DBMS Individual Project – National Park Database Service

CS 4513-002

Fall 2025

Dr. Egawati Panjei

Yale Gray

OUID: 113569043

Yale.p.gray-1@ou.edu

Tasks Performed — Page Number

Task 1. ER Diagram 1

Task 2. Relational Database Schemas 2

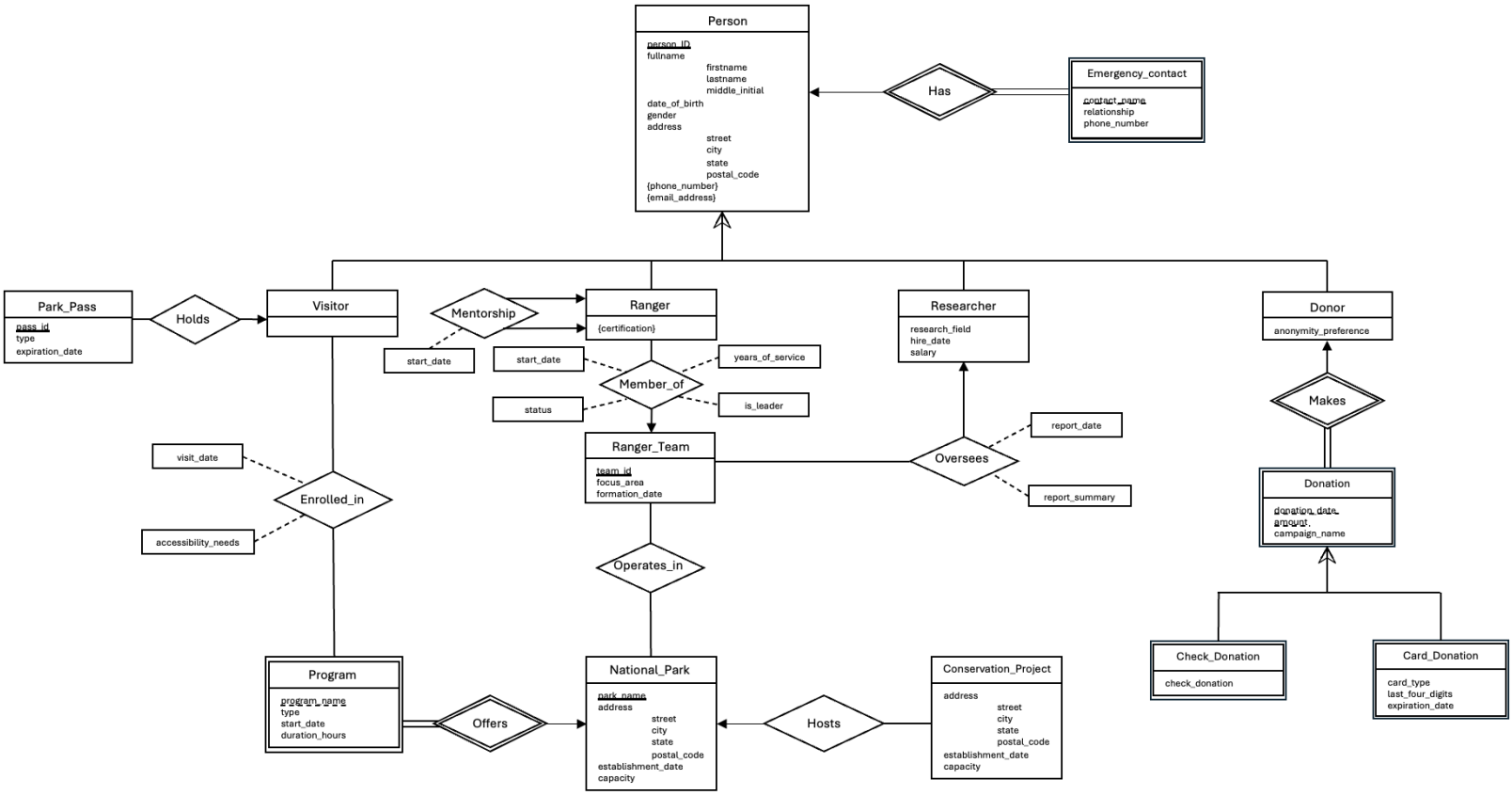
Task 3. 3-24
- 3.1. Discussion of storage structures for tables 3-24
- 3.2. Discussion of storage structures for table (Azure SQL) 3-24

Task 4. SQL statements and screenshots showing the creation of tables in Azure SQL database 25-46

Task 5. 47-72
- 5.1. SQL statements (and Transact SQL stored procedures, if any) implementing all queries (1-15 and error checking) 47-61
- 5.2. The Java source program and screenshots showing its successful compilation 62-72

Task 6. Java program execution 73-91
- 6.1. Screenshots showing the testing of query 1 73
- 6.2. Screenshots showing the testing of query 2 74
- 6.3. Screenshots showing the testing of query 3 75
- 6.4. Screenshots showing the testing of query 4 76
- 6.5. Screenshots showing the testing of query 5 77
- 6.6. Screenshots showing the testing of query 6 78
- 6.7. Screenshots showing the testing of query 7 79
- 6.8. Screenshots showing the testing of query 8 80
- 6.9. Screenshots showing the testing of query 9 81
- 6.10. Screenshots showing the testing of query 10 82
- 6.11. Screenshots showing the testing of query 11 83
- 6.12. Screenshots showing the testing of query 12 84
- 6.13. Screenshots showing the testing of query 13 85
- 6.14. Screenshots showing the testing of query 14 86
- 6.15. Screenshots showing the testing of query 15 87
- 6.16. Screenshots showing the testing of the import and export options 88-89
- 6.17. Screenshots showing the results of three types of errors 90
- 6.18. Screenshots showing the testing of the quit option 91

ER Diagram:


Person
person_ID
fullname
firstname
lastname
middle_initial
date_of_birth
gender
address
street
city
state
postal_code
{phone_number}
{email_address}
Has
Emergency_contact
contact_name
relationship
phone_number
Park_Pass
pass_id
type
expiration_date
Holds
Visitor
Mentorship
Ranger
{certification}
start_date
start_date
years_of_service
Member_of
status
is_leader
Researcher
research_field
hire_date
salary
Donor
anonymity_preference
Makes
Donation
donation_date
amount
campaign_name
Ranger_Team
team_id
focus_area
formation_date
Oversees
report_date
report_summary
visit_date
Enrolled_in
accessibility_needs
Operates_in
Program
program_name
type
start_date
duration_hours
Offers
National_Park
park_name
address
street
city
state
postal_code
establishment_date
capacity
Hosts
Conservation_Project
address
street
city
state
postal_code
establishment_date
capacity
Check_Donation
check_donation
Card_Donation
card_type
last_four_digits
expiration_date

Relational Database Schema:

Person (person_id: string, firstname: string, lastname: string, middle_initial: string, dob date, gender: string, street: string, city: string, state: string, postal_code: string, subscribed: boolean)
Phone (person_id: string [FK], phone_number: string)
Email (person_id: string [FK], email_address: string)
Emergency_Contact (person_id: string [FK], contact_name: string, relationship: string, phone_number: string)
Visitor (person_id: string [FK])
Ranger (person_id: string [FK])
Researcher (person_id: string [FK], research_field: string, hire_date: date, salary: integer)
Donor (person_id: string [FK], anonymity_preference: boolean)
National_Park (park_name: string, street: string, city: string, state: string, postal_code: string, establishment_date: date, capacity: integer)
Conservation_Project (project_id: string, name: string, start_date: date, budget: integer, park_name: string [FK])
Ranger_Team (team_id: string, focus_area: string, formation_date: date)
Member_of (person_id: string [FK], team_id: string [FK], start_date: date, status: string, years_of_service: integer, is_leader: boolean)
Operates_in (team_id: string [FK], park_name: string [FK])
Oversees (person_id: string, team_id: string, report_date: date, report_summary: string)
Certification (person_id: string [FK], certification_name: string)
Mentorship (mentor_id: string, mentee_id: string, mentorship_start_date: string)
Program (park_name: string [FK], program_name: string, type: string, start_date: date, duration_hours: integer)
Enrolled_in (person_id: string [FK], park_name: string, program_name: string, visit_date: date, accessibility_needs: string)
Park_Pass (pass_id: string, type: string, expiration_date: date, person_id: string [FK])
Donation (person_id: string [FK], donation_date: date, amount: integer, campaign_name: string)
Check_Donation (check_number: integer, person_id: string [FK], donation_date: date [FK], amount: integer [FK])
Card_Donation (card_type: string, last_four_digits: integer, expiration_date: date, person_id: string [FK], donation_date: date [FK], amount: integer [FK])

Discussion of Storage Structure for **Person** Table

Table Name	Query # and Type	Search Key(s)	Query Frequency	Selected File Org.
Person	Q1 – Insertion; Q8 – Selection; Q13 – Range Search	person_id, lastname, subscribed	Q1: 10x/day; Q8: 2x/week; Q13: 1x/week	B+ Tree

Justification
The Person table supports frequent equality lookups by person_id and occasional range or ordered searches by last_name and subscribed. B+ tree organization keeps data ordered by key, supporting equality and range scans in an efficient manner. Because of the frequency of these operations, the high read frequency favors a B+ tree over a heap or hash structure, as some overhead is preferred for the higher performance.

Discussion of Storage Structure for **Person** Table in Azure SQL Database

Table Name	Selected File Organization	Azure SQL Database Implementation
Person	B+ Tree	Clustered Index on person_id; Non-Clustered Index on (lastname, subscribed)

Justification
Azure SQL implements B+ Tree storage through clustered indexes. The clustered index on person_id orders rows for fast lookups and joins. The non-clustered index on (lastname, subscribed) directly accelerates the searches in queries 8 and 13.

Discussion of Storage Structure for **Phone** Table

Table Name	Query # and Type	Search Key(s)	Query Frequency	Selected File Org.
Phone	Q13 – Random Search	person_id	1x/week	B+ Tree

Justification
The Phone table supports fairly frequent lookups by person_id. Because person_id is a foreign key, a B+ Tree provides the lowest overhead of any structure for joins between tables. Insertions rarely occur in the Phone table, and. B+ Tree is highly efficient and optimized for fast lookups.

Discussion of Storage Structure for **Phone** Table in Azure SQL Database

Table Name	Selected File Organization	Azure SQL Database Implementation
Phone	B+ Tree	Clustered index on person_id; non-clustered on phone_number

Justification
Azure SQL uses clustered B+ Trees for both primary and foreign keys. Further adding a non-clustered index on phone_number improves the performance of lookups by phone_number, further increasing efficiency for lookups and joins.

Discussion of Storage Structure for **Email** Table

Table Name	Query # and Type	Search Key(s)	Query Frequency	Selected File Org.
Email	Q13 – Random Search	person_id	1x/week	B+ Tree

Justification
The Email table also supports frequent lookups on person_id. Because person_id is a foreign key, a B+ Tree structure would provide low overhead joins between tables. Insertions rarely occur in the Email table, which means a B+ Tree would excel in terms of minimizing lookup time.

Discussion of Storage Structure for **Email** Table in Azure SQL Database

Table Name	Selected File Organization	Azure SQL Database Implementation
Email	B+ Tree	Clustered index on person_id; non-clustered on email_address

Justification
Azure SQL portrays primary and foreign keys through clustered B+ Trees. The non-clustered index on email_address, similar to the one on phone_number, is simply there to further improve lookup times and join efficiency.

Discussion of Storage Structure for **Emergency_Contact** Table

Table Name	Query # and Type	Search Key(s)	Query Frequency	Selected File Org.
Emergency_ Contact	Q8 – Random Search	person_id	2x/week	Hash

Justification
Emergency_Contacts are looked up through the associated person_id when retrieving a person's profile. Searches are equality-based and not ordered, so a Hash structure gives the best performance. This table has relatively few insertions and deletions, so the simplicity/speed of a Hash works best. This guarantees quick searches and low overhead.

Discussion of Storage Structure for **Emergency_Contact** Table in Azure SQL Database

Table Name	Selected File Organization	Azure SQL Database Implementation
Emergency_Contact	Hash	Non-clustered index on person_id

Justification
By implementing a non-clustered index on person_id, Azure SQL can provide equality searches and remain efficient. This mirrors a Hash structure and the advantages of it. It minimizes the cost of lookups while remaining highly efficient.

Discussion of Storage Structure for **Visitor** Table

Table Name	Query # and Type	Search Key(s)	Query Frequency	Selected File Org.
Visitor	Q1 – Insertion; Q15 - Deletion	person_id	Q1: 10x/day; Q15: 1x/month	Heap

Justification
Visitor has frequent insertions when new visitors register as well as occasional deletions when inactive visitors are removed. Each of these operations depend simply on deleting entire entries rather than searching the tables themselves. A Heap would work best for this situation, as it avoids high-cost indexing and allows for the lowest cost deletions.

Discussion of Storage Structure for **Visitor** Table in Azure SQL Database

Table Name	Selected File Organization	Azure SQL Database Implementation
Visitor	Heap	No clustered index

Justification
As stated above, the Heap structure does not rely on indexing. This is more efficient for tables which are more heavy on insertion/deletion. The lack of indexing enables much hgher efficiency during these frequent operations.

Discussion of Storage Structure for **Ranger** Table

Table Name	Query # and Type	Search Key(s)	Query Frequency	Selected File Org.
Ranger	Q2 – Insertion; Q12 – Random Search	person_id; team_id	Q2: 2x/month; Q12: 2x/week	Hash

Justification
Ranger data is inserted when new rangers are added and is read frequently when eams and and their certifications are displayed. Access relies on person_id and team_id, which benefit from ordered organization. A B+ Tree supports both efficient insertions and fast random access by key while maintaining sorted structure. This ensures consistent performance during both changes and retrievals.

Discussion of Storage Structure for **Ranger** Table in Azure SQL Database

Table Name	Selected File Organization	Azure SQL Database Implementation
Ranger	B+ Tree	Clustered index on person_id; Non-clustered index on team_id

Justification
Azure SQL implements B+ Trees through clustered indexes. The clustered index on person_id optimized for direct lookups, while the non-clustered index on team_id allows for better performance on queries for team-based retrievals.

Discussion of Storage Structure for **Researcher** Table

Table Name	Query # and Type	Search Key(s)	Query Frequency	Selected File Org.
Researcher	Q5 – Insertion;Q6 – Insertion; Q14 – Random Search	person_id	Q5: 1x/year; Q6: 10x/month Q14: x/month	B+ Tree

Justification
Researcher records are rarely inserted but are retrieved regularly when updating salaries or assigning teams. Lookups and data updates depend on equality lookups using person_id, so an ordered structure works best. A B+ Tree supports stable random access while minimizing overhead during the infrequent inserts. This balances the need for monthly salary updates and rare insertions.

Discussion of Storage Structure for **Researcher** Table in Azure SQL Database

Table Name	Selected File Organization	Azure SQL Database Implementation
Researcher	B+ Tree	Clustered index on person_id

Justification
The clustered index on person_id organizes the data for efficient retrievals as well as updates. This structure maintains a low cost or lookups while also minimizing the overhead during inserts.

Discussion of Storage Structure for **Donor** Table

Table Name	Query # and Type	Search Key(s)	Query Frequency	Selected File Org.
Donor	Q4 – Insertion; Q11 – Range Search	person_id	Q4: 5x/day; Q11: 1x/month	B+ Tree

Justification
Donor data accessed primarily through equality matches on person_id when inserting new donations. A B+ Tree organization strikes a good balance between write and read operations while also keeping ordered access to keep locating donors and their anonymity_preference efficient.

Discussion of Storage Structure for **Donor** Table in Azure SQL Database

Table Name	Selected File Organization	Azure SQL Database Implementation
Donot	B+ Tree	Clustered index on person_id; non-clustered on anonymity_preference

Justification
Azure SQL implements a clustered B+ Tree for efficient lookups on person_id. The non-clustered index on anonymity_preference allows for quick filtering by this attribute.

Discussion of Storage Structure for **National_park** Table

Table Name	Query # and Type	Search Key(s)	Query Frequency	Selected File Org.
National_ Park	Q1 – Random Search; Q7 – Random Search; Q10 – Range Search	park_nam e	Q1: 10x/day Q7: 2x/month; Q10: 1x/week	B+ Tree

Justification
Parks are frequently retrieved and joined by park_name. A B+ Tree organization supports both equality and range searches while keeping all records in sorted order. This provides efficient access for any queries that filer by park_name. Since the records for these parks change infrequently, the overhead for keeping order is low.

Discussion of Storage Structure for **National_Park** Table in Azure SQL Database

Table Name	Selected File Organization	Azure SQL Database Implementation
National_Park	B+ Tree	Clustered index on park_name

Justification
Azure SQL implements B+ Trees as clustered indexes. Ordering by park_name optimizes both equality and range searches for parks, while also maintaining efficient joins with other tables.

Discussion of Storage Structure for **Conservation_Project** Table

Table Name	Query # and Type	Search Key(s)	Query Frequency	Selected File Org.
Conservation _Project	Unused	project_id	N/A	Heap

Justification
Since the table experiences no current query traffic, a Heap structure would be most functional. This keeps the table ready for future use, minimizing any unnecesarry overhead cost.

Discussion of Storage Structure for **Conservation_Project** Table in Azure SQL Database

Table Name	Selected File Organization	Azure SQL Database Implementation
Conservation_Project	Heap	No clustered index

Justification
Because the table currently supports no lookups or retrievals, an unordered heap table is the most storage-efficient and least maintenance-intensive choice. In Azure SQL, this means we should refrain from using a clustered index, keeping the table lightweight for potential future operations.

Discussion of Storage Structure for **Ranger_Team** Table

Table Name	Query # and Type	Search Key(s)	Query Frequency	Selected File Org.
Ranger_Team	Q2 – Insertion; Q3 – Insertion; Q12 – Random Search	team_id	Q2: 2x/month; Q3: 1x/month; Q12: 4x/year	Hash

Justification
The Ranger_Team table supports quality lookups by team_id when either inserting or retrieving records. Because the structure of the team changes infrequently, but equality search occur more often, a Hash organization would work well, allowing for constant-time access whilst minimizing overhead.

Discussion of Storage Structure for **Person** Table in Azure SQL Database

Table Name	Selected File Organization	Azure SQL Database Implementation
Ranger_team	Hash	Clustered index on team_id

Justification
Azure SQL accomplished hash structures and performance via a clustered index on team_id. His allows for quick lookups and efficient joins between Ranger_Tea, and other nearby tables.

Discussion of Storage Structure for **Member_of** Table

Table Name	Query # and Type	Search Key(s)	Query Frequency	Selected File Org.
Member_of	Q2 – insertion; 12 – Random Search	Person_id ; team_id	Q2: 2x/month; Q12: 4x/year	Hash

Justification
The Member_of table links between Ranger and Ranger_Team, and thus is mainly used for equality joins between the two. Because of the low frequency of insertions, a Hash structure would work well, allowing for efficient access to data.

Discussion of Storage Structure for **Member_of** Table in Azure SQL Database

Table Name	Selected File Organization	Azure SQL Database Implementation
Member_of	Hash	Non-clustered index on (team_id, person_id)

Justification
The non-clustered index on (team_id, person_id) enable Azure SQL to do efficient lookups and fast joins on the ranger and Ranger_Team tables.

Discussion of Storage Structure for **Operates_in** Table

Table Name	Query # and Type	Search Key(s)	Query Frequency	Selected File Org.
Operates_in	Unused	team_id, park_name	N/A	Heap

Justification
Since the table experiences no current query traffic, organizing it as a heap most appropriate. This keeps it undordered for the time being, but sets it up well for future use. This minimizes overhead while allowing future expansion if later versions of the database include functionality.

Discussion of Storage Structure for **Operates_in** Table in Azure SQL Database

Table Name	Selected File Organization	Azure SQL Database Implementation
Operates_in	Heap	No clustered index

Justification
Because the table currently supports no lookups or retrievals, an unordered heap table is the most storage-efficient and least maintenance-intensive choice. In Azure SQL, this means we should refrain from using a clustered index, keeping the table lightweight for potential future operations.

Discussion of Storage Structure for **Oversees** Table

Table Name	Query # and Type	Search Key(s)	Query Frequency	Selected File Org.
Oversees	Q5 – Insertion; Q6 - Insertion	person_id; team_id	Q5: 1x/year; Q6: 10x/month	B+ Tree

Justification
The Oversees table associates Researchers with Ranger_team, supporting multiple insertions. Because of the frequent ordered retrievals, a B+ Tree would work well. B+ Trees provide efficient search, especially in such frequent joins between tables.

Discussion of Storage Structure for **Oversees** Table in Azure SQL Database

Table Name	Selected File Organization	Azure SQL Database Implementation
Oversees	B+ Tree	Clustered index on (person_id, team_id)

Justification
The clustered index here would maintain ordered records in Azure SQL, allowing for efficient retrievals for researchers and their associated teams.

Discussion of Storage Structure for **Certification** Table

Table Name	Query # and Type	Search Key(s)	Query Frequency	Selected File Org.
Certification	Q2 – Insertion; Q12 – Random Search	person_id; certification_name	Q2: 2x/month; Q12: 4x/year	Hash

Justification
The Certification table stores Rangers' certifications, which are added during team assignment and retrieved when listing the member of the team. Bacause of the reliance on equality lookups, a Hash structure would work well, as it ensure constant-time lookups for joins which are equality focused.

Discussion of Storage Structure for **Certification** Table in Azure SQL Database

Table Name	Selected File Organization	Azure SQL Database Implementation
Certification	Hash	Non-clustered on (person_id, certification_name)

Justification
This non-clustered composite index enables Azure SQL to achieve Hash structure-like performance. It supports equality lookups on both person_id and certification_name, maximizing efficiency and minimizing overhead.

Discussion of Storage Structure for **Mentorship** Table

Table Name	Query # and Type	Search Key(s)	Query Frequency	Selected File Org.
Mentorship	Unused	Mentor_id, mentee_id	N/A	Hash

Justification
The Mentorship table tracks rangers' mentor to mentee relationships, but is currently not referenced in any of the implemented queries. Because it is currently unused for lookups or updates, there is no need for indexing or ordered access. A Heap structure, however, would minimize storage overhead and supports future use.

Discussion of Storage Structure for **Mentorship** Table in Azure SQL Database

Table Name	Selected File Organization	Azure SQL Database Implementation
Mentorship	Hash	No clustered index

Justification
When no clustered index is included, Azure SQL implements a Heap structure. Avoiding index management and allowing for cheap and efficient queries for future use.

Discussion of Storage Structure for **Program** Table

Table Name	Query # and Type	Search Key(s)	Query Frequency	Selected File Org.
Program	Q7 – Insertion; Q9: Random Search; Q10 – Range Search	park_name; program_name; start_date	Q7: 2x/month; Q9: 2x/week; Q10: 1x/month	B+ Tree

Justification
The Program table supports both equality and range lookups, by park_name, program_name, and start_date. These mized search operation lend itself to a B+ Tree structure, enabling order based search for range search, while simultaneously keeping equality searched efficient.

Discussion of Storage Structure for **Program** Table in Azure SQL Database

Table Name	Selected File Organization	Azure SQL Database Implementation
Program	B+ Tree	Clustered index on (park_name, program_name); non-clustered on start_date

Justification
The clustered index here supports equality searches on park_name and program_name, and the non-clustered supports range search on start_date. This keeps the B+ Tree structure efficient for various search types.

