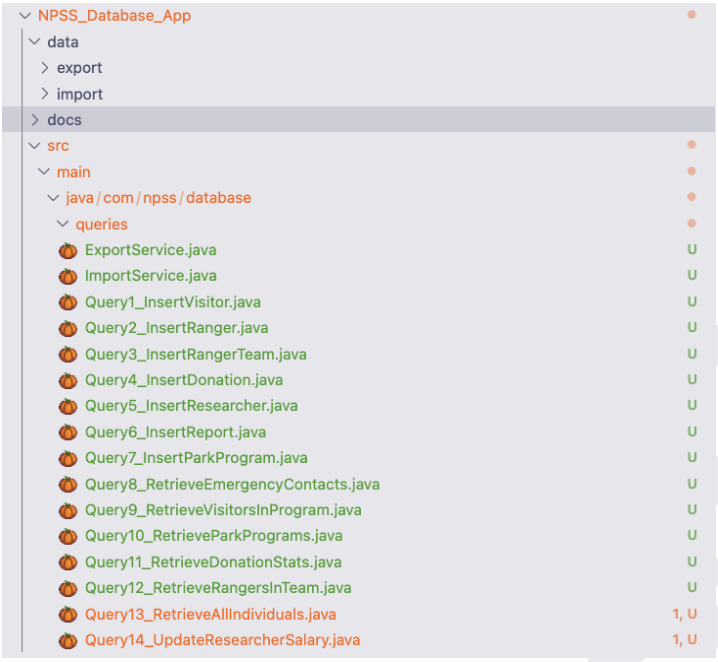
	Schema	Table Name	Type
1	dbo	Card_number	BASE TABLE
2	dbo	Check_donation	BASE TABLE
3	dbo	Conservation_projects	BASE TABLE
4	dbo	Donation	BASE TABLE
5	dbo	Donor	BASE TABLE
6	dbo	Emergency_contact	BASE TABLE
7	dbo	Individual	BASE TABLE
8	dbo	Individual_email_addresses	BASE TABLE
9	dbo	Individual_phone_numbers	BASE TABLE
10	dbo	National_parks	BASE TABLE

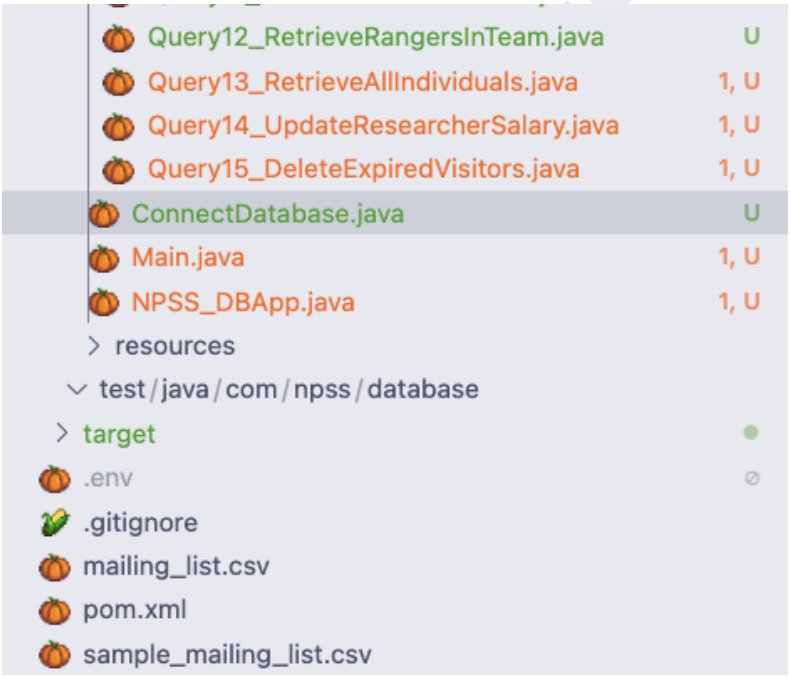
	Schema	Table Name	Created Date	Last Modified	Column Count
1	dbo	Card_number	2025-11-06 23:38:49.200	2025-11-06 23:38:49.200	4
2	dbo	Check_donation	2025-11-06 23:38:49.193	2025-11-06 23:38:49.193	2
3	dbo	Conservation_projects	2025-11-06 23:38:49.137	2025-11-06 23:38:49.137	5
4	dbo	Donation	2025-11-06 23:38:49.187	2025-11-06 23:38:49.200	5
5	dbo	Donor	2025-11-06 23:38:49.143	2025-11-06 23:38:49.190	2
6	dbo	Emergency_contact	2025-11-06 23:38:49.180	2025-11-06 23:38:49.180	4
7	dbo	Individual	2025-11-06 23:38:49.107	2025-11-06 23:38:49.217	11
8	dbo	Individual_email_addresses	2025-11-06 23:38:49.213	2025-11-06 23:38:49.213	2
9	dbo	Individual_phone_numbers	2025-11-06 23:38:49.210	2025-11-06 23:38:49.210	2
10	dbo	National_parks	2025-11-06 23:38:49.113	2025-11-07 00:08:31.250	7

	Total Tables
1	24

Task 5. Script File Showing the Entire Java Program and Its Successful Compilation

5.1.0 Java Program Code:





5.1.1 Connection To Database

```
package com.npss.database;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.net.URLDecoder;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.HashMap;
import java.util.Map;

public class ConnectDatabase {
   private static final String DB_URL = "DB_URL";
   private static final String DB_USERNAME = "DB_USERNAME";
   private static final String DB_PASSWORD  = "DB_PASSWORD";
   private static Map<String, String> envCache = null;

   /**
    * Loads environment variables from .env file if it exists
    * @return Map of environment variables from .env file
    */
   private static Map<String, String> loadEnvFile() {
      if (envCache != null) {
         return envCache;
      }

      Map<String, String> envMap = new HashMap<>();
      try {
         // Try multiple locations to find .env file
         Path envPath = null;

         // 1. Try current working directory
         Path currentDir = Paths.get(System.getProperty("user.dir"));
         envPath = currentDir.resolve(".env");

         // 2. If not found, try relative to the class location (for IDE runs)
         if (!envPath.toFile().exists()) {
            try {
               java.net.URL classUrl = ConnectDatabase.class.getResource("/");
               if (classUrl != null) {
                  String classPath = classUrl.getPath();
                  // Handle URL-encoded paths
```

```
                if (classPath.startsWith("file:")) {
                    classPath = classPath.substring(5);
                }
                // Decode URL encoding
                classPath = URLDecoder.decode(classPath, "UTF-8");
                Path classDir = Paths.get(classPath);
                // Navigate up from target/classes to project root
                if (classDir.toString().contains("target")) {
                    envPath = classDir.getParent().getParent().resolve(".env");
                } else {
                    envPath = classDir.resolve(".env");
                }
            }
        } catch (Exception e) {
            // If we can't determine class location, continue with other paths
        }
    }

    // 3. Try relative paths from current directory
    if (envPath == null || !envPath.toFile().exists()) {
        envPath = Paths.get(".env");
    }
    if (!envPath.toFile().exists()) {
        envPath = Paths.get("..", ".env");
    }
    if (!envPath.toFile().exists()) {
        envPath = Paths.get("..", "..", ".env");
    }

    // 4. Try in NPSS_Database_App directory
    if (!envPath.toFile().exists()) {
        Path projectRoot = currentDir;
        if (currentDir.getFileName().toString().equals("NPSS_Database_App")) {
            envPath = projectRoot.resolve(".env");
        } else {
            envPath = projectRoot.resolve("NPSS_Database_App").resolve(".env");
        }
    }

    if (envPath != null && envPath.toFile().exists()) {
        try (BufferedReader reader = new BufferedReader(new FileReader(envPath.toFile())))
{
            String line;
            while ((line = reader.readLine()) != null) {
```

```
                line = line.trim();
                // Skip empty lines and comments
                if (line.isEmpty() || line.startsWith("#")) {
                    continue;
                }
                int equalsIndex = line.indexOf('=');
                if (equalsIndex > 0) {
                    String key = line.substring(0, equalsIndex).trim();
                    String value = line.substring(equalsIndex + 1).trim();
                    if (!key.isEmpty() && !value.isEmpty()) {
                        envMap.put(key, value);
                    }
                }
            }
        }
    }
    } catch (IOException e) {
        // If .env file doesn't exist or can't be read, that's okay
        // We'll fall back to system environment variables
    }

    envCache = envMap;
    return envMap;
}

/**
 * Gets configuration value from environment variables (.env file or system env)
 * Checks .env file first, then falls back to system environment variables
 * @param envKey The env variable key
 * @param description Description of the error messages
 * @return The env variable value
 * @throws SQLException If the environment variable is not set
 */
private static String getEnvValue(String envKey, String description)throws SQLException{
    // First, try to load from .env file
    Map<String, String> envFile = loadEnvFile();
    String value = envFile.get(envKey);

    // If not found in .env file, try system environment variables
    if (value == null || value.trim().isEmpty()) {
        value = System.getenv(envKey);
    }

    if(value == null || value.trim().isEmpty() ){
```

```
            throw new SQLException(
                String.format("Missing required environment variable: %s (%s). Please set this in your
.env file or system environment variables.", envKey, description)
            );
        }
        return value.trim();
    }

    /**
     * Creates and returns a secure database connection using environment variables
     * @return Connection object to the database
     * @throws SQLException If the environment variable is not set
     */
    public static Connection getConnection() throws SQLException{
        try{
            Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");

            String url = getEnvValue(DB_URL, "Database connection URL");
            String username = getEnvValue(DB_USERNAME, "Database username");
            String password = getEnvValue(DB_PASSWORD, "Database password");

            // Create Connection
            Connection connection = DriverManager.getConnection(url,username,password);

            if(connection != null && !connection.isClosed()){
                System.out.println("Database connect successfully!");
                return connection;
            }else{
                throw new SQLException("Failed to connect the database. Try again.");
            }

        }catch(ClassNotFoundException e){
            throw new SQLException("JDBC Driver not found. Make sure mssql-jdbc is in your
classpath.", e);
        }catch(SQLException e){
            System.err.println("SQL Exception occurred:");
            System.err.println("  Message: " + e.getMessage());
            if (e.getSQLState() != null) {
                System.err.println("SQL State: " + e.getSQLState());
            }
            if (e.getErrorCode() != 0) {
                System.err.println("Error Code: " + e.getErrorCode());
            }
            throw e;
```

```
        }
    }

    /**
     * Closes a database connection
     * @param connection The connection to close
     */
    public static void closeConnection(Connection connection){
        if(connection != null){
            try {
                if(!connection.isClosed()){
                    connection.close();
                    System.out.println("Database connection closed!");
                }
            } catch (SQLException e) {
                System.err.println("Error closing database connection: " + e.getMessage());
            }
        }
    }

}
```

5.1.2 National Park Service System (NPSS) Database Application

```
package com.npss.database;
import java.sql.Connection;
import java.sql.SQLException;
import java.util.Scanner;
import com.npss.database.ConnectDatabase;
import com.npss.database.queries.*;
/**
* National Park Service System (NPSS) Database Application
*
* This application provides a menu-driven interface to interact with the NPSS database
* using JDBC and Azure SQL Database. It supports 15 queries, import/export functionality,
* and proper error handling.
*
* @author Astra Nguyen
* @version 1.0
*/

public class NPSS_DBApp {
    private Connection connection;
    private Scanner scanner;
```

```
    /*
     * Initial Constructor
     */
    public NPSS_DBApp(){
        scanner = new Scanner(System.in);
    }

    /**
     * Check to see if connect to Azure SQL Database
     * @return true if connection is successful, false otherwise
     * @throws SQLException If the environment variable is not set
     */
    public boolean connectToDatabase()throws SQLException{
        try{
            this.connection = ConnectDatabase.getConnection();
            if (this.connection != null && !this.connection.isClosed()) {
                System.out.println("Database connected successfully!");
                return true;
            }
        }catch(SQLException e){
            System.err.println("SQL Exception occurred:");
            System.err.println("  Message: " + e.getMessage());
            if (e.getSQLState() != null) {
                System.err.println("SQL State: " + e.getSQLState());
            }
            if (e.getErrorCode() != 0) {
                System.err.println("Error Code: " + e.getErrorCode());
            }
            return false;
        }
        return false;
    }

    /**
     * Displaying the NPPS Menu
     */
    public void displayMenu(){
        System.out.println("\n" + "=".repeat(60));
        System.out.println("WELCOME TO THE NATIONAL PARK SERVICE SYSTEM
DATABASE(NPSS)!");
        System.out.println("=".repeat(60));
        System.out.println("(1)  Insert a new visitor into the database and associate them with one
or more park programs");
```

```
        System.out.println("(2)  Insert a new ranger into the database and assign them to a ranger
team");
        System.out.println("(3)  Insert a new ranger team into the database and set its leader");
        System.out.println("(4)  Insert a new donation from a donor");
        System.out.println("(5)  Insert a new researcher into the database and associate them with
one or more ranger teams");
        System.out.println("(6)  Insert a report submitted by a ranger team to a researcher");
        System.out.println("(7)  Insert a new park program into the database for a specific park");
        System.out.println("(8)  Retrieve the names and contact information of all emergency
contacts for a specific person");
        System.out.println("(9)  Retrieve the list of visitors enrolled in a specific park program,
including their accessibility needs");
        System.out.println("(10) Retrieve all park programs for a specific park that started after a
given date");
        System.out.println("(11) Retrieve the total and average donation amount received in a
month from all anonymous donors");
        System.out.println("(12) Retrieve the list of rangers in a team, including their certifications,
years of service and their role in the team");
        System.out.println("(13) Retrieve the names, IDs, contact information, and newsletter
subscription status of all individuals in the database");
        System.out.println("(14) Update the salary of researchers overseeing more than one ranger
team by a 3% increase");
        System.out.println("(15) Delete visitors who have not enrolled in any park programs and
whose park passes have expired");
        System.out.println("(16) Import: Enter new teams from a data file until the file is empty");
        System.out.println("(17) Export: Retrieve names and mailing addresses of all people on the
mailing list");
        System.out.println("(18) Quit");
        System.out.println("=".repeat(60));
        System.out.print("Please select an option (1-18): ");
    }

    /**
     * Processes the user's menu choice and executes the corresponding action
     * @param choice The menu option selected by the user (1-18)
     */
    public void processMenuChoice(int choice){
        try {
            switch(choice){
                case 1:
                    Query1_InsertVisitor query1 = new Query1_InsertVisitor(connection, scanner);
                    query1.execute();
                    break;
                case 2:
```

```
                Query2_InsertRanger query2 = new Query2_InsertRanger(connection, scanner);
                query2.execute();
                break;
            case 3:
                Query3_InsertRangerTeam query3 = new Query3_InsertRangerTeam(connection,
scanner);
                query3.execute();
                break;
            case 4:
                Query4_InsertDonation query4 = new Query4_InsertDonation(connection, scanner);
                query4.execute();
                break;
            case 5:
                Query5_InsertResearcher query5 = new Query5_InsertResearcher(connection,
scanner);
                query5.execute();
                break;
            case 6:
                Query6_InsertReport query6 = new Query6_InsertReport(connection, scanner);
                query6.execute();
                break;
            case 7:
                Query7_InsertParkProgram query7 = new Query7_InsertParkProgram(connection,
scanner);
                query7.execute();
                break;
            case 8:
                Query8_RetrieveEmergencyContacts query8 = new
Query8_RetrieveEmergencyContacts(connection, scanner);
                query8.execute();
                break;
            case 9:
                Query9_RetrieveVisitorsInProgram query9 = new
Query9_RetrieveVisitorsInProgram(connection, scanner);
                query9.execute();
                break;
            case 10:
                Query10_RetrieveParkPrograms query10 = new
Query10_RetrieveParkPrograms(connection, scanner);
                query10.execute();
                break;
            case 11:
                Query11_RetrieveDonationStats query11 = new
Query11_RetrieveDonationStats(connection, scanner);
```

```
                query11.execute();
                break;
            case 12:
                Query12_RetrieveRangersInTeam query12 = new
Query12_RetrieveRangersInTeam(connection, scanner);
                query12.execute();
                break;
            case 13:
                Query13_RetrieveAllIndividuals query13 = new
Query13_RetrieveAllIndividuals(connection, scanner);
                query13.execute();
                break;
            case 14:
                Query14_UpdateResearcherSalary query14 = new
Query14_UpdateResearcherSalary(connection, scanner);
                query14.execute();
                break;
            case 15:
                Query15_DeleteExpiredVisitors query15 = new
Query15_DeleteExpiredVisitors(connection, scanner);
                query15.execute();
                break;
            case 16:
                ImportService importService = new ImportService(connection, scanner);
                importService.execute();
                break;
            case 17:
                ExportService exportService = new ExportService(connection, scanner);
                exportService.execute();
                break;
            default:
                System.out.println("\nInvalid choice! Please select an option between 1-18.");
                break;
        }
    } catch (SQLException e) {
        System.err.println("\nDatabase error occurred:");
        System.err.println("  Message: " + e.getMessage());
        if (e.getSQLState() != null) {
            System.err.println("  SQL State: " + e.getSQLState());
        }
        if (e.getErrorCode() != 0) {
            System.err.println("  Error Code: " + e.getErrorCode());
        }
    } catch (Exception e) {
```

```
            System.err.println("\nAn unexpected error occurred: " + e.getMessage());
            e.printStackTrace();
        }

        // Pause before showing menu again
        System.out.println("\nPress Enter to continue...");
        scanner.nextLine();
    }

    /**
     * Close the database connection and close input scanner
     */
    public void closeConnection(){
        if(connection != null){
            ConnectDatabase.closeConnection(connection);
            connection = null;
        }
        if (scanner != null) {
            scanner.close();
        }
    }

    /**
     * Main application loop that display menu and handle user input
     */
    public void run(){
        boolean running = true;
        while(running){
            displayMenu();

            try{
                if(scanner.hasNextInt()){
                    int choice = scanner.nextInt();
                    scanner.nextLine();

                    if(choice == 18){
                        running = false;
                        System.out.println("\nThank you for using NPSS Database System. Goodbye!");
                    }else if(choice >= 1 && choice <= 17){
                        processMenuChoice(choice);
                    }else{
                        System.out.println("\nInvalid choice! Please select an option between 1-18.");
                        System.out.println("Press Enter to continue...");
                        scanner.nextLine();
```

```
            }
        }else{
            System.out.println("\nInvalid input! Please enter a number between 1-18.");
            scanner.nextLine(); // Clear invalid input
        }
      }catch(Exception e){
        System.err.println("\nAn error occurred: " + e.getMessage());
        scanner.nextLine();
      }
    }
    closeConnection();
  }
}
```

