Database Management Systems

UE21CS351A

Heritage In Peril - Endangered Species Database

Project Report

Prerana Sanjay Kulkarni, PES1UG21CS451
Rajith M, PES1UG21CS476

Mentor: Prof. Bhargavi Mokashi

Table of Contents

Introduction

The Endangered Species Website is a comprehensive platform aimed at raising awareness about endangered species worldwide. The website employs a Database Management System (DBMS) powered by MySQL to efficiently manage and store critical information about endangered species, their habitats, and relevant analytics. The platform caters to two primary user roles: "Explorers" and "Collaborators."

User Roles

- **Explorer:** An Explorer is a user who gains access to limited functionalities on the website. They can interact with the map page, allowing them to explore endangered species based on regions. Additionally, they have access to the analytics page, which provides insightful data and trends regarding endangered species.

- **Collaborator:** Collaborators have advanced privileges, including all functionalities available to Explorers. In addition, Collaborators have full CRUD (Create, Read, Update, Delete) operations access. They can insert new species into the database, update existing species information, and delete species entries, contributing to the database's maintenance and accuracy.

DBMS Design and Functionality

- **Schema Design:** The MySQL database is structured to efficiently store species information, including details about the species, their habitats, geographical data, conservation status, and other relevant attributes.

- **Data Integrity:** The database employs data integrity constraints to ensure accuracy and consistency of information stored, preventing duplicate or erroneous entries.

- **Security Measures:** The system employs robust security protocols to safeguard sensitive species data. Access control mechanisms ensure that only authorized Collaborators can execute CRUD operations.

Use Cases

- **Explorer Use Case:** An Explorer logs in and navigates the map page, selecting a specific region to view endangered species within that area. They can explore species details, habitat information, and access analytics to understand population trends and conservation efforts.

- **Collaborator Use Case:** A Collaborator logs in and performs various operations such as adding a newly discovered species to the database, updating the conservation status of an endangered species based on recent research, or removing duplicate or erroneous entries to maintain database accuracy.