Discussion of Storage Structure for **Enrolled_in** Table

Table Name	Query # and Type	Search Key(s)	Query Frequency	Selected File Org.
Enrolled_in	Q1 – Insetion; Q9 – Random Search; Q15 – Random Search	person_id; park_name; program_name	Q1: 10x/day; Q9: 2x/week; Q15: 2x/year	B+ Tree

Justification
The Enrolled_in table supports both frequent insertions when visitors join programs, d equality based lookups for retrieving enrollment data. Because of the mix between insertions and equality lookups, a B+ Tree would prove the most balanced and fficient for such a range of operations

Discussion of Storage Structure for **Enrolled_in** Table in Azure SQL Database

Table Name	Selected File Organization	Azure SQL Database Implementation
Enrolled_in	B+ Tree	Clustered index on (person_id, park_name, program_name)

Justification
Azure SQL implements B+ Tree using clustered indexes. In this case, it represents the composite key, enabling efficient insertions and lookups for anything involving the attributes.

Discussion of Storage Structure for **Park_Pass** Table

Table Name	Query # and Type	Search Key(s)	Query Frequency	Selected File Org.
Park_Pass	Q15 - Deletion	person_id; expiration_date	Q15: 2x/year	B+ Tree

Justification
The park_Pass table is used to identify and remove visitors with an expired pass. It gets filtered by expiration_date. Because of deletions reliance on range checks, a B+ Tree would work best. It allows for efficient identification of expired passes.

Discussion of Storage Structure for **Park_Pass** Table in Azure SQL Database

Table Name	Selected File Organization	Azure SQL Database Implementation
Park_Pass	B+ Tree	Clustered index on expiration_date

Justification
The clustered index on expiration date would enable Azure SQL to implement a B+ Tree structure. This would maximize efficiency for the deletion of expired visitors.

Discussion of Storage Structure for **Donation** Table

Table Name	Query # and Type	Search Key(s)	Query Frequency	Selected File Org.
Donation	Q4 – Insertion; Q11 – ange Search	person_id; donation_ date	Q4: 5x/day; Q11: 1x/month	B+ Tree

Justification
The Donation table supports frequent insertions and range queries filtered by month and year . A B+ Tree structure supports chronological ordering and efficient grouping of donations while maintaining high efficiency insertion performance for new entries.

Discussion of Storage Structure for **Donation** Table in Azure SQL Database

Table Name	Selected File Organization	Azure SQL Database Implementation
Donation	B+ tree	Clustered index on (person_id, donation_date)

Justification
Azure SQL automatically clusters indexes to implement a B+ Tree structure. In this case, the index would be on person_id and donation_date, allowing for highly efficient retrieval by date and donor.

Discussion of Storage Structure for **Check_Donation** Table

Table Name	Query # and Type	Search Key(s)	Query Frequency	Selected File Org.
Check_Donation	Q4 - Insertion	check_number	Q4: 5x/day	Hash

Justification
Check_Donation is used only during donation insertions to log payments by check. Since access is equality-based by check_number with ordering or ranged queries, a Hash structure is the optimal choice, as it minimizes insertion overhead and lookup time.

Discussion of Storage Structure for **Person** Table in Azure SQL Database

Table Name	Selected File Organization	Azure SQL Database Implementation
Check_Donation	Hash	Clustered index on check_number

Justification
In Azure SQL, clustered are used to handle equality based searches, in this case on the check_number attribute. This enable the constant-time access provided by a Hash.

Discussion of Storage Structure for **Card_Donation** Table

Table Name	Query # and Type	Search Key(s)	Query Frequency	Selected File Org.
Card_Donation	Q4 - Insertion	Card_type; last_four_digits, expiration_date	Q4: 5x/day	B+ Tree

Justification
Card_Donation entries are created whenever donors use card payments. Although insertions are most frequent, maintaining a B+ Tree organization supports efficient sorting and validation by card type and expiration date, which can be useful for detecting duplicates or expiring cards. It is also useful for the composite key that is used by this table.

Discussion of Storage Structure for **Card_Donation** Table in Azure SQL Database

Table Name	Selected File Organization	Azure SQL Database Implementation
Card_Donation	B+ Tree	Clustered index on (card_type, last_four_digits, expiration_date)

Justification
Azure SQL automatically applies a clustered B+ Tree to composite primary keys. This structure ensures ordered storage and efficient filtering by card information, while minimizing cost during frequent insertions.

Table Creation for **Person** Table

```sql
-- 1. PERSON
CREATE TABLE Person (
    person_id INT IDENTITY(1,1) PRIMARY KEY CLUSTERED,
    firstname NVARCHAR(50) NOT NULL,
    lastname NVARCHAR(50) NOT NULL,
    middle_initial NVARCHAR(10),
    dob DATE NOT NULL,
    gender NVARCHAR(20),
    street NVARCHAR(100),
    city NVARCHAR(50),
    state NVARCHAR(50),
    postal_code NVARCHAR(10),
    subscribed BIT
);
GO
```

Query 1

Run | Cancel query | Save query | Export data as | Show only Editor

```sql
1   CREATE TABLE Person (
2       person_id INT IDENTITY(1,1) PRIMARY KEY CLUSTERED,
3       firstname NVARCHAR(50) NOT NULL,
4       lastname NVARCHAR(50) NOT NULL,
5       middle_initial NVARCHAR(10),
6       dob DATE NOT NULL,
7       gender NVARCHAR(20),
8       street NVARCHAR(100),
9       city NVARCHAR(50),
10      state NVARCHAR(50)
```

Results | Messages

```
Query succeeded: Affected rows: 0
```

Table Creation for **Phone** Table

```
-- 2. PHONE
CREATE TABLE Phone (
    person_id INT NOT NULL FOREIGN KEY REFERENCES Person(person_id),
    phone_number NVARCHAR(20) NOT NULL,
    PRIMARY KEY (person_id, phone_number)
);
GO
```

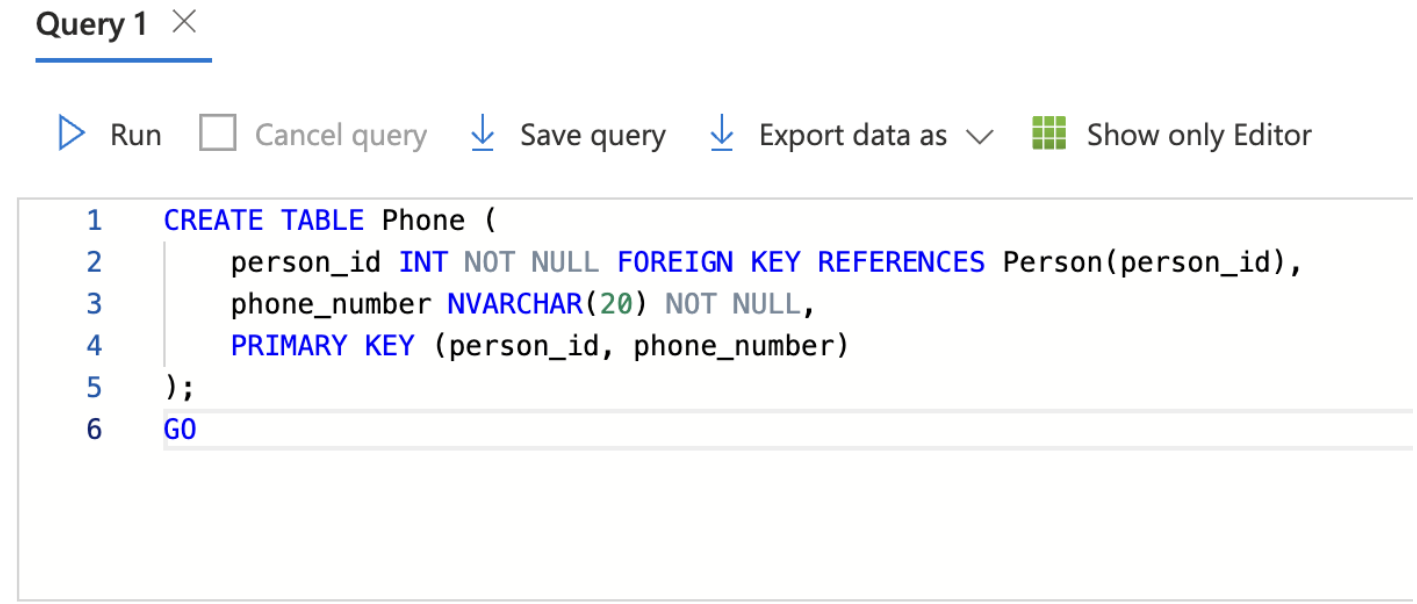


Table Creation for **Email** Table

```
-- 3. EMAIL
CREATE TABLE Email (
    person_id INT NOT NULL FOREIGN KEY REFERENCES Person(person_id),
    email_address NVARCHAR(100) NOT NULL,
    PRIMARY KEY (person_id, email_address)
);
GO
```

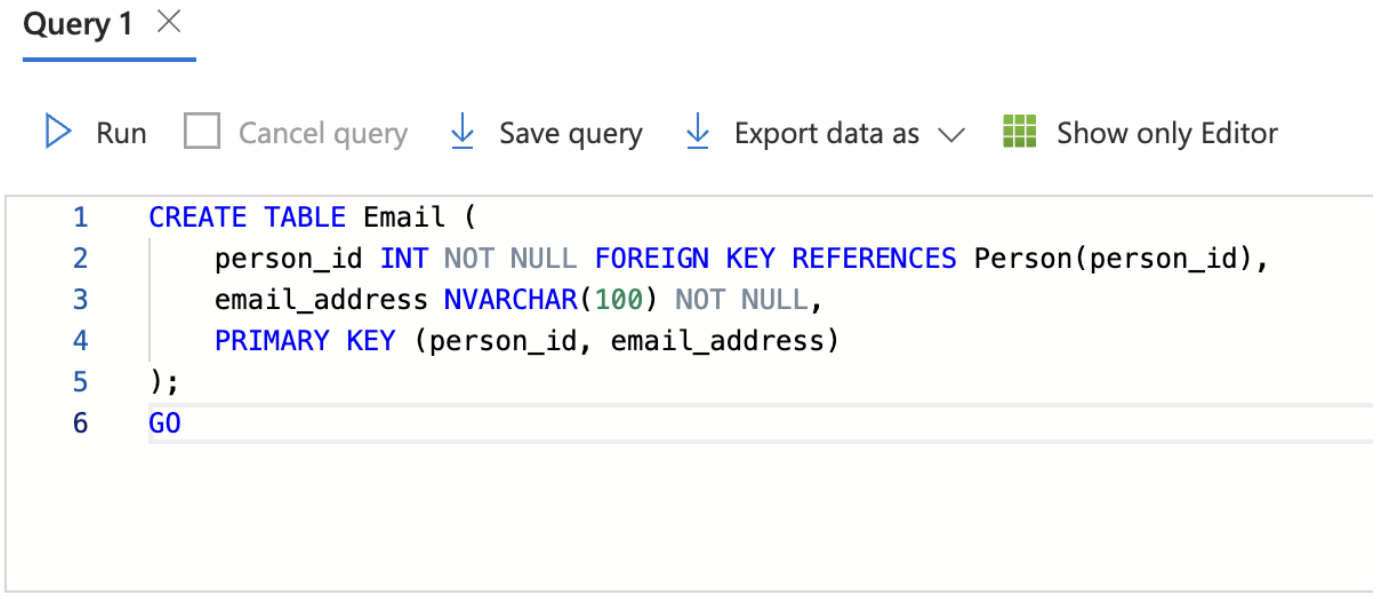


Table Creation for **Emergency_Contact** Table

```sql
-- 4. EMERGENCY CONTACT
CREATE TABLE Emergency_Contact (
    person_id INT NOT NULL FOREIGN KEY REFERENCES Person(person_id),
    contact_name NVARCHAR(100) NOT NULL,
    relationship NVARCHAR(50),
    phone_number NVARCHAR(20),
    PRIMARY KEY (person_id, contact_name)
);
GO
```

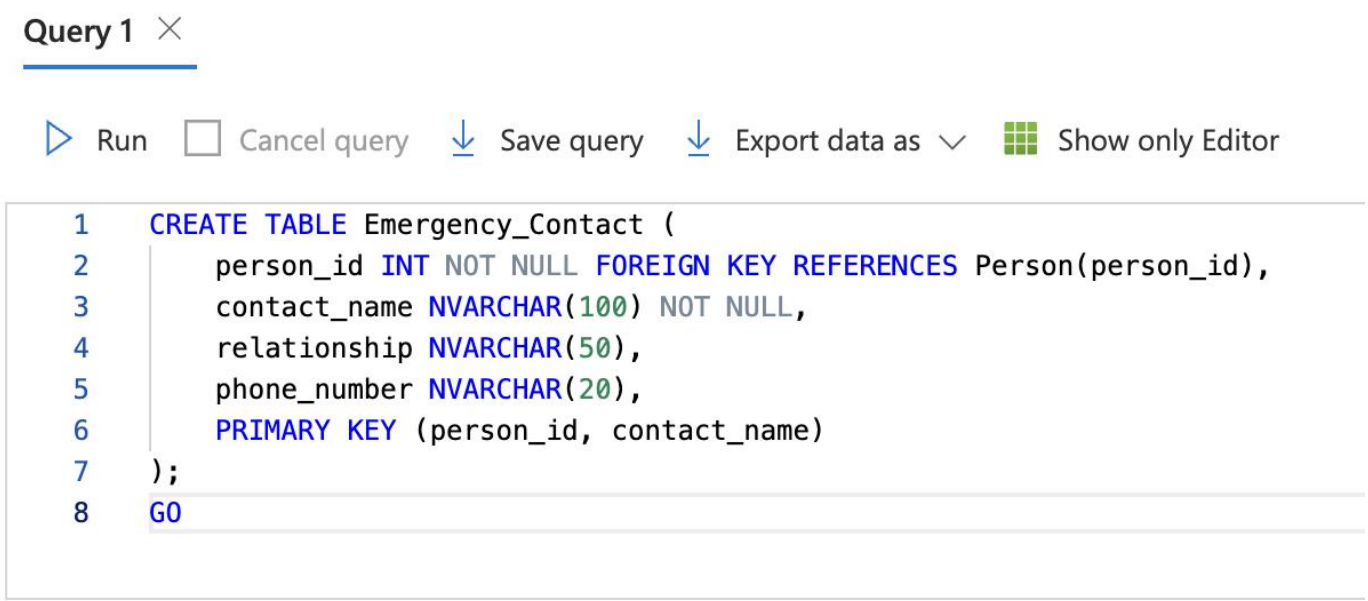


Table Creation for **Visitor** Table

```
-- 5. VISITOR
CREATE TABLE Visitor (
    person_id INT PRIMARY KEY FOREIGN KEY REFERENCES Person(person_id)
);
GO
```

Table Creation for **Ranger** Table

```
-- 6. RANGER
CREATE TABLE Ranger (
    person_id INT PRIMARY KEY FOREIGN KEY REFERENCES Person(person_id)
);
GO
```

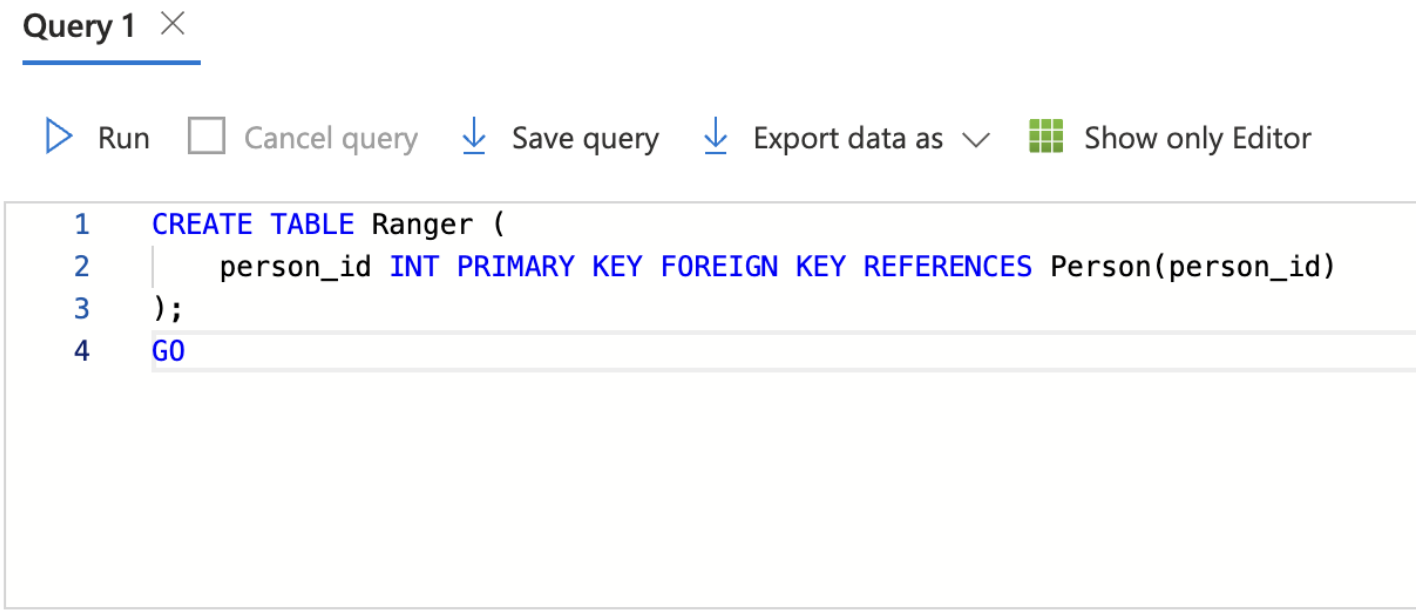


Table Creation for **Researcher** Table

```
-- 7. RESEARCHER
CREATE TABLE Researcher (
    person_id INT PRIMARY KEY FOREIGN KEY REFERENCES Person(person_id),
    research_field NVARCHAR(100),
    hire_date DATE,
    salary INT
);
GO
```

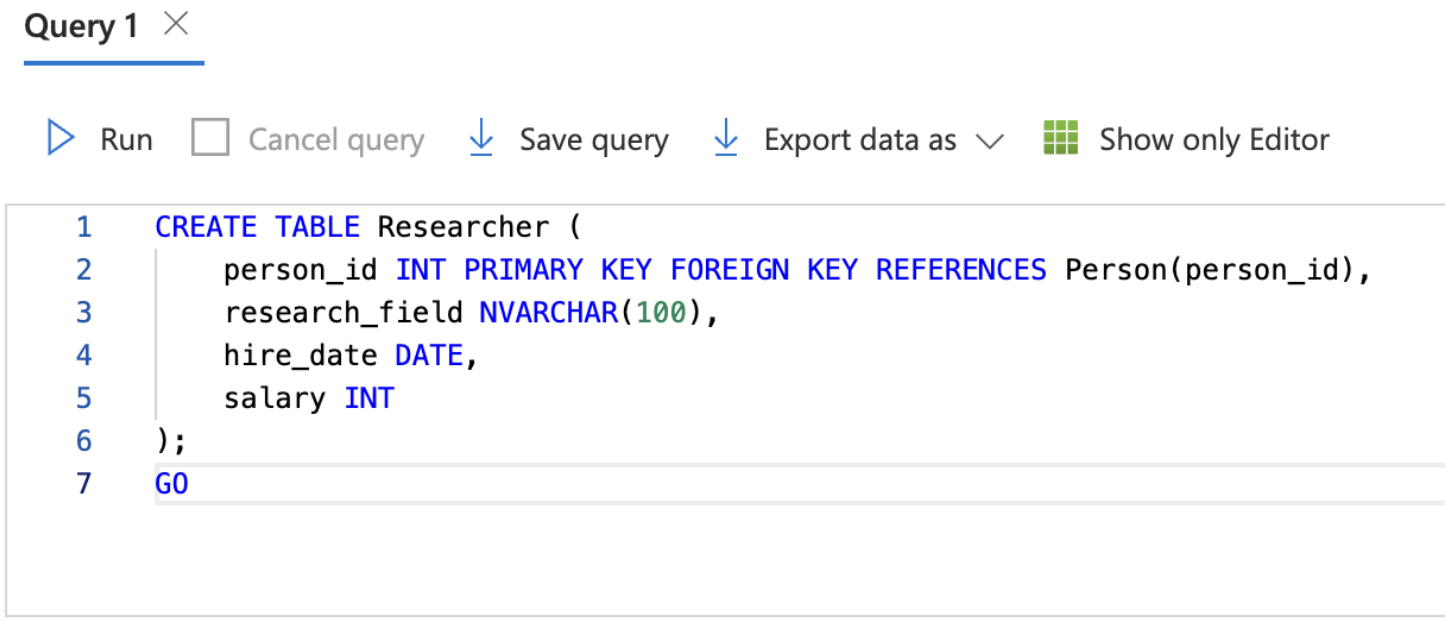


Table Creation for **Donor** Table

```
-- 8. DONOR
CREATE TABLE Donor (
    person_id INT PRIMARY KEY FOREIGN KEY REFERENCES Person(person_id),
    anonymity_preference BIT
);
GO
```

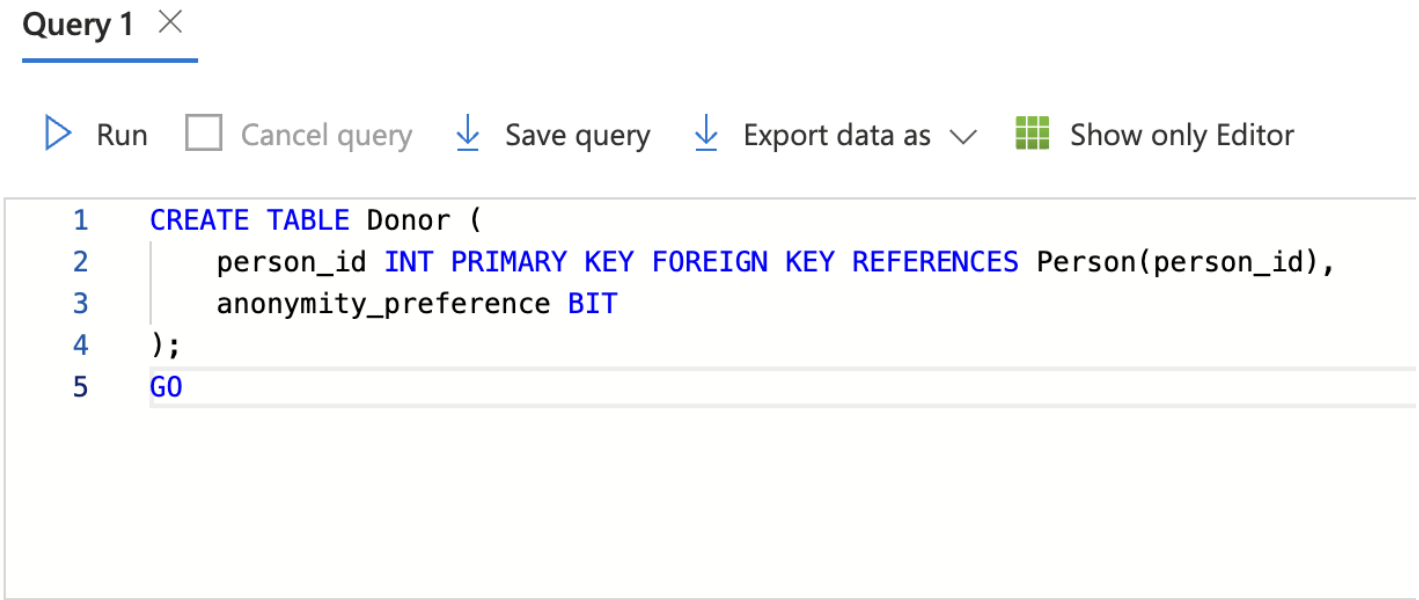


Table Creation for **National_Park** Table

```sql
-- 9. NATIONAL PARK
CREATE TABLE National_Park (
    park_name NVARCHAR(100) PRIMARY KEY CLUSTERED,
    street NVARCHAR(100),
    city NVARCHAR(50),
    state NVARCHAR(50),
    postal_code NVARCHAR(10),
    establishment_date DATE,
    capacity INT
);
GO
```