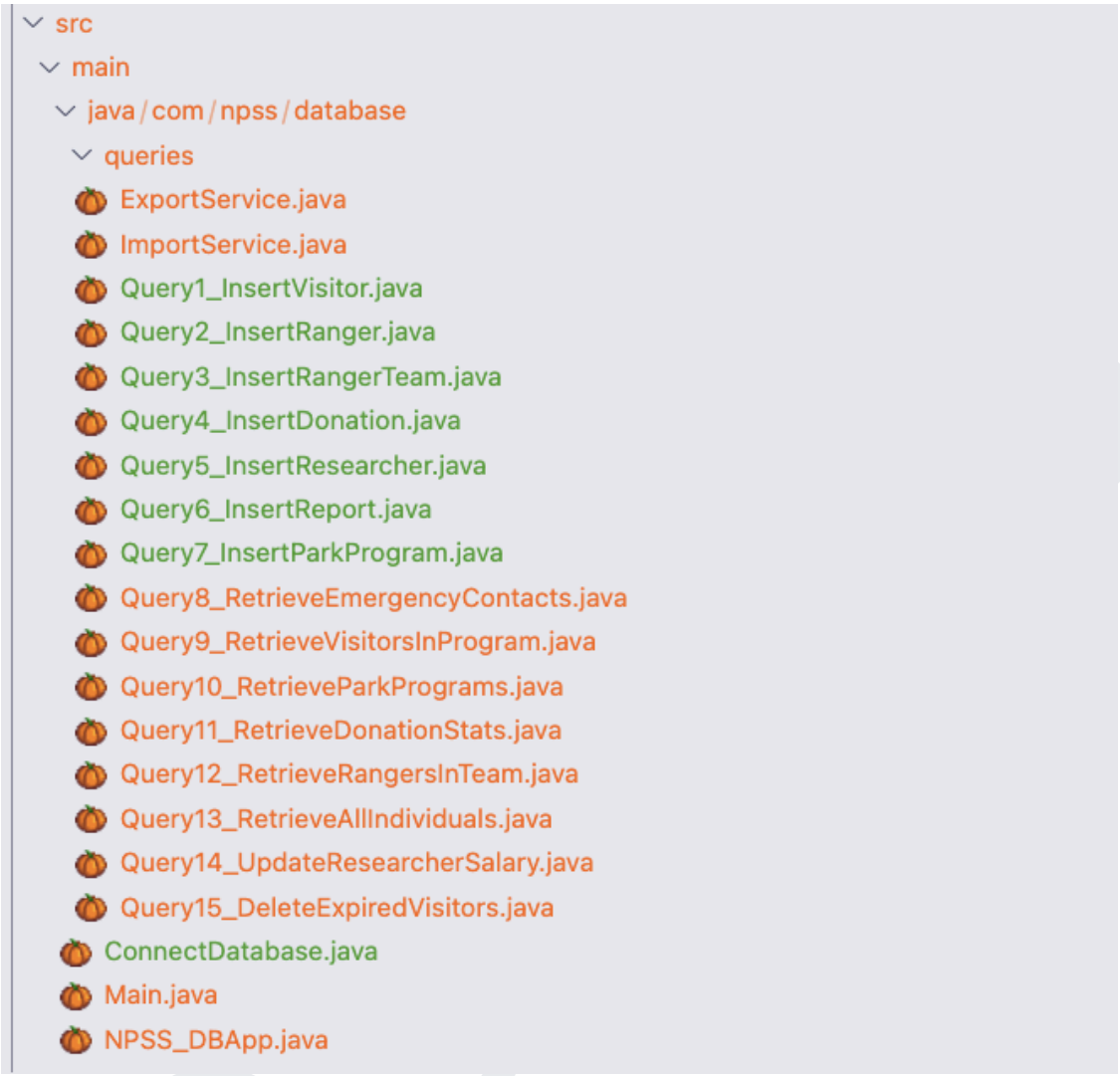
5.1.3 Java Pom & Its Dependencies

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
      http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <!-- Project Information -->
  <groupId>com.npss</groupId>
  <artifactId>npss-database-app</artifactId>
  <version>1.0.0</version>
  <packaging>jar</packaging>

  <name>NPSS Database Application</name>
  <description>National Park Service System Database Management Application</description>

  <!-- Java Version Configuration -->
  <properties>
    <maven.compiler.source>11</maven.compiler.source>
    <maven.compiler.target>11</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

    <!-- Dependency Versions -->
    <sqlserver.jdbc.version>12.4.2.jre11</sqlserver.jdbc.version>
    <mysql.jdbc.version>8.0.33</mysql.jdbc.version>
    <postgresql.jdbc.version>42.6.0</postgresql.jdbc.version>
    <junit.version>5.10.0</junit.version>
    <slf4j.version>2.0.9</slf4j.version>
    <logback.version>1.4.11</logback.version>
```

```
</properties>

<!-- Dependencies -->
<dependencies>
    <!--SQL Server JDBC Driver -->
    <dependency>
        <groupId>com.microsoft.sqlserver</groupId>
        <artifactId>mssql-jdbc</artifactId>
        <version>12.4.2.jre11</version>
    </dependency>

    <!-- Logging Framework: SLF4J API -->
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-api</artifactId>
        <version>${slf4j.version}</version>
    </dependency>

    <!-- Logging Implementation: Logback -->
    <dependency>
        <groupId>ch.qos.logback</groupId>
        <artifactId>logback-classic</artifactId>
        <version>${logback.version}</version>
    </dependency>

    <!-- JUnit 5 for Testing -->
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-api</artifactId>
        <version>${junit.version}</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-engine</artifactId>
        <version>${junit.version}</version>
        <scope>test</scope>
    </dependency>
</dependencies>

<!-- Build Configuration -->
<build>
    <plugins>
        <!-- Maven Compiler Plugin -->
```

```
        <plugin>
          <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-compiler-plugin</artifactId>
          <version>3.11.0</version>
          <configuration>
            <source>11</source>
            <target>11</target>
            <encoding>UTF-8</encoding>
          </configuration>
        </plugin>

        <!-- Maven Surefire Plugin (for running tests) -->
        <plugin>
          <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-surefire-plugin</artifactId>
          <version>3.1.2</version>
        </plugin>

        <!-- Maven Resources Plugin -->
        <plugin>
          <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-resources-plugin</artifactId>
          <version>3.3.1</version>
          <configuration>
            <encoding>UTF-8</encoding>
          </configuration>
        </plugin>
      </plugins>
    </build>
</project>
```

5.1.4 Queries Files



5.1.4.0 Query 1 - Insert a New Visitor

```
package com.npss.database.queries;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.util.Scanner;
/**
* Query 1: Insert a new visitor into the database and associate them with one or more park programs
*
* @author Astra Nguyen
* @version 1.0
```

```
*/

public class Query1_InsertVisitor {
  private Connection connection;
  private Scanner scanner;

  public Query1_InsertVisitor(Connection connection, Scanner scanner) {
     this.connection = connection;
     this.scanner = scanner;
  }

  /**
   * Executes Query 1: Insert a new visitor into the database
   * and associate them with one or more park programs
   *
   * @throws SQLException if a database error occurs
   */
  public void execute() throws SQLException {
     System.out.println("\n[Query 1] Insert a new visitor into the database and associate them
with one or more park programs");

     try{
        // Get user input
        System.out.print("Enter visitor ID number: ");
        String idNumber = scanner.nextLine().trim();

        System.out.print("Enter first name: ");
        String firstName = scanner.nextLine().trim();

        System.out.print("Enter last name: ");
        String lastName = scanner.nextLine().trim();

        System.out.print("Enter gender (M/F/O): ");
        String gender = scanner.nextLine().trim();

        System.out.print("Enter street address: ");
        String street = scanner.nextLine().trim();

        System.out.print("Enter city: ");
        String city = scanner.nextLine().trim();

        System.out.print("Enter state: ");
        String state = scanner.nextLine().trim();
```

```
        System.out.print("Enter postal code: ");
        String postalCode = scanner.nextLine().trim();

        System.out.print("Enter date of birth (YYYY-MM-DD): ");
        String dateOfBirth = scanner.nextLine().trim();

        System.out.print("Subscribe to newsletter? (true/false): ");
        boolean newsletterStatus = Boolean.parseBoolean(scanner.nextLine().trim());

        System.out.print("Enter visit date (YYYY-MM-DD) or press Enter for NULL: ");
        String visitDate = scanner.nextLine().trim();
        if (visitDate.isEmpty()) visitDate = null;

        System.out.print("Enter accessibility needs (or press Enter for NULL): ");
        String accessibilityNeeds = scanner.nextLine().trim();
        if (accessibilityNeeds.isEmpty()) accessibilityNeeds = null;

        // SQL queries
        String insertIndividualSQL =
            "INSERT INTO Individual(id_number, first_name, last_name, gender, street, city, state,
" +
            "postal_code, date_of_birth, newsletter_status) " +
            "VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)"; // ? prevents SQL injection, handles types safely
like null

        String insertVisitorSQL =
            "INSERT INTO Visitor(id_number, visit_date, accessibility_needs) " +
            "VALUES (?, ?, ?)";

        // Set parameters
        PreparedStatement parameterIndividual =
connection.prepareStatement(insertIndividualSQL);
        parameterIndividual.setString(1, idNumber);
        parameterIndividual.setString(2, firstName);
        parameterIndividual.setString(3, lastName);
        parameterIndividual.setString(4, gender);
        parameterIndividual.setString(5, street);
        parameterIndividual.setString(6, city);
        parameterIndividual.setString(7, state);
        parameterIndividual.setString(8, postalCode);
        parameterIndividual.setDate(9, java.sql.Date.valueOf(dateOfBirth));
        parameterIndividual.setBoolean(10, newsletterStatus);

        PreparedStatement parameterVisitor = connection.prepareStatement(insertVisitorSQL);
```

```
        parameterVisitor.setString(1, idNumber);
        if (visitDate != null) {
            parameterVisitor.setDate(2, java.sql.Date.valueOf(visitDate));
        } else {
            parameterVisitor.setNull(2, java.sql.Types.DATE);
        }
        parameterVisitor.setString(3, accessibilityNeeds);

        // Execute
        connection.setAutoCommit(false); // Start transaction

        try {
            // Execute inserts -- runs the sql insert
            parameterIndividual.executeUpdate();
            parameterVisitor.executeUpdate();

            // Associate with park programs
            System.out.print("How many park programs to enroll? (0 or more): ");
            int programCount = Integer.parseInt(scanner.nextLine().trim());

            if (programCount > 0) {
                String enrollProgramSQL =
                    "INSERT INTO Visitor_enrolls_program(visitor_id_number, program_name) " +
                    "VALUES (?, ?)";
                PreparedStatement pstmtEnroll =
connection.prepareStatement(enrollProgramSQL);

                for (int i = 0; i < programCount; i++) {
                    System.out.print("Enter program name " + (i + 1) + ": ");
                    String programName = scanner.nextLine().trim();
                    pstmtEnroll.setString(1, idNumber);
                    pstmtEnroll.setString(2, programName);
                    pstmtEnroll.executeUpdate();
                }
                pstmtEnroll.close();
            }

            connection.commit(); // Commit transaction
            System.out.println("Visitor inserted successfully!");

        } catch (SQLException e) {
            connection.rollback(); // Rollback on error
            throw e;
        } finally {
```

```
            connection.setAutoCommit(true); // Reset auto-commit
            parameterIndividual.close();
            parameterVisitor.close();
        }

    } catch (SQLException e) {
        System.err.println("Database error: " + e.getMessage());
        if (e.getSQLState() != null) {
            System.err.println("SQL State: " + e.getSQLState());
        }
        throw e;
    } catch (Exception e) {
        System.err.println("Error: " + e.getMessage());
    }
  }
}
```

5.1.4.1 Query 2 - Insert a New Ranger

```
package com.npss.database.queries;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.util.Scanner;

/**
* Query 2: Insert a new ranger into the database and assign them to a ranger team
*
* @author Astra Nguyen
* @version 1.0
*/
public class Query2_InsertRanger {
  private Connection connection;
  private Scanner scanner;

  public Query2_InsertRanger(Connection connection, Scanner scanner) {
      this.connection = connection;
      this.scanner = scanner;
  }

  /**
   * Executes Query 2: Insert a new ranger into the database and assign them to a ranger team
   *
   * @throws SQLException if a database error occurs
```

```
   */
  public void execute() throws SQLException {
      System.out.println("\n[Query 2] Insert a new ranger into the database and assign them to a
ranger team");

      try {
          // User's input
          System.out.print("Enter ranger ID number: ");
          String idNumber = scanner.nextLine().trim();

          System.out.print("Enter first name: ");
          String firstName = scanner.nextLine().trim();

          System.out.print("Enter last name: ");
          String lastName = scanner.nextLine().trim();

          System.out.print("Enter gender (M/F/O): ");
          String gender = scanner.nextLine().trim();

          System.out.print("Enter street address: ");
          String street = scanner.nextLine().trim();

          System.out.print("Enter city: ");
          String city = scanner.nextLine().trim();

          System.out.print("Enter state: ");
          String state = scanner.nextLine().trim();

          System.out.print("Enter postal code: ");
          String postalCode = scanner.nextLine().trim();

          System.out.print("Enter date of birth (YYYY-MM-DD): ");
          String dateOfBirth = scanner.nextLine().trim();

          System.out.print("Subscribe to newsletter? (true/false): ");
          boolean newsletterStatus = Boolean.parseBoolean(scanner.nextLine().trim());

          // Team assignment information
          System.out.print("Enter team ID to assign ranger to: ");
          String teamId = scanner.nextLine().trim();

          System.out.print("Enter start date (YYYY-MM-DD): ");
          String startDate = scanner.nextLine().trim();
```

```
        System.out.print("Enter status (e.g., Active, On Leave, etc.): ");
        String status = scanner.nextLine().trim();

        // Optional certifications
        System.out.print("How many certifications? (0 or more): ");
        int certCount = Integer.parseInt(scanner.nextLine().trim());

        // SQL queries
        String insertIndividualSQL =
            "INSERT INTO Individual(id_number, first_name, last_name, gender, street, city, state,
" +
            "postal_code, date_of_birth, newsletter_status) " +
            "VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";

        String insertRangerSQL =
            "INSERT INTO Ranger(id_number) " +
            "VALUES (?)";

        String assignTeamSQL =
            "INSERT INTO Ranger_assigned_ranger_team(ranger_id_number, team_id,
start_date, status) " +
            "VALUES (?, ?, ?, ?)";

        PreparedStatement pstmtIndividual = connection.prepareStatement(insertIndividualSQL);
        pstmtIndividual.setString(1, idNumber);
        pstmtIndividual.setString(2, firstName);
        pstmtIndividual.setString(3, lastName);
        pstmtIndividual.setString(4, gender);
        pstmtIndividual.setString(5, street);
        pstmtIndividual.setString(6, city);
        pstmtIndividual.setString(7, state);
        pstmtIndividual.setString(8, postalCode);
        pstmtIndividual.setDate(9, java.sql.Date.valueOf(dateOfBirth));
        pstmtIndividual.setBoolean(10, newsletterStatus);

        PreparedStatement pstmtRanger = connection.prepareStatement(insertRangerSQL);
        pstmtRanger.setString(1, idNumber);

        PreparedStatement pstmtAssignTeam = connection.prepareStatement(assignTeamSQL);
        pstmtAssignTeam.setString(1, idNumber);
        pstmtAssignTeam.setString(2, teamId);
        pstmtAssignTeam.setDate(3, java.sql.Date.valueOf(startDate));
        pstmtAssignTeam.setString(4, status);
```

```
    // Executing
    connection.setAutoCommit(false); // Start transaction

    try {
        pstmtIndividual.executeUpdate();
        pstmtRanger.executeUpdate();
        pstmtAssignTeam.executeUpdate();

        // Add certifications if any
        if (certCount > 0) {
            String insertCertSQL =
                "INSERT INTO Ranger_certifications(id_number, certification) " +
                "VALUES (?, ?)";
            PreparedStatement pstmtCert = connection.prepareStatement(insertCertSQL);

            for (int i = 0; i < certCount; i++) {
                System.out.print("Enter certification " + (i + 1) + ": ");
                String certification = scanner.nextLine().trim();
                pstmtCert.setString(1, idNumber);
                pstmtCert.setString(2, certification);
                pstmtCert.executeUpdate();
            }
            pstmtCert.close();
        }

        connection.commit(); // Commit transaction
        System.out.println("Ranger inserted and assigned to team successfully!");

    } catch (SQLException e) {
        connection.rollback(); // Rollback on error
        throw e; // Re-throw to be caught by outer catch
    } finally {
        connection.setAutoCommit(true); // Reset auto-commit
        pstmtIndividual.close();
        pstmtRanger.close();
        pstmtAssignTeam.close();
    }

} catch (SQLException e) {
    System.err.println("Database error: " + e.getMessage());
    if (e.getSQLState() != null) {
        System.err.println("SQL State: " + e.getSQLState());
    }
    throw e;
```

```
        } catch (Exception e) {
            System.err.println("Error: " + e.getMessage());
        }
    }
}
```

5.1.4.2 Query 3 - Insert a New Ranger Team

```
package com.npss.database.queries;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.util.Scanner;

/**
* Query 3: Insert a new ranger team into the database and set its leader
*
* @author Astra Nguyen
* @version 1.0
*/
public class Query3_InsertRangerTeam {
    private Connection connection;
    private Scanner scanner;

    public Query3_InsertRangerTeam(Connection connection, Scanner scanner) {
        this.connection = connection;
        this.scanner = scanner;
    }

    /**
     * Executes Query 3: Insert a new ranger team into the database and set its leader
     *
     * @throws SQLException if a database error occurs
     */
    public void execute() throws SQLException {
        System.out.println("\n[Query 3] Insert a new ranger team into the database and set its
leader");

        try {
            // User's input
            System.out.print("Enter team ID: ");
            String teamId = scanner.nextLine().trim();

            System.out.print("Enter formation date (YYYY-MM-DD): ");
```

```
String formationDate = scanner.nextLine().trim();

System.out.print("Enter focus date (YYYY-MM-DD) or press Enter for NULL: ");
String focusDate = scanner.nextLine().trim();
if (focusDate.isEmpty()) focusDate = null;

System.out.print("Enter team leader ID (ranger ID) or press Enter for NULL: ");
String teamLeader = scanner.nextLine().trim();
if (teamLeader.isEmpty()) teamLeader = null;

// SQL queries
String insertTeamSQL =
    "INSERT INTO Ranger_team(team_id, formation_date, focus_date, team_leader) " +
    "VALUES (?, ?, ?, ?)";

// Set parameters
PreparedStatement pstmtTeam = connection.prepareStatement(insertTeamSQL);
pstmtTeam.setString(1, teamId);
pstmtTeam.setDate(2, java.sql.Date.valueOf(formationDate));
if (focusDate != null) {
    pstmtTeam.setDate(3, java.sql.Date.valueOf(focusDate));
} else {
    pstmtTeam.setNull(3, java.sql.Types.DATE);
}
if (teamLeader != null) {
    pstmtTeam.setString(4, teamLeader);
} else {
    pstmtTeam.setNull(4, java.sql.Types.VARCHAR);
}

// Executing
connection.setAutoCommit(false);

try {
    pstmtTeam.executeUpdate();
    connection.commit();
    System.out.println("Ranger team inserted successfully!");

} catch (SQLException e) {
    connection.rollback();
    throw e;
} finally {
    connection.setAutoCommit(true);
    pstmtTeam.close();
```

```
        }

    } catch (SQLException e) {
        System.err.println("Database error: " + e.getMessage());
        if (e.getSQLState() != null) {
            System.err.println("SQL State: " + e.getSQLState());
        }
        throw e;
    } catch (Exception e) {
        System.err.println("Error: " + e.getMessage());
    }
  }
}
```

5.1.4.3 Query 4 - Insert a Donation for a Donor

```
package com.npss.database.queries;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Scanner;

/**
* Query 4: Insert a new donation from a donor
* If the donor doesn't exist, it will be created automatically
*
* @author Astra Nguyen
* @version 1.0
*/
public class Query4_InsertDonation {
  private Connection connection;
  private Scanner scanner;

  public Query4_InsertDonation(Connection connection, Scanner scanner) {
    this.connection = connection;
    this.scanner = scanner;
  }

  // Check if donor exists
  private boolean donorExists(String donorId) throws SQLException {
    String checkSQL = "SELECT id_number FROM Donor WHERE id_number = ?";
    try (PreparedStatement pstmt = connection.prepareStatement(checkSQL)) {
        pstmt.setString(1, donorId);
        try (ResultSet rs = pstmt.executeQuery()) {
```

```
            return rs.next();
        }
    }
}

// Check if individual exists first
private boolean individualExists(String idNumber) throws SQLException {
    String checkSQL = "SELECT id_number FROM Individual WHERE id_number = ?";
    try (PreparedStatement pstmt = connection.prepareStatement(checkSQL)) {
        pstmt.setString(1, idNumber);
        try (ResultSet rs = pstmt.executeQuery()) {
            return rs.next();
        }
    }
}

/**
 * Executes Query 4: Insert a new donation from a donor
 * If the donor doesn't exist, it will be created automatically
 *
 * @throws SQLException if a database error occurs
 */
public void execute() throws SQLException {
    System.out.println("\n[Query 4] Insert a new donation from a donor");

    try {
        // User's input
        System.out.print("Enter donation ID: ");
        String donationId = scanner.nextLine().trim();

        System.out.print("Enter donor ID number: ");
        String donorId = scanner.nextLine().trim();

        // Check if donor exists, if not, collect information to create it
        boolean donorNeedsCreation = !donorExists(donorId);
        boolean individualNeedsCreation = false;

        // Variables to store donor/individual information if needed
        String firstName = null, lastName = null, gender = null;
        String street = null, city = null, state = null, postalCode = null;
        String dateOfBirth = null;
        boolean newsletterStatus = false;
        String preference = null;
```

```
if (donorNeedsCreation) {
    System.out.println("\nDonor '" + donorId + "' does not exist. Creating new donor...");
    System.out.println("Please provide the following information:");

    // Check if Individual exists
    individualNeedsCreation = !individualExists(donorId);

    if (individualNeedsCreation) {
        System.out.println("Individual record not found. Creating Individual first...");