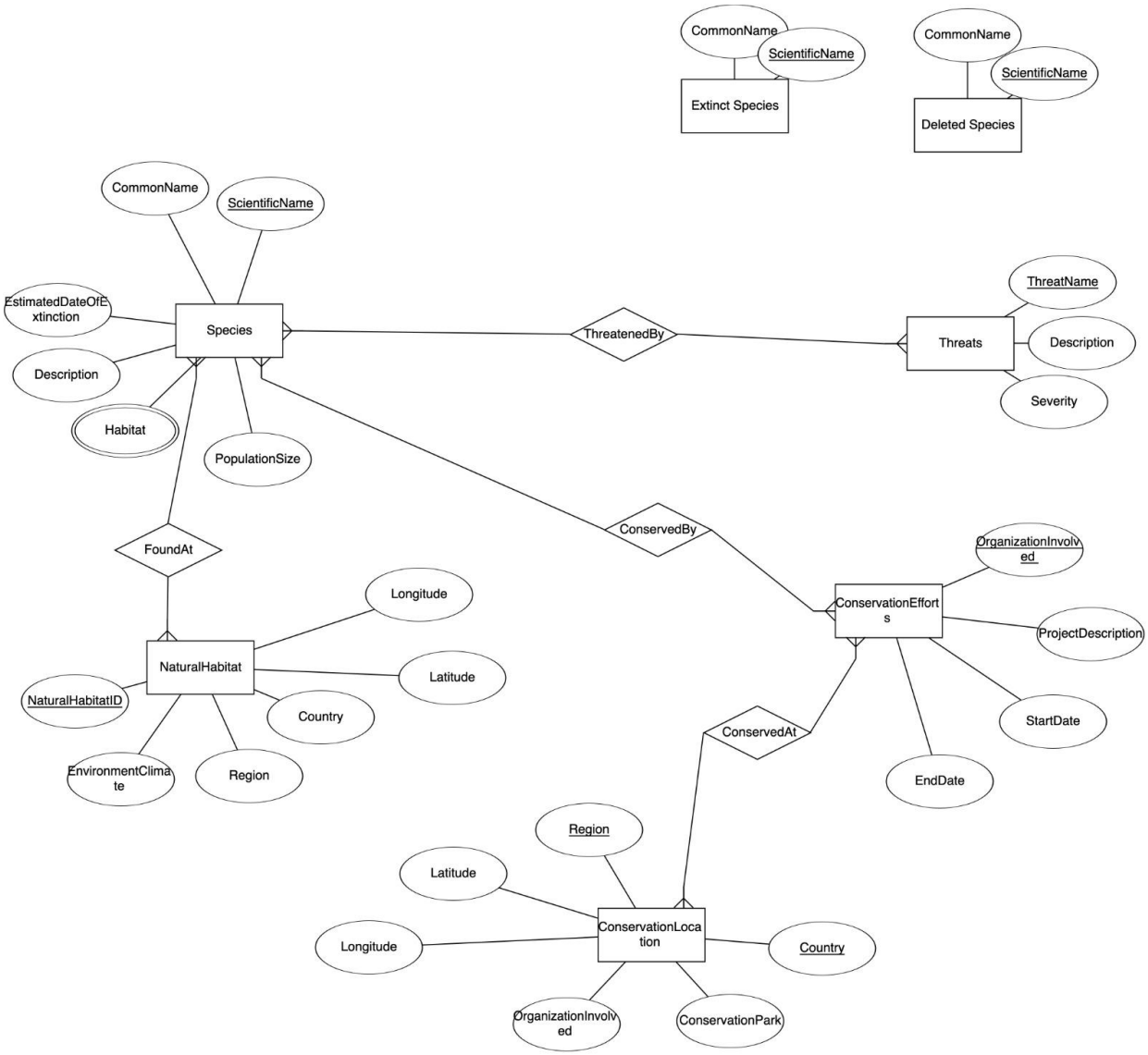
Interactive GUI and User Experience

- The website boasts an intuitive and user-friendly interface, providing seamless navigation for both Explorers and Collaborators.

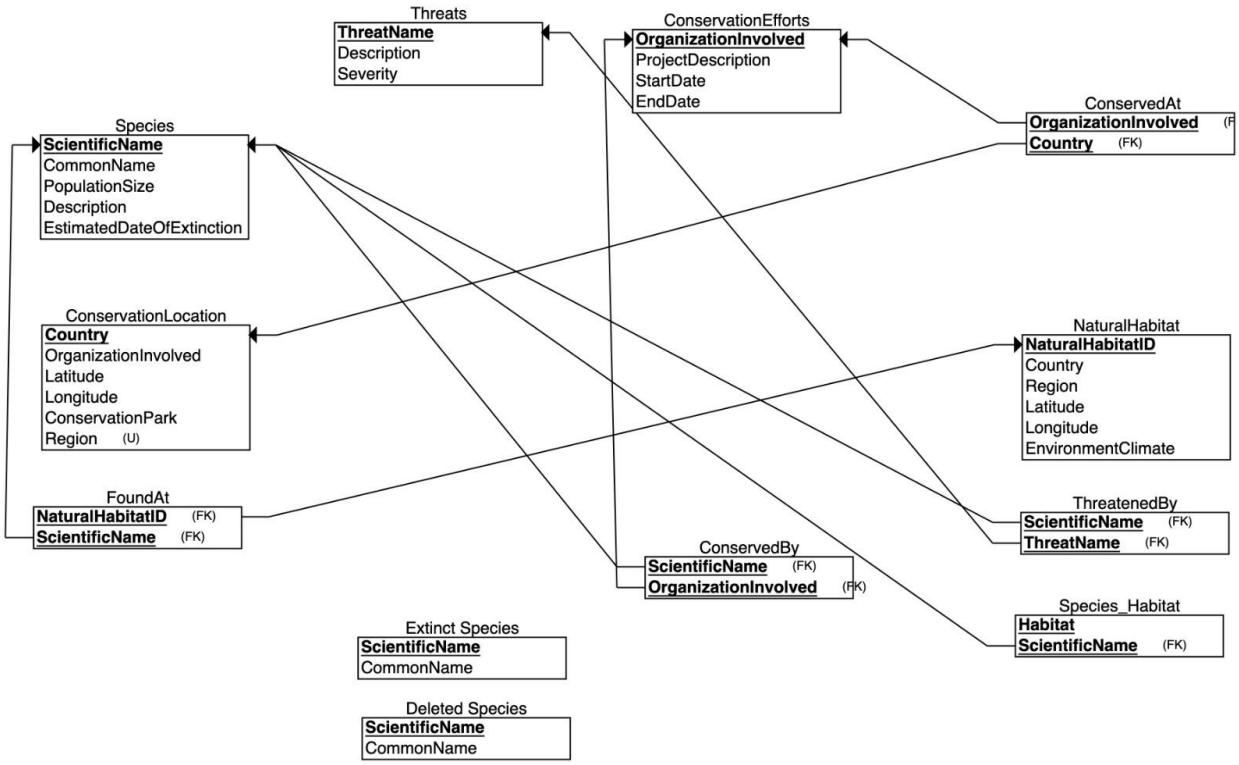
- The map interface is interactive, allowing users to click on regions to discover the endangered species inhabiting those areas. Additionally, the analytics page presents visually appealing and informative graphs, aiding users in understanding species trends and conservation efforts.

In conclusion, the Endangered Species Website leverages MySQL as its DBMS to manage, store, and facilitate efficient retrieval of endangered species data. Through distinct user roles and well-designed functionalities, the platform aims to raise awareness and aid conservation efforts for endangered species worldwide.