Query 1

Run | Cancel query | Save query | Export data as | Show only Editor

```sql
1   CREATE TABLE National_Park (
2       park_name NVARCHAR(100) PRIMARY KEY CLUSTERED,
3       street NVARCHAR(100),
4       city NVARCHAR(50),
5       state NVARCHAR(50),
6       postal_code NVARCHAR(10),
7       establishment_date DATE,
8       capacity INT
9   );
10  GO
```

Results | Messages

```
Query succeeded: Affected rows: 0
```

Table Creation for **Conservation_Project** Table

```
-- 10. CONSERVATION PROJECT
CREATE TABLE Conservation_Project (
    project_id INT IDENTITY(1,1) PRIMARY KEY CLUSTERED,
    name NVARCHAR(100),
    start_date DATE,
    budget INT,
    park_name NVARCHAR(100) FOREIGN KEY REFERENCES National_Park(park_name)
);
GO
```

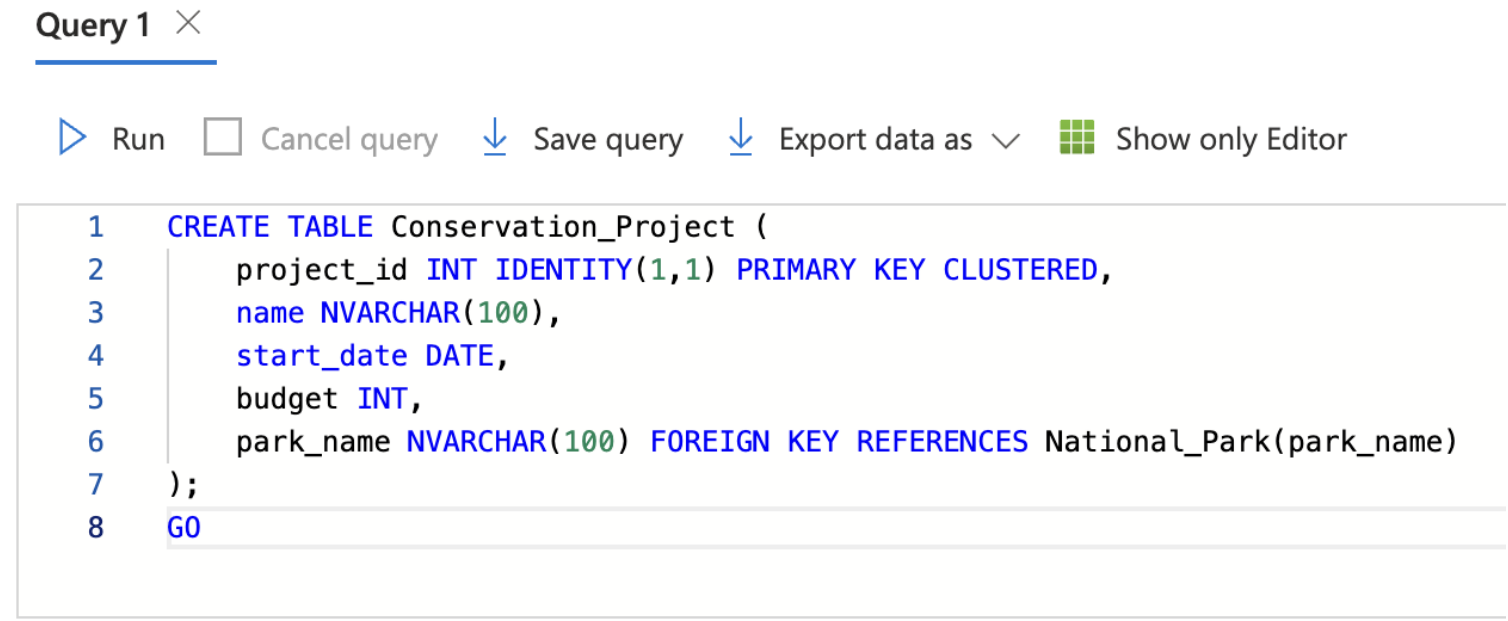


Table Creation for **Ranger_Team** Table

```
-- 11. RANGER TEAM
CREATE TABLE Ranger_Team (
    team_id INT IDENTITY(1,1) PRIMARY KEY CLUSTERED,
    focus_area NVARCHAR(100),
    formation_date DATE
);
GO
```

Query 1

Run | Cancel query | Save query | Export data as | Show only Editor

```
1  CREATE TABLE Ranger_Team (
2      team_id INT IDENTITY(1,1) PRIMARY KEY CLUSTERED,
3      focus_area NVARCHAR(100),
4      formation_date DATE
5  );
6  GO
```

Results | Messages

```
Query succeeded: Affected rows: 0
```

Table Creation for **Member_Of** Table

```
-- 12. MEMBER_OF
CREATE TABLE Member_of (
    person_id INT FOREIGN KEY REFERENCES Ranger(person_id),
    team_id INT FOREIGN KEY REFERENCES Ranger_Team(team_id),
    start_date DATE,
    status NVARCHAR(20),
    years_of_service INT,
    is_leader BIT,
    PRIMARY KEY (person_id, team_id)
);
GO
```

Query 1

Run | Cancel query | Save query | Export data as | Show only Editor

```
1   CREATE TABLE Member_of (
2       person_id INT FOREIGN KEY REFERENCES Ranger(person_id),
3       team_id INT FOREIGN KEY REFERENCES Ranger_Team(team_id),
4       start_date DATE,
5       status NVARCHAR(20),
6       years_of_service INT,
7       is_leader BIT,
8       PRIMARY KEY (person_id, team_id)
9   );
10  GO
```

Results | Messages

```
Query succeeded: Affected rows: 0
```

Table Creation for **Operates_In** Table

```
-- 13. OPERATES_IN
CREATE TABLE Operates_in (
    team_id INT FOREIGN KEY REFERENCES Ranger_Team(team_id),
    park_name NVARCHAR(100) FOREIGN KEY REFERENCES National_Park(park_name),
    PRIMARY KEY (team_id, park_name)
);
GO
```

Query 1

Run | Cancel query | Save query | Export data as | Show only Editor

```
1  CREATE TABLE Operates_in (
2      team_id INT FOREIGN KEY REFERENCES Ranger_Team(team_id),
3      park_name NVARCHAR(100) FOREIGN KEY REFERENCES National_Park(park_name),
4      PRIMARY KEY (team_id, park_name)
5  );
6  GO
```

Results | Messages

```
Query succeeded: Affected rows: 0
```

Table Creation for **Oversees** Table

```
-- 14. OVERSEES
CREATE TABLE Oversees (
    person_id INT FOREIGN KEY REFERENCES Researcher(person_id),
    team_id INT FOREIGN KEY REFERENCES Ranger_Team(team_id),
    report_date DATE,
    report_summary NVARCHAR(MAX),
    PRIMARY KEY (person_id, team_id, report_date)
);
GO
```

Query 1

Run | Cancel query | Save query | Export data as | Show only Editor

```
1   CREATE TABLE Oversees (
2       person_id INT FOREIGN KEY REFERENCES Researcher(person_id),
3       team_id INT FOREIGN KEY REFERENCES Ranger_Team(team_id),
4       report_date DATE,
5       report_summary NVARCHAR(MAX),
6       PRIMARY KEY (person_id, team_id, report_date)
7   );
8   GO
```

Results | Messages

```
Query succeeded: Affected rows: 0
```

Table Creation for **Certification** Table

```
-- 15. CERTIFICATION
CREATE TABLE Certification (
    person_id INT FOREIGN KEY REFERENCES Ranger(person_id),
    certification_name NVARCHAR(100),
    PRIMARY KEY (person_id, certification_name)
);
GO
```

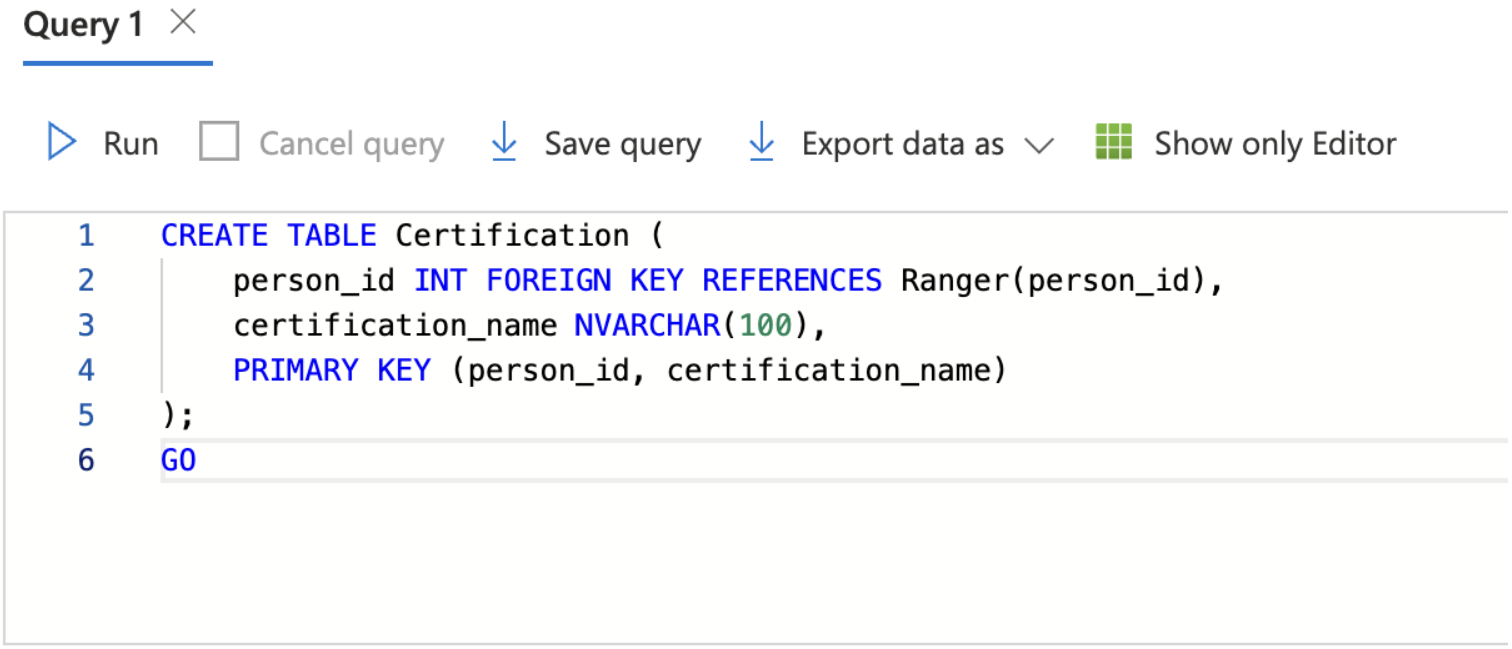


Table Creation for **Mentorship** Table

```
-- 16. MENTORSHIP
CREATE TABLE Mentorship (
    mentor_id INT FOREIGN KEY REFERENCES Ranger(person_id),
    mentee_id INT FOREIGN KEY REFERENCES Ranger(person_id),
    mentorship_start_date DATE,
    PRIMARY KEY (mentor_id, mentee_id)
);
GO
```

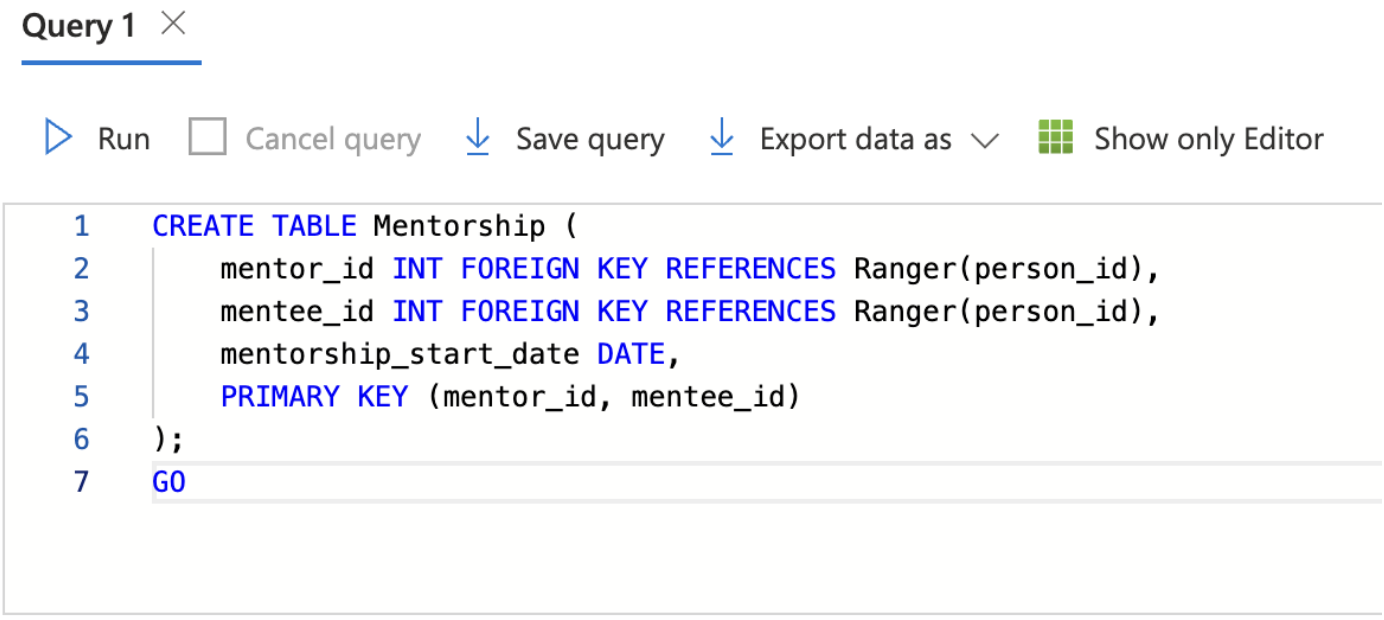


Table Creation for **Program** Table

```
-- 17. PROGRAM
CREATE TABLE Program (
    park_name NVARCHAR(100) FOREIGN KEY REFERENCES National_Park(park_name),
    program_name NVARCHAR(100),
    type NVARCHAR(50),
    start_date DATE,
    duration_hours INT,
    PRIMARY KEY CLUSTERED (park_name, program_name)
);
GO
```

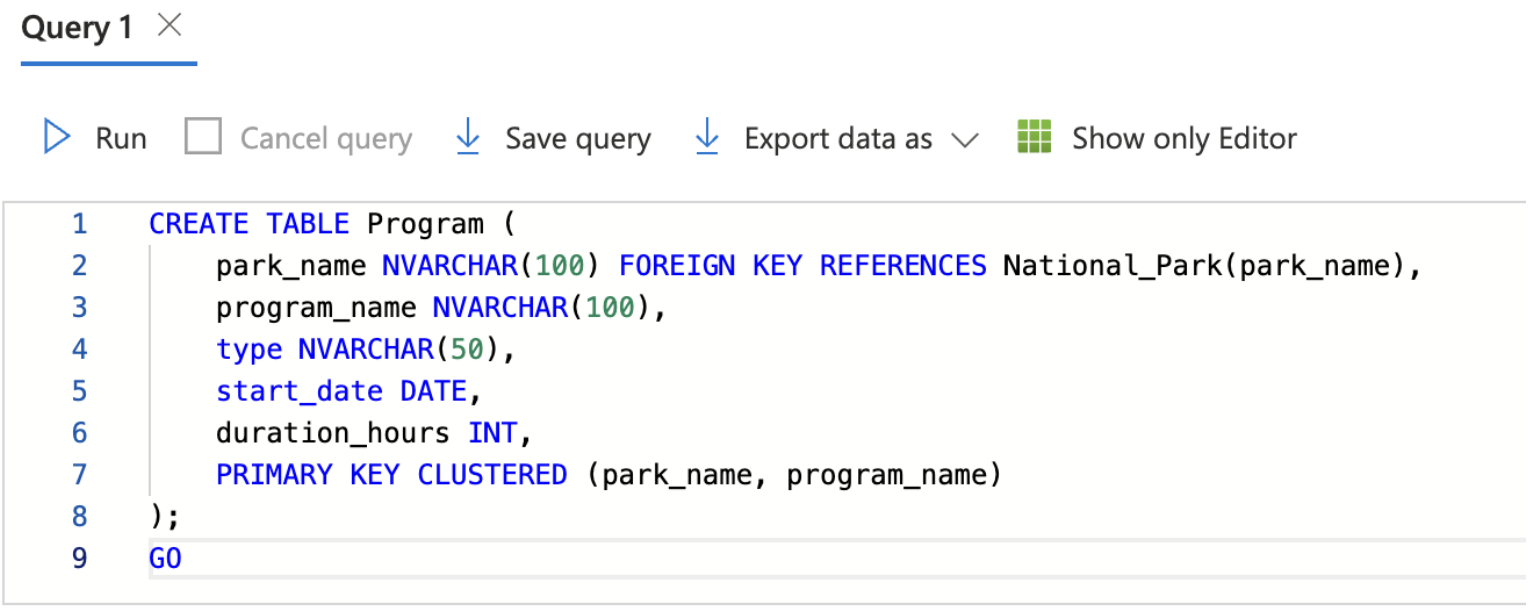


Table Creation for **Enrolled_In** Table

```sql
-- 18. ENROLLED_IN
CREATE TABLE Enrolled_in (
    person_id INT FOREIGN KEY REFERENCES Visitor(person_id),
    park_name NVARCHAR(100),
    program_name NVARCHAR(100),
    visit_date DATE,
    accessibility_needs NVARCHAR(200),
    PRIMARY KEY (person_id, park_name, program_name),
    FOREIGN KEY (park_name, program_name) REFERENCES Program(park_name, program_name)
);
GO
```

Query 1

Run | Cancel query | Save query | Export data as | Show only Editor

```sql
1   CREATE TABLE Enrolled_in (
2       person_id INT FOREIGN KEY REFERENCES Visitor(person_id),
3       park_name NVARCHAR(100),
4       program_name NVARCHAR(100),
5       visit_date DATE,
6       accessibility_needs NVARCHAR(200),
7       PRIMARY KEY (person_id, park_name, program_name),
8       FOREIGN KEY (park_name, program_name) REFERENCES Program(park_name, program_name)
9   );
10  GO
```

Results | Messages

```
Query succeeded: Affected rows: 0
```

Table Creation for **Park_Pass** Table

```
-- 19. PARK PASS
CREATE TABLE Park_Pass (
    pass_id INT IDENTITY(1,1) PRIMARY KEY CLUSTERED,
    type NVARCHAR(50),
    expiration_date DATE,
    person_id INT FOREIGN KEY REFERENCES Visitor(person_id)
);
GO
```

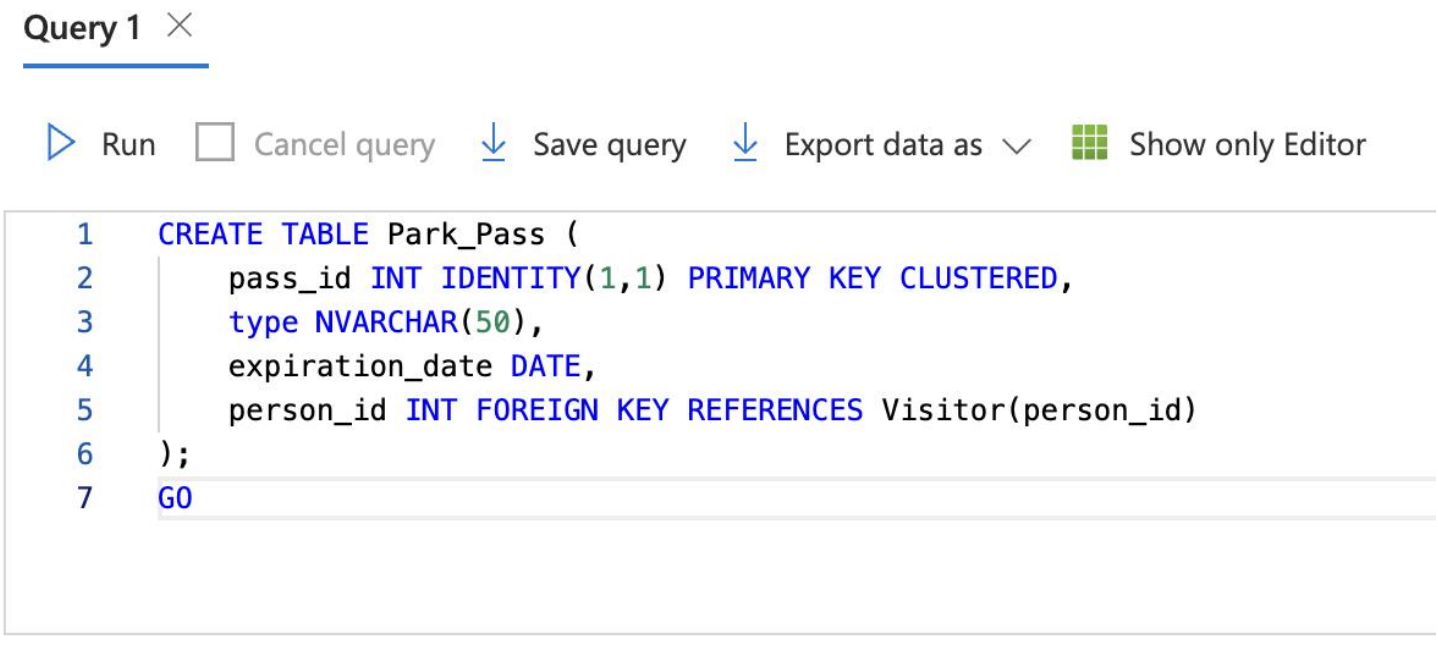


Table Creation for **Donation** Table

```
-- 20. DONATION
CREATE TABLE Donation (
    person_id INT FOREIGN KEY REFERENCES Donor(person_id),
    donation_date DATE,
    amount INT,
    campaign_name NVARCHAR(100),
    PRIMARY KEY (person_id, donation_date, amount)
);
GO
```