        System.out.print("Enter first name: ");
        firstName = scanner.nextLine().trim();

        System.out.print("Enter last name: ");
        lastName = scanner.nextLine().trim();

        System.out.print("Enter gender (M/F/O): ");
        gender = scanner.nextLine().trim();

        System.out.print("Enter street address: ");
        street = scanner.nextLine().trim();

        System.out.print("Enter city: ");
        city = scanner.nextLine().trim();

        System.out.print("Enter state: ");
        state = scanner.nextLine().trim();

        System.out.print("Enter postal code: ");
        postalCode = scanner.nextLine().trim();

        System.out.print("Enter date of birth (YYYY-MM-DD): ");
        dateOfBirth = scanner.nextLine().trim();

        System.out.print("Subscribe to newsletter? (true/false): ");
        newsletterStatus = Boolean.parseBoolean(scanner.nextLine().trim());
    } else {
        System.out.println("Individual record found for ID: " + donorId);
    }

    // Get donor preference
    System.out.print("Enter donor preference (or press Enter for NULL): ");
    preference = scanner.nextLine().trim();
    if (preference.isEmpty()) preference = null;
```

```
        } else {
            System.out.println("Donor '" + donorId + "' found in database.");
        }

        System.out.print("Enter donation date (YYYY-MM-DD): ");
        String donationDate = scanner.nextLine().trim();

        System.out.print("Enter donation amount: ");
        String amountStr = scanner.nextLine().trim();
        double amount = Double.parseDouble(amountStr);

        System.out.print("Enter campaign name (or press Enter for NULL): ");
        String campaignName = scanner.nextLine().trim();
        if (campaignName.isEmpty()) campaignName = null;

        System.out.print("Enter payment method (check/card): ");
        String paymentMethod = scanner.nextLine().trim().toLowerCase();

        // SQL queries
        String insertIndividualSQL =
            "INSERT INTO Individual(id_number, first_name, last_name, gender, street, city, state,
" +
            "postal_code, date_of_birth, newsletter_status) " +
            "VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";

        String insertDonorSQL = "INSERT INTO Donor(id_number, preference) VALUES (?, ?)";

        String insertDonationSQL =
            "INSERT INTO Donation(donation_id, donor_id_number, date, amount,
campaign_name) " +
            "VALUES (?, ?, ?, ?, ?)";

        // Executing
        connection.setAutoCommit(false);

        try {
            // Create Individual if needed
            if (individualNeedsCreation) {
                try (PreparedStatement pstmt = connection.prepareStatement(insertIndividualSQL))
{
                    pstmt.setString(1, donorId);
                    pstmt.setString(2, firstName);
                    pstmt.setString(3, lastName);
                    pstmt.setString(4, gender);
```

```
                pstmt.setString(5, street);
                pstmt.setString(6, city);
                pstmt.setString(7, state);
                pstmt.setString(8, postalCode);
                pstmt.setDate(9, java.sql.Date.valueOf(dateOfBirth));
                pstmt.setBoolean(10, newsletterStatus);

                int rowsAffected = pstmt.executeUpdate();
                System.out.println("Individual record created successfully! (Rows affected: " +
rowsAffected + ")");
            }
        }

        // Create Donor if needed
        if (donorNeedsCreation) {
            try (PreparedStatement pstmt = connection.prepareStatement(insertDonorSQL)) {
                pstmt.setString(1, donorId);
                if (preference != null) {
                    pstmt.setString(2, preference);
                } else {
                    pstmt.setNull(2, java.sql.Types.VARCHAR);
                }

                int rowsAffected = pstmt.executeUpdate();
                System.out.println("Donor '" + donorId + "' created successfully! (Rows affected: "
+ rowsAffected + ")");
            }
        }

        // Create donation
        try (PreparedStatement pstmtDonation =
connection.prepareStatement(insertDonationSQL)) {
            pstmtDonation.setString(1, donationId);
            pstmtDonation.setString(2, donorId);
            pstmtDonation.setDate(3, java.sql.Date.valueOf(donationDate));
            pstmtDonation.setDouble(4, amount);
            if (campaignName != null) {
                pstmtDonation.setString(5, campaignName);
            } else {
                pstmtDonation.setNull(5, java.sql.Types.VARCHAR);
            }

            int rows1 = pstmtDonation.executeUpdate();
```

```
            // Insert payment method details
            int rows2 = 0;
            if (paymentMethod.equals("check")) {
                System.out.print("Enter check number: ");
                String checkNumber = scanner.nextLine().trim();

                String insertCheckSQL =
                    "INSERT INTO Check_donation(donation_id, check_number) " +
                    "VALUES (?, ?)";
                try (PreparedStatement pstmtCheck =
connection.prepareStatement(insertCheckSQL)) {
                    pstmtCheck.setString(1, donationId);
                    pstmtCheck.setString(2, checkNumber);
                    rows2 = pstmtCheck.executeUpdate();
                }

            } else if (paymentMethod.equals("card")) {
                System.out.print("Enter card type: ");
                String cardType = scanner.nextLine().trim();

                System.out.print("Enter last four digits: ");
                String lastFour = scanner.nextLine().trim();

                System.out.print("Enter expiration date (YYYY-MM-DD): ");
                String expDate = scanner.nextLine().trim();

                String insertCardSQL =
                    "INSERT INTO Card_number(donation_id, card_type, last_four_digits,
expiration_date) " +
                    "VALUES (?, ?, ?, ?)";
                try (PreparedStatement pstmtCard =
connection.prepareStatement(insertCardSQL)) {
                    pstmtCard.setString(1, donationId);
                    pstmtCard.setString(2, cardType);
                    pstmtCard.setString(3, lastFour);
                    pstmtCard.setDate(4, java.sql.Date.valueOf(expDate));
                    rows2 = pstmtCard.executeUpdate();
                }
            }

            connection.commit();
            System.out.println("Donation inserted successfully! (Donation rows: " + rows1 + ",
Payment rows: " + rows2 + ")");
```

```
                // Verify the insert
                String verifySQL = "SELECT d.donation_id, d.amount, dr.id_number as donor_id " +
                                   "FROM Donation d " +
                                   "INNER JOIN Donor dr ON d.donor_id_number = dr.id_number " +
                                   "WHERE d.donation_id = ?";
                try (PreparedStatement verifyStmt = connection.prepareStatement(verifySQL)) {
                    verifyStmt.setString(1, donationId);
                    try (ResultSet rs = verifyStmt.executeQuery()) {
                        if (rs.next()) {
                            System.out.println("VERIFIED: Donation " + donationId + " ($" +
                                               String.format("%.2f", rs.getDouble("amount")) +
                                               ") from donor " + rs.getString("donor_id") + " is in the database");
                        } else {
                            System.out.println("WARNING: Donation " + donationId + " not found after
insert!");
                        }
                    }
                }
            }

        } catch (SQLException e) {
            connection.rollback();
            throw e;
        } finally {
            connection.setAutoCommit(true);
        }

    } catch (SQLException e) {
        System.err.println("Database error: " + e.getMessage());
        if (e.getSQLState() != null) {
            System.err.println("SQL State: " + e.getSQLState());
        }
        throw e;
    } catch (IllegalArgumentException e) {
        System.err.println("Invalid date format: " + e.getMessage());
        System.err.println("Please use YYYY-MM-DD format for dates.");
        throw new SQLException("Invalid input: " + e.getMessage(), e);
    } catch (Exception e) {
        System.err.println("Error: " + e.getMessage());
        e.printStackTrace();
        throw new SQLException("Unexpected error: " + e.getMessage(), e);
    }
  }
}
```

5.1.4.4 Query 5 - Insert a New Researcher

```
package com.npss.database.queries;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.util.Scanner;

/**
* Query 5: Insert a new researcher into the database and associate them with one or more
ranger teams
*
* @author Astra Nguyen
* @version 1.0
*/
public class Query5_InsertResearcher {
  private Connection connection;
  private Scanner scanner;

  public Query5_InsertResearcher(Connection connection, Scanner scanner) {
     this.connection = connection;
     this.scanner = scanner;
  }

  /**
   * Executes Query 5: Insert a new researcher into the database and associate them with one
or more ranger teams
   *
   * @throws SQLException if a database error occurs
   */
  public void execute() throws SQLException {
     System.out.println("\n[Query 5] Insert a new researcher into the database and associate
them with one or more ranger teams");

     try {
        // User's input
        System.out.print("Enter researcher ID number: ");
        String idNumber = scanner.nextLine().trim();

        System.out.print("Enter first name: ");
        String firstName = scanner.nextLine().trim();

        System.out.print("Enter last name: ");
        String lastName = scanner.nextLine().trim();
```

```
        System.out.print("Enter gender (M/F/O): ");
        String gender = scanner.nextLine().trim();

        System.out.print("Enter street address: ");
        String street = scanner.nextLine().trim();

        System.out.print("Enter city: ");
        String city = scanner.nextLine().trim();

        System.out.print("Enter state: ");
        String state = scanner.nextLine().trim();

        System.out.print("Enter postal code: ");
        String postalCode = scanner.nextLine().trim();

        System.out.print("Enter date of birth (YYYY-MM-DD): ");
        String dateOfBirth = scanner.nextLine().trim();

        System.out.print("Subscribe to newsletter? (true/false): ");
        boolean newsletterStatus = Boolean.parseBoolean(scanner.nextLine().trim());

        System.out.print("Enter research field: ");
        String researchField = scanner.nextLine().trim();

        System.out.print("Enter hire date (YYYY-MM-DD): ");
        String hireDate = scanner.nextLine().trim();

        System.out.print("Enter salary: ");
        String salaryStr = scanner.nextLine().trim();
        double salary = Double.parseDouble(salaryStr);

        System.out.print("How many ranger teams to associate? (1 or more): ");
        int teamCount = Integer.parseInt(scanner.nextLine().trim());

        // SQL queries
        String insertIndividualSQL =
            "INSERT INTO Individual(id_number, first_name, last_name, gender, street, city, state,
" +
            "postal_code, date_of_birth, newsletter_status) " +
            "VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";

        String insertResearcherSQL =
            "INSERT INTO Researcher(id_number, research_field, hire_date, salary) " +
            "VALUES (?, ?, ?, ?)";
```

```
        // Set parameters
        PreparedStatement pstmtIndividual = connection.prepareStatement(insertIndividualSQL);
        pstmtIndividual.setString(1, idNumber);
        pstmtIndividual.setString(2, firstName);
        pstmtIndividual.setString(3, lastName);
        pstmtIndividual.setString(4, gender);
        pstmtIndividual.setString(5, street);
        pstmtIndividual.setString(6, city);
        pstmtIndividual.setString(7, state);
        pstmtIndividual.setString(8, postalCode);
        pstmtIndividual.setDate(9, java.sql.Date.valueOf(dateOfBirth));
        pstmtIndividual.setBoolean(10, newsletterStatus);

        PreparedStatement pstmtResearcher =
connection.prepareStatement(insertResearcherSQL);
        pstmtResearcher.setString(1, idNumber);
        pstmtResearcher.setString(2, researchField);
        pstmtResearcher.setDate(3, java.sql.Date.valueOf(hireDate));
        pstmtResearcher.setDouble(4, salary);

        // Executing
        connection.setAutoCommit(false);

        try {
            pstmtIndividual.executeUpdate();
            pstmtResearcher.executeUpdate();

            // Associate with ranger teams
            String associateTeamSQL =
                "INSERT INTO Researcher_reports_ranger_team(researcher_id_number, team_id,
date, summary) " +
                "VALUES (?, ?, ?, ?)";
            PreparedStatement pstmtAssociate =
connection.prepareStatement(associateTeamSQL);

            for (int i = 0; i < teamCount; i++) {
                System.out.print("Enter team ID " + (i + 1) + ": ");
                String teamId = scanner.nextLine().trim();

                System.out.print("Enter report date (YYYY-MM-DD): ");
                String reportDate = scanner.nextLine().trim();

                System.out.print("Enter summary (or press Enter for NULL): ");
```

```
                String summary = scanner.nextLine().trim();
                if (summary.isEmpty()) summary = null;

                pstmtAssociate.setString(1, idNumber);
                pstmtAssociate.setString(2, teamId);
                pstmtAssociate.setDate(3, java.sql.Date.valueOf(reportDate));
                if (summary != null) {
                    pstmtAssociate.setString(4, summary);
                } else {
                    pstmtAssociate.setNull(4, java.sql.Types.VARCHAR);
                }
                pstmtAssociate.executeUpdate();
            }
            pstmtAssociate.close();

            connection.commit();
            System.out.println("Researcher inserted and associated with teams successfully!");

        } catch (SQLException e) {
            connection.rollback();
            throw e;
        } finally {
            connection.setAutoCommit(true);
            pstmtIndividual.close();
            pstmtResearcher.close();
        }

    } catch (SQLException e) {
        System.err.println("Database error: " + e.getMessage());
        if (e.getSQLState() != null) {
            System.err.println("SQL State: " + e.getSQLState());
        }
        throw e;
    } catch (Exception e) {
        System.err.println("Error: " + e.getMessage());
    }
  }
}
```

5.1.4.5 Query 6 - Insert a Report

```
package com.npss.database.queries;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
```

```
import java.sql.SQLException;
import java.util.Scanner;

/**
* Query 6: Insert a new report submitted by a ranger team to a researcher
* If a report already exists for this researcher-team pair, it will be updated
*
* @author Astra Nguyen
* @version 1.0
*/
public class Query6_InsertReport {
  private Connection connection;
  private Scanner scanner;

  public Query6_InsertReport(Connection connection, Scanner scanner) {
     this.connection = connection;
     this.scanner = scanner;
  }

  /**
   * Executes Query 6: Insert a report submitted by a ranger team to a researcher
   * Uses UPDATE if report exists, INSERT if it doesn't (UPSERT pattern)
   *
   * @throws SQLException if a database error occurs
   */
  public void execute() throws SQLException {
     System.out.println("\n[Query 6] Insert a report submitted by a ranger team to a researcher");

     try {
        // User's input
        System.out.print("Enter researcher ID number: ");
        String researcherId = scanner.nextLine().trim();

        System.out.print("Enter ranger team ID: ");
        String teamId = scanner.nextLine().trim();

        System.out.print("Enter report date (YYYY-MM-DD): ");
        String reportDate = scanner.nextLine().trim();

        System.out.print("Enter summary (or press Enter for NULL): ");
        String summary = scanner.nextLine().trim();
        if (summary.isEmpty()) summary = null;

        // SQL queries
```

```
        // SQL Server MERGE statement
        String mergeReportSQL =
            "MERGE Researcher_reports_ranger_team AS target " +
            "USING (SELECT ? AS researcher_id_number, ? AS team_id) AS source " +
            "ON target.researcher_id_number = source.researcher_id_number " +
            "   AND target.team_id = source.team_id " +
            "WHEN MATCHED THEN " +
            "    UPDATE SET date = ?, summary = ? " +
            "WHEN NOT MATCHED THEN " +
            "    INSERT (researcher_id_number, team_id, date, summary) " +
            "    VALUES (?, ?, ?, ?);";

        // Executing
        connection.setAutoCommit(false);

        try (PreparedStatement pstmtReport = connection.prepareStatement(mergeReportSQL))
{
            // Parameters for MERGE: source (2), update (2), insert (4)
            pstmtReport.setString(1, researcherId);  // source researcher_id
            pstmtReport.setString(2, teamId);        // source team_id
            pstmtReport.setDate(3, java.sql.Date.valueOf(reportDate));  // update date
            if (summary != null) {
                pstmtReport.setString(4, summary);    // update summary
            } else {
                pstmtReport.setNull(4, java.sql.Types.VARCHAR);
            }
            pstmtReport.setString(5, researcherId);  // insert researcher_id
            pstmtReport.setString(6, teamId);        // insert team_id
            pstmtReport.setDate(7, java.sql.Date.valueOf(reportDate));  // insert date
            if (summary != null) {
                pstmtReport.setString(8, summary);    // insert summary
            } else {
                pstmtReport.setNull(8, java.sql.Types.VARCHAR);
            }

            int rowsAffected = pstmtReport.executeUpdate();
            connection.commit();

            if (rowsAffected > 0) {
                System.out.println("Report inserted/updated successfully! (Rows affected: " +
rowsAffected + ")");

                // Verify the report
                String verifySQL = "SELECT researcher_id_number, team_id, date, summary " +
```

```
                        "FROM Researcher_reports_ranger_team " +
                        "WHERE researcher_id_number = ? AND team_id = ?";
            try (PreparedStatement verifyStmt = connection.prepareStatement(verifySQL)) {
                verifyStmt.setString(1, researcherId);
                verifyStmt.setString(2, teamId);
                try (ResultSet rs = verifyStmt.executeQuery()) {
                    if (rs.next()) {
                        System.out.println("VERIFIED: Report for researcher " + researcherId +
                                " and team " + teamId + " is in the database");
                        System.out.println("  Date: " + rs.getDate("date"));
                        System.out.println("  Summary: " + (rs.getString("summary") != null ?
rs.getString("summary") : "NULL"));
                    }
                }
            }
        } else {
            System.out.println("No rows affected. Report may not have been
inserted/updated.");
        }

    } catch (SQLException e) {
        connection.rollback();
        throw e;
    } finally {
        connection.setAutoCommit(true);
    }

  } catch (SQLException e) {
    System.err.println("Database error: " + e.getMessage());
    if (e.getSQLState() != null) {
        System.err.println("SQL State: " + e.getSQLState());
    }
    throw e;
  } catch (IllegalArgumentException e) {
    System.err.println("Invalid date format: " + e.getMessage());
    System.err.println("Please use YYYY-MM-DD format for dates.");
    throw new SQLException("Invalid input: " + e.getMessage(), e);
  } catch (Exception e) {
    System.err.println("Error: " + e.getMessage());
    e.printStackTrace();
    throw new SQLException("Unexpected error: " + e.getMessage(), e);
  }
 }
}
```

5.1.4.6 Query 7 - Insert a New Park Program

```
package com.npss.database.queries;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Scanner;

/**
* Query 7: Insert a new park program into the database for a specific park
* If the park doesn't exist, it will be created automatically
*
* @author Astra Nguyen
* @version 1.0
*/
public class Query7_InsertParkProgram {
  private Connection connection;
  private Scanner scanner;

  public Query7_InsertParkProgram(Connection connection, Scanner scanner) {
     this.connection = connection;
     this.scanner = scanner;
  }

  /**
   * Checks if a park exists in the database
   */
  private boolean parkExists(String parkName) throws SQLException {
     String checkSQL = "SELECT Name FROM National_parks WHERE Name = ?";
     try (PreparedStatement pstmt = connection.prepareStatement(checkSQL)) {
        pstmt.setString(1, parkName);
        try (ResultSet rs = pstmt.executeQuery()) {
           return rs.next();
        }
     }
  }


  /**
   * Executes Query 7: Insert a new park program into the database for a specific park
   * If the park doesn't exist, it will be created automatically
   *
   * @throws SQLException if a database error occurs
   */
```

```
  public void execute() throws SQLException {
    System.out.println("\n[Query 7] Insert a new park program into the database for a specific park");

    try {
      // User's input
      System.out.print("Enter program name: ");
      String programName = scanner.nextLine().trim();

      System.out.print("Enter program type: ");
      String programType = scanner.nextLine().trim();

      System.out.print("Enter start date (YYYY-MM-DD): ");
      String startDate = scanner.nextLine().trim();

      System.out.print("Enter duration (in days): ");
      String durationStr = scanner.nextLine().trim();
      int duration = Integer.parseInt(durationStr);

      System.out.print("Enter park name: ");
      String parkName = scanner.nextLine().trim();

      // Create park if it does not exist
      boolean parkNeedsCreation = !parkExists(parkName);
      String street = null, city = null, state = null, postalCode = null, establishmentDate = null;
      int capacity = 0;

      if (parkNeedsCreation) {
        System.out.println("\nPark '" + parkName + "' does not exist. Creating new park...");
        System.out.println("Please provide the following park information:");

        System.out.print("Enter street address: ");
        street = scanner.nextLine().trim();

        System.out.print("Enter city: ");
        city = scanner.nextLine().trim();

        System.out.print("Enter state: ");
        state = scanner.nextLine().trim();

        System.out.print("Enter postal code: ");
        postalCode = scanner.nextLine().trim();

        System.out.print("Enter establishment date (YYYY-MM-DD): ");
```

```
            establishmentDate = scanner.nextLine().trim();