1. E-R Diagram


CommonName
ScientificName
Extinct Species
CommonName
ScientificName
Deleted Species
CommonName
ScientificName
EstimatedDateOfExtinction
Species
ThreatenedBy
Threats
ThreatName
Description
Severity
Description
Habitat
PopulationSize
FoundAt
ConservedBy
OrganizationInvolved
ConservationEfforts
ProjectDescription
Longitude
NaturalHabitat
Latitude
NaturalHabitatID
Country
StartDate
ConservedAt
EnvironmentClimate
Region
EndDate
Region
Latitude
ConservationLocation
Longitude
Country
OrganizationInvolved
ConservationPark

2. Relational Schema



3. Normal Form

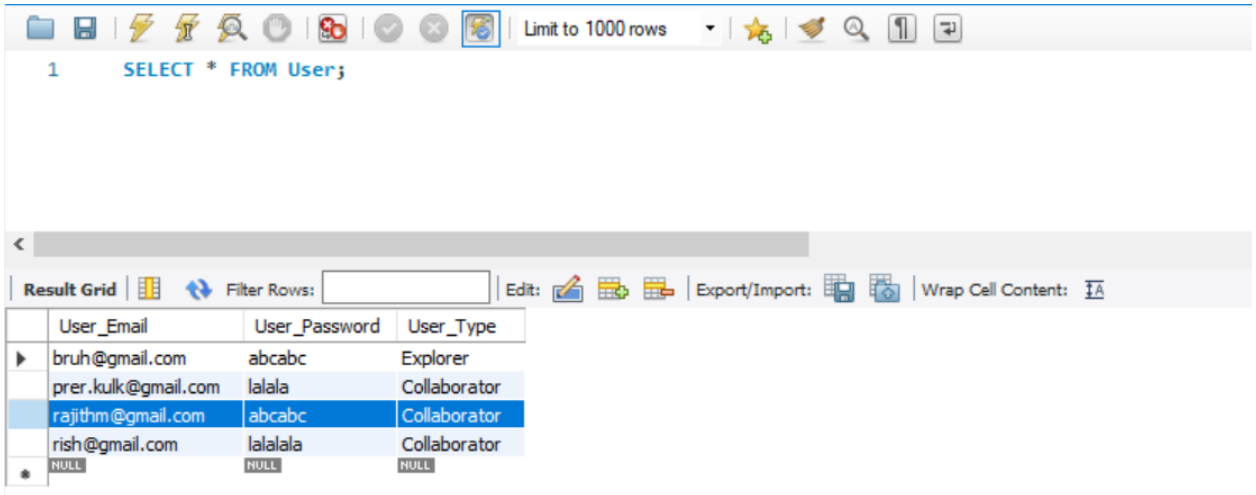
The database has been designed such that all the tables are in 3NF.

4. User Creation/ Varied Privileges

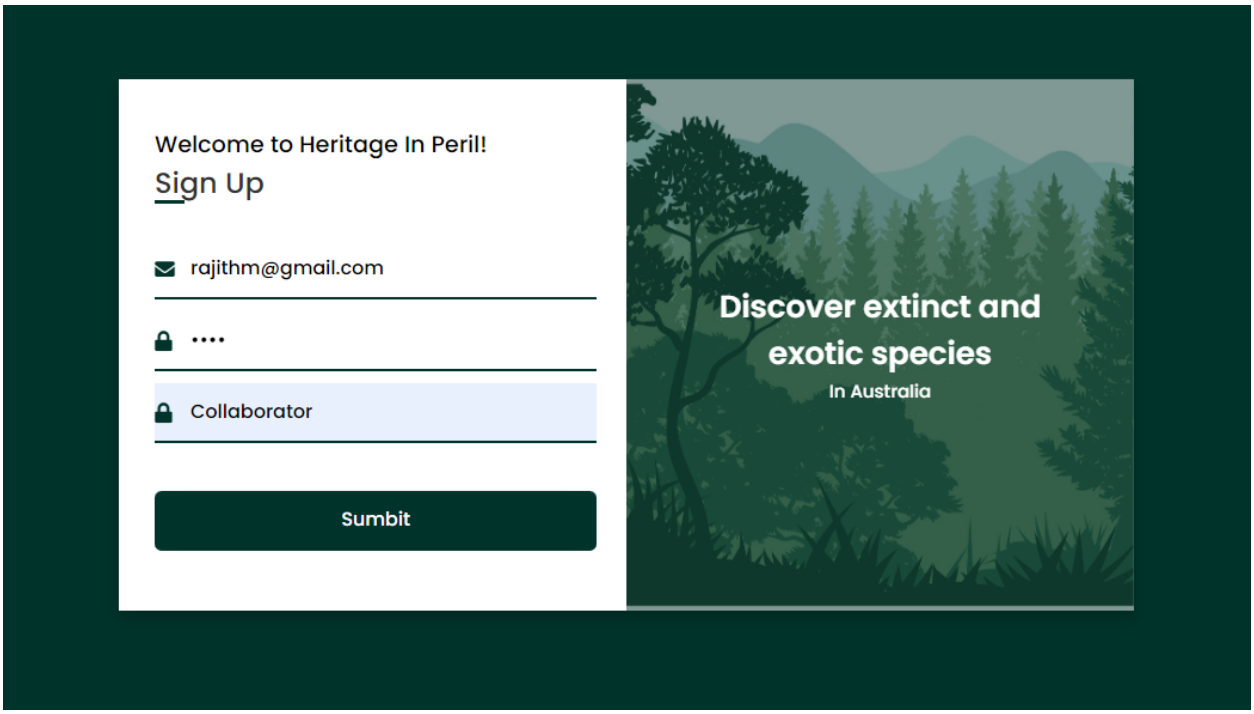
Within our database, we allow two roles:

1) **Explorer**: They only have permissions to view the database hence the SELECT privileges.
2) **Collaborator**: They are allowed to perform any of the standard CRUD operations to our database and hence are granted SELECT, INSERT, UPDATE, DELETE