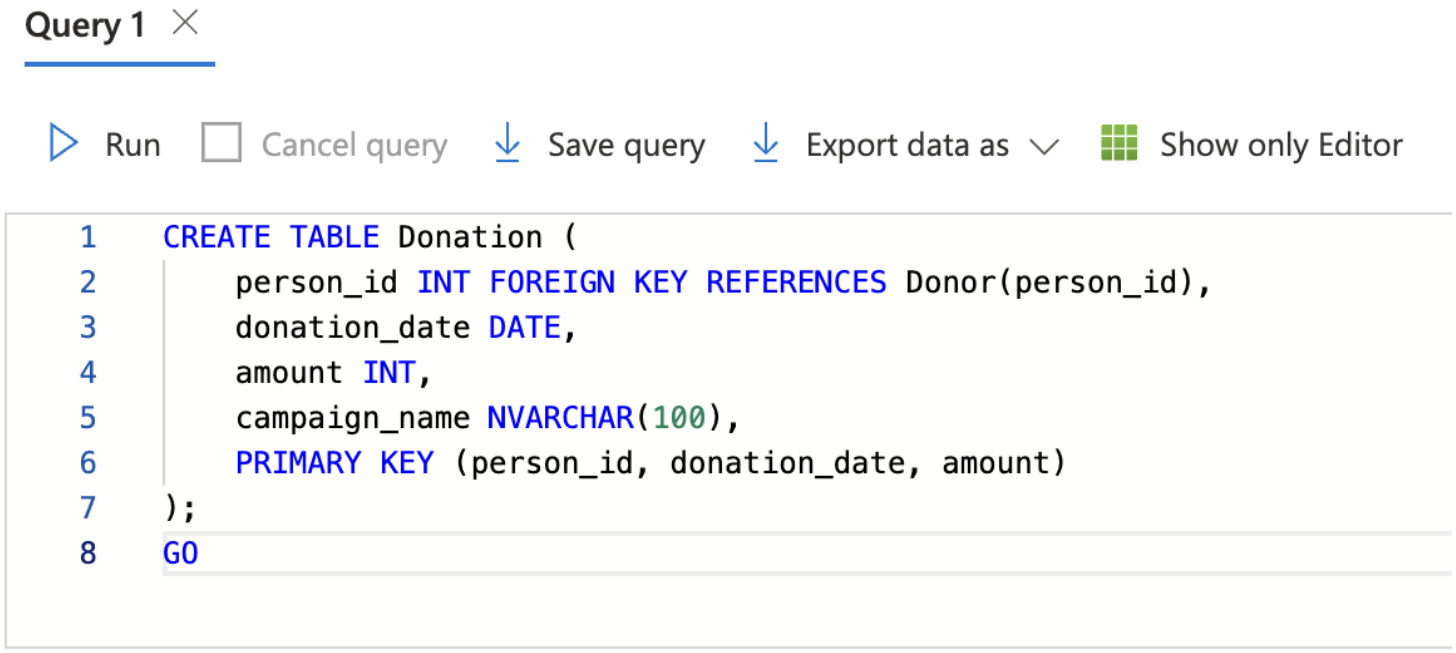


Table Creation for **Check_Donation** Table

```
-- 21. CHECK DONATION
CREATE TABLE Check_Donation (
    check_number INT PRIMARY KEY,
    person_id INT,
    donation_date DATE,
    amount INT,
    FOREIGN KEY (person_id, donation_date, amount)
        REFERENCES Donation(person_id, donation_date, amount)
);
GO
```

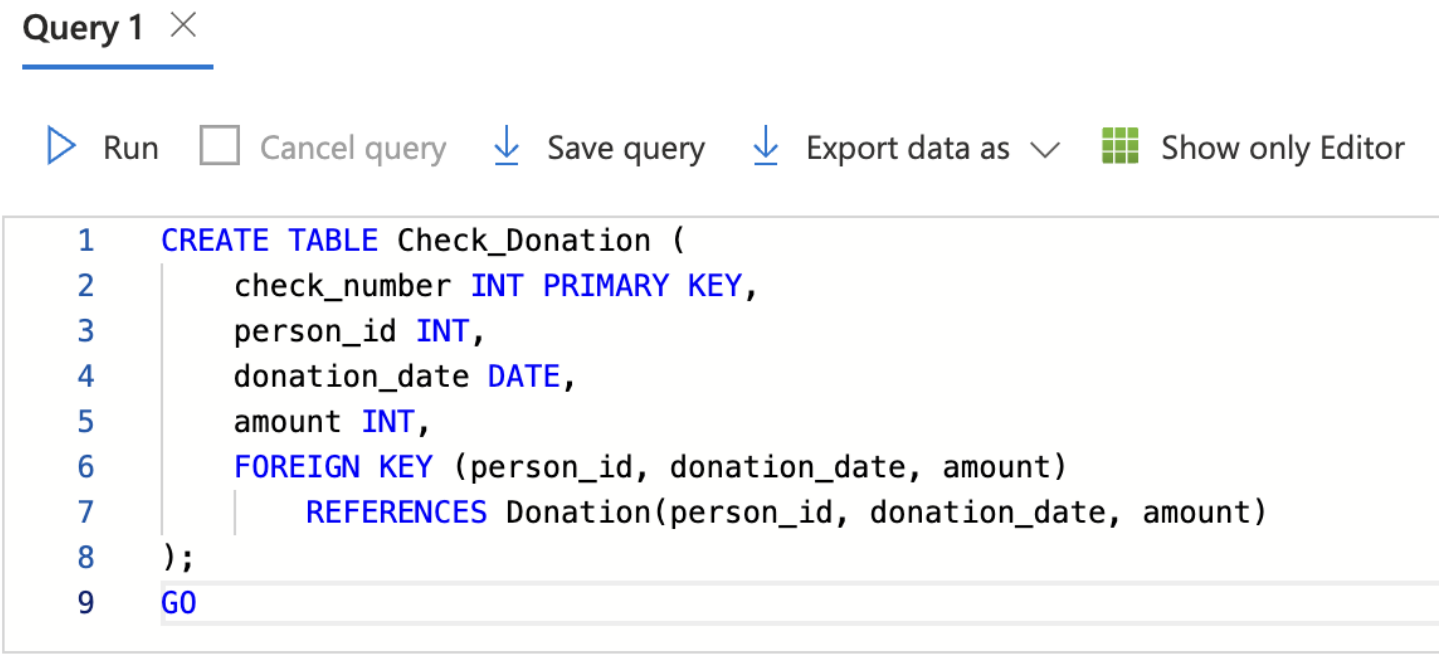
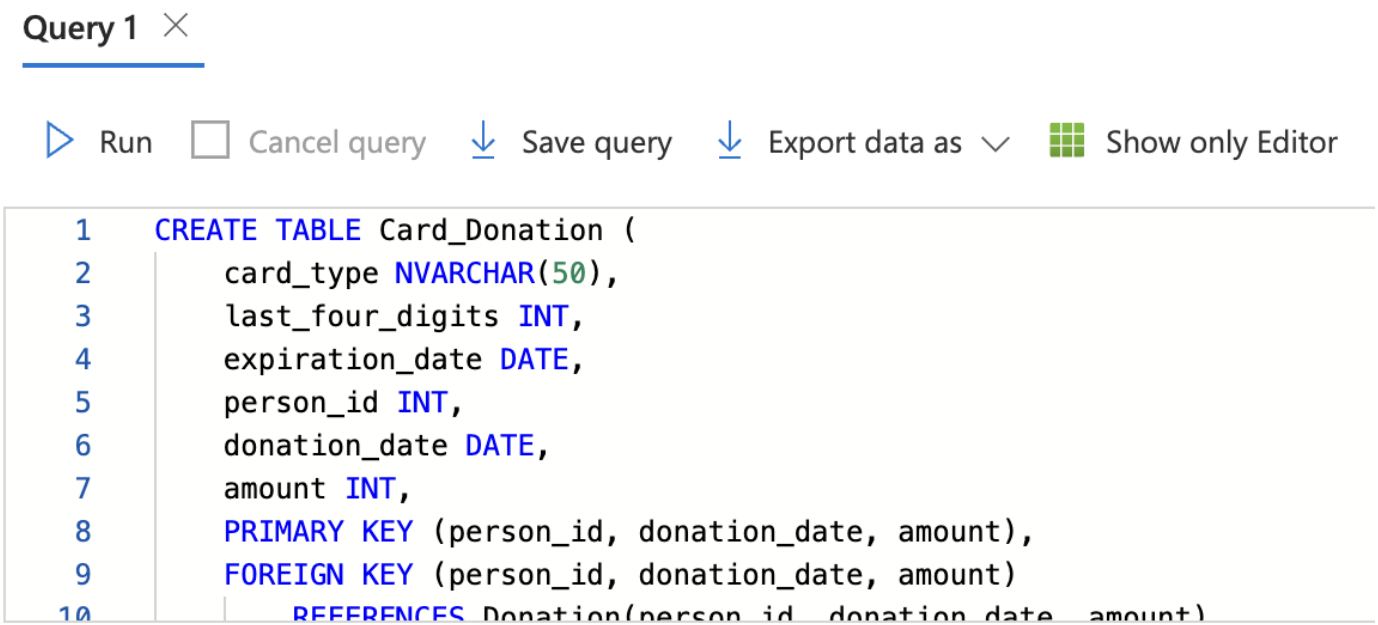


Table Creation for **Card_Donation** Table

```
-- 22. CARD DONATION
CREATE TABLE Card_Donation (
    card_type NVARCHAR(50),
    last_four_digits INT,
    expiration_date DATE,
    person_id INT,
    donation_date DATE,
    amount INT,
    PRIMARY KEY (person_id, donation_date, amount),
    FOREIGN KEY (person_id, donation_date, amount)
        REFERENCES Donation(person_id, donation_date, amount)
);
GO
```



Saved Procedure for **Query #1**: Insert Visitor w/ Error Checking Built-in

```sql
-----------------------------------------
-- Query 1: Insert Visitor
-----------------------------------------

CREATE OR ALTER PROCEDURE InsertNewVisitor
    @person_id INT,
    @park_name NVARCHAR(100),
    @program_name NVARCHAR(100),
    @visit_date DATE,
    @accessibility_needs NVARCHAR(100)
AS
BEGIN
    SET NOCOUNT ON;

    -- Validate input parameters
    IF @person_id IS NULL OR @person_id <= 0
    BEGIN
        RAISERROR('Invalid person_id: must be a positive integer.', 16, 1);
        RETURN;
    END

    IF @park_name IS NULL OR LTRIM(RTRIM(@park_name)) = ''
    BEGIN
        RAISERROR('Invalid park_name: cannot be null or empty.', 16, 1);
        RETURN;
    END

    IF @program_name IS NULL OR LTRIM(RTRIM(@program_name)) = ''
    BEGIN
        RAISERROR('Invalid program_name: cannot be null or empty.', 16, 1);
        RETURN;
    END

    IF @visit_date IS NULL
    BEGIN
        RAISERROR('Invalid visit_date: cannot be null.', 16, 1);
        RETURN;
    END

    -- Ensure the person exists
    IF NOT EXISTS (SELECT 1 FROM Person WHERE person_id = @person_id)
    BEGIN
        RAISERROR('The specified person_id does not exist in the Person table.', 16, 1);
        RETURN;
    END

    -- Ensure the park exists
    IF NOT EXISTS (SELECT 1 FROM National_Park WHERE park_name = @park_name)
    BEGIN
        RAISERROR('The specified park does not exist in the National_Park table.', 16, 1);
        RETURN;
    END

    -- Ensure the program exists under that park
    IF NOT EXISTS (
        SELECT 1
        FROM Program
        WHERE park_name = @park_name AND program_name = @program_name
    )
    BEGIN
        RAISERROR('The specified program does not exist for this park.', 16, 1);
        RETURN;
    END

    -- Ensure the person is not already a visitor
    IF EXISTS (SELECT 1 FROM Visitor WHERE person_id = @person_id)
    BEGIN
        IF EXISTS (
            SELECT 1
            FROM Enrolled_in
            WHERE person_id = @person_id
              AND park_name = @park_name
              AND program_name = @program_name
        )
        BEGIN
            RAISERROR('This visitor is already enrolled in the specified program.', 16, 1);
            RETURN;
        END
    END

    -- If not already a visitor, insert into Visitor
    IF NOT EXISTS (SELECT 1 FROM Visitor WHERE person_id = @person_id)
        INSERT INTO Visitor (person_id) VALUES (@person_id);

    -- Ensure they aren't already enrolled in another program on the same day (optional business rule)
    IF EXISTS (
        SELECT 1
        FROM Enrolled_in
        WHERE person_id = @person_id AND visit_date = @visit_date
    )
    BEGIN
        RAISERROR('This person already has an enrollment scheduled for the same date.', 16, 1);
        RETURN;
    END

    -- Insert enrollment record
    INSERT INTO Enrolled_in (person_id, park_name, program_name, visit_date, accessibility_needs)
    VALUES (@person_id, @park_name, @program_name, @visit_date, @accessibility_needs);

    -- Return success message
    SELECT CONCAT(
        'Success: Visitor (ID = ', @person_id,
        ') enrolled in "', @program_name,
        '" at park "', @park_name, '".'
    ) AS SuccessMessage;
END;
GO
```

Saved Procedure for **Query #2**: Insert Ranger w/ Error Checking Built-in

```sql
-----------------------------------------
-- Query 2: Insert Ranger
-----------------------------------------

CREATE OR ALTER PROCEDURE InsertNewRanger
    @person_id INT,
    @team_id INT,
    @start_date DATE,
    @status NVARCHAR(20),
    @certifications NVARCHAR(500) -- Comma-separated list
AS
BEGIN
    SET NOCOUNT ON;

    -- Validate parameters
    IF @person_id IS NULL OR @person_id <= 0
    BEGIN
        RAISERROR('Invalid person_id: must be a positive integer.', 16, 1);
        RETURN;
    END

    IF @team_id IS NULL OR @team_id <= 0
    BEGIN
        RAISERROR('Invalid team_id: must be a positive integer.', 16, 1);
        RETURN;
    END

    IF @start_date IS NULL
    BEGIN
        RAISERROR('Invalid start_date: cannot be null.', 16, 1);
        RETURN;
    END

    IF @status IS NULL OR LTRIM(RTRIM(@status)) = ''
    BEGIN
        RAISERROR('Invalid status: cannot be null or empty.', 16, 1);
        RETURN;
    END

    -- Check existence of person and team
    IF NOT EXISTS (SELECT 1 FROM Person WHERE person_id = @person_id)
    BEGIN
        RAISERROR('Person does not exist in the database.', 16, 1);
        RETURN;
    END

    IF NOT EXISTS (SELECT 1 FROM Ranger_Team WHERE team_id = @team_id)
    BEGIN
        RAISERROR('Ranger team does not exist.', 16, 1);
        RETURN;
    END

    -- Insert into Ranger if not exists
    IF NOT EXISTS (SELECT 1 FROM Ranger WHERE person_id = @person_id)
        INSERT INTO Ranger (person_id) VALUES (@person_id);

    -- Handle certifications (simplified with STRING_SPLIT)
    IF (@certifications IS NOT NULL AND LTRIM(RTRIM(@certifications)) <> '')
    BEGIN
        INSERT INTO Certification (person_id, certification_name)
        SELECT DISTINCT @person_id, LTRIM(RTRIM(value))
        FROM STRING_SPLIT(@certifications, ',')
        WHERE LTRIM(RTRIM(value)) <> ''
          AND NOT EXISTS (
              SELECT 1 FROM Certification
              WHERE person_id = @person_id
              AND certification_name = LTRIM(RTRIM(value))
          );
    END

    -- Prevent duplicate team assignment
    IF EXISTS (SELECT 1 FROM Member_of WHERE person_id = @person_id)
    BEGIN
        RAISERROR('This ranger is already assigned to a ranger team.', 16, 1);
        RETURN;
    END

    -- Insert into Member_of
    INSERT INTO Member_of (person_id, team_id, start_date, status, years_of_service, is_leader)
    VALUES (@person_id, @team_id, @start_date, @status, 0, 0);

    -- Return success
    SELECT CONCAT(
        'Success: Ranger (ID=', @person_id,
        ') assigned to team ', @team_id,
        ' with status "', @status, '". Certifications added if provided.'
    ) AS SuccessMessage;
END;
GO
```

Saved Procedure for **Query #3**: Insert Ranger Team w/ Error Checking Built-in

```sql
----------------------------------------
-- Query 3: Insert Ranger Team
----------------------------------------

CREATE OR ALTER PROCEDURE InsertNewRangerTeam
    @focus_area NVARCHAR(100),
    @formation_date DATE,
    @leader_id INT
AS
BEGIN
    SET NOCOUNT ON;

    -- Validate inputs
    IF @focus_area IS NULL OR LTRIM(RTRIM(@focus_area)) = ''
    BEGIN
        RAISERROR('Invalid focus_area: cannot be null or empty.', 16, 1);
        RETURN;
    END

    IF @formation_date IS NULL
    BEGIN
        RAISERROR('Invalid formation_date: cannot be null.', 16, 1);
        RETURN;
    END

    IF @leader_id IS NULL OR @leader_id <= 0
    BEGIN
        PRINT 'No valid leader ID provided. Proceeding with NULL leader.';
        SET @leader_id = NULL;
    END

    -- Prevent duplicate focus area names
    IF EXISTS (SELECT 1 FROM Ranger_Team WHERE focus_area = @focus_area)
    BEGIN
        RAISERROR('A ranger team with this focus area already exists.', 16, 1);
        RETURN;
    END

    -- Insert new ranger team
    INSERT INTO Ranger_Team (focus_area, formation_date)
    VALUES (@focus_area, @formation_date);

    DECLARE @team_id INT = (SELECT MAX(team_id) FROM Ranger_Team);

    SELECT CONCAT(
        'Success: New ranger team (ID=', @team_id,
        ') created for focus area "', @focus_area,
        '". Leader assignment skipped.'
    ) AS SuccessMessage;
END;
GO
```

Saved Procedure for **Query #4**: Insert Donation w/ Error Checking Built-in

```sql
-----------------------------------------
-- Query 4: Insert New Donation from a Donor
-----------------------------------------

CREATE OR ALTER PROCEDURE InsertNewDonation
    @person_id INT,
    @donation_date DATE,
    @amount DECIMAL(12, 2),
    @campaign_name NVARCHAR(100),
    @check_number INT = NULL,
    @card_type NVARCHAR(20) = NULL,
    @last_four_digits INT = NULL,
    @expiration_date DATE = NULL
AS
BEGIN
    SET NOCOUNT ON;

    -- Validate input parameters
    IF @person_id IS NULL OR @person_id <= 0
    BEGIN
        RAISERROR('Invalid person_id: must be a positive integer.', 16, 1);
        RETURN;
    END

    IF @donation_date IS NULL
    BEGIN
        RAISERROR('Invalid donation_date: cannot be null.', 16, 1);
        RETURN;
    END

    IF @amount IS NULL OR @amount <= 0
    BEGIN
        RAISERROR('Invalid amount: must be greater than zero.', 16, 1);
        RETURN;
    END

    IF @campaign_name IS NULL OR LTRIM(RTRIM(@campaign_name)) = ''
    BEGIN
        RAISERROR('Invalid campaign_name: cannot be null or empty.', 16, 1);
        RETURN;
    END

    -- Ensure donor exists
    IF NOT EXISTS (SELECT 1 FROM Donor WHERE person_id = @person_id)
    BEGIN
        RAISERROR('Donor does not exist. The person must be registered as a donor first.', 16, 1);
        RETURN;
    END

    -- Prevent duplicate donations on the same date and amount
    IF EXISTS (
        SELECT 1 FROM Donation
        WHERE person_id = @person_id AND donation_date = @donation_date AND amount = @amount
    )
    BEGIN
        RAISERROR('Duplicate donation detected for this donor with the same date and amount.', 16, 1);
        RETURN;
    END

    -- Insert into Donation table
    INSERT INTO Donation (person_id, donation_date, amount, campaign_name)
    VALUES (@person_id, @donation_date, @amount, @campaign_name);

    -- Determine payment method (Check or Card)
    IF @check_number IS NOT NULL
    BEGIN
        INSERT INTO Check_Donation (check_number, person_id, donation_date, amount)
        VALUES (@check_number, @person_id, @donation_date, @amount);
    END
    ELSE IF @card_type IS NOT NULL AND @last_four_digits IS NOT NULL AND @expiration_date IS NOT NULL
    BEGIN
        INSERT INTO Card_Donation (card_type, last_four_digits, expiration_date, person_id, donation_date, amount)
        VALUES (@card_type, @last_four_digits, @expiration_date, @person_id, @donation_date, @amount);
    END
    ELSE
    BEGIN
        RAISERROR('Invalid payment method: provide either check_number or full card details.', 16, 1);
        RETURN;
    END

    -- Return success message
    SELECT CONCAT(
        'Success: Donation of $', @amount, ' from donor (ID=', @person_id,
        ') added to campaign "', @campaign_name, '".'
    ) AS SuccessMessage;
END;
GO
```

Saved Procedure for **Query #5**: Insert Researcher w/ Error Checking Built-in

```sql
-----------------------------------------
-- Query 5: Insert Researcher
-----------------------------------------

CREATE OR ALTER PROCEDURE InsertNewResearcher
    @person_id INT,
    @research_field NVARCHAR(100),
    @hire_date DATE,
    @salary INT,
    @team_ids NVARCHAR(MAX)
AS
BEGIN
    SET NOCOUNT ON;

    -- Validate parameters
    IF @person_id IS NULL OR @person_id <= 0
    BEGIN
        RAISERROR('Invalid person_id: must be a positive integer.', 16, 1);
        RETURN;
    END

    IF @research_field IS NULL OR LTRIM(RTRIM(@research_field)) = ''
    BEGIN
        RAISERROR('Invalid research_field: cannot be null or empty.', 16, 1);
        RETURN;
    END

    IF @hire_date IS NULL
    BEGIN
        RAISERROR('Invalid hire_date: cannot be null.', 16, 1);
        RETURN;
    END

    IF @salary IS NULL OR @salary <= 0
    BEGIN
        RAISERROR('Invalid salary: must be greater than zero.', 16, 1);
        RETURN;
    END

    IF @team_ids IS NULL OR LTRIM(RTRIM(@team_ids)) = ''
    BEGIN
        RAISERROR('Invalid team_ids: at least one team ID must be provided.', 16, 1);
        RETURN;
    END

    -- Ensure person exists
    IF NOT EXISTS (SELECT 1 FROM Person WHERE person_id = @person_id)
    BEGIN
        RAISERROR('Specified person does not exist in the Person table.', 16, 1);
        RETURN;
    END

    -- Prevent duplicate researcher entry
    IF EXISTS (SELECT 1 FROM Researcher WHERE person_id = @person_id)
    BEGIN
        RAISERROR('This person is already registered as a Researcher.', 16, 1);
        RETURN;
    END

    -- Insert researcher record
    INSERT INTO Researcher (person_id, research_field, hire_date, salary)
    VALUES (@person_id, @research_field, @hire_date, @salary);

    -- Associate researcher with one or more ranger teams
    DECLARE @team_id NVARCHAR(20);
    DECLARE @pos INT;

    WHILE LEN(@team_ids) > 0
    BEGIN
        SET @pos = CHARINDEX(',', @team_ids);
        IF @pos > 0
        BEGIN
            SET @team_id = LTRIM(RTRIM(LEFT(@team_ids, @pos - 1)));
            SET @team_ids = SUBSTRING(@team_ids, @pos + 1, LEN(@team_ids) - @pos);
        END
        ELSE
        BEGIN
            SET @team_id = LTRIM(RTRIM(@team_ids));
            SET @team_ids = '';
        END

        IF NOT EXISTS (SELECT 1 FROM Ranger_Team WHERE team_id = @team_id)
        BEGIN
            RAISERROR('One or more provided team IDs do not exist in Ranger_Team.', 16, 1);
            RETURN;
        END

        IF EXISTS (
            SELECT 1 FROM Oversees
            WHERE person_id = @person_id AND team_id = @team_id
        )
        BEGIN
            RAISERROR('Researcher is already overseeing one of the provided teams.', 16, 1);
            RETURN;
        END

        INSERT INTO Oversees (person_id, team_id, report_date, report_summary)
        VALUES (@person_id, @team_id, @hire_date, 'Initial oversight report pending.');
    END

    -- Return success message
    SELECT CONCAT(
        'Success: Researcher (ID=', @person_id,
        ') added and assigned to provided ranger team(s).'
    ) AS SuccessMessage;
END;
GO
```