            System.out.print("Enter capacity: ");
            String capacityStr = scanner.nextLine().trim();
            capacity = Integer.parseInt(capacityStr);
        } else {
            System.out.println("Park '" + parkName + "' found in database.");
        }

        // SQL queries
        String insertParkSQL =
            "INSERT INTO National_parks(Name, Street, City, State, Postal_code,
Establishment_date, Capacity) " +
            "VALUES (?, ?, ?, ?, ?, ?, ?)";

        String insertProgramSQL =
            "INSERT INTO Program(program_name, type, start_date, duration) " +
            "VALUES (?, ?, ?, ?)";

        String linkParkProgramSQL =
            "INSERT INTO National_parks_offers_program(park_name, program_name) " +
            "VALUES (?, ?)";

        // Executing
        connection.setAutoCommit(false);

        try {
            // Create park if it doesn't exist
            if (parkNeedsCreation) {
                try (PreparedStatement pstmtPark = connection.prepareStatement(insertParkSQL))
{
                    pstmtPark.setString(1, parkName);
                    pstmtPark.setString(2, street);
                    pstmtPark.setString(3, city);
                    pstmtPark.setString(4, state);
                    pstmtPark.setString(5, postalCode);
                    pstmtPark.setDate(6, java.sql.Date.valueOf(establishmentDate));
                    pstmtPark.setInt(7, capacity);

                    int parkRows = pstmtPark.executeUpdate();
                    System.out.println("Park '" + parkName + "' created successfully! (Rows affected:
" + parkRows + ")");
                }
            }
```

```
            // Create program
            try (PreparedStatement pstmtProgram =
connection.prepareStatement(insertProgramSQL)) {
                pstmtProgram.setString(1, programName);
                pstmtProgram.setString(2, programType);
                pstmtProgram.setDate(3, java.sql.Date.valueOf(startDate));
                pstmtProgram.setInt(4, duration);

                int rows1 = pstmtProgram.executeUpdate();

                // Link program to park
                try (PreparedStatement pstmtLink =
connection.prepareStatement(linkParkProgramSQL)) {
                    pstmtLink.setString(1, parkName);
                    pstmtLink.setString(2, programName);

                    int rows2 = pstmtLink.executeUpdate();

                    connection.commit();
                    System.out.println("Park program inserted successfully! (Program rows: " + rows1
+ ", Link rows: " + rows2 + ")");

                    // Verify the insert
                    String verifySQL = "SELECT p.program_name, p.type, np.name as park_name "
+
                                       "FROM Program p " +
                                       "INNER JOIN National_parks_offers_program npop ON
p.program_name = npop.program_name " +
                                       "INNER JOIN National_parks np ON npop.park_name = np.name " +
                                       "WHERE p.program_name = ?";
                    try (PreparedStatement verifyStmt = connection.prepareStatement(verifySQL)) {
                        verifyStmt.setString(1, programName);
                        try (ResultSet rs = verifyStmt.executeQuery()) {
                            if (rs.next()) {
                                System.out.println("VERIFIED: Program '" + programName + "' is now
linked to park '" +
                                                   rs.getString("park_name") + "' in the database");
                            } else {
                                System.out.println("WARNING: Program '" + programName + "' not found
after insert!");
                            }
                        }
                    }
```

```
                }
            }

        } catch (SQLException e) {
            connection.rollback();
            throw e;
        } finally {
            connection.setAutoCommit(true);
        }

    } catch (SQLException e) {
        System.err.println("Database error: " + e.getMessage());
        if (e.getSQLState() != null) {
            System.err.println("SQL State: " + e.getSQLState());
        }
        throw e;
    } catch (IllegalArgumentException e) {
        System.err.println("Invalid date format: " + e.getMessage());
        System.err.println("Please use YYYY-MM-DD format for dates.");
        throw new SQLException("Invalid input: " + e.getMessage(), e);
    } catch (Exception e) {
        System.err.println("Error: " + e.getMessage());
        e.printStackTrace();
        throw new SQLException("Unexpected error: " + e.getMessage(), e);
    }
  }
}
```

5.1.4.7 Query 8 - Retrieve the Names and Contact Information of All Emergency Contacts

```
package com.npss.database.queries;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Scanner;

/**
* Query 8: Retrieve the names and contact information of all emergency contacts for a specific person
*
* This query leverages the IX_emergency_contact_id_number index for efficient lookups.
*
```

```
* @author Astra Nguyen
* @version 1.0
*/
public class Query8_RetrieveEmergencyContacts {
  private Connection connection;
  private Scanner scanner;

  public Query8_RetrieveEmergencyContacts(Connection connection, Scanner scanner) {
     this.connection = connection;
     this.scanner = scanner;
  }

  /**
   * Executes Query 8: Retrieve the names and contact information of all emergency contacts
for a specific person
   *
   * @throws SQLException if a database error occurs
   */
  public void execute() throws SQLException {
     System.out.println("\n[Query 8] Retrieve the names and contact information of all
emergency contacts for a specific person");

     try {
        // Get user input
        System.out.print("Enter the person's ID number: ");
        String idNumber = scanner.nextLine().trim();

        if (idNumber.isEmpty()) {
           System.out.println("Error: ID number cannot be empty.");
           return;
        }

        // SQL query - leverages IX_emergency_contact_id_number index
        // ec is table alias for Emergency_contact table
        // Also joins with Individual to get the person's name for context
        String SQL =
           "SELECT ec.name AS emergency_contact_name, " +
           "       ec.relationship, " +
           "       ec.phone_number, " +
           "       CONCAT(i.first_name, ' ', i.last_name) AS person_name " + //first name & last
name from individual alias
           "FROM Emergency_contact ec " +
           "INNER JOIN Individual i ON ec.id_number = i.id_number " + //joins with individual
table
```

```
            "WHERE ec.id_number = ? " +
            "ORDER BY ec.relationship, ec.name"; //sort results

        // Fill in the variables
        PreparedStatement pstmt = connection.prepareStatement(SQL);
        pstmt.setString(1, idNumber);

        ResultSet rs = pstmt.executeQuery();

        // Display results
        boolean hasResults = false;
        String personName = null;

        System.out.println("Emergency Contacts");

        while (rs.next()) {
            if (!hasResults) {
                personName = rs.getString("person_name");
                System.out.println("- Person: " + personName + " (ID: " + idNumber + ")");
                hasResults = true;
            }

            System.out.println("- Contact Name: " + rs.getString("emergency_contact_name"));
            System.out.println("- Relationship: " + rs.getString("relationship"));
            System.out.println("- Phone Number: " + rs.getString("phone_number"));
        }

        if (!hasResults) {
            System.out.println("- No emergency contacts found for ID number: " + idNumber);
        }

        rs.close();
        pstmt.close();

    } catch (SQLException e) {
        System.err.println("Database error: " + e.getMessage());
        if (e.getSQLState() != null) {
            System.err.println("SQL State: " + e.getSQLState());
        }
        throw e;
    } catch (Exception e) {
        System.err.println("Error: " + e.getMessage());
    }
}
```

```
}
```

5.1.4.8 Query 9 - Retrieve the list of visitors enrolled in a specific park program, including their accessibility needs

```
package com.npss.database.queries;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Scanner;

/**
* Query 9: Retrieve the list of visitors enrolled in a specific park program, including their accessibility needs
*
* This query leverages the following indexes for optimal performance:
* - IX_visitor_enrolls_program_program_name (for filtering by program)
* - IX_visitor_enrolls_program_visitor_id_number (for joining with Visitor)
* - Clustered index on Individual.id_number (for joining with Individual)
*
* @author Astra Nguyen
* @version 1.0
*/
public class Query9_RetrieveVisitorsInProgram {
  private Connection connection;
  private Scanner scanner;

  public Query9_RetrieveVisitorsInProgram(Connection connection, Scanner scanner) {
    this.connection = connection;
    this.scanner = scanner;
  }

  /**
   * Executes Query 9: Retrieve the list of visitors enrolled in a specific park program, including their accessibility needs
   *
   * @throws SQLException if a database error occurs
   */
  public void execute() throws SQLException {
    System.out.println("\n[Query 9] Retrieve the list of visitors enrolled in a specific park program, including their accessibility needs");

    try {
```

```
        // Get user input
        System.out.print("Enter the program name: ");
        String programName = scanner.nextLine().trim();

        // SQL query - leverages indexes on Visitor_enrolls_program and joins with Visitor and
Individual
        String SQL =
            "SELECT i.id_number, " +
            "       i.first_name, " +
            "       i.last_name, " +
            "       i.first_name + ' ' + i.last_name AS full_name, " +
            "       v.accessibility_needs, " +
            "       v.visit_date " +
            "FROM Visitor_enrolls_program vep " +
            "INNER JOIN Visitor v ON vep.visitor_id_number = v.id_number " +
            "INNER JOIN Individual i ON v.id_number = i.id_number " +
            "WHERE vep.program_name = ? " +
            "ORDER BY i.last_name, i.first_name"; // sort the results

        // Fill in the variables & displaying
        PreparedStatement pstmt = connection.prepareStatement(SQL);
        pstmt.setString(1, programName);

        ResultSet rs = pstmt.executeQuery();

        boolean hasResults = false;

        System.out.println("Visitors Enrolled in Program: " + programName);

        while (rs.next()) {
            if (!hasResults) {
                hasResults = true;
            }

            System.out.println("ID Number: " + rs.getString("id_number"));
            System.out.println("Name: " + rs.getString("full_name"));

            String accessibilityNeeds = rs.getString("accessibility_needs");
            if (accessibilityNeeds != null && !accessibilityNeeds.isEmpty()) {
                System.out.println("Accessibility Needs: " + accessibilityNeeds);
            } else {
                System.out.println("Accessibility Needs: None ");
            }
```

```
            java.sql.Date visitDate = rs.getDate("visit_date");
            if (visitDate != null) {
                System.out.println("Visit Date: " + visitDate.toString());
            } else {
                System.out.println("Visit Date: Not specified");
            }
        }

        if (!hasResults) {
            System.out.println("No visitors found enrolled in program: " + programName);
        }

        rs.close();
        pstmt.close();

    } catch (SQLException e) {
        System.err.println("Database error: " + e.getMessage());
        if (e.getSQLState() != null) {
            System.err.println("SQL State: " + e.getSQLState());
        }
        throw e;
    } catch (Exception e) {
        System.err.println("Error: " + e.getMessage());
    }
  }
}
```

5.1.4.9 Query 10 - Retrieve Park Programs For a Specific Park That Started After A Given Date

```
package com.npss.database.queries;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Scanner;

/**
* Query 10: Retrieve all park programs for a specific park that started after a given date
*
* This query leverages the following indexes for optimal performance:
* - IX_national_parks_offers_program_park_name (for filtering by park)
* - IX_national_parks_offers_program_program_name (for joining with Program)
* - IX_program_start_date (for range query on start_date)
```

```
*
* @author Astra Nguyen
* @version 1.0
*/
public class Query10_RetrieveParkPrograms {
  private Connection connection;
  private Scanner scanner;

  public Query10_RetrieveParkPrograms(Connection connection, Scanner scanner) {
     this.connection = connection;
     this.scanner = scanner;
  }

  /**
   * Executes Query 10: Retrieve all park programs for a specific park that started after a given
date
   *
   * @throws SQLException if a database error occurs
   */
  public void execute() throws SQLException {
     System.out.println("\n[Query 10] Retrieve all park programs for a specific park that started
after a given date");

     try {
        // Get user input
        System.out.print("Enter the park name: ");
        String parkName = scanner.nextLine().trim();

        if (parkName.isEmpty()) {
           System.out.println("Error: Park name cannot be empty.");
           return;
        }

        System.out.print("Enter the start date (YYYY-MM-DD) - programs starting after this date
will be shown: ");
        String startDateStr = scanner.nextLine().trim();

        if (startDateStr.isEmpty()) {
           System.out.println("Error: Start date cannot be empty.");
           return;
        }

        java.sql.Date startDate;
        try {
```

```
    startDate = java.sql.Date.valueOf(startDateStr);
} catch (IllegalArgumentException e) {
    System.out.println("Error: Invalid date format. Please use YYYY-MM-DD format.");
    return;
}

// SQL query - leverages indexes on National_parks_offers_program and Program
// Uses range query on start_date which benefits from IX_program_start_date index
String SQL =
    "SELECT p.program_name, " +
    "       p.type, " +
    "       p.start_date, " +
    "       p.duration " +
    "FROM National_parks_offers_program npop " +
    "INNER JOIN Program p ON npop.program_name = p.program_name " +
    "WHERE npop.park_name = ? " +
    "  AND p.start_date > ? " +
    "ORDER BY p.start_date, p.program_name";

// Fill in the variables
PreparedStatement pstmt = connection.prepareStatement(SQL);
pstmt.setString(1, parkName);
pstmt.setDate(2, startDate);

ResultSet rs = pstmt.executeQuery();

// Display results
boolean hasResults = false;

System.out.println("Park Programs for: " + parkName);
System.out.println("Programs starting after: " + startDate.toString());

while (rs.next()) {
    if (!hasResults) {
        hasResults = true;
    }

    System.out.println("Program Name: " + rs.getString("program_name"));
    System.out.println("Type: " + rs.getString("type"));
    System.out.println("Start Date: " + rs.getDate("start_date").toString());
    System.out.println("Duration: " + rs.getInt("duration") + " days");
}

if (!hasResults) {
```

```
            System.out.println("No programs found for park '" + parkName + "' starting after " +
startDate.toString());
        }
        rs.close();
        pstmt.close();

    } catch (SQLException e) {
        System.err.println("Database error: " + e.getMessage());
        if (e.getSQLState() != null) {
            System.err.println("SQL State: " + e.getSQLState());
        }
        throw e;
    } catch (Exception e) {
        System.err.println("Error: " + e.getMessage());
    }
  }
}
```

5.1.4.10 Query 11 - Retrieve The Total And Average Donation Amount Received In A Month From All Anonymous Donors

```
package com.npss.database.queries;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Scanner;

/**
* Query 11: Retrieve the total and average donation amount received in a month from all
anonymous donors
*
* This query leverages the following indexes for optimal performance:
* - IX_donation_date (for filtering by month/year)
* - IX_donation_donor_id_number (for joining with Donor)
* - IX_donor_preference (for filtering anonymous donors)
* - Clustered index on Donor.id_number (for grouping by donor)
*
* @author Astra Nguyen
* @version 1.0
*/
public class Query11_RetrieveDonationStats {
  private Connection connection;
  private Scanner scanner;
```

```
  public Query11_RetrieveDonationStats(Connection connection, Scanner scanner) {
      this.connection = connection;
      this.scanner = scanner;
  }

  /**
   * Executes Query 11: Retrieve the total and average donation amount received in a month
from all anonymous donors
   *
   * @throws SQLException if a database error occurs
   */
  public void execute() throws SQLException {
      System.out.println("\n[Query 11] Retrieve the total and average donation amount received
in a month from all anonymous donors");

      try{
          // Get user input
          System.out.println("Enter the month (MM)");
          String month = scanner.nextLine().trim();
          System.out.print("Enter the year (YYYY): ");
          String year = scanner.nextLine().trim();

          if(month.isEmpty() || year.isEmpty()){
              System.out.println("Error: Month and year cannot be empty.");
              return;
          }

          // Logic check
          int monthInt, yearInt;
          try {
              monthInt = Integer.parseInt(month);
              yearInt = Integer.parseInt(year);

              if (monthInt < 1 || monthInt > 12) {
                  System.out.println("Error: Month must be between 01 and 12.");
                  return;
              }
          } catch (NumberFormatException e) {
              System.out.println("Error: Invalid number format.");
              return;
          }

          // SQL query - Retrieve the total and average donation amount received in a
```

```
        // month from all anonymous donors
        String SQL = "SELECT "  +
        "dr.id_number AS donor_id, " +
        "SUM(d.amount) AS total_amount, " +
        "AVG(d.amount) AS average_amount, " +
        "COUNT(d.donation_id) AS donation_count " +
        "FROM Donation d " +
        "INNER JOIN Donor dr ON d.donor_id_number = dr.id_number " +
        "WHERE YEAR(d.date) = ? " +
        " AND MONTH(d.date) = ? " +
        "AND (dr.preference IS NULL OR dr.preference = 'Anonymous') " +
        "GROUP BY dr.id_number " + "ORDER BY SUM(d.amount) DESC";

        // Fill in the variables
        PreparedStatement pstmt = connection.prepareStatement(SQL);
        pstmt.setInt(1, yearInt);
        pstmt.setInt(2, monthInt);

        ResultSet rs = pstmt.executeQuery();

        // Display results
        boolean hasResults = false;

        System.out.println("Anonymous Donor Statistics for " + month + "/" + year);
        System.out.println("Sorted by Total Amount (Descending)");

        while (rs.next()) {
            hasResults = true;

            System.out.println("Donor ID: " + rs.getString("donor_id"));
            System.out.println("Total Amount: $" + String.format("%.2f",
rs.getDouble("total_amount")));
            System.out.println("Average Amount: $" + String.format("%.2f",
rs.getDouble("average_amount")));
            System.out.println("Number of Donations: " + rs.getInt("donation_count"));
        }

        if (!hasResults) {
            System.out.println("No anonymous donations found for " + month + "/" + year);
        }


        rs.close();
        pstmt.close();
```

```
        }catch(SQLException e){
            System.err.println("Database error: " + e.getMessage());
            if (e.getSQLState() != null) {
                System.err.println("SQL State: " + e.getSQLState());
            }
            throw e;
        }catch(Exception e){
            System.err.println("Error: " + e.getMessage());
        }
    }
}
```

5.1.4.11 Query 12 - Retrieve The List of Rangers In A Team, Including Their Certifications, Years of Service And Their Role in The Team

```
package com.npss.database.queries;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Scanner;

/**
* Query 12: Retrieve the list of rangers in a team, including their certifications, years of service and their role in the team
*
* This query leverages the following indexes for optimal performance:
* - IX_ranger_assigned_ranger_team_team_id (for filtering by team)
* - IX_ranger_assigned_ranger_team_ranger_id_number (for joining with Ranger)
* - IX_ranger_certifications_id_number (for retrieving certifications)
* - Clustered index on Individual.id_number (for joining with Individual)
*
* @author Astra Nguyen
* @version 1.0
*/
public class Query12_RetrieveRangersInTeam {
    private Connection connection;
    private Scanner scanner;

    public Query12_RetrieveRangersInTeam(Connection connection, Scanner scanner) {
        this.connection = connection;
        this.scanner = scanner;
```

```
  }

  /**
   * Executes Query 12: Retrieve the list of rangers in a team, including their certifications, years of service and their role in the team
   *
   * @throws SQLException if a database error occurs
   */
  public void execute() throws SQLException {
      System.out.println("\n[Query 12] Retrieve the list of rangers in a team, including their certifications, years of service and their role in the team");

      try {
          // Get user input
          System.out.print("Enter the team ID: ");
          String teamId = scanner.nextLine().trim();

          if (teamId.isEmpty()) {
              System.out.println("Error: Team ID cannot be empty.");
              return;
          }

          // SQL query - Retrieve rangers in team with certifications, years of service, and status
          // Uses LEFT JOIN for certifications since a ranger may have no certifications
          String SQL =
              "SELECT DISTINCT " +
              "    i.id_number, " +
              "    i.first_name, " +
              "    i.last_name, " +
              "    i.first_name + ' ' + i.last_name AS full_name, " +
              "    rart.status, " +
              "    rart.years_of_service, " +
              "    rc.certification " +
              "FROM Ranger_assigned_ranger_team rart " +
              "INNER JOIN Ranger r ON rart.ranger_id_number = r.id_number " +
              "INNER JOIN Individual i ON r.id_number = i.id_number " +
              "LEFT JOIN Ranger_certifications rc ON r.id_number = rc.id_number " +
              "WHERE rart.team_id = ? " +
              "ORDER BY i.last_name, i.first_name, rc.certification";

          // Fill in the variables
          PreparedStatement pstmt = connection.prepareStatement(SQL);
          pstmt.setString(1, teamId);
```

```
ResultSet rs = pstmt.executeQuery();

// Display results
boolean hasResults = false;
String currentRangerId = null;
String currentRangerName = null;
String currentStatus = null;
Integer currentYearsOfService = null;
boolean firstRanger = true;
boolean firstCertification = true;

System.out.println("Rangers in Team: " + teamId);

while (rs.next()) {
    String rangerId = rs.getString("id_number");