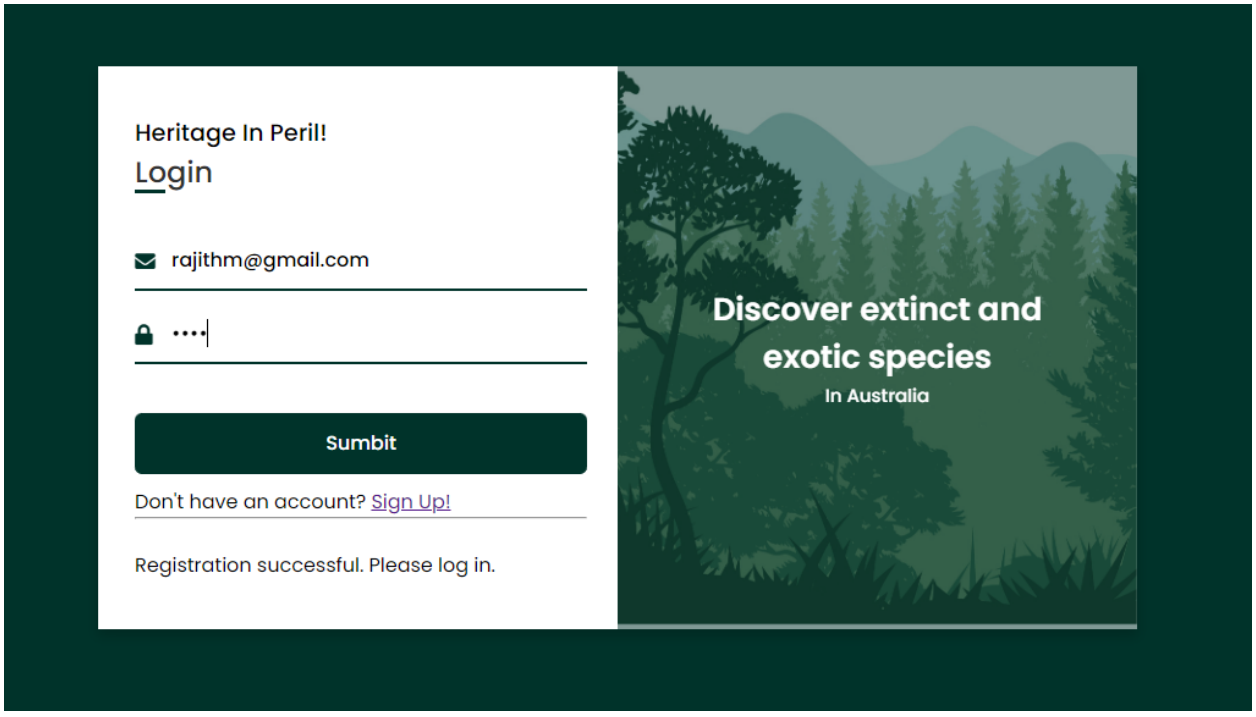
```
mysql> CREATE ROLE "Explorer";
Query OK, 0 rows affected (0.02 sec)

mysql> CREATE ROLE "Collaborator";
Query OK, 0 rows affected (0.01 sec)

mysql> GRANT SELECT, INSERT, UPDATE, DELETE ON heritage_in_peril.* TO "Collaborator";
Query OK, 0 rows affected (0.01 sec)

mysql> GRANT SELECT ON heritage_in_peril.* TO "Explorer";
Query OK, 0 rows affected (0.01 sec)

mysql>
```