Saved Procedure for **Query #6**: Insert Report w/ Error Checking Built-in

```
-----------------------------------------
-- Query 6: Insert Report
-----------------------------------------

CREATE OR ALTER PROCEDURE InsertNewReport
    @person_id INT,
    @team_id INT,
    @report_date DATE,
    @report_summary NVARCHAR(MAX)
AS
BEGIN
    SET NOCOUNT ON;

    -- Validate parameters
    IF @person_id IS NULL OR @person_id <= 0
    BEGIN
        RAISERROR('Invalid person_id: must be a positive integer.', 16, 1);
        RETURN;
    END

    IF @team_id IS NULL OR @team_id <= 0
    BEGIN
        RAISERROR('Invalid team_id: must be a positive integer.', 16, 1);
        RETURN;
    END

    IF @report_date IS NULL
    BEGIN
        RAISERROR('Invalid report_date: cannot be null.', 16, 1);
        RETURN;
    END

    IF @report_summary IS NULL OR LTRIM(RTRIM(@report_summary)) = ''
    BEGIN
        RAISERROR('Invalid report_summary: cannot be null or empty.', 16, 1);
        RETURN;
    END

    -- Ensure researcher exists
    IF NOT EXISTS (SELECT 1 FROM Researcher WHERE person_id = @person_id)
    BEGIN
        RAISERROR('The specified researcher does not exist in the Researcher table.', 16, 1);
        RETURN;
    END

    -- Ensure ranger team exists and is overseen by the researcher
    IF NOT EXISTS (SELECT 1 FROM Ranger_Team WHERE team_id = @team_id)
    BEGIN
        RAISERROR('The specified ranger team does not exist in the Ranger_Team table.', 16, 1);
        RETURN;
    END

    IF NOT EXISTS (SELECT 1 FROM Oversees WHERE person_id = @person_id AND team_id = @team_id)
    BEGIN
        RAISERROR('This researcher does not oversee the specified ranger team.', 16, 1);
        RETURN;
    END

    IF EXISTS (SELECT 1 FROM Oversees WHERE person_id = @person_id AND team_id = @team_id AND report_date = @report_date)
    BEGIN
        RAISERROR('A report for this ranger team and researcher already exists on this date.', 16, 1);
        RETURN;
    END

    -- Insert report
    INSERT INTO Oversees (person_id, team_id, report_date, report_summary)
    VALUES (@person_id, @team_id, @report_date, @report_summary);

    -- Return success message
    SELECT CONCAT(
        'Success: Report submitted by ranger team (ID=', @team_id,
        ') to researcher (ID=', @person_id,
        ') on ', CONVERT(VARCHAR(10), @report_date, 120), '.'
    ) AS SuccessMessage;
END;
GO
```

Saved Procedure for **Query #7**: Insert Program w/ Error Checking Built-in

```sql
-----------------------------------------
-- Query 7: Insert Program
-----------------------------------------

CREATE OR ALTER PROCEDURE InsertNewProgram
    @park_name NVARCHAR(100),
    @program_name NVARCHAR(100),
    @type NVARCHAR(50),
    @start_date DATE,
    @duration_hours INT
AS
BEGIN
    SET NOCOUNT ON;

    -- Validate parameters
    IF @park_name IS NULL OR LTRIM(RTRIM(@park_name)) = ''
    BEGIN
        RAISERROR('Invalid park_name: cannot be null or empty.', 16, 1);
        RETURN;
    END

    IF @program_name IS NULL OR LTRIM(RTRIM(@program_name)) = ''
    BEGIN
        RAISERROR('Invalid program_name: cannot be null or empty.', 16, 1);
        RETURN;
    END

    IF @type IS NULL OR LTRIM(RTRIM(@type)) = ''
    BEGIN
        RAISERROR('Invalid type: must be specified.', 16, 1);
        RETURN;
    END

    IF @start_date IS NULL
    BEGIN
        RAISERROR('Invalid start_date: cannot be null.', 16, 1);
        RETURN;
    END

    IF @duration_hours IS NULL OR @duration_hours <= 0
    BEGIN
        RAISERROR('Invalid duration_hours: must be greater than zero.', 16, 1);
        RETURN;
    END

    -- Ensure park exists
    IF NOT EXISTS (SELECT 1 FROM National_Park WHERE park_name = @park_name)
    BEGIN
        RAISERROR('The specified park does not exist in the National_Park table.', 16, 1);
        RETURN;
    END

    -- Prevent duplicate program names within the same park
    IF EXISTS (SELECT 1 FROM Program WHERE park_name = @park_name AND program_name = @program_name)
    BEGIN
        RAISERROR('A program with this name already exists for the specified park.', 16, 1);
        RETURN;
    END

    -- Ensure start_date is not in the past
    IF @start_date < GETDATE()
    BEGIN
        RAISERROR('Invalid start_date: program cannot start in the past.', 16, 1);
        RETURN;
    END

    -- Insert new program
    INSERT INTO Program (park_name, program_name, type, start_date, duration_hours)
    VALUES (@park_name, @program_name, @type, @start_date, @duration_hours);

    -- Return success message
    SELECT CONCAT(
        'Success: Program "', @program_name, '" added to park "', @park_name,
        '" with start date ', CONVERT(VARCHAR(10), @start_date, 120),
        ' and duration ', @duration_hours, ' hours.'
    ) AS SuccessMessage;
END;
GO
```

Saved Procedure for **Query #8**: Retrieve Emergency Contacts w/ Error Checking Built-in

```sql
----------------------------------------
-- Query 8: Retrieve Emergency Contacts
----------------------------------------

CREATE OR ALTER PROCEDURE RetrieveEmergencyContacts
    @person_id INT
AS
BEGIN
    SET NOCOUNT ON;

    -- Validate parameters
    IF @person_id IS NULL OR @person_id <= 0
    BEGIN
        RAISERROR('Invalid person_id: must be a positive integer.', 16, 1);
        RETURN;
    END

    -- Ensure person exists
    IF NOT EXISTS (SELECT 1 FROM Person WHERE person_id = @person_id)
    BEGIN
        RAISERROR('The specified person does not exist in the Person table.', 16, 1);
        RETURN;
    END

    -- Ensure emergency contacts exist for the person
    IF NOT EXISTS (SELECT 1 FROM Emergency_Contact WHERE person_id = @person_id)
    BEGIN
        RAISERROR('This person does not have any emergency contacts on record.', 16, 1);
        RETURN;
    END

    -- Retrieve emergency contacts
    SELECT
        ec.contact_name AS [Contact Name],
        ec.relationship AS [Relationship],
        ec.phone_number AS [Phone Number]
    FROM Emergency_Contact ec
    WHERE ec.person_id = @person_id
    ORDER BY ec.contact_name;
END;
GO
```

Saved Procedure for **Query #9**: Retrieve Enrolled Visitors w/ Error Checking Built-in

```
----------------------------------------
-- Query 9: Retrieve Enrolled Visitors
----------------------------------------

CREATE OR ALTER PROCEDURE RetrieveEnrolledVisitors
    @park_name NVARCHAR(100),
    @program_name NVARCHAR(100)
AS
BEGIN
    SET NOCOUNT ON;

    -- Validate parameters
    IF @park_name IS NULL OR LTRIM(RTRIM(@park_name)) = ''
    BEGIN
        RAISERROR('Invalid park_name: cannot be null or empty.', 16, 1);
        RETURN;
    END

    IF @program_name IS NULL OR LTRIM(RTRIM(@program_name)) = ''
    BEGIN
        RAISERROR('Invalid program_name: cannot be null or empty.', 16, 1);
        RETURN;
    END

    -- Ensure park exists
    IF NOT EXISTS (SELECT 1 FROM National_Park WHERE park_name = @park_name)
    BEGIN
        RAISERROR('The specified park does not exist in the National_Park table.', 16, 1);
        RETURN;
    END

    -- Ensure program exists for the park
    IF NOT EXISTS (SELECT 1 FROM Program WHERE park_name = @park_name AND program_name = @program_name)
    BEGIN
        RAISERROR('The specified program does not exist for the given park.', 16, 1);
        RETURN;
    END

    -- Ensure there are enrolled visitors
    IF NOT EXISTS (SELECT 1 FROM Enrolled_in WHERE park_name = @park_name AND program_name = @program_name)
    BEGIN
        RAISERROR('No visitors are currently enrolled in this program.', 16, 1);
        RETURN;
    END

    -- Retrieve enrolled visitors
    SELECT
        p.person_id AS [Visitor ID],
        CONCAT(p.firstname, ' ', p.lastname) AS [Visitor Name],
        e.visit_date AS [Visit Date],
        e.accessibility_needs AS [Accessibility Needs]
    FROM Enrolled_in e
    INNER JOIN Person p ON e.person_id = p.person_id
    WHERE e.park_name = @park_name AND e.program_name = @program_name
    ORDER BY p.lastname, p.firstname;
END;
GO
```

Saved Procedure for **Query #10**: Retrieve Park Programs After Date w/ Error Checking Built-in

```sql
----------------------------------------
-- Query 10: Retrieve Park Programs After Date
----------------------------------------

CREATE OR ALTER PROCEDURE RetrieveParkProgramsAfterDate
    @park_name NVARCHAR(100),
    @given_date DATE
AS
BEGIN
    SET NOCOUNT ON;

    -- Validate parameters
    IF @park_name IS NULL OR LTRIM(RTRIM(@park_name)) = ''
    BEGIN
        RAISERROR('Invalid park_name: cannot be null or empty.', 16, 1);
        RETURN;
    END

    IF @given_date IS NULL
    BEGIN
        RAISERROR('Invalid given_date: cannot be null.', 16, 1);
        RETURN;
    END

    -- Ensure park exists
    IF NOT EXISTS (SELECT 1 FROM National_Park WHERE park_name = @park_name)
    BEGIN
        RAISERROR('The specified park does not exist in the National_Park table.', 16, 1);
        RETURN;
    END

    -- Ensure there are programs for the park
    IF NOT EXISTS (SELECT 1 FROM Program WHERE park_name = @park_name)
    BEGIN
        RAISERROR('No programs exist for the specified park.', 16, 1);
        RETURN;
    END

    -- Ensure there are programs after the given date
    IF NOT EXISTS (SELECT 1 FROM Program WHERE park_name = @park_name AND start_date > @given_date)
    BEGIN
        RAISERROR('No programs found that start after the specified date.', 16, 1);
        RETURN;
    END

    -- Retrieve programs starting after the given date
    SELECT
        program_name AS [Program Name],
        type AS [Program Type],
        start_date AS [Start Date],
        duration_hours AS [Duration (Hours)]
    FROM Program
    WHERE park_name = @park_name AND start_date > @given_date
    ORDER BY start_date ASC;
END;
GO
```

Saved Procedure for **Query #11**: Retrieve Anonymous Donations w/ Error Checking Built-in

```sql
-----------------------------------------
-- Query 11: Retrieve Anonymous Donations
-----------------------------------------

CREATE OR ALTER PROCEDURE RetrieveAnonymousDonationsSummary
    @month INT,
    @year INT
AS
BEGIN
    SET NOCOUNT ON;

    -- Validate parameters
    IF @month IS NULL OR @month < 1 OR @month > 12
    BEGIN
        RAISERROR('Invalid month: must be between 1 and 12.', 16, 1);
        RETURN;
    END

    IF @year IS NULL OR @year < 1900
    BEGIN
        RAISERROR('Invalid year: must be a valid year greater than 1900.', 16, 1);
        RETURN;
    END

    -- Ensure there are anonymous donors
    IF NOT EXISTS (SELECT 1 FROM Donor WHERE anonymity_preference = 1)
    BEGIN
        RAISERROR('No anonymous donors exist in the Donor table.', 16, 1);
        RETURN;
    END

    -- Ensure there are anonymous donations for the specified month and year
    IF NOT EXISTS (
        SELECT 1
        FROM Donation d
        INNER JOIN Donor dn ON d.person_id = dn.person_id
        WHERE dn.anonymity_preference = 1
        AND MONTH(d.donation_date) = @month
        AND YEAR(d.donation_date) = @year
    )
    BEGIN
        RAISERROR('No anonymous donations found for the specified month and year.', 16, 1);
        RETURN;
    END

    -- Retrieve summary of anonymous donations
    SELECT
        YEAR(d.donation_date) AS [Donation Year],
        MONTH(d.donation_date) AS [Donation Month],
        SUM(d.amount) AS [Total Donation],
        AVG(d.amount) AS [Average Donation],
        COUNT(*) AS [Number of Donations]
    FROM Donation d
    INNER JOIN Donor dn ON d.person_id = dn.person_id
    WHERE dn.anonymity_preference = 1
      AND MONTH(d.donation_date) = @month
      AND YEAR(d.donation_date) = @year
    GROUP BY YEAR(d.donation_date), MONTH(d.donation_date)
    ORDER BY [Total Donation] DESC;
END;
GO
```

Saved Procedure for **Query #12**: Retrieve Rangers in Team w/ Error Checking Built-in

```
-----------------------------------------
-- Query 12: Retrieve Rangers in Team
-----------------------------------------

CREATE OR ALTER PROCEDURE RetrieveRangersInTeam
    @team_id INT
AS
BEGIN
    SET NOCOUNT ON;

    -- Validate parameters
    IF @team_id IS NULL OR @team_id <= 0
    BEGIN
        RAISERROR('Invalid team_id: must be a positive integer.', 16, 1);
        RETURN;
    END

    -- Ensure ranger team exists
    IF NOT EXISTS (SELECT 1 FROM Ranger_Team WHERE team_id = @team_id)
    BEGIN
        RAISERROR('The specified ranger team does not exist in the Ranger_Team table.', 16, 1);
        RETURN;
    END

    -- Ensure there are rangers assigned to the team
    IF NOT EXISTS (SELECT 1 FROM Member_of WHERE team_id = @team_id)
    BEGIN
        RAISERROR('No rangers are currently assigned to this team.', 16, 1);
        RETURN;
    END

    -- Retrieve rangers in the specified team
    SELECT
        p.person_id AS [Ranger ID],
        CONCAT(p.firstname, ' ', p.lastname) AS [Ranger Name],
        ISNULL(c.certification_name, 'None') AS [Certification],
        m.years_of_service AS [Years of Service],
        CASE WHEN m.is_leader = 1 THEN 'Leader' ELSE 'Member' END AS [Role]
    FROM Member_of m
    INNER JOIN Person p ON m.person_id = p.person_id
    LEFT JOIN Certification c ON p.person_id = c.person_id
    WHERE m.team_id = @team_id
    ORDER BY [Role] DESC, [Years of Service] DESC, [Ranger Name];
END;
GO
```

Saved Procedure for **Query #13**: Retrieve All Individuals w/ Error Checking Built-in

```
----------------------------------------
-- Query 13: Retrieve All Individuals
----------------------------------------

CREATE OR ALTER PROCEDURE RetrieveAllIndividuals
AS
BEGIN
    SET NOCOUNT ON;

    -- Ensure there are individuals in the Person table
    IF NOT EXISTS (SELECT 1 FROM Person)
    BEGIN
        RAISERROR('No individuals exist in the Person table.', 16, 1);
        RETURN;
    END

    -- Retrieve all individuals with contact details and subscription status
    SELECT
        p.person_id AS [Person ID],
        CONCAT(p.firstname, ' ', p.lastname) AS [Full Name],
        ISNULL(ph.phone_number, 'N/A') AS [Phone Number],
        ISNULL(e.email_address, 'N/A') AS [Email Address],
        CASE
            WHEN p.subscribed = 1 THEN 'Subscribed'
            ELSE 'Not Subscribed'
        END AS [Newsletter Status],
        p.city AS [City],
        p.state AS [State]
    FROM Person p
    LEFT JOIN Phone ph ON p.person_id = ph.person_id
    LEFT JOIN Email e ON p.person_id = e.person_id
    ORDER BY p.lastname, p.firstname;
END;
GO
```

Saved Procedure for **Query #14**: Update Researchers' Salary w/ Error Checking Built-in

```sql
----------------------------------------
-- Query 14: Update Researchers Salary
----------------------------------------

CREATE OR ALTER PROCEDURE UpdateResearchersSalary
AS
BEGIN
    SET NOCOUNT ON;

    -- Ensure there are researchers in the Researcher table
    IF NOT EXISTS (SELECT 1 FROM Researcher)
    BEGIN
        RAISERROR('No researchers found in the Researcher table.', 16, 1);
        RETURN;
    END

    -- Ensure there are researchers overseeing more than one ranger team
    IF NOT EXISTS (
        SELECT person_id
        FROM Oversees
        GROUP BY person_id
        HAVING COUNT(DISTINCT team_id) > 1
    )
    BEGIN
        RAISERROR('No researchers oversee more than one ranger team.', 16, 1);
        RETURN;
    END

    -- Update salary by 3% for researchers overseeing more than one ranger team
    UPDATE Researcher
    SET salary = salary * 1.03
    WHERE person_id IN (
        SELECT person_id
        FROM Oversees
        GROUP BY person_id
        HAVING COUNT(DISTINCT team_id) > 1
    );

    -- Return updated researchers with new salaries
    SELECT
        r.person_id AS [Researcher ID],
        CONCAT(p.firstname, ' ', p.lastname) AS [Researcher Name],
        r.salary AS [New Salary]
    FROM Researcher r
    INNER JOIN Person p ON r.person_id = p.person_id
    WHERE r.person_id IN (
        SELECT person_id
        FROM Oversees
        GROUP BY person_id
        HAVING COUNT(DISTINCT team_id) > 1
    );
END;
GO
```

Saved Procedure for **Query #15**: Delete Inactive Visitors

```
-----------------------------------------
-- Query 15: Delete Inactive Visitors
-----------------------------------------

CREATE OR ALTER PROCEDURE DeleteInactiveVisitors
AS
BEGIN
    SET NOCOUNT ON;

    -- Delete park passes for visitors who are not enrolled and whose passes are expired
    DELETE FROM Park_Pass
    WHERE person_id IN (
        SELECT v.person_id
        FROM Visitor v
        LEFT JOIN Enrolled_in e ON v.person_id = e.person_id
        WHERE e.person_id IS NULL
          AND v.person_id IN (
              SELECT person_id FROM Park_Pass WHERE expiration_date < GETDATE()
          )
    );

    -- Delete visitors who now have no enrollments and no passes
    DELETE FROM Visitor
    WHERE person_id NOT IN (SELECT person_id FROM Enrolled_in)
      AND person_id NOT IN (SELECT person_id FROM Park_Pass);

    DECLARE @deleted INT = @@ROWCOUNT;

    -- Return the number of deleted visitors
    SELECT CONCAT(@deleted, ' visitor(s) deleted who had expired passes and no enrollments.') AS ResultMessage;
END;
GO
```

Java Source Program (Pt. 1)

```java
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.Statement;
import java.util.Scanner;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.DriverManager;

public class Gray_Yale_IP_Task5b {
    // Database credentials
    final static String HOSTNAME = "gray0188.database.windows.net";
    final static String DBNAME = "cs-dsa-4513-sql-db";
    final static String USERNAME = "gray0188";
    final static String PASSWORD = "OkcThunder2025";
    // Database connection string
    final static String URL =
        String.format("jdbc:sqlserver://%s:1433;database=%s;user=%s;password=%s;encrypt=true;trustServerCertificate=false;hostNameInCertificate=*.database.windows.net;loginTimeout=30;",
            HOSTNAME, DBNAME, USERNAME, PASSWORD);
    // User input prompt
    final static String PROMPT =
            "1) Insert New Visitor and Associate with Park Program\n" +
            "2) Insert New Ranger and Assign to Team\n" +
            "3) Insert New Ranger Team and Set Leader\n" +
            "4) Insert New Donation from Donor\n" +
            "5) Insert New Researcher and Associate with Teams\n" +
            "6) Insert New Report from Ranger Team to Researcher\n" +
            "7) Insert New Park Program for a Park\n" +
            "8) Retrieve Emergency Contacts for a Person\n" +
            "9) Retrieve Visitors Enrolled in a Specific Park Program\n" +
            "10) Retrieve Park Programs Starting After a Given Date\n" +
            "11) Retrieve Total and Average Anonymous Donations\n" +
            "12) Retrieve Rangers in a Team with Certifications and Roles\n" +
            "13) Retrieve All Individuals (Mailing List)\n" +
            "14) Update Salary of Researchers Overseeing Multiple Teams\n" +
            "15) Delete Inactive Visitors with Expired Park Passes\n" +
            "16) Import New Ranger Teams from File\n" +
            "17) Export Mailing Addresses to File\n" +
            "18) Quit\n";

    public static void main(String[] args) throws SQLException {

        System.out.println("WELCOME TO THE NATIONAL PARK SERVICE SYSTEM DATABASE");

        final Scanner sc = new Scanner(System.in); // Scanner is used to collect the user input

        String option = ""; // Initialize user option selection as nothing

        while (!option.equals("18")) { // Loop for option selection

            System.out.println(PROMPT); // Print the available options

            option = sc.next(); // Read in the user option selection

            switch (option) {
                case "1" -> InsertNewVisitor(sc); // Query 1: Insert New Visitor
                case "2" -> InsertNewRanger(sc); // Query 2: Insert New Ranger
                case "3" -> InsertNewRangerTeam(sc); // Query 3: Insert New Ranger Team
                case "4" -> InsertNewDonation(sc); // Query 4: Insert New Donation
                case "5" -> InsertNewResearcher(sc); // Query 5: Insert New Researcher
                case "6" -> InsertNewReport(sc); // Query 6: Insert New Report
                case "7" -> InsertNewProgram(sc); // Query 7: Insert New Park Program
                case "8" -> RetrieveEmergencyContacts(sc); // Query 8: Retrieve Emergency Contacts
                case "9" -> RetrieveEnrolledVisitors(sc); // Query 9: Retrieve Enrolled Visitors
                case "10" -> RetrieveParkProgramsAfterDate(sc); // Query 10: Retrieve Park Programs After Date
                case "11" -> RetrieveAnonymousDonationsSummary(sc); // Query 11: Retrieve Anonymous Donations Summary
                case "12" -> RetrieveRangersInTeam(sc); // Query 12: Retrieve Rangers in a Team
                case "13" -> RetrieveMailingList(); // Query 13: Retrieve All Individuals (Mailing List)
                case "14" -> UpdateResearchersSalary(); // Query 14: Update Salary of Researchers
                case "15" -> DeleteInactiveVisitors(); // Query 15: Delete Inactive Visitors
                case "16" -> ImportNewTeamsFromFile(sc); // Query 16: Import New Ranger Teams from File
                case "17" -> ExportMailingAddressesToFile(sc); // Query 17: Export Mailing Addresses to File
                case "18" -> System.out.println("Exiting program."); // Quit
                default -> System.out.println("Invalid option. Please try again.");
            }
        }
        sc.close(); // Close the scanner before exiting the application
    }

// Query 1 – Insert a new visitor and associate them with one or more park programs
    private static void InsertNewVisitor(Scanner sc) {
        try (Connection conn = DriverManager.getConnection(URL)) {

            // Collect user input
            System.out.print("Enter Visitor ID (Person ID): ");
            int personId = sc.nextInt();
            sc.nextLine();
```