    // If this is a new ranger, display ranger info
    if (currentRangerId == null || !rangerId.equals(currentRangerId)) {
        if (!firstRanger) {
            System.out.println(); // New line after certifications
        }
        firstRanger = false;
        firstCertification = true;
        hasResults = true;

        currentRangerId = rangerId;
        currentRangerName = rs.getString("full_name");
        currentStatus = rs.getString("status");
        currentYearsOfService = rs.getInt("years_of_service");

        System.out.println("ID Number: " + currentRangerId);
        System.out.println("Name: " + currentRangerName);
        System.out.println("Status (Role): " + currentStatus);
        System.out.println("Years of Service: " + currentYearsOfService);
        System.out.print("Certifications: ");
    }

    // Display certification
    // May not have any certifications
    String certification = rs.getString("certification");
    if (certification != null && !certification.isEmpty()) {
        if (firstCertification) {
            // First certification for this ranger
            System.out.print(certification);
```

```
                    firstCertification = false;
                } else {
                    // Additional certification
                    System.out.print(", " + certification);
                }
            } else {
                // No certifications found
                if (firstCertification) {
                    System.out.print("None");
                    firstCertification = false;
                }
            }
        }

        if (hasResults) {
            System.out.println(); // New line after last certification
        } else {
            System.out.println("No rangers found in team: " + teamId);
        }

        rs.close();
        pstmt.close();

    } catch (SQLException e) {
        System.err.println("Database error: " + e.getMessage());
        if (e.getSQLState() != null) {
            System.err.println("SQL State: " + e.getSQLState());
        }
        throw e;
    } catch (Exception e) {
        System.err.println("Error: " + e.getMessage());
    }
  }
}
```

5.1.4.12 Query 13 - Retrieve the Names, IDs, Contact Information, and Newsletter Subscription Status of All Individuals

```
package com.npss.database.queries;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Scanner;
```

```
/**
* Query 13: Retrieve the names, IDs, contact information, and newsletter subscription status of all individuals in the database
*
* This query leverages the following indexes for optimal performance:
* - Clustered index on Individual.id_number (primary key)
* - IX_individual_phone_numbers_id_number (for retrieving phone numbers)
* - IX_individual_email_addresses_id_number (for retrieving email addresses)
*
* @author Astra Nguyen
* @version 1.0
*/
public class Query13_RetrieveAllIndividuals {
  private Connection connection;
  private Scanner scanner;

  public Query13_RetrieveAllIndividuals(Connection connection, Scanner scanner) {
     this.connection = connection;
     this.scanner = scanner;
  }

  /**
   * Executes Query 13: Retrieve the names, IDs, contact information, and newsletter subscription status of all individuals in the database
   *
   * @throws SQLException if a database error occurs
   */
  public void execute() throws SQLException {
     System.out.println("\n[Query 13] Retrieve the names, IDs, contact information, and newsletter subscription status of all individuals in the database");

     try {
        // SQL query - Retrieve all individuals with their contact information
        // Uses LEFT JOIN for phone numbers and emails since they are multi-valued attributes
        String SQL =
           "SELECT " +
           "    i.id_number, " +
           "    i.first_name, " +
           "    i.last_name, " +
           "    i.first_name + ' ' + i.last_name AS full_name, " +
           "    i.newsletter_status, " +
           "    ipn.phone_number, " +
           "    iea.email_address " +
```

```
            "FROM Individual i " +
            "LEFT JOIN Individual_phone_numbers ipn ON i.id_number = ipn.id_number " +
            "LEFT JOIN Individual_email_addresses iea ON i.id_number = iea.id_number " +
            "ORDER BY i.last_name, i.first_name, ipn.phone_number, iea.email_address";

        // Fill in the variables
        PreparedStatement pstmt = connection.prepareStatement(SQL);

        ResultSet rs = pstmt.executeQuery();

        // Display results
        boolean hasResults = false;
        String currentIndividualId = null;
        String currentIndividualName = null;
        Boolean currentNewsletterStatus = null;
        boolean firstIndividual = true;
        boolean firstPhone = true;
        boolean firstEmail = true;

        System.out.println("All Individuals in Database");

        while (rs.next()) {
            String individualId = rs.getString("id_number");

            if (currentIndividualId == null || !individualId.equals(currentIndividualId)) {
                if (!firstIndividual) {
                    System.out.println(); // New line after contact info
                }
                firstIndividual = false;
                firstPhone = true;
                firstEmail = true;
                hasResults = true;

                currentIndividualId = individualId;
                currentIndividualName = rs.getString("full_name");
                currentNewsletterStatus = rs.getBoolean("newsletter_status");

                System.out.println("ID Number: " + currentIndividualId);
                System.out.println("Name: " + currentIndividualName);
                System.out.println("Newsletter Status: " + (currentNewsletterStatus ? "Subscribed" :
"Not Subscribed"));
                System.out.print("Phone Numbers: ");
            }
```

```
    // Display phone number, may be null
    String phoneNumber = rs.getString("phone_number");
    if (phoneNumber != null && !phoneNumber.isEmpty()) {
        if (firstPhone) {
            // First phone number for this individual
            System.out.print(phoneNumber);
            firstPhone = false;
        } else {
            // Additional phone number
            System.out.print(", " + phoneNumber);
        }
    } else {
        // No phone numbers found
        if (firstPhone) {
            System.out.print("None");
            firstPhone = false;
        }
    }

    // Display email address
    String emailAddress = rs.getString("email_address");
    if (emailAddress != null && !emailAddress.isEmpty()) {
        if (firstEmail) {
            System.out.print(" | Email: " + emailAddress);
            firstEmail = false;
        } else {
            // Additional email
            System.out.print(", " + emailAddress);
        }
    } else {
        // No email found
        if (firstEmail) {
            System.out.print(" | Email: None");
            firstEmail = false;
        }
    }
}

if (hasResults) {
    System.out.println(); // New line after last contact info
} else {
    System.out.println("No individuals found in the database.");
}
```

```
            rs.close();
            pstmt.close();

        } catch (SQLException e) {
            System.err.println("Database error: " + e.getMessage());
            if (e.getSQLState() != null) {
                System.err.println("SQL State: " + e.getSQLState());
            }
            throw e;
        } catch (Exception e) {
            System.err.println("Error: " + e.getMessage());
        }
    }
}
```

5.1.4.13 Query 14 - Update The Salary of Researchers

```
package com.npss.database.queries;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.util.Scanner;

/**
* Query 14: Update the salary of researchers overseeing more than one ranger team by a 3%
increase
*
* This query leverages the following indexes for optimal performance:
* - IX_researcher_reports_ranger_team_researcher_id_number (for grouping by researcher)
* - IX_researcher_reports_ranger_team_team_id (for counting teams)
* - Clustered index on Researcher.id_number (for updating salary)
*
* @author Astra Nguyen
* @version 1.0
*/
public class Query14_UpdateResearcherSalary {
    private Connection connection;
    private Scanner scanner;

    public Query14_UpdateResearcherSalary(Connection connection, Scanner scanner) {
        this.connection = connection;
        this.scanner = scanner;
    }

    /**
```

```
    * Executes Query 14: Update the salary of researchers overseeing more than one ranger
team by a 3% increase
    *
    * @throws SQLException if a database error occurs
    */
   public void execute() throws SQLException {
      System.out.println("\n[Query 14] Update the salary of researchers overseeing more than
one ranger team by a 3% increase");

      try {
         // SQL query - Update salary by 3% for researchers overseeing more than one team
         // Uses subquery to find researchers with COUNT(team_id) > 1
         String SQL =
            "UPDATE Researcher " +
            "SET salary = salary * 1.03 " +
            "WHERE id_number IN (" +
               "SELECT researcher_id_number " +
               "FROM Researcher_reports_ranger_team " +
               "GROUP BY researcher_id_number " +
               "HAVING COUNT(DISTINCT team_id) > 1" +
            ")";

         // Executing
         connection.setAutoCommit(false);

         PreparedStatement pstmt = connection.prepareStatement(SQL);

         try {
            int rowsAffected = pstmt.executeUpdate();
            connection.commit();

            if (rowsAffected > 0) {
               System.out.println("Updated salary for " + rowsAffected + " researcher(s)
successfully!");
               System.out.println("Salary increased by 3% for researchers overseeing more than
one ranger team.");
            } else {
               System.out.println("No researchers found overseeing more than one ranger team.");
            }

         } catch (SQLException e) {
            connection.rollback();
            throw e;
         } finally {
```

```
            connection.setAutoCommit(true);
            pstmt.close();
        }

    } catch (SQLException e) {
        System.err.println("Database error: " + e.getMessage());
        if (e.getSQLState() != null) {
            System.err.println("SQL State: " + e.getSQLState());
        }
        throw e;
    } catch (Exception e) {
        System.err.println("Error: " + e.getMessage());
    }
  }
}
```

5.1.4.14 Query 15 - Delete Visitors Who Have Not Enrolled In Any Park Programs and Whose Park Passes Have Expired

```
package com.npss.database.queries;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.util.Scanner;

/**
* Query 15: Delete visitors who have not enrolled in any park programs and whose park passes
have expired
*
* This query leverages the following indexes for optimal performance:
* - IX_visitor_holds_park_passes_visitor_id_number (for joining with Visitor)
* - IX_visitor_holds_park_passes_pass_id (for joining with Park_passes)
* - IX_park_passes_expiration_date (for filtering expired passes)
* - IX_visitor_enrolls_program_visitor_id_number (for checking program enrollment)
* - Clustered index on Visitor.id_number (for deletion)
*
* @author Astra Nguyen
* @version 1.0
*/
public class Query15_DeleteExpiredVisitors {
  private Connection connection;
  private Scanner scanner;

  public Query15_DeleteExpiredVisitors(Connection connection, Scanner scanner) {
```

```
        this.connection = connection;
        this.scanner = scanner;
    }

    /**
     * Executes Query 15: Delete visitors who have not enrolled in any park programs and whose
park passes have expired
     *
     * @throws SQLException if a database error occurs
     */
    public void execute() throws SQLException {
        System.out.println("\n[Query 15] Delete visitors who have not enrolled in any park programs
and whose park passes have expired");

        try {
            // SQL query - Delete visitors with expired passes who are not enrolled in programs
            // Uses subquery to find eligible visitors for deletion
            String SQL =
                "DELETE FROM Visitor " +
                "WHERE id_number IN (" +
                    "SELECT DISTINCT v.id_number " +
                    "FROM Visitor v " +
                    "INNER JOIN Visitor_holds_park_passes vhpp ON v.id_number =
vhpp.visitor_id_number " +
                    "INNER JOIN Park_passes pp ON vhpp.pass_id = pp.pass_id " +
                    "WHERE pp.expiration_date < GETDATE() " +
                    "AND v.id_number NOT IN (" +
                        "SELECT visitor_id_number FROM Visitor_enrolls_program" +
                    ")" +
                ")";

            // Executing
            connection.setAutoCommit(false);

            PreparedStatement pstmt = connection.prepareStatement(SQL);

            try {
                int rowsAffected = pstmt.executeUpdate();
                connection.commit();

                if (rowsAffected > 0) {
                    System.out.println("Deleted " + rowsAffected + " expired visitor(s) successfully!");
                } else {
                    System.out.println("No expired visitors found to delete.");
```

```
            }

        } catch (SQLException e) {
            connection.rollback();
            throw e;
        } finally {
            connection.setAutoCommit(true);
            pstmt.close();
        }

    } catch (SQLException e) {
        System.err.println("Database error: " + e.getMessage());
        if (e.getSQLState() != null) {
            System.err.println("SQL State: " + e.getSQLState());
        }
        throw e;
    } catch (Exception e) {
        System.err.println("Error: " + e.getMessage());
    }
  }
}
```

5.1.5 Import Service

```
package com.npss.database.queries;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.util.Scanner;

/**
* Import Service: Enter new teams from a data file until the file is empty
*
* Expected file format (CSV, one team per line):
* team_id,formation_date,focus_date,team_leader
*
* Note: Empty values for focus_date or team_leader will be treated as NULL
*
* @author Astra Nguyen
```

```
* @version 1.0
*/
public class ImportService {
  private Connection connection;
  private Scanner scanner;

  public ImportService(Connection connection, Scanner scanner) {
     this.connection = connection;
     this.scanner = scanner;
  }

  /**
   * Executes the import functionality: Enter new teams from a data file until the file is empty
   *
   * @throws SQLException if a database error occurs
   */
  public void execute() throws SQLException {
     System.out.println("\n[Import] Enter new teams from a data file until the file is empty");
     System.out.print("Please enter the input file name: ");
     String inputFileName = scanner.nextLine().trim();

     if (inputFileName.isEmpty()) {
        System.out.println("Error: File name cannot be empty.");
        return;
     }

     // Try to find the file in current directory or project root
     Path filePath = findFile(inputFileName);

     if (filePath == null || !Files.exists(filePath)) {
        System.err.println("Error: File '" + inputFileName + "' not found.");
        System.err.println("Please ensure the file exists in the current directory or project root.");
        return;
     }

     System.out.println("Reading from file: " + filePath.toAbsolutePath());

     // SQL query for inserting teams
     String insertTeamSQL =
        "INSERT INTO Ranger_team(team_id, formation_date, focus_date, team_leader) " +
        "VALUES (?, ?, ?, ?)";

     int totalLines = 0;
     int successCount = 0;
```

```
        int errorCount = 0;

        try (BufferedReader reader = new BufferedReader(new FileReader(filePath.toFile()))) {
            String line;
            boolean isFirstLine = true;

            while ((line = reader.readLine()) != null) {
                totalLines++;
                line = line.trim();

                // Skip empty lines
                if (line.isEmpty()) {
                    continue;
                }

                // Skip header line if present
                if (isFirstLine && line.toLowerCase().startsWith("team_id")) {
                    isFirstLine = false;
                    continue;
                }
                isFirstLine = false;

                // Parse CSV line
                String[] parts = parseCSVLine(line);

                if (parts.length < 2) {
                    System.err.println("Line " + totalLines + ": Invalid format (expected at least 2 fields).
Skipping: " + line);
                    errorCount++;
                    continue;
                }

                String teamId = parts[0].trim();
                String formationDateStr = parts[1].trim();
                String focusDateStr = (parts.length > 2 && !parts[2].trim().isEmpty()) ? parts[2].trim() :
null;
                String teamLeader = (parts.length > 3 && !parts[3].trim().isEmpty()) ? parts[3].trim() :
null;

                // Validate required fields
                if (teamId.isEmpty() || formationDateStr.isEmpty()) {
                    System.err.println("Line " + totalLines + ": Missing required fields (team_id or
formation_date). Skipping: " + line);
                    errorCount++;
```

```
                continue;
            }

            // Insert team
            try {
                connection.setAutoCommit(false);

                try (PreparedStatement pstmt = connection.prepareStatement(insertTeamSQL)) {
                    pstmt.setString(1, teamId);
                    pstmt.setDate(2, java.sql.Date.valueOf(formationDateStr));

                    if (focusDateStr != null && !focusDateStr.isEmpty()) {
                        pstmt.setDate(3, java.sql.Date.valueOf(focusDateStr));
                    } else {
                        pstmt.setNull(3, java.sql.Types.DATE);
                    }

                    if (teamLeader != null && !teamLeader.isEmpty()) {
                        pstmt.setString(4, teamLeader);
                    } else {
                        pstmt.setNull(4, java.sql.Types.VARCHAR);
                    }

                    pstmt.executeUpdate();
                    connection.commit();

                    System.out.println("Imported team: " + teamId + " (formation: " + formationDateStr
+ ")");
                    successCount++;
                }
            } catch (SQLException e) {
                connection.rollback();
                System.err.println("Line " + totalLines + ": Failed to import team '" + teamId + "': " +
e.getMessage());
                errorCount++;
            } catch (IllegalArgumentException e) {
                connection.rollback();
                System.err.println("Line " + totalLines + ": Invalid date format for team '" + teamId +
"': " + e.getMessage());
                System.err.println("   Expected format: YYYY-MM-DD");
                errorCount++;
            } finally {
                connection.setAutoCommit(true);
            }
```

```
        }

        // Display summary result
        System.out.println("Total lines processed: " + totalLines);
        System.out.println("Successfully imported: " + successCount);
        System.out.println("Errors: " + errorCount);

        if (successCount > 0) {
            System.out.println("\nImport completed successfully!");
        }

    } catch (IOException e) {
        System.err.println("Error reading file: " + e.getMessage());
        throw new SQLException("File I/O error: " + e.getMessage(), e);
    }
}

/**
 * Finds the file in current directory or project root
 */
private Path findFile(String fileName) {
    // Try current directory first
    Path currentDir = Paths.get(System.getProperty("user.dir"));
    Path filePath = currentDir.resolve(fileName);

    if (Files.exists(filePath)) {
        return filePath;
    }

    // Try project root first
    if (currentDir.getFileName().toString().equals("NPSS_Database_App")) {
        return filePath;
    } else {
        // Try going up one level to find NPSS_Database_App
        Path projectRoot = currentDir.resolve("NPSS_Database_App");
        if (Files.exists(projectRoot)) {
            Path projectFile = projectRoot.resolve(fileName);
            if (Files.exists(projectFile)) {
                return projectFile;
            }
        }
    }
    return filePath;
}
```

```
    /**
     * Parses a CSV line, handling quoted fields
     */
    private String[] parseCSVLine(String line) {
        // Simple CSV parser - splits by comma, trims whitespace
        // For more complex CSV (with quoted fields containing commas), use a CSV library
        return line.split(",");
    }
}
```

5.1.6 Export Service

```
package com.npss.database.queries;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Scanner;

/**
* Export Service: Retrieve names and mailing addresses of all people on the mailing list
* and output them to a data file instead of screen
*
* The mailing list consists of all individuals with newsletter_status = true
*
* @author Astra Nguyen
* @version 1.0
*/
public class ExportService {
    private Connection connection;
    private Scanner scanner;

    public ExportService(Connection connection, Scanner scanner) {
        this.connection = connection;
        this.scanner = scanner;
    }

    /**
```

```
    * Executes the export functionality: Retrieve names and mailing addresses of all people on
the mailing list
    * and output them to a data file instead of screen
    *
    * @throws SQLException if a database error occurs
    */
   public void execute() throws SQLException {
      System.out.println("\n[Export] Retrieve names and mailing addresses of all people on the
mailing list");
      System.out.print("Please enter the output file name: ");
      String outputFileName = scanner.nextLine().trim();

      if (outputFileName.isEmpty()) {
         System.out.println("Error: File name cannot be empty.");
         return;
      }

      // Determine file path (current directory or project root)
      Path filePath = findFile(outputFileName);

      // SQL query to retrieve all people on the mailing list (newsletter_status = true)
      String SQL =
         "SELECT " +
         "    CONCAT(i.first_name, ' ', i.last_name) AS full_name, " +
         "    i.street, " +
         "    i.city, " +
         "    i.state, " +
         "    i.postal_code " +
         "FROM Individual i " +
         "WHERE i.newsletter_status = 1 " +
         "ORDER BY i.last_name, i.first_name";

      int recordCount = 0;

      try (PreparedStatement pstmt = connection.prepareStatement(SQL);
           ResultSet rs = pstmt.executeQuery();
           BufferedWriter writer = new BufferedWriter(new FileWriter(filePath.toFile()))) {

         // Write header line
         writer.write("Name,Street,City,State,Postal Code");
         writer.newLine();

         // Write each record
         while (rs.next()) {
```

```
            String fullName = rs.getString("full_name");
            String street = rs.getString("street");
            String city = rs.getString("city");
            String state = rs.getString("state");
            String postalCode = rs.getString("postal_code");

            // Format as CSV: Name,Street,City,State,Postal Code
            writer.write(escapeCSV(fullName) + ",");
            writer.write(escapeCSV(street) + ",");
            writer.write(escapeCSV(city) + ",");
            writer.write(escapeCSV(state) + ",");
            writer.write(escapeCSV(postalCode));
            writer.newLine();

            recordCount++;
        }

        System.out.println("Export completed successfully!");
        System.out.println("File saved to: " + filePath.toAbsolutePath());
        System.out.println("Total records exported: " + recordCount);

    } catch (IOException e) {
        System.err.println("Error writing to file: " + e.getMessage());
        throw new SQLException("File I/O error: " + e.getMessage(), e);
    } catch (SQLException e) {
        System.err.println("Database error: " + e.getMessage());
        if (e.getSQLState() != null) {
            System.err.println("SQL State: " + e.getSQLState());
        }
        throw e;
    }
}