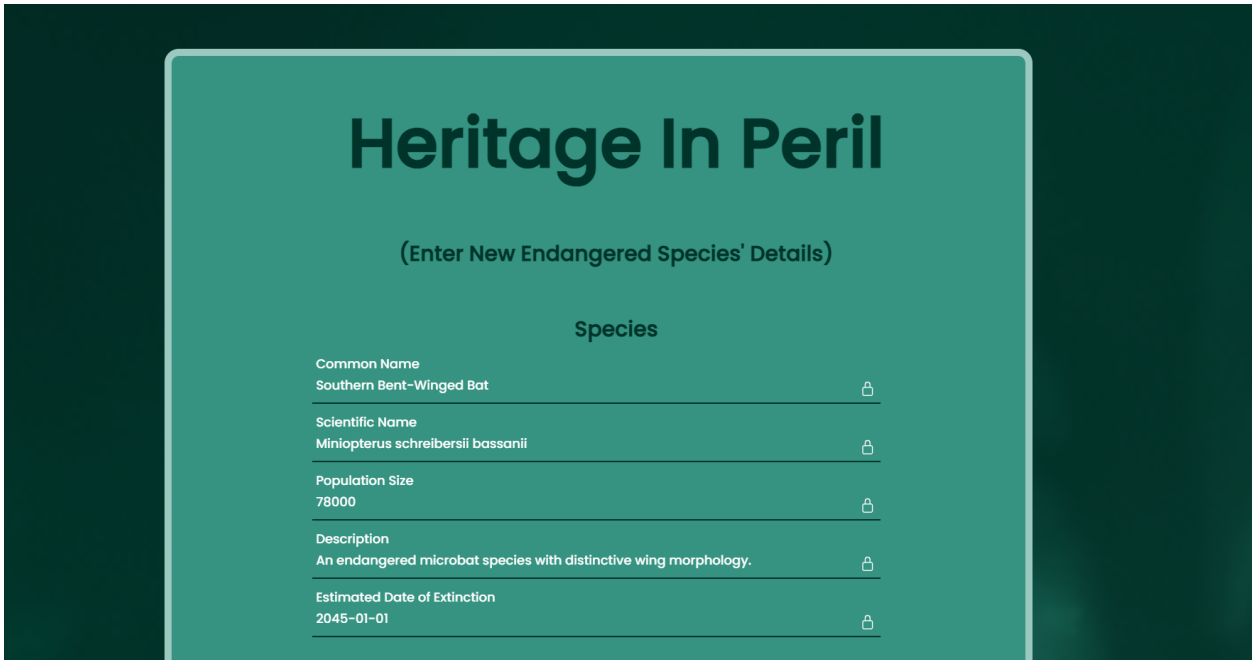


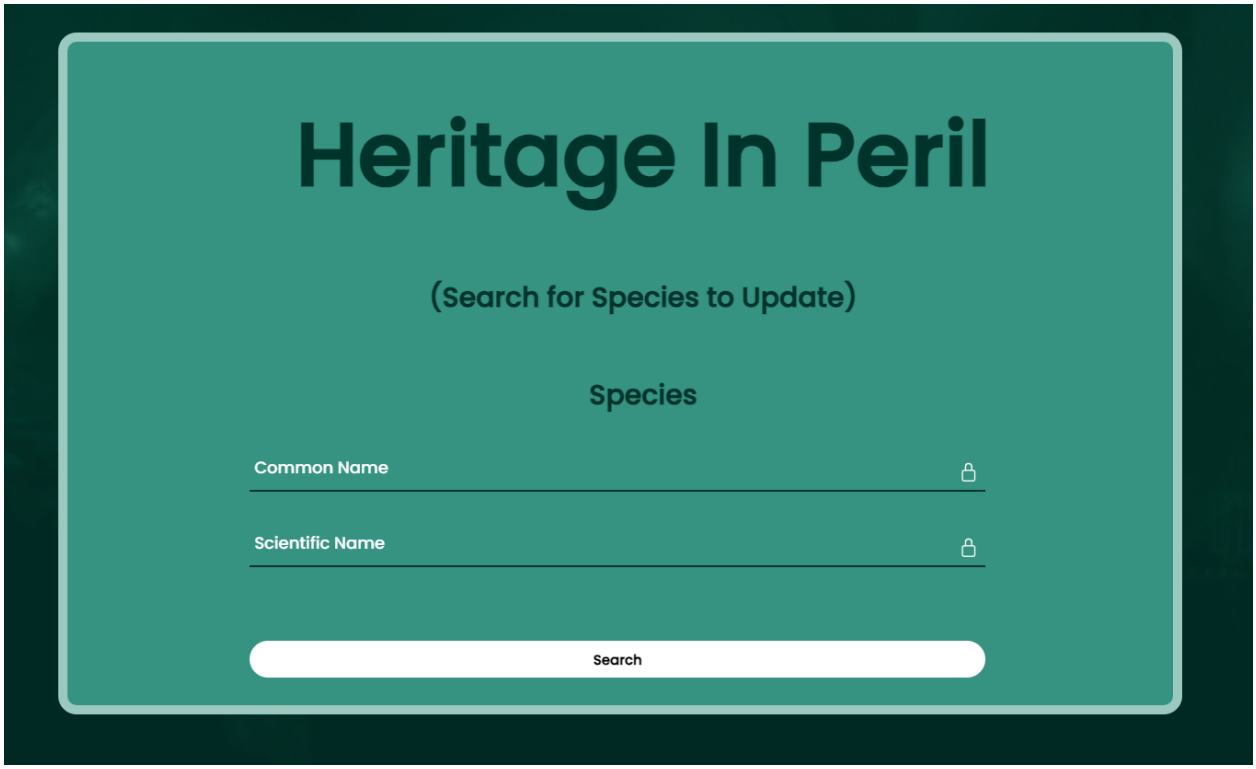
The register page in our frontend allows the user to create an account and either sign-up as a Collaborator or an Explorer.