Java Source Program (Pt. 2)

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

        System.out.print("Enter program name: ");
        String programName = sc.nextLine();

        System.out.print("Enter visit date (YYYY-MM-DD): ");
        String visitDate = sc.nextLine();

        System.out.print("Enter accessibility needs (or 'None'): ");
        String accessibility = sc.nextLine();

        // Prepare and execute stored procedure
        try (CallableStatement cs = conn.prepareCall("{call InsertNewVisitor(?,?,?,?,?)}")) {
            cs.setInt(1, personId);
            cs.setString(2, parkName);
            cs.setString(3, programName);
            cs.setString(4, visitDate);
            cs.setString(5, accessibility);

            cs.execute();

            System.out.println("Visitor inserted successfully and associated with program");
        }

    } catch (SQLException e) {
        System.out.println("Error inserting visitor: " + e.getMessage());
    }
}

// Query 2: Insert New Ranger
private static void InsertNewRanger(Scanner sc) {
    try (Connection conn = DriverManager.getConnection(URL)) {

        // Collect user input
        System.out.print("Enter Ranger person ID: ");
        int personId = sc.nextInt();
        sc.nextLine();

        System.out.print("Enter team ID: ");
        int teamId = sc.nextInt();
        sc.nextLine();

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

        System.out.print("Enter status (Active/Inactive): ");
        String status = sc.nextLine();

        System.out.print("Enter certifications (comma-separated or leave blank): ");
        String certifications = sc.nextLine();

        // Prepare and execute stored procedure
        try (CallableStatement cs = conn.prepareCall("{call InsertNewRanger(?,?,?,?,?)}")) {
            cs.setInt(1, personId);
            cs.setInt(2, teamId);
            cs.setString(3, startDate);
            cs.setString(4, status);
            cs.setString(5, certifications);

            cs.execute();
            System.out.println("Ranger inserted successfully and assigned to team.");
        }

    } catch (SQLException e) {
        System.out.println("Error inserting ranger: " + e.getMessage());
    }
}

// Query 3: Insert New Ranger Team
private static void InsertNewRangerTeam(Scanner sc) {
    try (Connection conn = DriverManager.getConnection(URL)) {

        // Collect user input
        System.out.print("Enter focus area: ");
        sc.nextLine();
        String focusArea = sc.nextLine();

        System.out.print("Enter formation date (YYYY-MM-DD): ");
        String formationDate = sc.nextLine();

        System.out.print("Enter leader ID: ");
        int leaderId = sc.nextInt();
        sc.nextLine();

        // Prepare and execute stored procedure
        try (CallableStatement cs = conn.prepareCall("{call InsertNewRangerTeam(?,?,?)}")) {
            cs.setString(1, focusArea);
            cs.setString(2, formationDate);
            cs.setInt(3, leaderId);
```

Java Source Program (Pt. 3)

```
            cs.execute();
            System.out.println("Ranger team inserted successfully.");
        }

    } catch (SQLException e) {
        System.out.println("Error inserting ranger team: " + e.getMessage());
    }
}

// Query 4: Insert New Donation
private static void InsertNewDonation(Scanner sc) {
    try (Connection conn = DriverManager.getConnection(URL)) {

        // Collect user input
        System.out.print("Enter donor person ID: ");
        int personId = sc.nextInt();
        sc.nextLine();

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

        System.out.print("Enter amount: ");
        double amount = sc.nextDouble();
        sc.nextLine();

        System.out.print("Enter campaign name: ");
        String campaignName = sc.nextLine();

        System.out.print("Enter check number (or 0 if not applicable): ");
        int checkNumber = sc.nextInt();
        sc.nextLine();

        System.out.print("Enter card type (or 'NULL'): ");
        String cardType = sc.nextLine();

        System.out.print("Enter last four digits (or 0 if not applicable): ");
        int lastFour = sc.nextInt();
        sc.nextLine();

        System.out.print("Enter expiration date (YYYY-MM-DD or leave blank): ");
        String expDate = sc.nextLine();
        if (expDate.isBlank()) expDate = null;
        // Prepare and execute stored procedur
        try (CallableStatement cs = conn.prepareCall("{call InsertNewDonation(?,?,?,?,?,?,?,?)}")) {
            cs.setInt(1, personId);
            cs.setString(2, donationDate);
            cs.setDouble(3, amount);
            cs.setString(4, campaignName);
            cs.setInt(5, checkNumber);
            cs.setString(6, cardType);
            cs.setInt(7, lastFour);
            cs.setString(8, expDate);

            cs.execute();
            System.out.println("Donation inserted successfully.");
        }

    } catch (SQLException e) {
        System.out.println("Error inserting donation: " + e.getMessage());
    }
}

// Query 5: Insert New Researcher
private static void InsertNewResearcher(Scanner sc) {
    try (Connection conn = DriverManager.getConnection(URL)) {

        // Collect users input
        System.out.print("Enter Researcher person ID: ");
        int personId = sc.nextInt();
        sc.nextLine();

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

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

        System.out.print("Enter salary: ");
        int salary = sc.nextInt();
        sc.nextLine();

        System.out.print("Enter associated team IDs (comma-separated): ");
        String teamIds = sc.nextLine();

        // Prepare and execute stored procedure
        try (CallableStatement cs = conn.prepareCall("{call InsertNewResearcher(?,?,?,?,?)}")) {
            cs.setInt(1, personId);
            cs.setString(2, researchField);
            cs.setString(3, hireDate);
            cs.setInt(4, salary);
```

Java Source Program (Pt. 3)

```
            cs.execute();
            System.out.println("Ranger team inserted successfully.");
        }

    } catch (SQLException e) {
        System.out.println("Error inserting ranger team: " + e.getMessage());
    }
}

// Query 4: Insert New Donation
private static void InsertNewDonation(Scanner sc) {
    try (Connection conn = DriverManager.getConnection(URL)) {

        // Collect user input
        System.out.print("Enter donor person ID: ");
        int personId = sc.nextInt();
        sc.nextLine();

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

        System.out.print("Enter amount: ");
        double amount = sc.nextDouble();
        sc.nextLine();

        System.out.print("Enter campaign name: ");
        String campaignName = sc.nextLine();

        System.out.print("Enter check number (or 0 if not applicable): ");
        int checkNumber = sc.nextInt();
        sc.nextLine();

        System.out.print("Enter card type (or 'NULL'): ");
        String cardType = sc.nextLine();

        System.out.print("Enter last four digits (or 0 if not applicable): ");
        int lastFour = sc.nextInt();
        sc.nextLine();

        System.out.print("Enter expiration date (YYYY-MM-DD or leave blank): ");
        String expDate = sc.nextLine();
        if (expDate.isBlank()) expDate = null;
        // Prepare and execute stored procedur
        try (CallableStatement cs = conn.prepareCall("{call InsertNewDonation(?,?,?,?,?,?,?,?)}")) {
            cs.setInt(1, personId);
            cs.setString(2, donationDate);
            cs.setDouble(3, amount);
            cs.setString(4, campaignName);
            cs.setInt(5, checkNumber);
            cs.setString(6, cardType);
            cs.setInt(7, lastFour);
            cs.setString(8, expDate);

            cs.execute();
            System.out.println("Donation inserted successfully.");
        }

    } catch (SQLException e) {
        System.out.println("Error inserting donation: " + e.getMessage());
    }
}

// Query 5: Insert New Researcher
private static void InsertNewResearcher(Scanner sc) {
    try (Connection conn = DriverManager.getConnection(URL)) {

        // Collect users input
        System.out.print("Enter Researcher person ID: ");
        int personId = sc.nextInt();
        sc.nextLine();

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

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

        System.out.print("Enter salary: ");
        int salary = sc.nextInt();
        sc.nextLine();

        System.out.print("Enter associated team IDs (comma-separated): ");
        String teamIds = sc.nextLine();

        // Prepare and execute stored procedure
        try (CallableStatement cs = conn.prepareCall("{call InsertNewResearcher(?,?,?,?,?)}")) {
            cs.setInt(1, personId);
            cs.setString(2, researchField);
            cs.setString(3, hireDate);
            cs.setInt(4, salary);
```

Java Source Program (Pt. 4)

```
            cs.setString(5, teamIds);

            cs.execute();
            System.out.println("Researcher inserted successfully and assigned to teams.");
        }

    } catch (SQLException e) {
        System.out.println("Error inserting researcher: " + e.getMessage());
    }
}

// Query 6: Insert New Report
private static void InsertNewReport(Scanner sc) {
    try (Connection conn = DriverManager.getConnection(URL)) {

        // Collect user input
        System.out.print("Enter researcher ID: ");
        int researcherId = sc.nextInt();
        sc.nextLine();

        System.out.print("Enter ranger team ID: ");
        int teamId = sc.nextInt();
        sc.nextLine();

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

        System.out.print("Enter report summary: ");
        String summary = sc.nextLine();

        // Prepare and execute stored procedure
        try (CallableStatement cs = conn.prepareCall("{call InsertNewReport(?,?,?,?)}")) {
            cs.setInt(1, researcherId);
            cs.setInt(2, teamId);
            cs.setString(3, reportDate);
            cs.setString(4, summary);

            cs.execute();
            System.out.println("Report inserted successfully.");
        }

    } catch (SQLException e) {
        System.out.println("Error inserting report: " + e.getMessage());
    }
}

// Query 7: Insert New Program
private static void InsertNewProgram(Scanner sc) {
    try (Connection conn = DriverManager.getConnection(URL)) {

        // Collect user input
        System.out.print("Enter park name: ");
        sc.nextLine();
        String parkName = sc.nextLine();

        System.out.print("Enter program name: ");
        String programName = sc.nextLine();

        System.out.print("Enter type: ");
        String type = sc.nextLine();

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

        System.out.print("Enter duration in hours: ");
        int duration = sc.nextInt();
        sc.nextLine();

        // Prepare and execute stored procedure
        try (CallableStatement cs = conn.prepareCall("{call InsertNewProgram(?,?,?,?,?)}")) {
            cs.setString(1, parkName);
            cs.setString(2, programName);
            cs.setString(3, type);
            cs.setString(4, startDate);
            cs.setInt(5, duration);

            cs.execute();
            System.out.println("Park program inserted successfully.");
        }

    } catch (SQLException e) {
        System.out.println("Error inserting program: " + e.getMessage());
    }
}

// Query 8: Retrieve Emergency Contacts
private static void RetrieveEmergencyContacts(Scanner sc) {
    try (Connection conn = DriverManager.getConnection(URL)) {
```

Java Source Program (Pt. 5)

```
        // Collect user input
        System.out.print("Enter person ID: ");
        int personId = sc.nextInt();
        sc.nextLine();

        // Prepare and execute stored procedure
        try (CallableStatement cs = conn.prepareCall("{call RetrieveEmergencyContacts(?)}")) {
            cs.setInt(1, personId);

            boolean hasResult = cs.execute();

            // Process result set
            if (hasResult) {
                try (ResultSet rs = cs.getResultSet()) {
                    System.out.println("\n--- Emergency Contacts ---");
                    // Print the header
                    System.out.printf("%-20s %-15s %-15s%n", "Contact Name", "Relationship", "Phone Number");
                    // Print the data
                    while (rs.next()) {
                        String name = rs.getString("Contact Name");
                        String relation = rs.getString("Relationship");
                        String phone = rs.getString("Phone Number");
                        System.out.printf("%-20s %-15s %-15s%n", name, relation, phone);
                    }
                }
            } else {
                System.out.println("No contacts found for this person.");
            }

        }

    } catch (SQLException e) {
        System.out.println("Error retrieving emergency contacts: " + e.getMessage());
    }
}

// Query 9: Retrieve Enrolled Visitors
private static void RetrieveEnrolledVisitors(Scanner sc) {
    try (Connection conn = DriverManager.getConnection(URL)) {

        // Collect user input
        System.out.print("Enter park name: ");
        sc.nextLine();
        String parkName = sc.nextLine();

        System.out.print("Enter program name: ");
        String programName = sc.nextLine();

        // Prepare and execute stored procedure
        try (CallableStatement cs = conn.prepareCall("{call RetrieveEnrolledVisitors(?,?)}")) {
            cs.setString(1, parkName);
            cs.setString(2, programName);

            boolean hasResult = cs.execute();

            // Process result set
            if (hasResult) {
                try (ResultSet rs = cs.getResultSet()) {
                    System.out.println("\n--- Enrolled Visitors ---");
                    // Print the header
                    System.out.printf("%-10s %-25s %-15s %-25s%n", "Visitor ID", "Visitor Name", "Visit Date", "Accessibility Needs");
                    // Print the data
                    while (rs.next()) {
                        int id = rs.getInt("Visitor ID");
                        String name = rs.getString("Visitor Name");
                        String date = rs.getString("Visit Date");
                        String access = rs.getString("Accessibility Needs");
                        System.out.printf("%-10d %-25s %-15s %-25s%n", id, name, date, access);
                    }
                }
            } else {
                System.out.println("No enrolled visitors found for the specified program.");
            }

        }

    } catch (SQLException e) {
        System.out.println("Error retrieving enrolled visitors: " + e.getMessage());
    }
}

// Query 10: Retrieve Park Programs after Date
private static void RetrieveParkProgramsAfterDate(Scanner sc) {
    try (Connection conn = DriverManager.getConnection(URL)) {

        // Collect user input
        System.out.print("Enter park name: ");
        sc.nextLine();
        String parkName = sc.nextLine();
```

Java Source Program (Pt. 6)

```
        System.out.print("Enter comparison date (YYYY-MM-DD): ");
        String date = sc.nextLine();

        // Prepare and execute stored procedure
        try (CallableStatement cs = conn.prepareCall("{call RetrieveParkProgramsAfterDate(?,?)}")) {
            cs.setString(1, parkName);
            cs.setString(2, date);

            boolean hasResult = cs.execute();

            // Process result set
            if (hasResult) {
                try (ResultSet rs = cs.getResultSet()) {
                    System.out.println("\n--- Park Programs After " + date + " ---");
                    // Print the header
                    System.out.printf("%-25s %-20s %-15s %-15s%n",  "Program Name", "Program Type", "Start Date", "Duration (Hours)");
                    // Print the data
                    while (rs.next()) {
                        String progName = rs.getString("Program Name");
                        String type = rs.getString("Program Type");
                        String start = rs.getString("Start Date");
                        int duration = rs.getInt("Duration (Hours)");
                        System.out.printf("%-25s %-20s %-15s %-15d%n", progName, type, start, duration);
                    }
                }
            } else {
                System.out.println("No programs found that start after the given date.");
            }

        }

    } catch (SQLException e) {
        System.out.println("Error retrieving park programs: " + e.getMessage());
    }
}

// Query 11: Retrieve Anonymous Donations Summary
private static void RetrieveAnonymousDonationsSummary(Scanner sc) {
    try (Connection conn = DriverManager.getConnection(URL)) {

        // Collect user input
        System.out.print("Enter month (1-12): ");
        int month = sc.nextInt();

        System.out.print("Enter year (e.g., 2025): ");
        int year = sc.nextInt();
        sc.nextLine();

        // Prepare and execute stored procedure
        try (CallableStatement cs = conn.prepareCall("{call RetrieveAnonymousDonationsSummary(?,?)}")) {
            cs.setInt(1, month);
            cs.setInt(2, year);

            boolean hasResult = cs.execute();

            // Process result set
            if (hasResult) {
                try (ResultSet rs = cs.getResultSet()) {
                    System.out.println("\n--- Anonymous Donations Summary ---");
                    // Print header
                    System.out.printf("%-15s %-15s %-20s %-20s %-15s%n", "Year", "Month", "Total Donation", "Average Donation", "Donations Count");
                    // Print all data
                    while (rs.next()) {
                        int dYear = rs.getInt("Donation Year");
                        int dMonth = rs.getInt("Donation Month");
                        double total = rs.getDouble("Total Donation");
                        double avg = rs.getDouble("Average Donation");
                        int count = rs.getInt("Number of Donations");
                        System.out.printf("%-15d %-15d %-20.2f %-20.2f %-15d%n", dYear, dMonth, total, avg, count);
                    }
                }
            } else {
                System.out.println("No anonymous donations found for the given month and year.");
            }

        }

    } catch (SQLException e) {
        System.out.println("Error retrieving anonymous donation summary: " + e.getMessage());
    }
}

// Query 12: Retrieve Rangers in Team
private static void RetrieveRangersInTeam(Scanner sc) {
    try (Connection conn = DriverManager.getConnection(URL)) {

        // Collect user input
        System.out.print("Enter team ID: ");
        int teamId = sc.nextInt();
        sc.nextLine();
```

Java Source Program (Pt. 7)

```
        // Prepare and execute stored procedure
        try (CallableStatement cs = conn.prepareCall("{call RetrieveRangersInTeam(?)}")) {
            cs.setInt(1, teamId);

            boolean hasResult = cs.execute();

            if (hasResult) {
                try (ResultSet rs = cs.getResultSet()) {
                    System.out.println("\n--- Rangers in Team " + teamId + " ---");
                    // Print the header
                    System.out.printf("%-10s %-25s %-25s %-20s %-10s%n", "Ranger ID", "Ranger Name", "Certification", "Years of Service", "Role");
                    // Print the data
                    while (rs.next()) {
                        int rangerId = rs.getInt("Ranger ID");
                        String rangerName = rs.getString("Ranger Name");
                        String cert = rs.getString("Certification");
                        int years = rs.getInt("Years of Service");
                        String role = rs.getString("Role");
                        System.out.printf("%-10d %-25s %-25s %-20d %-10s%n",
                                rangerId, rangerName, cert, years, role);
                    }
                }
            } else {
                System.out.println("No rangers found for the given team.");
            }

        }

    } catch (SQLException e) {
        System.out.println("Error retrieving rangers: " + e.getMessage());
    }
}

// Query 13: Retrieve all Individuals
private static void RetrieveMailingList() {
    try (Connection conn = DriverManager.getConnection(URL);
         CallableStatement cs = conn.prepareCall("{call RetrieveAllIndividuals()}")) {
        // Execute stored procedure
        boolean hasResult = cs.execute();
        // Process result set
        if (hasResult) {
            try (ResultSet rs = cs.getResultSet()) {
                System.out.println("\n--- Mailing List ---");
                // Print the header
                System.out.printf("%-10s %-25s %-20s %-30s %-15s %-15s%n", "Person ID", "Full Name", "Phone", "Email", "City", "State");
                // Print the data
                while (rs.next()) {
                    int id = rs.getInt("Person ID");
                    String name = rs.getString("Full Name");
                    String phone = rs.getString("Phone Number");
                    String email = rs.getString("Email Address");
                    String city = rs.getString("City");
                    String state = rs.getString("State");
                    // Format and print each record
                    System.out.printf("%-10d %-25s %-20s %-30s %-15s %-15s%n",
                            id, name, phone, email, city, state);
                }
            }
        } else {
            System.out.println("No individuals found in the database.");
        }

    } catch (SQLException e) {
        System.out.println("Error retrieving mailing list: " + e.getMessage());
    }
}


// Query 14: Update researchers salary
    private static void UpdateResearchersSalary() {
    try (Connection conn = DriverManager.getConnection(URL);
         CallableStatement cs = conn.prepareCall("{call UpdateResearchersSalary()}")) {
        // Execute stored procedure
        boolean hasResult = cs.execute();
        // Process result set
        if (hasResult) {
            try (ResultSet rs = cs.getResultSet()) {
                System.out.println("\n--- Researchers With Updated Salaries ---");
                System.out.printf("%-15s %-25s %-15s%n", "Researcher ID", "Researcher Name", "New Salary");
                // Print the data
                while (rs.next()) {
                    int id = rs.getInt("Researcher ID");
                    String name = rs.getString("Researcher Name");
                    double salary = rs.getDouble("New Salary");
                    System.out.printf("%-15d %-25s %-15.2f%n", id, name, salary);
                }
            }
        } else {
            System.out.println("No researchers met the criteria for a salary update.");
        }
```

Java Source Program (Pt. 8)

```
    } catch (SQLException e) {
        System.out.println("Error updating researcher salaries: " + e.getMessage());
    }
}


// Query 15: Delete inactive visitors
private static void DeleteInactiveVisitors() {
    try (Connection conn = DriverManager.getConnection(URL);
         CallableStatement cs = conn.prepareCall("{call DeleteInactiveVisitors()}")) {
        // Execute stored procedure
        boolean hasResult = cs.execute();
        // Process result set
        if (hasResult) {
            try (ResultSet rs = cs.getResultSet()) {
                System.out.println("\n--- Deletion Summary ---");
                while (rs.next()) {
                    String msg = rs.getString("ResultMessage");
                    System.out.println(msg);
                }
            }
        } else {
            System.out.println("No inactive visitors were deleted.");
        }

    } catch (SQLException e) {
        System.out.println("Error deleting inactive visitors: " + e.getMessage());
    }
}

// Case 16:
private static void ImportNewTeamsFromFile(Scanner sc) {
    System.out.print("Enter input file name (e.g., teams.txt): ");
    sc.nextLine(); // clear buffer
    String fileName = sc.nextLine();
    // Read from file and insert teams
    try (Connection conn = DriverManager.getConnection(URL);
         BufferedReader reader = new BufferedReader(new FileReader(fileName))) {

        String line;
        int count = 0;

        System.out.println("\nImporting new ranger teams...\n");
        // Read each line from the file
        while ((line = reader.readLine()) != null) {
            String[] parts = line.split(",");
            if (parts.length != 3) { // Each line must have exactly 3 parts
                System.out.println("Skipping invalid line: " + line);
                continue;
            }
            // Parse team details
            String focusArea = parts[0].trim();
            String formationDate = parts[1].trim();
            int leaderId;
            // Validate leader ID
            try {
                leaderId = Integer.parseInt(parts[2].trim());
            } catch (NumberFormatException e) {
                System.out.println("Invalid leader ID in line: " + line);
                continue;
            }
            // Call stored procedure to insert the team
            try (CallableStatement cs = conn.prepareCall("{call InsertNewRangerTeam(?,?,?)}")) {
                cs.setString(1, focusArea);
                cs.setString(2, formationDate);
                cs.setInt(3, leaderId);
                cs.execute();
                count++;
            } catch (SQLException e) {
                System.out.println("Failed to insert team: " + e.getMessage());
            }
        }

        System.out.println("\nImport complete. " + count + " teams inserted.");