/**
 * Finds the file path in current directory or project root
 */
private Path findFile(String fileName) {
    // Try current directory first
    Path currentDir = Paths.get(System.getProperty("user.dir"));
    Path filePath = currentDir.resolve(fileName);

    // If in the project root, use it
    if (currentDir.getFileName().toString().equals("NPSS_Database_App")) {
        return filePath;
```

```
        } else {
            // Try going up one level to find NPSS_Database_App
            Path projectRoot = currentDir.resolve("NPSS_Database_App");
            if (Files.exists(projectRoot)) {
                return projectRoot.resolve(fileName);
            }
        }
        return filePath;
    }

    /**
     * Escapes CSV values (handles commas, quotes, and NULL values)
     */
    private String escapeCSV(String value) {
        if (value == null) {
            return "";
        }
        // If value contains comma, quote, or newline, wrap in quotes and escape quotes
        if (value.contains(",") || value.contains("\"") || value.contains("\n")) {
            return "\"" + value.replace("\"", "\"\"") + "\"";
        }
        return value;
    }
}
```

Task 6. Java Program Execution

Query 3 → Query 7 → Query 1 → Query 2 → Query 5 → Query 6 → Query 4

Query 3:

Creating Ranger Team #1

```
============================================================
WELCOME TO THE NATIONAL PARK SERVICE SYSTEM DATABASE(NPSS)!
============================================================
(1)  Insert a new visitor into the database and associate them with one or more park progr
ams
(2)  Insert a new ranger into the database and assign them to a ranger team
(3)  Insert a new ranger team into the database and set its leader
(4)  Insert a new donation from a donor
(5)  Insert a new researcher into the database and associate them with one or more ranger
teams
(6)  Insert a report submitted by a ranger team to a researcher
(7)  Insert a new park program into the database for a specific park
(8)  Retrieve the names and contact information of all emergency contacts for a specific p
erson
(9)  Retrieve the list of visitors enrolled in a specific park program, including their ac
cessibility needs
(10) Retrieve all park programs for a specific park that started after a given date
(11) Retrieve the total and average donation amount received in a month from all anonymous
 donors
(12) Retrieve the list of rangers in a team, including their certifications, years of serv
ice and their role in the team
(13) Retrieve the names, IDs, contact information, and newsletter subscription status of a
ll individuals in the database
(14) Update the salary of researchers overseeing more than one ranger team by a 3% increas
e
(15) Delete visitors who have not enrolled in any park programs and whose park passes have
 expired
(16) Import: Enter new teams from a data file until the file is empty
(17) Export: Retrieve names and mailing addresses of all people on the mailing list
(18) Quit
============================================================
Please select an option (1-18): 3

[Query 3] Insert a new ranger team into the database and set its leader
Enter team ID: T001
Enter formation date (YYYY-MM-DD): 2020-01-01
Enter focus date (YYYY-MM-DD) or press Enter for NULL: 2020-02-01
Enter team leader ID (ranger ID) or press Enter for NULL:
Ranger team inserted successfully! (Rows affected: 1)
```

Creating Ranger Team #2

```
============================================================
WELCOME TO THE NATIONAL PARK SERVICE SYSTEM DATABASE(NPSS)!
============================================================
(1)  Insert a new visitor into the database and associate them with one or more park progr
ams
(2)  Insert a new ranger into the database and assign them to a ranger team
(3)  Insert a new ranger team into the database and set its leader
(4)  Insert a new donation from a donor
(5)  Insert a new researcher into the database and associate them with one or more ranger
teams
(6)  Insert a report submitted by a ranger team to a researcher
(7)  Insert a new park program into the database for a specific park
(8)  Retrieve the names and contact information of all emergency contacts for a specific p
erson
(9)  Retrieve the list of visitors enrolled in a specific park program, including their ac
cessibility needs
(10) Retrieve all park programs for a specific park that started after a given date
(11) Retrieve the total and average donation amount received in a month from all anonymous
 donors
(12) Retrieve the list of rangers in a team, including their certifications, years of serv
ice and their role in the team
(13) Retrieve the names, IDs, contact information, and newsletter subscription status of a
ll individuals in the database
(14) Update the salary of researchers overseeing more than one ranger team by a 3% increas
e
(15) Delete visitors who have not enrolled in any park programs and whose park passes have
 expired
(16) Import: Enter new teams from a data file until the file is empty
(17) Export: Retrieve names and mailing addresses of all people on the mailing list
(18) Quit
============================================================
Please select an option (1-18): 3

[Query 3] Insert a new ranger team into the database and set its leader
Enter team ID: T002
Enter formation date (YYYY-MM-DD): 2021-03-15
Enter focus date (YYYY-MM-DD) or press Enter for NULL: 2021-12-04
Enter team leader ID (ranger ID) or press Enter for NULL:
Ranger team inserted successfully! (Rows affected: 1)
```

Creating Ranger Team #3

```
(11) Retrieve the total and average donation amount received in a month from all anonymous
 donors
(12) Retrieve the list of rangers in a team, including their certifications, years of serv
ice and their role in the team
(13) Retrieve the names, IDs, contact information, and newsletter subscription status of a
ll individuals in the database
(14) Update the salary of researchers overseeing more than one ranger team by a 3% increas
e
(15) Delete visitors who have not enrolled in any park programs and whose park passes have
 expired
(16) Import: Enter new teams from a data file until the file is empty
(17) Export: Retrieve names and mailing addresses of all people on the mailing list
(18) Quit
============================================================
Please select an option (1-18): 3

[Query 3] Insert a new ranger team into the database and set its leader
Enter team ID: T003
Enter formation date (YYYY-MM-DD): 2022-06-01
Enter focus date (YYYY-MM-DD) or press Enter for NULL: 2022-07-01
Enter team leader ID (ranger ID) or press Enter for NULL:
Ranger team inserted successfully! (Rows affected: 1)
```

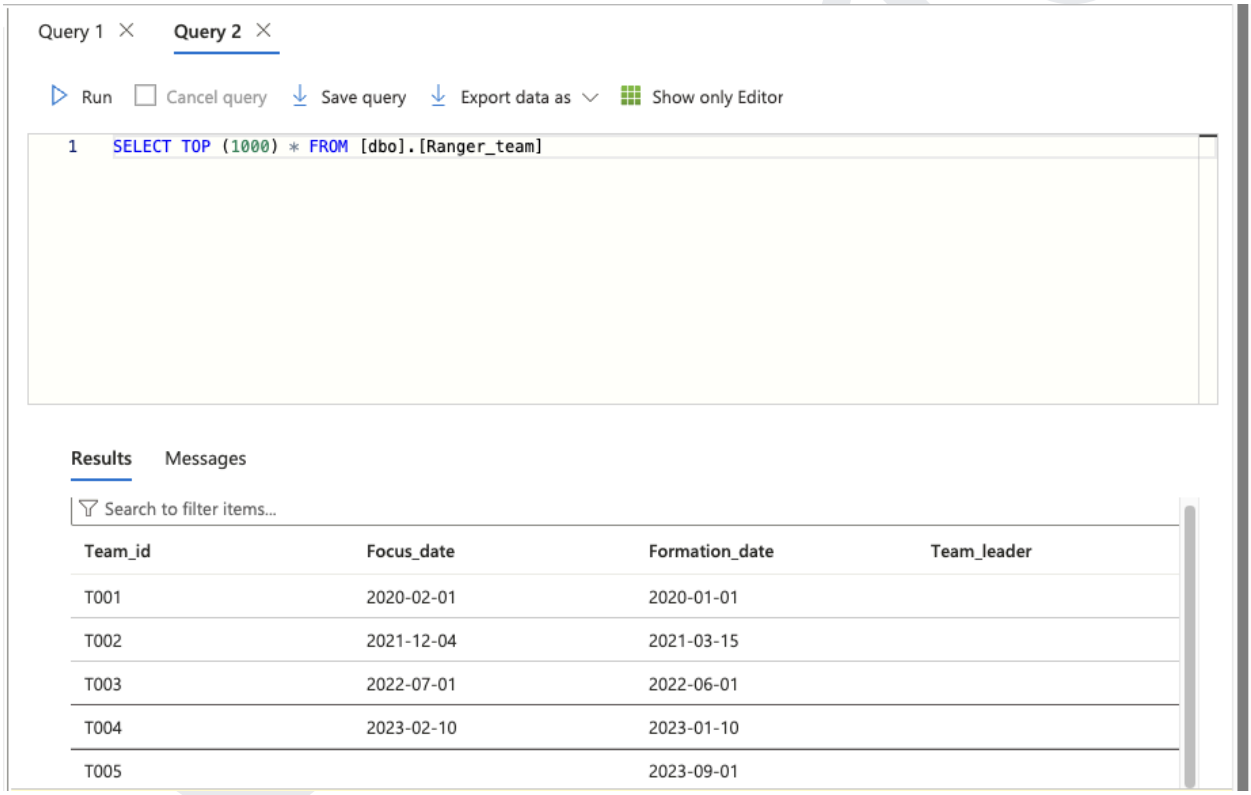
Creating Ranger Team #4

```
(12) Retrieve the list of rangers in a team, including their certifications, years of serv
ice and their role in the team
(13) Retrieve the names, IDs, contact information, and newsletter subscription status of a
ll individuals in the database
(14) Update the salary of researchers overseeing more than one ranger team by a 3% increas
e
(15) Delete visitors who have not enrolled in any park programs and whose park passes have
 expired
(16) Import: Enter new teams from a data file until the file is empty
(17) Export: Retrieve names and mailing addresses of all people on the mailing list
(18) Quit
============================================================
Please select an option (1-18): 3

[Query 3] Insert a new ranger team into the database and set its leader
Enter team ID: T004
Enter formation date (YYYY-MM-DD): 2023-01-10
Enter focus date (YYYY-MM-DD) or press Enter for NULL: 2023-02-10
Enter team leader ID (ranger ID) or press Enter for NULL:
Ranger team inserted successfully! (Rows affected: 1)
```

Creating Ranger Team #5

```
(15) Delete visitors who have not enrolled in any park programs and whose park passes have
 expired
(16) Import: Enter new teams from a data file until the file is empty
(17) Export: Retrieve names and mailing addresses of all people on the mailing list
(18) Quit
============================================================
Please select an option (1-18): 3

[Query 3] Insert a new ranger team into the database and set its leader
Enter team ID: T005
Enter formation date (YYYY-MM-DD): 2023-09-01
Enter focus date (YYYY-MM-DD) or press Enter for NULL:
Enter team leader ID (ranger ID) or press Enter for NULL:
Ranger team inserted successfully! (Rows affected: 1)
✓ VERIFIED: Team T005 is now in the database

Press Enter to continue...
```

Result:



Query 7:

Insert a new park program #1 into the database for a specific park, if the park does not exist create it.

```
ll individuals in the database
(14) Update the salary of researchers overseeing more than one ranger team by a 3% increas
e
(15) Delete visitors who have not enrolled in any park programs and whose park passes have
 expired
(16) Import: Enter new teams from a data file until the file is empty
(17) Export: Retrieve names and mailing addresses of all people on the mailing list
(18) Quit
============================================================
Please select an option (1-18): 7

[Query 7] Insert a new park program into the database for a specific park
Enter program name: Wildlife Watching Tour
Enter program type: Educational
Enter start date (YYYY-MM-DD): 2024-06-01
Enter duration (in days): 3
Enter park name: Yellowstone National Park

Park 'Yellowstone National Park' does not exist. Creating new park...
Please provide the following park information:
Enter street address: 1 Park Road
Enter city: Yellowstone
Enter state: WY
Enter postal code: 82190
Enter establishment date (YYYY-MM-DD): 1872-03-01
Enter capacity: 10000
Park 'Yellowstone National Park' created successfully! (Rows affected: 1)
Park program inserted successfully! (Program rows: 1, Link rows: 1)
VERIFIED: Program 'Wildlife Watching Tour' is now linked to park 'Yellowstone National Par
k' in the database

Press Enter to continue...
```

Insert a new park program #2 into the database for a specific park, if the park does not exist create it.

```
(16) Import: Enter new teams from a data file until the file is empty
(17) Export: Retrieve names and mailing addresses of all people on the mailing list
(18) Quit
============================================================
Please select an option (1-18): 7

[Query 7] Insert a new park program into the database for a specific park
Enter program name: Hiking Adventure
Enter program type: Recreational
Enter start date (YYYY-MM-DD): 2024-07-15
Enter duration (in days): 5
Enter park name: Yosemite National Park

Park 'Yosemite National Park' does not exist. Creating new park...
Please provide the following park information:
Enter street address: 9035 Village Dr
Enter city: Yosemite Valley
Enter state: CA
Enter postal code: 95389
Enter establishment date (YYYY-MM-DD): 1890-10-01
Enter capacity: 8000
Park 'Yosemite National Park' created successfully! (Rows affected: 1)
Park program inserted successfully! (Program rows: 1, Link rows: 1)
VERIFIED: Program 'Hiking Adventure' is now linked to park 'Yosemite National Park' in the
 database

Press Enter to continue...
```

Insert a new park program #3 into the database for a specific park, if the park does not exist create it.

```
(18) Quit
============================================================
Please select an option (1-18): 7

[Query 7] Insert a new park program into the database for a specific park
Enter program name: Nature Photography Event
Enter program type: Educational
Enter start date (YYYY-MM-DD): 2024-08-01
Enter duration (in days): 3
Enter park name: Grand Canyon National Park

Park 'Grand Canyon National Park' does not exist. Creating new park...
Please provide the following park information:
Enter street address: 20 South Entrance Road
Enter city: Grand Canyon
Enter state: AZ
Enter postal code: 86023
Enter establishment date (YYYY-MM-DD): 1919-02-26
Enter capacity: 12000
Park 'Grand Canyon National Park' created successfully! (Rows affected: 1)
Park program inserted successfully! (Program rows: 1, Link rows: 1)
VERIFIED: Program 'Nature Photography Event' is now linked to park 'Grand Canyon National
Park' in the database

Press Enter to continue...
```

Insert a new park program #4 into the database for a specific park, if the park does not exist create it.

```
(17) Export: Retrieve names and mailing addresses of all people on the mailing list
(18) Quit
============================================================
Please select an option (1-18): 7

[Query 7] Insert a new park program into the database for a specific park
Enter program name: Camping Edcuation Basic
Enter program type: Recreational
Enter start date (YYYY-MM-DD): 2024-09-10
Enter duration (in days): 4
Enter park name: Zion National Park

Park 'Zion National Park' does not exist. Creating new park...
Please provide the following park information:
Enter street address: 31 Zion Park Blvd
Enter city: Springdale
Enter state: UT
Enter postal code: 84767
Enter establishment date (YYYY-MM-DD): 1919-11-19
Enter capacity: 5000
Park 'Zion National Park' created successfully! (Rows affected: 1)
Park program inserted successfully! (Program rows: 1, Link rows: 1)
VERIFIED: Program 'Camping Edcuation Basic' is now linked to park 'Zion National Park' in
the database

Press Enter to continue...
```

Insert a new park program #5 into the database for a specific park, if the park does not exist create it.

```
(t individuals in the database
(14) Update the salary of researchers overseeing more than one ranger team by a 3% increas
e
(15) Delete visitors who have not enrolled in any park programs and whose park passes have
 expired
(16) Import: Enter new teams from a data file until the file is empty
(17) Export: Retrieve names and mailing addresses of all people on the mailing list
(18) Quit
============================================================
Please select an option (1-18): 7

[Query 7] Insert a new park program into the database for a specific park
Enter program name: Bird Watching Expedition
Enter program type: Educational
Enter start date (YYYY-MM-DD): 2024-10-05
Enter duration (in days): 6
Enter park name: Zion National Park
Park 'Zion National Park' found in database.
Park program inserted successfully! (Program rows: 1, Link rows: 1)
VERIFIED: Program 'Bird Watching Expedition' is now linked to park 'Zion National Park' in
 the database

Press Enter to continue...
```

Result:

```
1   SELECT TOP (1000) * FROM [dbo].[National_parks_offers_program]
```

Results Messages

Search to filter items...

Park_name	Program_name
Zion National Park	Bird Watching Expedition
Zion National Park	Camping Edcuation Basic
Yosemite National Park	Hiking Adventure
Grand Canyon National Park	Nature Photography Event
Yellowstone National Park	Wildlife Watching Tour

Query 1:

Insert Visistor #1

```
============================================================
Please select an option (1-18): 1

[Query 1] Insert a new visitor into the database and associate them with one or more park
programs
Enter visitor ID number: V001
Enter first name: John
Enter last name: Ma
Enter gender (M/F/O): M
Enter street address: 123 Main St
Enter city: Springfield
Enter state: IL
Enter postal code: 62701
Enter date of birth (YYYY-MM-DD): 1990-01-15
Subscribe to newsletter? (true/false): true
Enter visit date (YYYY-MM-DD) or press Enter for NULL: 2024-01-10
Enter accessibility needs (or press Enter for NULL): Wheelchair access
How many park programs to enroll? (0 or more): 1
Enter program name 1: Wildlife Watching Tour
Visitor inserted successfully! (Individual rows: 1, Visitor rows: 1)
```

Insert Visistor #2

```
============================================================
Please select an option (1-18): 1

[Query 1] Insert a new visitor into the database and associate them with one or more park
programs
Enter visitor ID number: V002
Enter first name: Jane
Enter last name: Smith
Enter gender (M/F/O): F
Enter street address: 456 Oak Ave
Enter city: Chicago
Enter state: IL
Enter postal code: 60601
Enter date of birth (YYYY-MM-DD): 1985-05-20
Subscribe to newsletter? (true/false): False
Enter visit date (YYYY-MM-DD) or press Enter for NULL: 2024-02-15
Enter accessibility needs (or press Enter for NULL):
How many park programs to enroll? (0 or more): 0
Visitor inserted successfully! (Individual rows: 1, Visitor rows: 1)
```

Insert Visistor #3

```
 expired
(16) Import: Enter new teams from a data file until the file is empty
(17) Export: Retrieve names and mailing addresses of all people on the mailing list
(18) Quit
============================================================
Please select an option (1-18): 1

[Query 1] Insert a new visitor into the database and associate them with one or more park
programs
Enter visitor ID number: V003
Enter first name: Bobby
Enter last name: Johnson
Enter gender (M/F/O): M
Enter street address: 789 Pine Rd
Enter city: Peoria
Enter state: Il
Enter postal code: 61601
Enter date of birth (YYYY-MM-DD): 1992-08-30
Subscribe to newsletter? (true/false): True
Enter visit date (YYYY-MM-DD) or press Enter for NULL:
Enter accessibility needs (or press Enter for NULL): Hearing aid
How many park programs to enroll? (0 or more): 2
Enter program name 1: Hiking Adventure
Enter program name 2: Nature Photography Event
Visitor inserted successfully!
```

Insert Visistor #4

```
============================================================
Please select an option (1-18): 1

[Query 1] Insert a new visitor into the database and associate them with one or more park
programs
Enter visitor ID number: V004
Enter first name: Alice
Enter last name: Williams
Enter gender (M/F/O): F
Enter street address: 321 Elm St
Enter city: Rockford
Enter state: IL
Enter postal code: 61101
Enter date of birth (YYYY-MM-DD): 1988-12-05
Subscribe to newsletter? (true/false): true
Enter visit date (YYYY-MM-DD) or press Enter for NULL: 2024-03-20
Enter accessibility needs (or press Enter for NULL):
How many park programs to enroll? (0 or more): 1
Enter program name 1: Hiking adventure
Visitor inserted successfully!
```

Insert Visistor #5

```
(16) Import: Enter new teams from a data file until the file is empty
(17) Export: Retrieve names and mailing addresses of all people on the mailing list
(18) Quit
============================================================
Please select an option (1-18): 1

[Query 1] Insert a new visitor into the database and associate them with one or more park
programs
Enter visitor ID number: V005
Enter first name: Zoe
Enter last name: Sabardu
Enter gender (M/F/O): F
Enter street address: 654 Maple Dr
Enter city: Montreal
Enter state: Canada
Enter postal code: 60540
Enter date of birth (YYYY-MM-DD): 2003-05-09
Subscribe to newsletter? (true/false): True
Enter visit date (YYYY-MM-DD) or press Enter for NULL: 2024-04-01
Enter accessibility needs (or press Enter for NULL):
How many park programs to enroll? (0 or more): 0
Visitor inserted successfully!
```

Result:

Results Messages

	Id_number	First_name	Middle_initial	Last_name	Gender	Street	City	State	Postal_code
1	V001	John	*NULL*	Ma	M	123 Main St	Springfield	IL	62701
2	V002	Jane	*NULL*	Smith	F	456 Oak Ave	Chicago	IL	60601
3	V003	Bobby	*NULL*	Johnson	M	789 Pine Rd	Peoria	Il	61601
4	V004	Alice	*NULL*	Williams	F	321 Elm St	Rockford	IL	61101
5	V005	Zoe	*NULL*	Sabardu	F	654 Maple Dr	Montreal	Canada	60540

Results grid

State	Postal_code	Date_of_birth	Newsletter_status
IL	62701	1990-01-15	1
IL	60601	1985-05-20	0
Il	61601	1992-08-30	1
IL	61101	1988-12-05	1
Canada	60540	2003-05-09	1

Query 2:

Insert a new ranger #1 into the database and assign them to a ranger team

```
(16) Import: Enter new teams from a data file until the file is empty
(17) Export: Retrieve names and mailing addresses of all people on the mailing list
(18) Quit
============================================================
Please select an option (1-18): 2

[Query 2] Insert a new ranger into the database and assign them to a ranger team
Enter ranger ID number: R001
Enter first name: Mike
Enter last name: Ranger
Enter gender (M/F/O): M
Enter street address: 100 Park Way
Enter city: Springfield
Enter state: IL
Enter postal code: 62701
Enter date of birth (YYYY-MM-DD): 1980-01-10
Subscribe to newsletter? (true/false): True
Enter team ID to assign ranger to: T001
Enter start date (YYYY-MM-DD): 2020-01-01
Enter status (e.g., Active, On Leave, etc.): Active
How many certifications? (0 or more): 2
Enter certification 1: Wildlife Management
Enter certification 2: First Aid
Ranger inserted and assigned to team successfully!