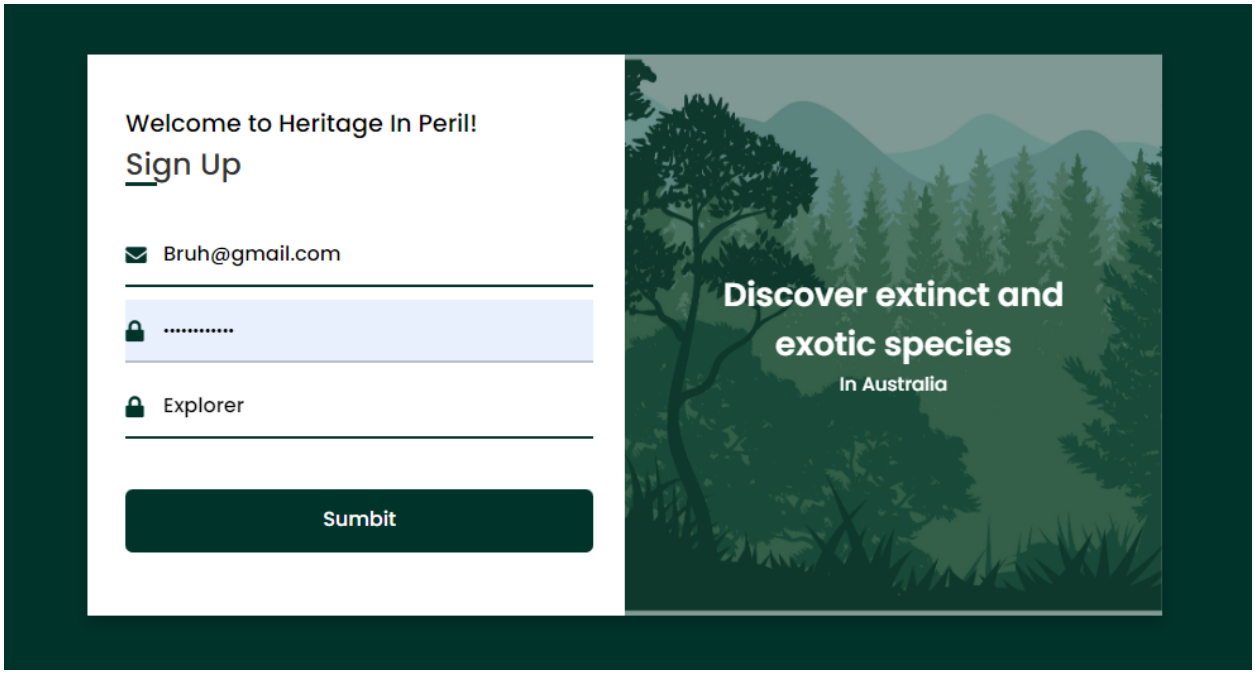
They can then login through our login page:



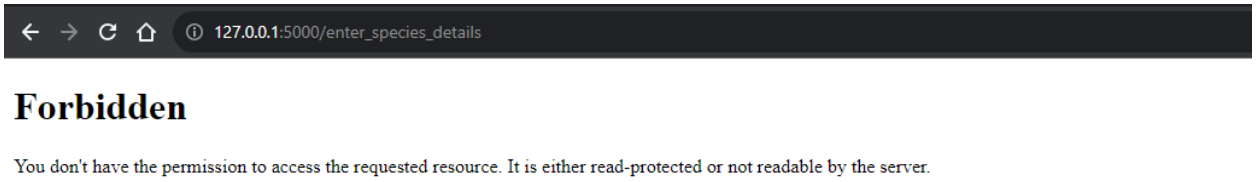




Since the user has logged in as an “Collaborator” they can access any of the pages that enable CRUD operations



Signing up as an “Explorer” and then trying to access the CRUD operation pages:



We return a 403 Message specifying that their attempts are invalid

5. Triggers

The TrackSpeciesDataChanges trigger maintains a log that keeps track of all the updates make to some species in the database.

```
DELIMITER //

CREATE TRIGGER TrackSpeciesDataChanges
AFTER UPDATE ON species
FOR EACH ROW
BEGIN
    DECLARE change_info TEXT;

    -- Check if PopulationSize changed
    IF OLD.PopulationSize <> NEW.PopulationSize THEN
              SET change_info = CONCAT('PopulationSize changed from ',
OLD.PopulationSize, ' to ', NEW.PopulationSize);
              INSERT INTO SpeciesChangeLog (ScientificName, CommonName,
ChangeInfo, ChangeDateTime)
        VALUES (NEW.ScientificName, NEW.CommonName, change_info, NOW());
    END IF;

    -- Check if EstimatedDateOfExtinction changed
    IF OLD.EstimatedDateOfExtinction <> NEW.EstimatedDateOfExtinction THEN
          SET change_info = CONCAT('EstimatedDateOfExtinction changed from
', OLD.EstimatedDateOfExtinction, ' to ', NEW.EstimatedDateOfExtinction);
              INSERT INTO SpeciesChangeLog (ScientificName, CommonName,
ChangeInfo, ChangeDateTime)
        VALUES (NEW.ScientificName, NEW.CommonName, change_info, NOW());
    END IF;

    -- Repeat similar checks for other fields and log changes if needed
END;
//
DELIMITER ;
```