    } catch (IOException e) {
        System.out.println("Error reading file: " + e.getMessage());
    } catch (SQLException e) {
        System.out.println("Database error during import: " + e.getMessage());
    }
}

private static void ExportMailingAddressesToFile(Scanner sc) {
    System.out.print("Enter output file name (e.g., output.txt): ");
    sc.nextLine(); // clear buffer
    String filePath = sc.nextLine();
    // Export mailing list to file
    try (Connection conn = DriverManager.getConnection(URL);
         CallableStatement cs = conn.prepareCall("{call RetrieveAllIndividuals()}");
         BufferedWriter writer = new BufferedWriter(new FileWriter(filePath))) {
```

Java Source Program (Pt. 9)

```
        // Execute stored procedure
        boolean hasResult = cs.execute();

        if (hasResult) {
            try (ResultSet rs = cs.getResultSet()) {
                writer.write("Person ID,Full Name,Phone,Email,City,State\n"); // header
                // Write each record to the file
                while (rs.next()) {
                    int id = rs.getInt("Person ID");
                    String name = rs.getString("Full Name");
                    String phone = rs.getString("Phone Number");
                    String email = rs.getString("Email Address");
                    String city = rs.getString("City");
                    String state = rs.getString("State");
                    // Format and write each record
                    writer.write(String.format("%d,%s,%s,%s,%s,%s\n",
                            id, name, phone, email, city, state));
                }
                System.out.println("Mailing list successfully exported to: " + filePath);
            }
        } else {
            System.out.println("No mailing list data found to export.");
        }

    } catch (SQLException e) {
        System.out.println("Database error while exporting mailing list: " + e.getMessage());
    } catch (IOException e) {
        System.out.println("File writing error: " + e.getMessage());
    }
}
```

Java Source Program Compilation

Gray_Yale_IP_Task5b.java

```java
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.Statement;
import java.util.Scanner;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.DriverManager;

public class Gray_Yale_IP_Task5b {
    // Database credentials
    final static String HOSTNAME = "gray0188.database.windows.net";
    final static String DBNAME = "cs-dsa-4513-sql-db";
    final static String USERNAME = "gray0188";
    final static String PASSWORD = "OkcThunder2025";
    // Database connection string
    final static String URL =
        String.format("jdbc:sqlserver://%s:1433;database=%s;user=%s;password=%s;encrypt=true;trustServerCertificate=false;hostNameInCertificate=*.d
            HOSTNAME, DBNAME, USERNAME, PASSWORD);
    // User input prompt
    final static String PROMPT =
            "1) Insert New Visitor and Associate with Park Program\n" +
            "2) Insert New Ranger and Assign to Team\n" +
```

Problems | Javadoc | Declaration | Console

Gray_Yale_IP_Task5b [Java Application] /Applications/Eclipse.app/Contents/Eclipse/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx.aarch64_21.0.8.v20250724-1412/jre/bin/java (Nov 11, 2025, 8:35:09 AM elap

```
WELCOME TO THE NATIONAL PARK SERVICE SYSTEM DATABASE
1) Insert New Visitor and Associate with Park Program
2) Insert New Ranger and Assign to Team
3) Insert New Ranger Team and Set Leader
4) Insert New Donation from Donor
5) Insert New Researcher and Associate with Teams
6) Insert New Report from Ranger Team to Researcher
7) Insert New Park Program for a Park
8) Retrieve Emergency Contacts for a Person
9) Retrieve Visitors Enrolled in a Specific Park Program
10) Retrieve Park Programs Starting After a Given Date
11) Retrieve Total and Average Anonymous Donations
12) Retrieve Rangers in a Team with Certifications and Roles
13) Retrieve All Individuals (Mailing List)
14) Update Salary of Researchers Overseeing Multiple Teams
15) Delete Inactive Visitors with Expired Park Passes
16) Import New Ranger Teams from File
17) Export Mailing Addresses to File
18) Quit
```

Testing for **Query #1**

Before Testing

Query 1

Run | Cancel query | Save query | Export data as | Show only Editor

```
1  SELECT * FROM Visitor;
2  SELECT * FROM Enrolled_in;
```

Results | Messages

```
Query succeeded: Affected rows: 0Affected rows: 0
```

After Testing

Query 1

Run | Cancel query | Save query | Export data as | Show only Editor

```
1  SELECT * FROM Visitor;
2  SELECT * FROM Enrolled_in ORDER BY person_id;
```

Results | Messages

person_id	park_name	program_name	visit_date	accessibility_needs
5	Yellowstone	Conservation 101	2025-06-15	None
6	Yellowstone	Conservation 101	2025-07-01	Wheelchair Access
7	Yellowstone	Wildlife Watch	2025-08-12	Hearing Assistance
8	Yosemite	Nature Hike	2025-09-05	Service Animal
9	Yosemite	Wildfire Preparedness Tr...	2026-01-15	Visual Aid Required

```
18) Quit

1
Enter Visitor ID (Person ID): 5
Enter park name: Yellowstone
Enter program name: Conservation 101
Enter visit date (YYYY-MM-DD): 2025-06-15
Enter accessibility needs (or 'None'): None
Visitor inserted successfully and associated with program
```

```
18) Quit

1
Enter Visitor ID (Person ID): 6
Enter park name: Yellowstone
Enter program name: Conservation 101
Enter visit date (YYYY-MM-DD): 2025-07-01
Enter accessibility needs (or 'None'): Wheelchair Access
Visitor inserted successfully and associated with program
```

```
18) Quit

1
Enter Visitor ID (Person ID): 7
Enter park name: Yellowstone
Enter program name: Wildlife Watch
Enter visit date (YYYY-MM-DD): 2025-08-12
Enter accessibility needs (or 'None'): Hearing Assistance
Visitor inserted successfully and associated with program
```

```
18) Quit

1
Enter Visitor ID (Person ID): 8
Enter park name: Yosemite
Enter program name: Nature Hike
Enter visit date (YYYY-MM-DD): 2025-09-05
Enter accessibility needs (or 'None'): Service Animal
Visitor inserted successfully and associated with program
```

```
18) Quit

1
Enter Visitor ID (Person ID): 9
Enter park name: Yosemite
Enter program name: Wildfire Preparedness Training
Enter visit date (YYYY-MM-DD): 2026-01-15
Enter accessibility needs (or 'None'): Visual Aid Required
Visitor inserted successfully and associated with program
```

Testing for **Query #2**

Before Testing

Query 1

Run | Cancel query | Save query | Export data as | Show only Editor

```
1  SELECT * FROM Ranger;
2  SELECT * FROM Member_of ORDER BY team_id;
```

Results | Messages

```
Query succeeded: Affected rows: 6Affected rows: 0
```

After Testing

Query 1

Save query | Export data as | Show only Editor

```
1  SELECT * FROM Ranger;
2  SELECT * FROM Member_of ORDER BY team_id;
```

Results | Messages

person_id	team_id	start_date	status	years_of_service	is_leader
10	19	2023-06-20	Active	0	False
11	20	2023-07-10	Active	0	False
12	21	2023-08-05	Active	0	False
13	22	2023-09-01	Active	0	False
4	23	2023-10-10	Active	0	False

```
18) Quit

2
Enter Ranger person ID: 10
Enter team ID: 19
Enter start date (YYYY-MM-DD): 2023-06-20
Enter status (Active/Inactive): Active
Enter certifications (comma-separated or leave blank): Fire Safety
Ranger inserted successfully and assigned to team.
```

```
18) Quit

2
Enter Ranger person ID: 11
Enter team ID: 20
Enter start date (YYYY-MM-DD): 2023-07-10
Enter status (Active/Inactive): Active
Enter certifications (comma-separated or leave blank): Fire Safety
Ranger inserted successfully and assigned to team.
```

```
18) Quit

2
Enter Ranger person ID: 12
Enter team ID: 21
Enter start date (YYYY-MM-DD): 2023-08-05
Enter status (Active/Inactive): Active
Enter certifications (comma-separated or leave blank): Rescue Ops
Ranger inserted successfully and assigned to team.
```

```
18) Quit

2
Enter Ranger person ID: 13
Enter team ID: 22
Enter start date (YYYY-MM-DD): 2023-09-01
Enter status (Active/Inactive): Active
Enter certifications (comma-separated or leave blank): Trail Maintenance
Ranger inserted successfully and assigned to team.
```

```
18) Quit

2
Enter Ranger person ID: 4
Enter team ID: 23
Enter start date (YYYY-MM-DD): 2023-10-10
Enter status (Active/Inactive): Active
Enter certifications (comma-separated or leave blank): Wilderness Survival
Ranger inserted successfully and assigned to team.
```

Testing for **Query #3**

Before Testing

Query 1

Run | Cancel query | Save query | Export data as | Show only Editor

```
1  SELECT * FROM Ranger_Team;
2  SELECT * FROM Member_of ORDER BY team_id;
```

Results | Messages

person_id	team_id	start_date	status	years_of_service	is_leader
10	19	2023-06-20	Active	0	False
11	20	2023-07-10	Active	0	False
12	21	2023-08-05	Active	0	False
13	22	2023-09-01	Active	0	False
4	23	2023-10-10	Active	0	False

After Testing

Query 1

Run | Cancel query | Save query | Export data as | Show only Editor

```
1  SELECT
2      rt.team_id,
3      rt.focus_area,
4      rt.formation_date,
5      m.person_id AS leader_id,
6      m.is_leader
7  FROM Ranger_Team rt
8  LEFT JOIN Member_of m ON rt.team_id = m.team_id
9  ORDER BY rt.team_id;
```

Results | Messages

24	Wildlife Research	2024-01-10	11	True
25	Trail Maintenance	2024-02-12	12	True
26	Flood Response	2024-03-25	13	True
27	Habitat Restoration	2024-04-18		
28	Search & Rescue	2024-05-22	14	True
29	Wilderness Fire Response	2024-06-30	15	True

```
18) Quit

3
Enter focus area: Trail Maintenance
Enter formation date (YYYY-MM-DD): 2024-02-12
Enter leader ID: 11
Ranger team inserted successfully.
```

```
18) Quit

3
Enter focus area: Flood Response
Enter formation date (YYYY-MM-DD): 2024-03-25
Enter leader ID: 12
Ranger team inserted successfully.
```

```
18) Quit

3
Enter focus area: Habitat Restoration
Enter formation date (YYYY-MM-DD): 2024-04-18
Enter leader ID: 13
Ranger team inserted successfully.
```

```
18) Quit

3
Enter focus area: Search & Rescue
Enter formation date (YYYY-MM-DD): 2024-05-22
Enter leader ID: 14
Ranger team inserted successfully.
```

```
18) Quit

3
Enter focus area: Wilderness Fire Response
Enter formation date (YYYY-MM-DD): 2024-06-30
Enter leader ID: 15
Ranger team inserted successfully.
```

Testing for **Query #4**

Before Testing

Query 1

Run | Cancel query | Save query | Export data as | Show only Editor

```
1   DELETE FROM Check_Donation;
2   DELETE FROM Card_Donation;
3   DELETE FROM Donation;
```

Results | Messages

```
Query succeeded: Affected rows: 2
```

After Testing

Query 1

Run | Cancel query | Save query | Export data as | Show only Editor

```
1   SELECT *
2   FROM Check_Donation
3   WHERE check_number <> 0
4   ORDER BY check_number;
```

Results | Messages

Search to filter items...

check_number	person_id	donation_date	amount
101	2	2025-02-01	100
102	2	2025-02-03	75
104	2	2025-02-04	120
105	2	2025-02-05	300

Query 1

Run | Cancel query | Save query | Export data as | Show only Editor

```
1   SELECT * FROM Card_Donation
```

Results | Messages

Search to filter items...

card_type	last_four_digits	expiration_date	person_id	donation_date	amount
MasterCard	9876	2028-09-01	2	2025-03-10	300

```
18) Quit

4
Enter donor person ID: 2
Enter donation date (YYYY-MM-DD): 2025-02-01
Enter amount: 100
Enter campaign name: Wildlife Fund
Enter check number (or 0 if not applicable): 101
Enter card type (or 'NULL'): NULL
Enter last four digits (or 0 if not applicable): 0
Enter expiration date (YYYY-MM-DD or leave blank):
Donation inserted successfully.
```

```
18) Quit

4
Enter donor person ID: 2
Enter donation date (YYYY-MM-DD): 2025-03-10
Enter amount: 300
Enter campaign name: Wildlife Preservation
Enter check number (or 0 if not applicable): 0
Enter card type (or 'NULL'): MasterCard
Enter last four digits (or 0 if not applicable): 9876
Enter expiration date (YYYY-MM-DD or leave blank): 2028-09-01
```

```
18) Quit

4
Enter donor person ID: 2
Enter donation date (YYYY-MM-DD): 2025-02-03
Enter amount: 75
Enter campaign name: Fire Recovery
Enter check number (or 0 if not applicable): 102
Enter card type (or 'NULL'): NULL
Enter last four digits (or 0 if not applicable): 0
Enter expiration date (YYYY-MM-DD or leave blank):
Donation inserted successfully.
```

```
18) Quit

4
Enter donor person ID: 2
Enter donation date (YYYY-MM-DD): 2025-02-04
Enter amount: 120
Enter campaign name: Wildlife Rescue
Enter check number (or 0 if not applicable): 104
Enter card type (or 'NULL'): NULL
Enter last four digits (or 0 if not applicable): 0
Enter expiration date (YYYY-MM-DD or leave blank):
Donation inserted successfully.
```

```
18) Quit

4
Enter donor person ID: 2
Enter donation date (YYYY-MM-DD): 2025-02-05
Enter amount: 300
Enter campaign name: Habitat Restoration
Enter check number (or 0 if not applicable): 105
Enter card type (or 'NULL'): NULL
Enter last four digits (or 0 if not applicable): 0
Enter expiration date (YYYY-MM-DD or leave blank):
Donation inserted successfully.
```

Testing for **Query #5**

Before Testing

Query 1

Run | Cancel query | Save query | Export data as | Show only Editor

```
1  SELECT * FROM Researcher ORDER BY person_id;
2  SELECT * FROM Oversees ORDER BY team_id;
```

Results | Messages

```
Query succeeded: Affected rows: 2Affected rows: 0
```

After Testing

Query 1

Run | Cancel query | Save query | Export data as | Show only Editor

```
1  SELECT * FROM Researcher ORDER BY person_id;
2  SELECT * FROM Oversees ORDER BY team_id;
```

Results | Messages

person_id	team_id	report_date	report_summary
16	24	2021-04-10	Initial oversight report pending
17	25	2022-02-25	Initial oversight report pending
18	26	2020-06-30	Initial oversight report pending
19	27	2019-11-05	Initial oversight report pending
20	28	2023-01-15	Initial oversight report pending

```
18) Quit

5
Enter Researcher person ID: 16
Enter research field: Ecology
Enter hire date (YYYY-MM-DD): 2021-04-10
Enter salary: 72000
Enter associated team IDs (comma-separated): 24
Researcher inserted successfully and assigned to teams.
```

```
18) Quit

5
Enter Researcher person ID: 17
Enter research field: Hydrology
Enter hire date (YYYY-MM-DD): 2022-02-25
Enter salary: 68000
Enter associated team IDs (comma-separated): 25
Researcher inserted successfully and assigned to teams.
```

```
18) Quit

5
Enter Researcher person ID: 18
Enter research field: Forestry
Enter hire date (YYYY-MM-DD): 2020-06-30
Enter salary: 75500
Enter associated team IDs (comma-separated): 26
Researcher inserted successfully and assigned to teams.
```

```
18) Quit

5
Enter Researcher person ID: 19
Enter research field: Wildlife Biology
Enter hire date (YYYY-MM-DD): 2019-11-05
Enter salary: 81000
Enter associated team IDs (comma-separated): 27
Researcher inserted successfully and assigned to teams.
```

```
18) Quit

5
Enter Researcher person ID: 20
Enter research field: Environmental Science
Enter hire date (YYYY-MM-DD): 2023-01-15
Enter salary: 69000
Enter associated team IDs (comma-separated): 28
Researcher inserted successfully and assigned to teams.
```

Testing for **Query #6**

Before Testing

Query 1

Run | Cancel query | Save query | Export data as | Show only Editor

```
1  SELECT * FROM Oversees ORDER BY report_date;
```

Results | Messages

person_id	team_id	report_date	report_summary
19	27	2019-11-05	Initial oversight report pending
18	26	2020-06-30	Initial oversight report pending
16	24	2021-04-10	Initial oversight report pending
17	25	2022-02-25	Initial oversight report pending
20	28	2023-01-15	Initial oversight report pending

After Testing

Query 1

Run | Cancel query | Save query | Export data as | Show only Editor

```
1  SELECT * FROM Oversees ORDER BY report_date;
```

Results | Messages

20	28	2023-01-15	Initial oversight report pending
17	25	2024-06-01	Water quality metrics reviewed
18	26	2024-07-22	Trail maintenance and erosion
19	27	2024-08-10	Habitat restoration efforts incr
20	28	2024-09-05	Search & rescue readiness aud
16	24	2025-05-15	Wildlife census data collected

```
18) Quit

6
Enter researcher ID: 16
Enter ranger team ID: 24
Enter report date (YYYY-MM-DD): 2025-05-15
Enter report summary: Wildlife census data collected and analyzed
Report inserted successfully.
```

```
18) Quit

6
Enter researcher ID: 17
Enter ranger team ID: 25
Enter report date (YYYY-MM-DD): 2024-06-01
Enter report summary: Water quality metrics reviewed; samples stable.
Report inserted successfully.
```

```
18) Quit

6
Enter researcher ID: 18
Enter ranger team ID: 26
Enter report date (YYYY-MM-DD): 2024-07-22
Enter report summary: Trail maintenance and erosion prevention successful.
Report inserted successfully.
```

```
18) Quit

6
Enter researcher ID: 19
Enter ranger team ID: 27
Enter report date (YYYY-MM-DD): 2024-08-10
Enter report summary: Habitat restoration efforts increased biodiversity.
Report inserted successfully.
```

```
18) Quit

6
Enter researcher ID: 20
Enter ranger team ID: 28
Enter report date (YYYY-MM-DD): 2024-09-05
Enter report summary: Search & rescue readiness audit completed.
Report inserted successfully.
```

Testing for **Query #7**

Before Testing

Query 1

Run | Cancel query | Save query | Export data as | Show only Editor

```
1  SELECT * FROM Program ORDER BY park_name, start_date;
```

Results | Messages

park_name	program_name	type	start_date	duration_hours
Grand Canyon	Trail Safety	Training	2025-03-15	2
Yellowstone	Wildlife Watch	Education	2025-03-12	3
Yellowstone	Conservation 101	Workshop	2025-04-01	4
Yosemite	Nature Hike	Tour	2025-05-20	5
Yosemite	Wildfire Preparedness Tr...	Safety Workshop	2026-07-10	3

After Testing

Query 1

Run | Cancel query | Save query | Export data as | Show only Editor

```
1  SELECT * FROM Program ORDER BY start_date;
```

Results | Messages

Yosemite	Wildfire Preparedness Tr...	Safety Workshop	2026-07-10	3
Grand Canyon	Desert Ecology Workshop	Education	2026-08-05	3
Yellowstone	Fire Safety Awareness	Safety	2026-09-12	2
Yosemite	Junior Ranger Program	Training	2026-10-20	4
Grand Canyon	Night Sky Exploration	Tour	2026-11-15	3
Yellowstone	Winter Wildlife Study	Research	2026-12-01	5

```
18) Quit

7
Enter park name: Grand Canyon
Enter program name: Desert Ecology Workshop
Enter type: Education
Enter start date (YYYY-MM-DD): 2026-08-05
Enter duration in hours: 3
Park program inserted successfully.
```

```
18) Quit

7
Enter park name: Yellowstone
Enter program name: Fire Safety Awareness
Enter type: Safety
Enter start date (YYYY-MM-DD): 2026-09-12
Enter duration in hours: 2
Park program inserted successfully.
```

```
18) Quit

7
Enter park name: Yosemite
Enter program name: Junior Ranger Program
Enter type: Training
Enter start date (YYYY-MM-DD): 2026-10-20
Enter duration in hours: 4
Park program inserted successfully.
```

```
18) Quit

7
Enter park name: Grand Canyon
Enter program name: Night Sky Exploration
Enter type: Tour
Enter start date (YYYY-MM-DD): 2026-11-15
Enter duration in hours: 3
Park program inserted successfully.
```

```
18) Quit

7
Enter park name: Yellowstone
Enter program name: Winter Wildlife Study
Enter type: Research
Enter start date (YYYY-MM-DD): 2026-12-01
Enter duration in hours: 5
Park program inserted successfully.
```

Testing for **Query #8**

Query 1

Run | Cancel query | Save query | Export data as | Show only Editor

```
    ec.person_id,
    p.firstname,
    p.lastname,
    ec.contact_name,
    ec.relationship,
    ec.phone_number
FROM Emergency_Contact ec
JOIN Person p
    ON ec.person_id = p.person_id
ORDER BY ec.person_id, ec.contact_name;
```

Results | Messages

Search to filter items...

person_id	firstname	lastname	contact_name	relationship	phone_numb
2	Ava	Smith	Ben Smith	Brother	405-555-888
3	Liam	Johnson	Kate Johnson	Wife	405-555-999
4	Olivia	Brown	Ethan Brown	Friend	405-555-666
5	Noah	Davis	Lily Davis	Spouse	405-555-444

```
18) Quit

8
Enter person ID: 2

--- Emergency Contacts ---
Contact Name         Relationship    Phone Number
Ben Smith            Brother         405-555-8888
```

```
18) Quit

8
Enter person ID: 3

--- Emergency Contacts ---
Contact Name         Relationship    Phone Number
Kate Johnson         Wife            405-555-9999
```

Testing for **Query #9**

Query 1

Run | Cancel query | Save query | Export data as | Show only Editor

```
1   SELECT
2       e.person_id,
3       p.firstname,
4       p.lastname,
5       e.park_name,
6       e.program_name,
7       e.visit_date,
8       e.accessibility_needs
9   FROM Enrolled_in e
10  JOIN Person p
```

Results | Messages

person_id	firstname	lastname	park_name	program_name	visit_date
5	Noah	Davis	Yellowstone	Conservation 101	2025-06-15
6	Zoe	Miller	Yellowstone	Conservation 101	2025-07-01
7	Ethan	Moore	Yellowstone	Wildlife Watch	2025-08-12
8	Sophia	Lopez	Yosemite	Nature Hike	2025-09-05
9	Ethan	Green	Yosemite	Wildfire Preparednes...	2026-01-15

```
18) Quit

9
Enter park name: Yellowstone
Enter program name: Conservation 101

--- Enrolled Visitors ---
Visitor ID Visitor Name              Visit Date      Accessibility Needs
5          Noah Davis                2025-06-15      None
6          Zoe Miller                2025-07-01      Wheelchair Access
```

```
18) Quit

9
Enter park name: Yosemite
Enter program name: Nature Hike

--- Enrolled Visitors ---
Visitor ID Visitor Name              Visit Date      Accessibility Needs
8          Sophia Lopez              2025-09-05      Service Animal
```

Testing for **Query #10**

Query 1 ×

dbo.RetrieveParkProgramsAfterDate

Run | Cancel query | Save query | Export data as | Show only Editor