Press Enter to continue...
```

Insert a new ranger #2 into the database and assign them to a ranger team

```
(16) Import: Enter new teams from a data file until the file is empty
(17) Export: Retrieve names and mailing addresses of all people on the mailing list
(18) Quit
============================================================
Please select an option (1-18): 2

[Query 2] Insert a new ranger into the database and assign them to a ranger team
Enter ranger ID number: R002
Enter first name: Sarah
Enter last name: Forest
Enter gender (M/F/O): F
Enter street address: 200 Trail Rd
Enter city: Chicago
Enter state: IL
Enter postal code: 60601
Enter date of birth (YYYY-MM-DD): 1985-05-15
Subscribe to newsletter? (true/false): false
Enter team ID to assign ranger to: T002
Enter start date (YYYY-MM-DD): 2021-03-15
Enter status (e.g., Active, On Leave, etc.): Active
How many certifications? (0 or more): 1
Enter certification 1: Search and Rescue
Ranger inserted and assigned to team successfully!

Press Enter to continue...
```

Insert a new ranger #3 into the database and assign them to a ranger team

```
Please select an option (1-18): 2

[Query 2] Insert a new ranger into the database and assign them to a ranger team
Enter ranger ID number: R003
Enter first name: Tom
Enter last name: Sarp
Enter gender (M/F/O): M
Enter street address: 300 Peak Ave
Enter city: Peoria
Enter state: IL
Enter postal code: 61601
Enter date of birth (YYYY-MM-DD): 1990-08-20
Subscribe to newsletter? (true/false): true
Enter team ID to assign ranger to: T003
Enter start date (YYYY-MM-DD): 2022-06-01
Enter status (e.g., Active, On Leave, etc.): On leave
How many certifications? (0 or more): 0
Database error: The INSERT statement conflicted with the CHECK constraint "CK_ranger_assig
ned_ranger_team_status". The conflict occurred in database "CS-DSA-4513-SQL-DB", table "db
o.Ranger_assigned_ranger_team", column 'Status'.
SQL State: 23000

Database error occurred:
  Message: The INSERT statement conflicted with the CHECK constraint "CK_ranger_assigned_r
anger_team_status". The conflict occurred in database "CS-DSA-4513-SQL-DB", table "dbo.Ran
ger_assigned_ranger_team", column 'Status'.
  SQL State: 23000
  Error Code: 547
```

Insert a new ranger #4 into the database and assign them to a ranger team

```
(18) Quit
============================================================
Please select an option (1-18): 2

[Query 2] Insert a new ranger into the database and assign them to a ranger team
Enter ranger ID number: R004
Enter first name: Lisa
Enter last name: nguyen
Enter gender (M/F/O): F
Enter street address: 400 Stream Blvd
Enter city: Rockford
Enter state: IL
Enter postal code: 61101
Enter date of birth (YYYY-MM-DD): 1988-11-10
Subscribe to newsletter? (true/false): True
Enter team ID to assign ranger to: T004
Enter start date (YYYY-MM-DD): 2023-01-10
Enter status (e.g., Active, On Leave, etc.): Active
How many certifications? (0 or more): 3
Enter certification 1: Wildlife Management
Enter certification 2: First Aid
Enter certification 3: Emergency Response
Ranger inserted and assigned to team successfully!

Press Enter to continue...
```

Insert a new ranger #5 into the database and assign them to a ranger team

```
============================================================
Please select an option (1-18): 2

[Query 2] Insert a new ranger into the database and assign them to a ranger team
Enter ranger ID number: R005
Enter first name: Brandon
Enter last name: Valley
Enter gender (M/F/O): M
Enter street address: 500 Canyon Dr
Enter city: Naperville
Enter state: IL
Enter postal code: 60540
Enter date of birth (YYYY-MM-DD): 1992-04-25
Subscribe to newsletter? (true/false): false
Enter team ID to assign ranger to: T005
Enter start date (YYYY-MM-DD): 2023-09-01
Enter status (e.g., Active, On Leave, etc.): Active
How many certifications? (0 or more): 1
Enter certification 1: 1
ConservationRanger inserted and assigned to team successfully!
```

Result:

Results Messages

	Team_id	Focus_date	Formation_date	Team_lea…
1	T001	2020-02-01	2020-01-01	NULL
2	T002	2021-12-04	2021-03-15	NULL
3	T003	2022-07-01	2022-06-01	NULL
4	T004	2023-02-10	2023-01-10	NULL
5	T005	NULL	2023-09-01	NULL

Results Messages

	Id_num…	Certificat…
1	R001	First Aid
2	R001	Wildlife Manag…
3	R002	Search and Res…
4	R004	Emergency Resp…
5	R004	First Aid
6	R004	Wildlife Manag…
7	R005	1

Results Messages

	Ranger…	Team_id	Start_date	Status	Years_of_…
1	R001	T001	2020-01-01	Active	5
2	R002	T002	2021-03-15	Active	4
3	R004	T004	2023-01-10	Active	2
4	R005	T005	2023-09-01	Active	2

Results Messages

	Id_num…	First_name	Middle_initial	Last_name	Gender	Street	City	State	Postal_
1	R001	Mike	NULL	Ranger	M	100 Park Way	Springfield	IL	6270
2	R002	Sarah	NULL	Forest	F	200 Trail Rd	Chicago	IL	6060
3	R004	Lisa	NULL	nguyen	F	400 Stream Bl…	Rockford	IL	6110
4	R005	Brandon	NULL	Valley	M	500 Canyon Dr	Naperville	IL	6054
5	V001	John	NULL	Ma	M	123 Main St	Springfield	IL	6270
6	V002	Jane	NULL	Smith	F	456 Oak Ave	Chicago	IL	6060
7	V003	Bobby	NULL	Johnson	M	789 Pine Rd	Peoria	Il	6160
8	V004	Alice	NULL	Williams	F	321 Elm St	Rockford	IL	6110

Query 5:

Insert a new researcher #1 into the database and associate them with one or more ranger teams

```
============================================================
Please select an option (1-18): 5

[Query 5] Insert a new researcher into the database and associate them with one or more ra
nger teams
Enter researcher ID number: RES001
Enter first name: Dr. Emily
Enter last name: Science
Enter gender (M/F/O): F
Enter street address: 600 Research Ave
Enter city: Springfield
Enter state: IL
Enter postal code: 62701
Enter date of birth (YYYY-MM-DD): 1975-01-20
Subscribe to newsletter? (true/false): True
Enter research field: Wildlife Biology
Enter hire date (YYYY-MM-DD): 2018-01-15
Enter salary: 75000.00
How many ranger teams to associate? (1 or more): 2
Enter team ID 1: T001
Enter report date (YYYY-MM-DD): 2024-01-20
Enter summary (or press Enter for NULL): Monthly wildlife observation report
Enter team ID 2: T002
Enter report date (YYYY-MM-DD): 2024-02-15
Enter summary (or press Enter for NULL):
Researcher inserted and associated with teams successfully!
```

Insert a new researcher #2 into the database and associate them with one or more ranger teams

```
============================================================
Please select an option (1-18): 5

[Query 5] Insert a new researcher into the database and associate them with one or more ra
nger teams
Enter researcher ID number: RES002
Enter first name: Dr. James
Enter last name: Ecology
Enter gender (M/F/O): M
Enter street address: 700 Study Blvd
Enter city: Chicago
Enter state: IL
Enter postal code: 60601
Enter date of birth (YYYY-MM-DD): 1980-05-10
Subscribe to newsletter? (true/false): False
Enter research field: Environmental Science
Enter hire date (YYYY-MM-DD): 2019-03-01
Enter salary: 80000.00
How many ranger teams to associate? (1 or more): 1
Enter team ID 1: T003
Enter report date (YYYY-MM-DD): 2024-03-10
Enter summary (or press Enter for NULL): Quarterly conservation status update
Researcher inserted and associated with teams successfully!
```

Insert a new researcher #3 into the database and associate them with one or more ranger teams

```
(18) Quit
============================================================
Please select an option (1-18): 5

[Query 5] Insert a new researcher into the database and associate them with one or more ra
nger teams
Enter researcher ID number: RES003
Enter first name: Dr. Maria
Enter last name: Conservation
Enter gender (M/F/O): F
Enter street address: 800 Analysis Dr
Enter city: Peoria
Enter state: IL
Enter postal code: 61601
Enter date of birth (YYYY-MM-DD): 1978-08-25
Subscribe to newsletter? (true/false): True
Enter research field: Conservation Biology
Enter hire date (YYYY-MM-DD): 2020-06-15
Enter salary: 95000.00
How many ranger teams to associate? (1 or more): 3
Enter team ID 1: T001
Enter report date (YYYY-MM-DD): 2024-01-25
Enter summary (or press Enter for NULL): Ecosystem health assessment
Enter team ID 2: T002
Enter report date (YYYY-MM-DD): 2024-02-20
Enter summary (or press Enter for NULL): Species population study
Enter team ID 3: T004
Enter report date (YYYY-MM-DD): 2024-03-15
Enter summary (or press Enter for NULL):
Researcher inserted and associated with teams successfully!

Press Enter to continue...
```

Insert a new researcher #4 into the database and associate them with one or more ranger teams

```
============================================================
Please select an option (1-18): 5

[Query 5] Insert a new researcher into the database and associate them with one or more ra
nger teams
Enter researcher ID number: RES004
Enter first name: Dr. Robert
Enter last name: Botany
Enter gender (M/F/O): M
Enter street address: 900 Plant St
Enter city: Rockford
Enter state: Il
Enter postal code: 61101
Enter date of birth (YYYY-MM-DD): 1982-11-15
Subscribe to newsletter? (true/false): True
Enter research field: Plant Ecology
Enter hire date (YYYY-MM-DD): 2021-09-01
Enter salary: 78000.00
How many ranger teams to associate? (1 or more): 1
Enter team ID 1: T005
Enter report date (YYYY-MM-DD): 2024-04-05
Enter summary (or press Enter for NULL):
Researcher inserted and associated with teams successfully!
```

Insert a new researcher #5 into the database and associate them with one or more ranger teams

```
(17) Export: Retrieve names and mailing addresses of all people on the mailing list
(18) Quit
============================================================
Please select an option (1-18): 5

[Query 5] Insert a new researcher into the database and associate them with one or more ra
nger teams
Enter researcher ID number: RES005
Enter first name: Dr. Zoe
Enter last name: Zoology
Enter gender (M/F/O): F
Enter street address: 1000 Animal Ways
Enter city: Montreal
Enter state: Canada
Enter postal code: 60540
Enter date of birth (YYYY-MM-DD): 1985-04-30
Subscribe to newsletter? (true/false): False
Enter research field: Animal Behavior
Enter hire date (YYYY-MM-DD): 2022-01-10
Enter salary: 92000.00
How many ranger teams to associate? (1 or more): 2
Enter team ID 1: T003
Enter report date (YYYY-MM-DD): 2024-05-01
Enter summary (or press Enter for NULL): Annual research findings summary
Enter team ID 2: T004
Enter report date (YYYY-MM-DD): 2024-05-15
Enter summary (or press Enter for NULL): Behavioral patterns analysis
Researcher inserted and associated with teams successfully!
```

Result:

Results Messages

	Id_num…	Research_f…	Hire_date	Salary
1	RES001	Wildlife Biolo…	2018-01-15	75000.00
2	RES002	Environmental …	2019-03-01	80000.00
3	RES003	Conservation B…	2020-06-15	95000.00
4	RES004	Plant Ecology	2021-09-01	78000.00
5	RES005	Animal Behavior	2022-01-10	92000.00

Results Messages

	Resear…	Team_id	Date	Summary
1	RES001	T001	2024-01-20	Monthly wildli…
2	RES001	T002	2024-02-15	*NULL*
3	RES002	T003	2024-03-10	Quarterly cons…
4	RES003	T001	2024-01-25	Ecosystem heal…
5	RES003	T002	2024-02-20	Species popula…
6	RES003	T004	2024-03-15	*NULL*
7	RES004	T005	2024-04-05	*NULL*
8	RES005	T003	2024-05-01	Annual researc…
9	RES005	T004	2024-05-15	Behavioral pat…

PROBLEMS OUTPUT TERMINAL TASKS

Query 6:

Insert a report #1

```
Please select an option (1-18): 6

[Query 6] Insert a report submitted by a ranger team to a researcher
Enter researcher ID number: RES001
Enter ranger team ID: T001
Enter report date (YYYY-MM-DD): 2024-06-01
Enter summary (or press Enter for NULL): Updated monthly wildlife observation report
Report inserted/updated successfully! (Rows affected: 1)
VERIFIED: Report for researcher RES001 and team T001 is in the database
  Date: 2024-06-01
  Summary: Updated monthly wildlife observation report
```

Insert a report #2

```
Please select an option (1-18): 6

[Query 6] Insert a report submitted by a ranger team to a researcher
Enter researcher ID number: RES002
Enter ranger team ID: T003
Enter report date (YYYY-MM-DD): 2024-06-15
Enter summary (or press Enter for NULL): Updated quarterly conservation status
Report inserted/updated successfully! (Rows affected: 1)
VERIFIED: Report for researcher RES002 and team T003 is in the database
  Date: 2024-06-15
  Summary: Updated quarterly conservation status
```

Insert a report #3

```
Please select an option (1-18): 6

[Query 6] Insert a report submitted by a ranger team to a researcher
Enter researcher ID number: RES003
Enter ranger team ID: T001
Enter report date (YYYY-MM-DD): 2024-07-01
Enter summary (or press Enter for NULL): Updated ecosystem health assessment
Report inserted/updated successfully! (Rows affected: 1)
VERIFIED: Report for researcher RES003 and team T001 is in the database
  Date: 2024-07-01
  Summary: Updated ecosystem health assessment
```

Insert a report #4

```
Please select an option (1-18): 6

[Query 6] Insert a report submitted by a ranger team to a researcher
Enter researcher ID number: RES004
Enter ranger team ID: T005
Enter report date (YYYY-MM-DD): 2024-07-15
Enter summary (or press Enter for NULL): Updated plant ecology findings
Report inserted/updated successfully! (Rows affected: 1)
VERIFIED: Report for researcher RES004 and team T005 is in the database
  Date: 2024-07-15
  Summary: Updated plant ecology findings
```

Insert a report #5

```
============================================================
Please select an option (1-18): 6

[Query 6] Insert a report submitted by a ranger team to a researcher
Enter researcher ID number: RES005
Enter ranger team ID: T003
Enter report date (YYYY-MM-DD): 2024-08-01
Enter summary (or press Enter for NULL): Updated annual research findings summary
Report inserted/updated successfully! (Rows affected: 1)
VERIFIED: Report for researcher RES005 and team T003 is in the database
  Date: 2024-08-01
  Summary: Updated annual research findings summary
```

Results:

Results Messages

	Resear…	Team_id	Date	Summary
1	RES001	T001	2024-06-01	Updated monthl…
2	RES001	T002	2024-02-15	*NULL*
3	RES002	T003	2024-06-15	Updated quarte…
4	RES003	T001	2024-07-01	Updated ecosys…
5	RES003	T002	2024-02-20	Species popula…
6	RES003	T004	2024-03-15	*NULL*
7	RES004	T005	2024-07-15	Updated plant …
8	RES005	T003	2024-08-01	Updated annual…
9	RES005	T004	2024-05-15	Behavioral pat…

Query 4:

Insert donor #1

```
------------------------------------------------------------
Please select an option (1-18): 4

[Query 4] Insert a new donation from a donor
Enter donation ID: D004
Enter donor ID number: DONOR004

Donor 'DONOR004' does not exist. Creating new donor...
Please provide the following information:
Individual record not found. Creating Individual first...
Enter first name: Mary
Enter last name: Williams
Enter gender (M/F/O): F
Enter street address: 321 Elm St
Enter city: aswdfawfw St
Enter state: IL
Enter postal code: 61101
Enter date of birth (YYYY-MM-DD): 1985-11-25
Subscribe to newsletter? (true/false): True
Enter donor preference (or press Enter for NULL): Education Programs
Enter donation date (YYYY-MM-DD): 2024-04-05
Enter donation amount: 750.00
Enter campaign name (or press Enter for NULL): Summer Fundraiser
Enter payment method (check/card): Card
Individual record created successfully! (Rows affected: 1)
Donor 'DONOR004' created successfully! (Rows affected: 1)
Enter card type: MasterCard
Enter last four digits: 5678
Enter expiration date (YYYY-MM-DD): 2026-06-30
Donation inserted successfully! (Donation rows: 1, Payment rows: 1)
VERIFIED: Donation D004 ($750.00) from donor DONOR004 is in the database
```

Insert donor #2

```
============================================================
Please select an option (1-18): 4

[Query 4] Insert a new donation from a donor
Enter donation ID: D002
Enter donor ID number: DONOR002

Donor 'DONOR002' does not exist. Creating new donor...
Please provide the following information:
Individual record not found. Creating Individual first...
Enter first name: Jane
Enter last name: Doe
Enter gender (M/F/O): F
Enter street address: 456 Oak Ave
Enter city: Oklahoma
Enter state: OK
Enter postal code: 73015
Enter date of birth (YYYY-MM-DD): 1975-08-20
Subscribe to newsletter? (true/false): True
Enter donor preference (or press Enter for NULL): Environmental Protection
Enter donation date (YYYY-MM-DD): 2024-02-20
Enter donation amount: 1000.00
Enter campaign name (or press Enter for NULL): Spring Campaign
Enter payment method (check/card): card
Individual record created successfully! (Rows affected: 1)
Donor 'DONOR002' created successfully! (Rows affected: 1)
Enter card type: Visa
Enter last four digits: 5792
Enter expiration date (YYYY-MM-DD): 2025-12-31
Donation inserted successfully! (Donation rows: 1, Payment rows: 1)
VERIFIED: Donation D002 ($1000.00) from donor DONOR002 is in the database

Press Enter to continue...
```

Insert donor #3

```
(13) Quit
============================================================
Please select an option (1-18): 4

[Query 4] Insert a new donation from a donor
Enter donation ID: D001
Enter donor ID number: DONOR001

Donor 'DONOR001' does not exist. Creating new donor...
Please provide the following information:
Individual record not found. Creating Individual first...
Enter first name: John
Enter last name: Smith
Enter gender (M/F/O): M
Enter street address: 123 Main St
Enter city: Springfield
Enter state: IL
Enter postal code: 62701
Enter date of birth (YYYY-MM-DD): 1980-05-15
Subscribe to newsletter? (true/false): true
Enter donor preference (or press Enter for NULL): Wildlife Conservation
Enter donation date (YYYY-MM-DD): 2024-01-15
Enter donation amount: 700.00
Enter campaign name (or press Enter for NULL): Check
Enter payment method (check/card): Check
Individual record created successfully! (Rows affected: 1)
Donor 'DONOR001' created successfully! (Rows affected: 1)
Enter check number: 8465
Donation inserted successfully! (Donation rows: 1, Payment rows: 1)
VERIFIED: Donation D001 ($700.00) from donor DONOR001 is in the database
```

Insert donor #4

```
(18) Quit
============================================================
Please select an option (1-18): 4

[Query 4] Insert a new donation from a donor
Enter donation ID: D120
Enter donor ID number: DONOR120

Donor 'DONOR120' does not exist. Creating new donor...
Please provide the following information:
Individual record not found. Creating Individual first...
Enter first name: Robert
Enter last name: Johnson
Enter gender (M/F/O): M
Enter street address: 789 Pine Rd
Enter city: Peoria
Enter state: TX
Enter postal code: 16010
Enter date of birth (YYYY-MM-DD): 1990-03-10
Subscribe to newsletter? (true/false): False
Enter donor preference (or press Enter for NULL): Park Maintenance
Enter donation date (YYYY-MM-DD): 2024-03-10
Enter donation amount: 1000.00
Enter campaign name (or press Enter for NULL):
Enter payment method (check/card): Check
Individual record created successfully! (Rows affected: 1)
Donor 'DONOR120' created successfully! (Rows affected: 1)
Enter check number: CHK004
Donation inserted successfully! (Donation rows: 1, Payment rows: 1)
VERIFIED: Donation D120 ($1000.00) from donor DONOR120 is in the database

Press Enter to continue...
```

Insert donor #5

```
Please select an option (1-18): 4

[Query 4] Insert a new donation from a donor
Enter donation ID: D777
Enter donor ID number: DONOR777

Donor 'DONOR777' does not exist. Creating new donor...
Please provide the following information:
Individual record not found. Creating Individual first...
Enter first name: David
Enter last name: Sabardu
Enter gender (M/F/O): M
Enter street address: 654 Maple Dr
Enter city: Cad
Enter state: Canada
Enter postal code: 71239
Enter date of birth (YYYY-MM-DD): 1978-07-08
Subscribe to newsletter? (true/false): True
Enter donor preference (or press Enter for NULL): General Support
Enter donation date (YYYY-MM-DD): 2024-05-20
Enter donation amount: 1200.00
Enter campaign name (or press Enter for NULL):
Enter payment method (check/card): Check
Individual record created successfully! (Rows affected: 1)
Donor 'DONOR777' created successfully! (Rows affected: 1)
Enter check number: CHK97
Donation inserted successfully! (Donation rows: 1, Payment rows: 1)
VERIFIED: Donation D777 ($1200.00) from donor DONOR777 is in the database

Press Enter to continue...
```

Result:

Results Messages

	Id_num…	Preference
1	DONOR001	Wildlife Conse…
2	DONOR002	Environmental …
3	DONOR004	Education Prog…
4	DONOR120	Park Maintenan…
5	DONOR777	General Support

Results Messages

	Donati…	Donor_id_n…	Date	Amount	Campaign_…
1	D001	DONOR001	2024-01-15	700.00	Check
2	D002	DONOR002	2024-02-20	1000.00	Spring Campai…
3	D004	DONOR004	2024-04-05	750.00	Summer Fundra…
4	D120	DONOR120	2024-03-10	1000.00	*NULL*
5	D777	DONOR777	2024-05-20	1200.00	*NULL*

Results Messages

	Donati…	Check_numb…
1	D001	8465
2	D120	CHK004
3	D777	CHK97

Query 8:

Retrieve Emergency Contacts #1

```
============================================================
Please select an option (1-18): 8

[Query 8] Retrieve the names and contact information of all emergency contacts for a speci
fic person
Enter the person's ID number: V001
Emergency Contacts
- No emergency contacts found for ID number: V001
```

Retrieve Emergency Contacts #2

```
============================================================
Please select an option (1-18): 8

[Query 8] Retrieve the names and contact information of all emergency contacts for a speci
fic person
Enter the person's ID number: V002
Emergency Contacts
- Person: Jane Smith (ID: V002)
- Contact Name: Astra Sabardu
- Relationship: Sister
- Phone Number: 555-0102
- Contact Name: John Smith
- Relationship: Spouse
- Phone Number: 555-0101

Press Enter to continue...
```

Result:

Results Messages

	Id_num…	Name	Relationship	Phone_numb…
1	V002	John Smith	Spouse	555-0101
2	V002	Astra Sabardu	Sister	555-0102

Query 9:

Retrieve Visitors in Program #1

```
============================================================
Please select an option (1-18): 9

[Query 9] Retrieve the list of visitors enrolled in a specific park program, including the
ir accessibility needs
Enter the program name: Wildlife Watching Tour
Visitors Enrolled in Program: Wildlife Watching Tour
ID Number: V001
Name: John Ma
Accessibility Needs: Wheelchair access
Visit Date: 2024-01-10
```

Retrieve Visitors in Program #2

```
============================================================
Please select an option (1-18): 9

[Query 9] Retrieve the list of visitors enrolled in a specific park program, including the
ir accessibility needs
Enter the program name: Hiking Adventure
Visitors Enrolled in Program: Hiking Adventure
ID Number: V003
Name: Bobby Johnson
Accessibility Needs: Hearing aid
Visit Date: Not specified
ID Number: V004
Name: Alice Williams
Accessibility Needs: None
Visit Date: 2024-03-20
```

Result:

Results Messages

	Id_num…	Visit_date	Accessibility…
1	V001	2024-01-10	Wheelchair access
2	V002	2024-02-15	*NULL*
3	V003	*NULL*	Hearing aid
4	V004	2024-03-20	*NULL*
5	V005	2024-04-01	*NULL*

Results Messages

	Visito…	Program_na…
1	V001	Wildlife Watch…
2	V003	Hiking Adventu…
3	V003	Nature Photogr…
4	V004	Hiking adventu…

Query 10:

Retrieve Park Programs #1

```
============================================================
Please select an option (1-18): 10

[Query 10] Retrieve all park programs for a specific park that started after a given date
Enter the park name: Yellowstone National Park
Enter the start date (YYYY-MM-DD) - programs starting after this date will be shown: 2024-
01-01
Park Programs for: Yellowstone National Park
Programs starting after: 2024-01-01
Program Name: Wildlife Watching Tour
Type: Educational
Start Date: 2024-06-01
Duration: 3 days
```

Retrieve Park Programs #2

```
============================================================
Please select an option (1-18): 10

[Query 10] Retrieve all park programs for a specific park that started after a given date
Enter the park name: Yosemite National Park
Enter the start date (YYYY-MM-DD) - programs starting after this date will be shown: 2024-
06-01
Park Programs for: Yosemite National Park
Programs starting after: 2024-06-01
Program Name: Hiking Adventure
Type: Recreational
Start Date: 2024-07-15
Duration: 5 days
```

Query 11:

Retrieve Donation

```
Please select an option (1-18): 11

[Query 11] Retrieve the total and average donation amount received in a month from all ano
nymous donors
Enter the month (MM)
01
Enter the year (YYYY): 2024
Anonymous Donor Statistics for 01/2024
Sorted by Total Amount (Descending)
No anonymous donations found for 01/2024
```

Query 12:

Retrieve Rangers in Team

```
Please select an option (1-18): 12
T
[Query 12] Retrieve the list of rangers in a team, including their certifications, years o
f service and their role in the team
Enter the team ID: T001
Rangers in Team: TT001
No rangers found in team: TT001
```

Query 13:

Retrieve All Individuals

```
Please select an option (1-18): 13

[Query 13] Retrieve the names, IDs, contact information, and newsletter subscription statu
s of all individuals in the database
All Individuals in Database
ID Number: RES004
Name: Dr. Robert Botany
Newsletter Status: Subscribed
Phone Numbers: None | Email: None
--------------------------------------------------------------------------------
ID Number: RES003
Name: Dr. Maria Conservation
Newsletter Status: Subscribed
Phone Numbers: None | Email: None
--------------------------------------------------------------------------------
ID Number: DONOR002
Name: Jane Doe
Newsletter Status: Subscribed
Phone Numbers: None | Email: None
--------------------------------------------------------------------------------
ID Number: RES002
Name: Dr. James Ecology
Newsletter Status: Not Subscribed
Phone Numbers: None | Email: None
--------------------------------------------------------------------------------
ID Number: R002
Name: Sarah Forest
Newsletter Status: Not Subscribed
Phone Numbers: None | Email: None
--------------------------------------------------------------------------------
ID Number: V003
Name: Bobby Johnson
Newsletter Status: Subscribed
Phone Numbers: None | Email: None
--------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
ID Number: DONOR120
Name: Robert Johnson
Newsletter Status: Not Subscribed
Phone Numbers: None | Email: None
--------------------------------------------------------------------------------
ID Number: V001
Name: John Ma
Newsletter Status: Subscribed
Phone Numbers: None | Email: None
--------------------------------------------------------------------------------
ID Number: R004
Name: Lisa nguyen
Newsletter Status: Subscribed
Phone Numbers: None | Email: None
--------------------------------------------------------------------------------
ID Number: R001
Name: Mike Ranger
Newsletter Status: Subscribed
Phone Numbers: None | Email: None
--------------------------------------------------------------------------------
ID Number: DONOR777
Name: David Sabardu
Newsletter Status: Subscribed
Phone Numbers: None | Email: None
--------------------------------------------------------------------------------
ID Number: V005
Name: Zoe Sabardu
Newsletter Status: Subscribed
Phone Numbers: None | Email: None
--------------------------------------------------------------------------------
ID Number: RES001
Name: Dr. Emily Science
Newsletter Status: Subscribed
Phone Numbers: None | Email: None
--------------------------------------------------------------------------------
ID Number: V002
Name: Jane Smith
Newsletter Status: Not Subscribed
Phone Numbers: None | Email: None
```

```
--------------------------------------------------------------------------------
ID Number: RES001
Name: Dr. Emily Science
Newsletter Status: Subscribed
Phone Numbers: None | Email: None
--------------------------------------------------------------------------------
ID Number: V002
Name: Jane Smith
Newsletter Status: Not Subscribed
Phone Numbers: None | Email: None
--------------------------------------------------------------------------------
ID Number: DONOR001
Name: John Smith
Newsletter Status: Subscribed
Phone Numbers: None | Email: None
--------------------------------------------------------------------------------
ID Number: R005
Name: Brandon Valley
Newsletter Status: Not Subscribed
Phone Numbers: None | Email: None
--------------------------------------------------------------------------------
ID Number: V004
Name: Alice Williams
Newsletter Status: Subscribed
Phone Numbers: None | Email: None
--------------------------------------------------------------------------------
ID Number: DONOR004
Name: Mary Williams
Newsletter Status: Subscribed
Phone Numbers: None | Email: None
--------------------------------------------------------------------------------
ID Number: RES005
Name: Dr. Zoe Zoology
Newsletter Status: Not Subscribed
Phone Numbers: None | Email: None

Press Enter to continue...
```

Query 14:

Update Researcher Salary

```
============================================================
Please select an option (1-18): 14

[Query 14] Update the salary of researchers overseeing more than one ranger team by a 3% i
ncrease
Updated salary for 3 researcher(s) successfully!
Salary increased by 3% for researchers overseeing more than one ranger team.
```

Result:

Results Messages

	Id_num…	Research_f…	Hire_date	Salary
1	RES001	Wildlife Biolo…	2018-01-15	77250.00
2	RES002	Environmental …	2019-03-01	80000.00
3	RES003	Conservation B…	2020-06-15	97850.00
4	RES004	Plant Ecology	2021-09-01	78000.00
5	RES005	Animal Behavior	2022-01-10	94760.00

Query 15:

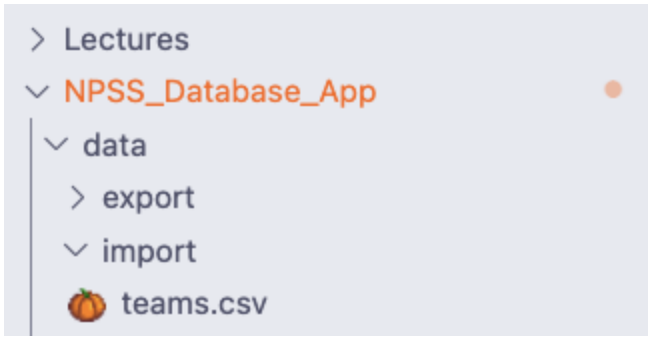
Delete Expired Visitors

```
============================================================
Please select an option (1-18): 15

[Query 15] Delete visitors who have not enrolled in any park programs and whose park passe
s have expired
No expired visitors found to delete.

Press Enter to continue...
```

Query 16:

Import - Enter new teams from a data file

Import Data Information:
team_id,formation_date,focus_date,team_leader
T100,2024-01-15,2024-06-01,R001
T101,2024-02-20,,R002
T102,2024-03-10,2024-07-15,
T103,2024-04-05,
T104,2024-05-12,2024-08-20,R003
T105,2024-06-01,2024-09-01,R004
T106,2024-07-15,R005
T107,2024-08-20,2024-12-31,
T108,2024-09-10,
T109,2024-10-01,2025-01-15,R001



Result:

```
============================================================
Please select an option (1-18): 16

[Import] Enter new teams from a data file until the file is empty
Please enter the input file name: teams.csv
Reading from file: /Users/astra.wkeav/Documents/Database_managements/NPSS_Database_App/tea
ms.csv
Imported team: T100 (formation: 2024-01-15)
Imported team: T101 (formation: 2024-02-20)
Imported team: T102 (formation: 2024-03-10)
Imported team: T103 (formation: 2024-04-05)
Line 6: Failed to import team 'T104': The INSERT statement conflicted with the FOREIGN KEY
 constraint "FK_ranger_team_team_leader". The conflict occurred in database "CS-DSA-4513-S
QL-DB", table "dbo.Ranger", column 'Id_number'.
Imported team: T105 (formation: 2024-06-01)
Imported team: T106 (formation: 2024-07-15)
Imported team: T107 (formation: 2024-08-20)
Imported team: T108 (formation: 2024-09-10)
Imported team: T109 (formation: 2024-10-01)
Total lines processed: 12
Successfully imported: 9
Errors: 1

Import completed successfully!
```

Results Messages

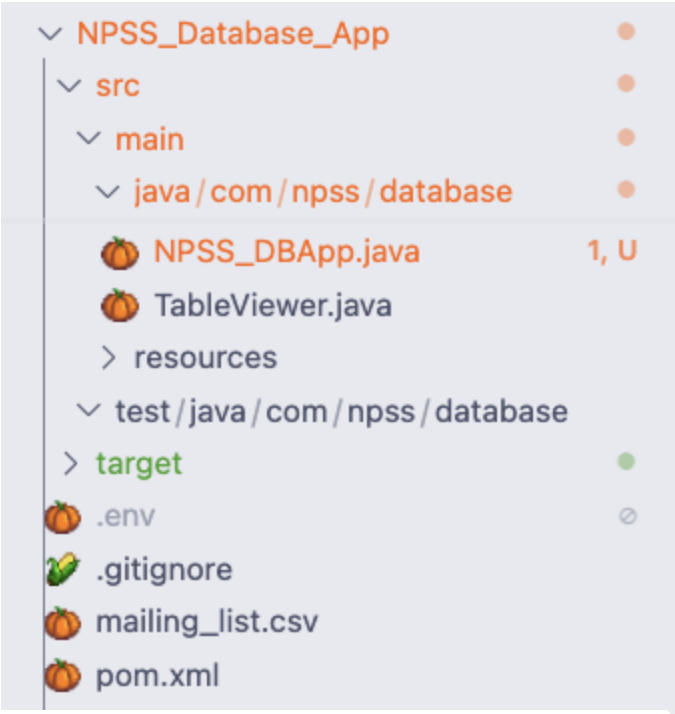
	Team_id	Focus_date	Formation_date	Team_leader
1	T001	2020-02-01	2020-01-01	*NULL*
2	T002	2021-12-04	2021-03-15	*NULL*
3	T003	2022-07-01	2022-06-01	*NULL*
4	T004	2023-02-10	2023-01-10	*NULL*
5	T005	*NULL*	2023-09-01	*NULL*
6	T100	2024-06-01	2024-01-15	R001
7	T101	*NULL*	2024-02-20	R002
8	T102	2024-07-15	2024-03-10	*NULL*
9	T103	*NULL*	2024-04-05	*NULL*
1…	T105	2024-09-01	2024-06-01	R004
1…	T106	*NULL*	2024-07-15	R005
1…	T107	2024-12-31	2024-08-20	*NULL*
1…	T108	*NULL*	2024-09-10	*NULL*
1…	T109	2025-01-15	2024-10-01	R001

Query 17:

Export - Retrieve names and mailing addresses

```
============================================================
Please select an option (1-18): 17

[Export] Retrieve names and mailing addresses of all people on the mailing list
Please enter the output file name: mailing_list.csv
Export completed successfully!
File saved to: /Users/astra.wkeav/Documents/Database_managements/NPSS_Database_App/mailing
_list.csv
Total records exported: 13

Press Enter to continue...
```

Result:



NPSS_Database_App > mailing_list.csv

```
1   Name,Street,City,State,Postal Code
2   Dr. Robert Botany,900 Plant St,Rockford,Il,61101
3   Dr. Maria Conservation,800 Analysis Dr,Peoria,IL,61601
4   Jane Doe,456 Oak Ave,Oklahoma,OK,73015
5   Bobby Johnson,789 Pine Rd,Peoria,Il,61601
6   John Ma,123 Main St,Springfield,IL,62701
7   Lisa nguyen,400 Stream Blvd,Rockford,IL,61101
8   Mike Ranger,100 Park Way,Springfield,IL,62701
9   David Sabardu,654 Maple Dr,Cad,Canada,71239
10  Zoe Sabardu,654 Maple Dr,Montreal,Canada,60540
11  Dr. Emily Science,600 Research Ave,Springfield,IL,62701
12  John Smith,123 Main St,Springfield,IL,62701
13  Alice Williams,321 Elm St,Rockford,IL,61101
14  Mary Williams,321 Elm St,aswdfawfw St,IL,61101
15
```

Error Testing:

Error Test 1 - Primary Key Violation

```
Subscribe to newsletter? (true/false): True
Enter visit date (YYYY-MM-DD) or press Enter for NULL: 2024-04-01
Enter accessibility needs (or press Enter for NULL):
Database error: Violation of PRIMARY KEY constraint 'PK_individual'. Cannot insert duplica
te key in object 'dbo.Individual'. The duplicate key value is (V005).
SQL State: 23000

Database error occurred:
  Message: Violation of PRIMARY KEY constraint 'PK_individual'. Cannot insert duplicate ke
y in object 'dbo.Individual'. The duplicate key value is (V005).
  SQL State: 23000
  Error Code: 2627
```

Error Test 2 - Invalid Date Format

```
Please select an option (1-18): 3

[Query 3] Insert a new ranger team into the database and set its leader
Enter team ID: TEST_ERROR
Enter formation date (YYYY-MM-DD): 2020/01/01
Enter focus date (YYYY-MM-DD) or press Enter for NULL: 2020/01/01
Enter team leader ID (ranger ID) or press Enter for NULL: 1234
Invalid date format: null
Please use YYYY-MM-DD format for dates.

Database error occurred:
  Message: Invalid input: null

Press Enter to continue...
```

Error Test 3 - Invalid Amount Format & Insert Statement Conflicted With The Check

```
Please select an option (1-18): 4

[Query 4] Insert a new donation from a donor
Enter donation ID: Error_test
Enter donor ID number:  DONOR001
Donor 'DONOR001' found in database.
Enter donation date (YYYY-MM-DD): 2024-12-01
Enter donation amount: -100.00
Enter campaign name (or press Enter for NULL):
Enter payment method (check/card): Check
Database error: The INSERT statement conflicted with the CHECK constraint "CK_donation_amo
unt". The conflict occurred in database "CS-DSA-4513-SQL-DB", table "dbo.Donation", column
 'Amount'.
SQL State: 23000

Database error occurred:
  Message: The INSERT statement conflicted with the CHECK constraint "CK_donation_amount".
 The conflict occurred in database "CS-DSA-4513-SQL-DB", table "dbo.Donation", column 'Amo
unt'.
  SQL State: 23000
  Error Code: 547
```

Quit

```
============================================================
Please select an option (1-18): 18

Thank you for using NPSS Database System. Goodbye!
Database connection closed!
```