```
1  SELECT
2      park_name,
3      program_name,
4      type,
5      start_date,
6      duration_hours
7  FROM Program
8  ORDER BY park_name, start_date;
```

Results | Messages

park_name	program_name	type	start_date	duration_hours
Grand Canyon	Trail Safety	Training	2025-03-15	2
Grand Canyon	Desert Ecology Workshop	Education	2026-08-05	3
Grand Canyon	Night Sky Exploration	Tour	2026-11-15	3
Yellowstone	Wildlife Watch	Education	2025-03-12	3
Yellowstone	Conservation 101	Workshop	2025-04-01	4
Yellowstone	Fire Safety Awareness	Safety	2026-09-12	2
Yellowstone	Winter Wildlife Study	Research	2026-12-01	5
Yosemite	Nature Hike	Tour	2025-05-20	5
Yosemite	Wildfire Preparedness Tr...	Safety Workshop	2026-07-10	3
Yosemite	Junior Ranger Program	Training	2026-10-20	4

```
18) Quit

10
Enter park name: Yellowstone
Enter comparison date (YYYY-MM-DD): 2025-03-15

--- Park Programs After 2025-03-15 ---
Program Name              Program Type         Start Date      Duration (Hours)
Conservation 101          Workshop             2025-04-01      4
Fire Safety Awareness     Safety               2026-09-12      2
Winter Wildlife Study     Research             2026-12-01      5
```

```
18) Quit

10
Enter park name: Yosemite
Enter comparison date (YYYY-MM-DD): 2025-05-01

--- Park Programs After 2025-05-01 ---
Program Name              Program Type         Start Date      Duration (Hours)
Nature Hike               Tour                 2025-05-20      5
Wildfire Preparedness Training Safety Workshop      2026-07-10      3
Junior Ranger Program     Training             2026-10-20      4
```

Testing for **Query #11**

Query 1

Run | Cancel query | Save query | Export data as | Show only Editor

```
    p.firstname,
    p.lastname,
    d.donation_date,
    d.amount,
    d.campaign_name,
    dn.anonymity_preference
FROM Donation d
JOIN Donor dn ON d.person_id = dn.person_id
JOIN Person p ON dn.person_id = p.person_id
WHERE MONTH(donation_date) = 2 AND YEAR(donation_date) = 2025;
```

Results | Messages

person_id	firstname	lastname	donation_date	amount	campaign_name	anonymity_preference
2	Ava	Smith	2025-02-01	100	Wildlife Fund	True
2	Ava	Smith	2025-02-03	75	Fire Recovery	True
2	Ava	Smith	2025-02-04	120	Wildlife Rescue	True
2	Ava	Smith	2025-02-04	150	Visitor Safety	True
2	Ava	Smith	2025-02-05	300	Habitat Restoration	True

```
18) Quit

11
Enter month (1-12): 2
Enter year (e.g., 2025): 2025

--- Anonymous Donations Summary ---
Year            Month           Total Donation      Average Donation    Donations Count
2025            2               745.00              149.00              5
```

Testing for **Query #12**

Query 1

Run | Cancel query | Save query | Export data as | Show only Editor

```
    p.firstname,
    p.lastname,
    m.years_of_service,
    CASE WHEN m.is_leader = 1 THEN 'Leader' ELSE 'Member' END AS role,
    c.certification_name
FROM Member_of m
JOIN Ranger r ON m.person_id = r.person_id
JOIN Person p ON r.person_id = p.person_id
LEFT JOIN Certification c ON r.person_id = c.person_id
WHERE m.team_id = 24;  -- or whichever team you're testing
```

Results | Messages

Search to filter items...

team_id	person_id	firstname	lastname	years_of_service	role	certification_name
24	11	Test	Visitor	0	Leader	Fire Safety

```
18) Quit

12
Enter team ID: 24

--- Rangers in Team 24 ---
Ranger ID  Ranger Name          Certification          Years of Service    Role
11         Test Visitor         Fire Safety            0                   Leader
```

Testing for **Query #13**

Results Messages

Search to filter items...

Person ID	Full Name	Phone Number	Email Address	City	State	Newsletter Status
2	Ava Smith	405-555-2222	ava.smith@example.com	Tulsa	OK	Not Subscribed
3	Liam Johnson	405-555-3333	liam.johnson@example....	Oklahoma City	OK	Subscribed
4	Olivia Brown	405-555-4444	olivia.brown@example.c...	Stillwater	OK	Subscribed
5	Noah Davis	405-555-5555	noah.davis@example.com	Lawton	OK	Not Subscribed
6	Zoe Miller			Norman	OK	Subscribed
7	Ethan Moore			Edmond	OK	Subscribed
8	Sophia Lopez			Tulsa	OK	Subscribed
9	Ethan Green			Norman	OK	Subscribed
10	Test Visitor			Norman	OK	Not Subscribed
11	Test Visitor			Norman	OK	Not Subscribed
12	Temp DeleteMe			Norman	OK	Not Subscribed
13	Temp DeleteMe			Norman	OK	Not Subscribed
14	Test Expired			Norman	OK	Not Subscribed
15	Temp Expired			Norman	OK	Not Subscribed
16	Final Tester			Norman	OK	Not Subscribed
17	Jordan Lee			Bozeman	MT	Subscribed
18	Morgan Carter			Flagstaff	AZ	Subscribed
19	Taylor Nguyen			Boulder	CO	Not Subscribed
20	Casey Brooks			Eugene	OR	Subscribed

```
18) Quit

13

--- Mailing List ---
Person ID  Full Name              Phone              Email                      City           State
20         Casey Brooks           N/A                N/A                        Eugene         OR
4          Olivia Brown           405-555-4444       olivia.brown@example.com   Stillwater     OK
18         Morgan Carter          N/A                N/A                        Flagstaff      AZ
5          Noah Davis             405-555-5555       noah.davis@example.com     Lawton         OK
12         Temp DeleteMe          N/A                N/A                        Norman         OK
13         Temp DeleteMe          N/A                N/A                        Norman         OK
15         Temp Expired           N/A                N/A                        Norman         OK
14         Test Expired           N/A                N/A                        Norman         OK
9          Ethan Green            N/A                N/A                        Norman         OK
3          Liam Johnson           405-555-3333       liam.johnson@example.com   Oklahoma City  OK
17         Jordan Lee             N/A                N/A                        Bozeman        MT
8          Sophia Lopez           N/A                N/A                        Tulsa          OK
6          Zoe Miller             N/A                N/A                        Norman         OK
7          Ethan Moore            N/A                N/A                        Edmond         OK
19         Taylor Nguyen          N/A                N/A                        Boulder        CO
2          Ava Smith              405-555-2222       ava.smith@example.com      Tulsa          OK
16         Final Tester           N/A                N/A                        Norman         OK
10         Test Visitor           N/A                N/A                        Norman         OK
11         Test Visitor           N/A                N/A                        Norman         OK
```

Testing for **Query #14**

person_id	researcher_name	salary	teams_overseen
16	Final Tester	72000	2

```
18) Quit

14

--- Researchers With Updated Salaries ---
Researcher ID   Researcher Name             New Salary
16              Final Tester                74160.00
```

Testing for **Query #15**

Query 1

Run | Cancel query | Save query | Export data as | Show only Editor

```
1  SELECT v.person_id, pp.expiration_date
2  FROM Visitor v
3  JOIN Park_Pass pp ON v.person_id = pp.person_id;
```

Results | Messages

Search to filter items...

person_id	expiration_date
21	2023-05-01

```
18) Quit

15

--- Deletion Summary ---
1 visitor(s) deleted who had expired passes and no enrollments.
```

Query 1

Run | Cancel query | Save query | Export data as | Show only Editor

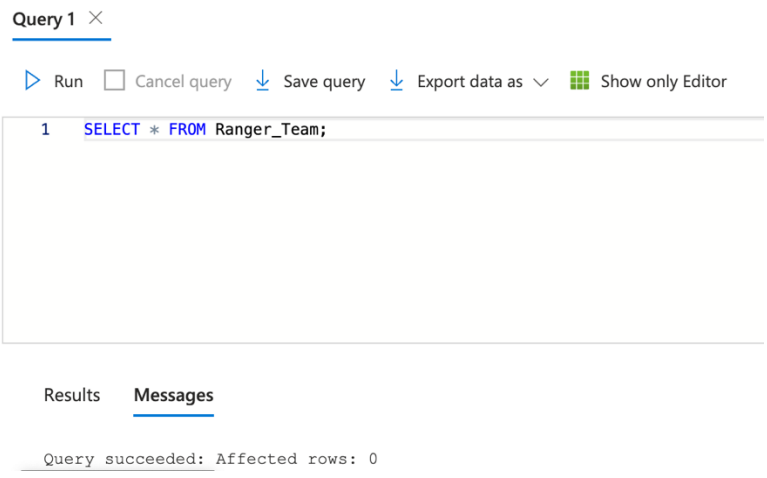
```
1  SELECT v.person_id, pp.expiration_date
2  FROM Visitor v
3  JOIN Park_Pass pp ON v.person_id = pp.person_id;
```

Results | Messages

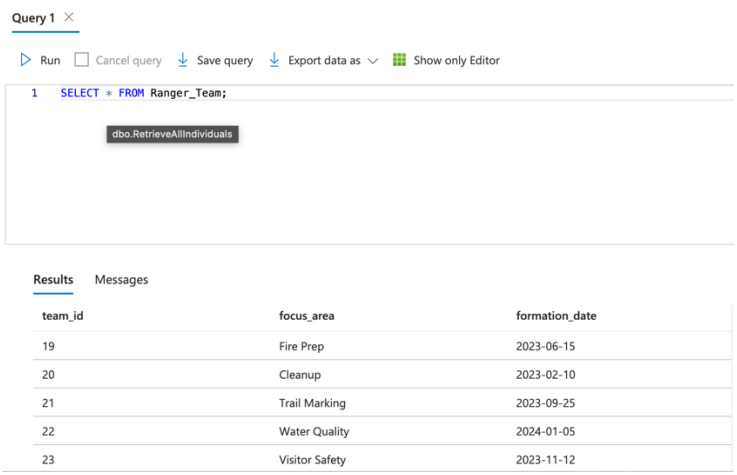
```
Query succeeded: Affected rows: 0
```

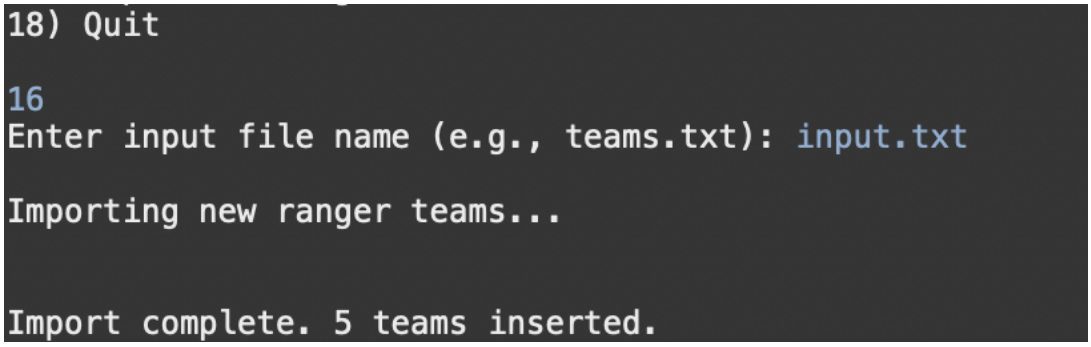
Testing for **Query #16**

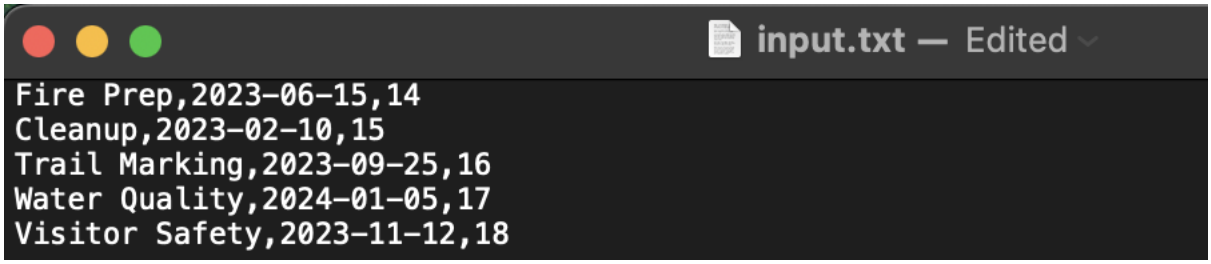
Before Testing



After Testing

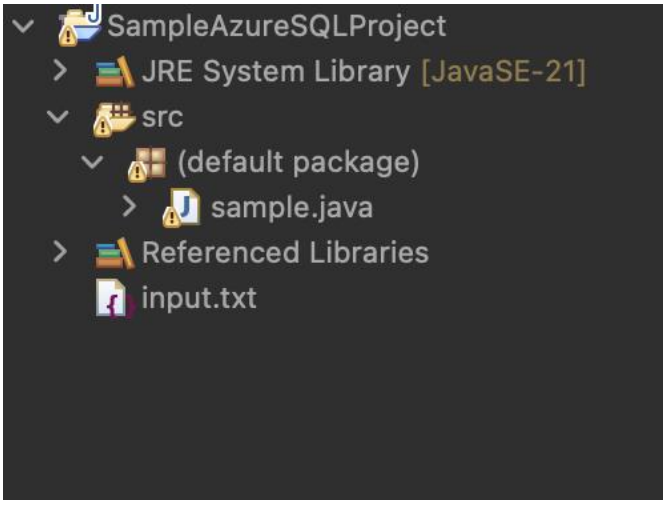




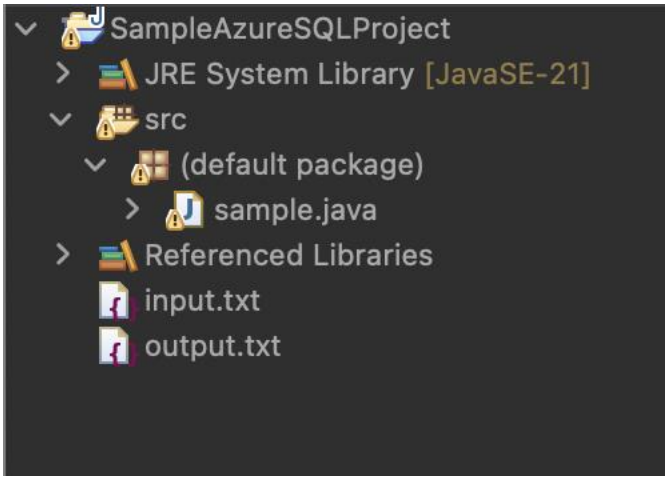


Testing for **Query #17**

Before Testing



After Testing



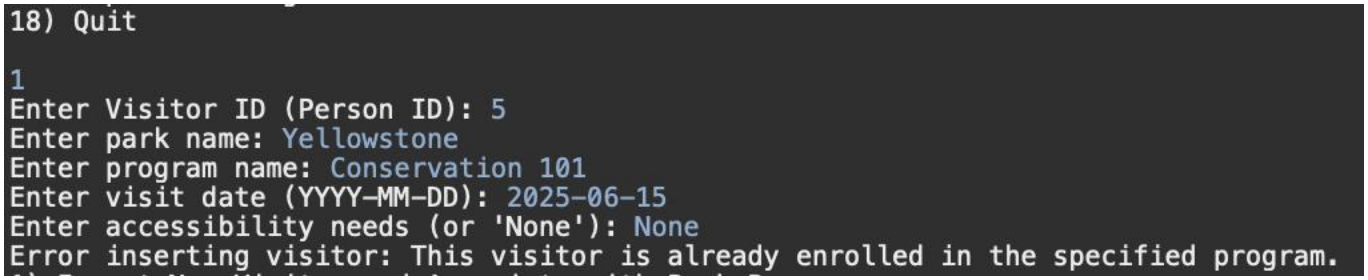

```
18) Quit

17
Enter output file name (e.g., output.txt): output.txt
Mailing list successfully exported to: output.txt
```

sample.java | input.txt | output.txt

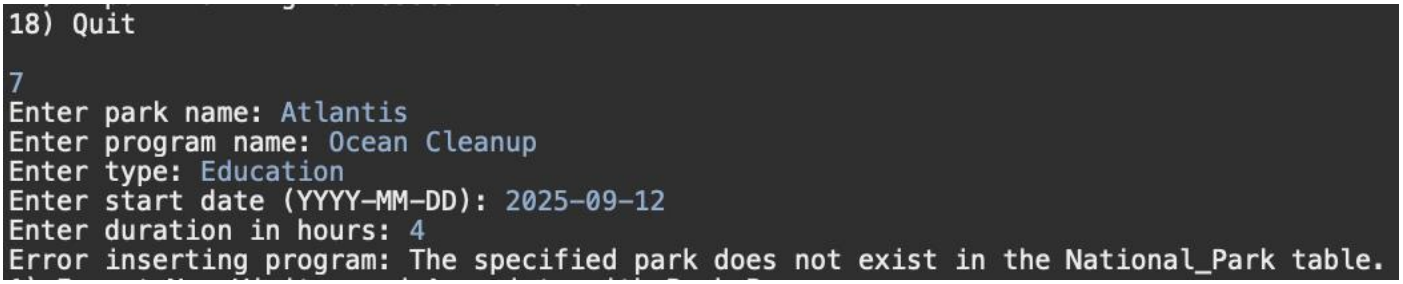
```
1  Person ID,Full Name,Phone,Email,City,State
2  4,Olivia Brown,405-555-4444,olivia.brown@example.com,Stillwater,OK
3  5,Noah Davis,405-555-5555,noah.davis@example.com,Lawton,OK
4  12,Temp DeleteMe,N/A,N/A,Norman,OK
5  13,Temp DeleteMe,N/A,N/A,Norman,OK
6  15,Temp Expired,N/A,N/A,Norman,OK
7  14,Test Expired,N/A,N/A,Norman,OK
8  9,Ethan Green,N/A,N/A,Norman,OK
9  3,Liam Johnson,405-555-3333,liam.johnson@example.com,Oklahoma City,OK
10 8,Sophia Lopez,N/A,N/A,Tulsa,OK
11 6,Zoe Miller,N/A,N/A,Norman,OK
12 7,Ethan Moore,N/A,N/A,Edmond,OK
13 2,Ava Smith,405-555-2222,ava.smith@example.com,Tulsa,OK
14 16,Final Tester,N/A,N/A,Norman,OK
15 10,Test Visitor,N/A,N/A,Norman,OK
16 11,Test Visitor,N/A,N/A,Norman,OK
```

Testing for 3 Types of Error Catching

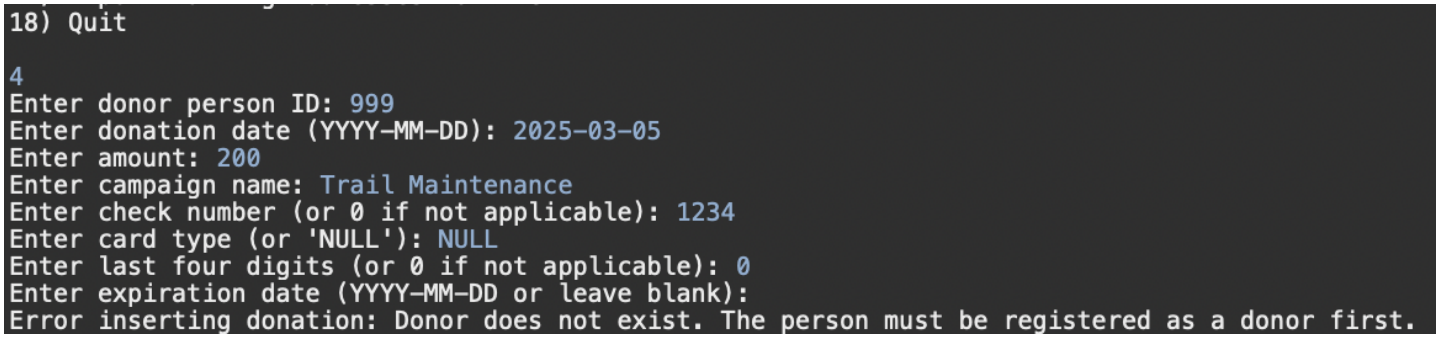
Error Test 1 – Duplicate Enrollment



Error Test 2 – Invalid Park Name

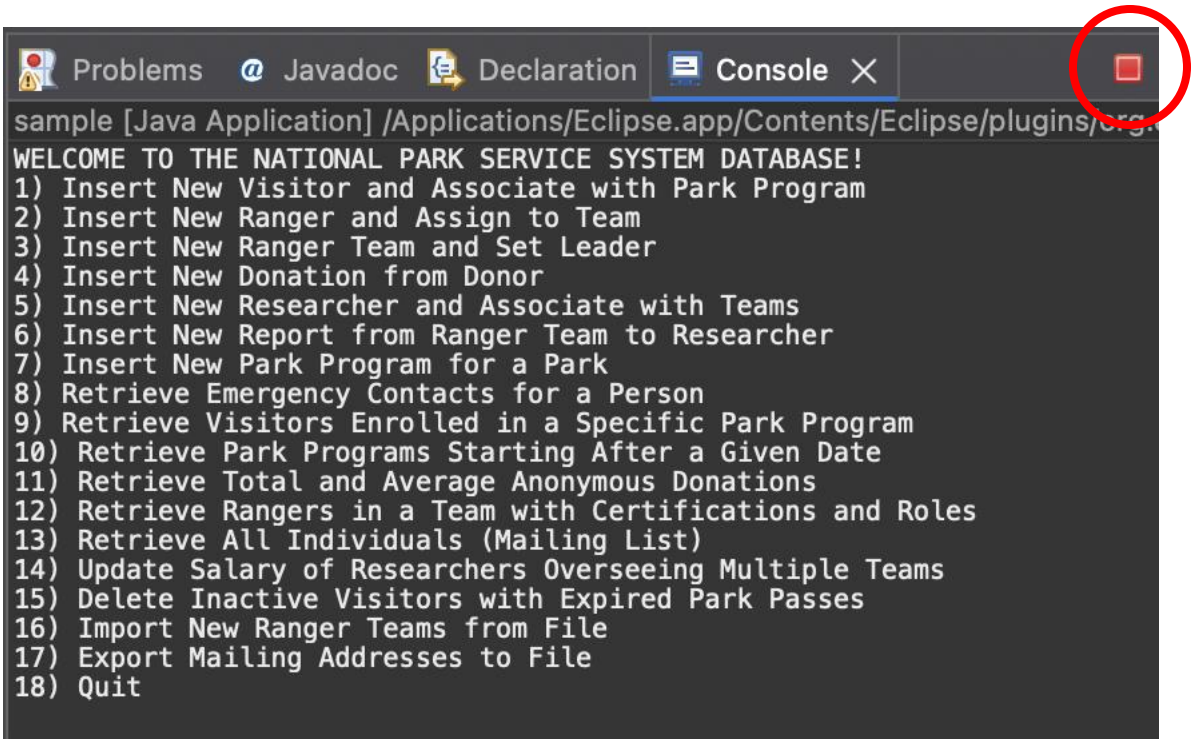


Error Test 3 – Invalid Donor person_id



Testing for ‘Quit’ Function

Before ‘Quit’



After ‘Quit’