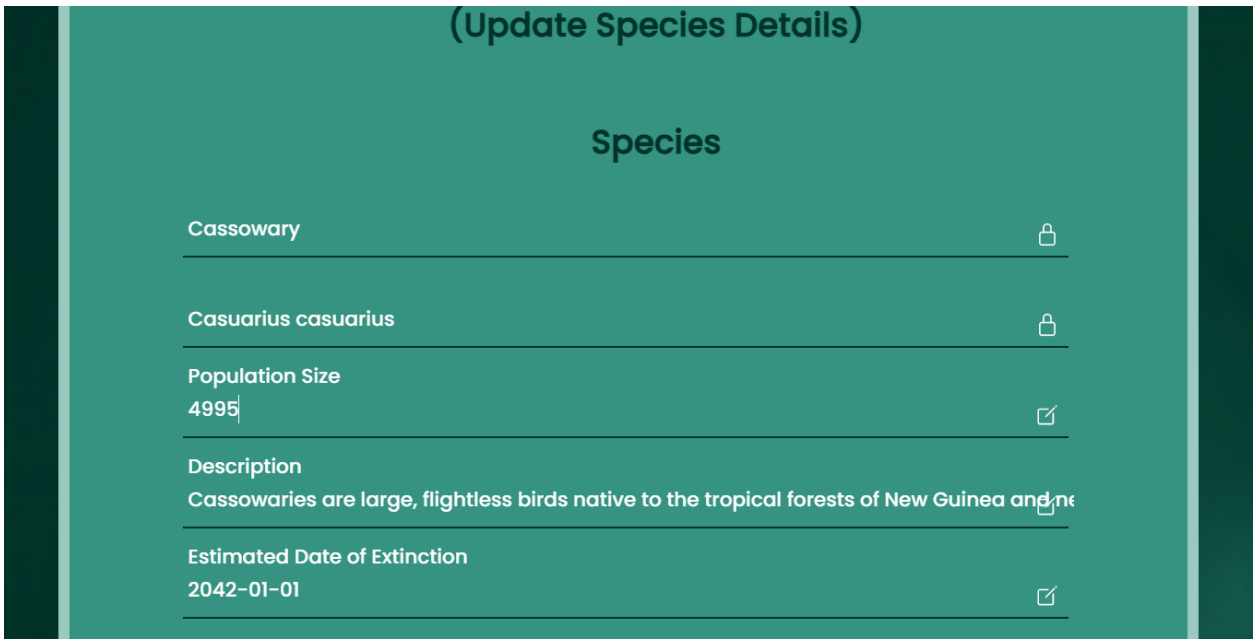
To demonstrate this, let us change thepopulation of the Cassowary.
This is the population of the cassowary before the update:

```
1    SELECT * FROM Species;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

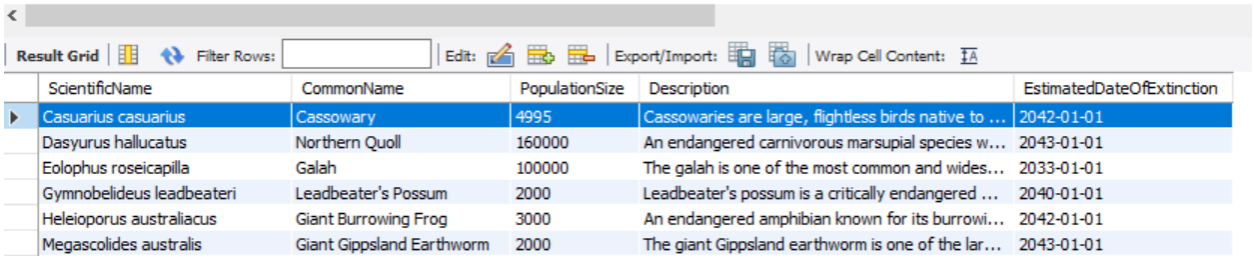
ScientificName	CommonName	PopulationSize	Description	EstimatedDateOfExtinction
Casuarius casuarius	Cassowary	5000	Cassowaries are large, flightless birds native to ...	2042-01-01
Dasyurus hallucatus	Northern Quoll	160000	An endangered carnivorous marsupial species w...	2043-01-01
Eolophus roseicapilla	Galah	100000	The galah is one of the most common and wides...	2033-01-01
Gymnobelideus leadbeateri	Leadbeater's Possum	2000	Leadbeater's possum is a critically endangered ...	2040-01-01
Heleioporus australiacus	Giant Burrowing Frog	3000	An endangered amphibian known for its burrowi...	2042-01-01
Megascolides australis	Giant Gippsland Earthworm	2000	The giant Gippsland earthworm is one of the lar...	2043-01-01
Miniopterus schreibersii bassanii	Southern Bent-Winged Bat	78000	An endangered microbat species with distinctive...	2045-01-01
Ornithorhynchus anatinus	Platypus	10000	The platypus is a unique egg-laying mammal nat...	2055-01-01

We change the population of the cassowary (on the update page of the website)



The edit is reflected in the database:

```
1    SELECT * FROM Species;
```

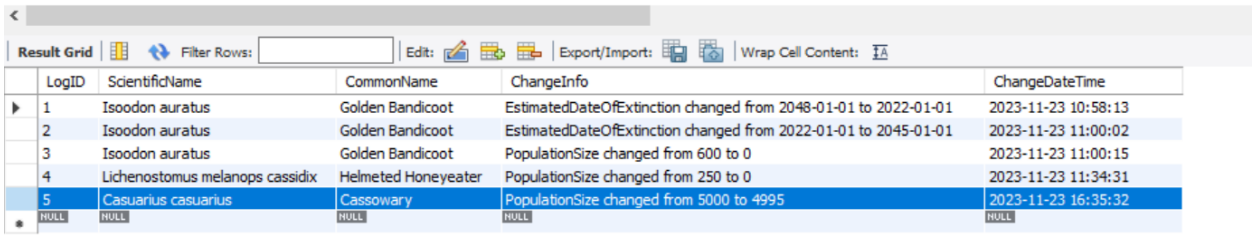


Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

ScientificName	CommonName	PopulationSize	Description	EstimatedDateOfExtinction
Casuarius casuarius	Cassowary	4995	Cassowaries are large, flightless birds native to ...	2042-01-01
Dasyurus hallucatus	Northern Quoll	160000	An endangered carnivorous marsupial species w...	2043-01-01
Eolophus roseicapilla	Galah	100000	The galah is one of the most common and wides...	2033-01-01
Gymnobelideus leadbeateri	Leadbeater's Possum	2000	Leadbeater's possum is a critically endangered ...	2040-01-01
Heleioporus australiacus	Giant Burrowing Frog	3000	An endangered amphibian known for its burrowi...	2042-01-01
Megascolides australis	Giant Gippsland Earthworm	2000	The giant Gippsland earthworm is one of the lar...	2043-01-01

The edit log is updated by the trigger:

```
1    SELECT * FROM SpeciesChangeLog;
```



Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

LogID	ScientificName	CommonName	ChangeInfo	ChangeDateTime
1	Isoodon auratus	Golden Bandicoot	EstimatedDateOfExtinction changed from 2048-01-01 to 2022-01-01	2023-11-23 10:58:13
2	Isoodon auratus	Golden Bandicoot	EstimatedDateOfExtinction changed from 2022-01-01 to 2045-01-01	2023-11-23 11:00:02
3	Isoodon auratus	Golden Bandicoot	PopulationSize changed from 600 to 0	2023-11-23 11:00:15
4	Lichenostomus melanops cassidix	Helmeted Honeyeater	PopulationSize changed from 250 to 0	2023-11-23 11:34:31
5	Casuarius casuarius	Cassowary	PopulationSize changed from 5000 to 4995	2023-11-23 16:35:32
NULL	NULL	NULL	NULL	NULL

6. Procedures/Functions

In our project, we have made use of procedures for C-U-D (Create, Update, Delete) operations. We also use a procedure to know when a species has gone extinct, so it can be appended to the table of extinct species.

We can see the following procedures in our project:

- **Procedure to update all the relations in the database, once a new species is entered into the database:**

```
DELIMITER //

CREATE PROCEDURE UpdateRelationTables (
    IN p_ScientificName VARCHAR(255),
    IN p_CommonName VARCHAR(255),
    IN p_Country VARCHAR(255),
    IN p_Region VARCHAR(255),
    IN p_ThreatName VARCHAR(255),
    IN p_OrganizationName VARCHAR(255),
    IN p_ConservationLocationCountry VARCHAR(255),
    IN p_ConservationLocationRegion VARCHAR(255)
)
BEGIN
    -- Update FoundAt table if the entry doesn't already exist
    IF NOT EXISTS (
        SELECT 1 FROM FoundAt
        WHERE NaturalHabitatCountry = p_Country
        AND NaturalHabitatRegion = p_Region
        AND SpeciesScientificName = p_ScientificName
    ) THEN
                    INSERT  INTO  FoundAt  (NaturalHabitatCountry,
NaturalHabitatRegion, SpeciesScientificName)
```

```
        VALUES (p_Country, p_Region, p_ScientificName);
    END IF;

     -- Update ThreatenedBy table if the entry doesn't already
exist
    IF NOT EXISTS (
        SELECT 1 FROM ThreatenedBy
        WHERE SpeciesScientificName = p_ScientificName
        AND ThreatName = p_ThreatName
    ) THEN
            INSERT INTO ThreatenedBy (SpeciesScientificName,
ThreatName)
        VALUES (p_ScientificName, p_ThreatName);
    END IF;

     -- Update ConservedAt table if the entry doesn't already
exist
    IF NOT EXISTS (
        SELECT 1 FROM ConservedAt
        WHERE OrganizationName = p_OrganizationName
                        AND ConservationLocationCountry =
p_ConservationLocationCountry
                         AND ConservationLocationRegion =
p_ConservationLocationRegion
    ) THEN
                INSERT INTO ConservedAt (OrganizationName,
ConservationLocationCountry, ConservationLocationRegion)
                                VALUES (p_OrganizationName,
p_ConservationLocationCountry, p_ConservationLocationRegion);
    END IF;

     -- Update ConservedBy table if the entry doesn't already
exist
    IF NOT EXISTS (
        SELECT 1 FROM ConservedBy
```

```
        WHERE OrganizationName = p_OrganizationName
        AND SpeciesScientificName = p_ScientificName
    ) THEN
                    INSERT  INTO  ConservedBy  (OrganizationName,
SpeciesScientificName)
        VALUES (p_OrganizationName, p_ScientificName);
    END IF;
END //

DELIMITER ;
```

- **Procedure to update the attributes associated with a species:**

```
DELIMITER //

CREATE PROCEDURE UpdateSpeciesDetails (
    IN p_CommonName VARCHAR(255),
    IN p_ScientificName VARCHAR(255),
    IN p_PopulationSize INT,
    IN p_Description TEXT,
    IN p_EstimatedExtinctionDate DATE,
    IN p_Country VARCHAR(255),
    IN p_Region VARCHAR(255),
    IN p_Latitude DECIMAL(10, 6),
    IN p_Longitude DECIMAL(10, 6),
    IN p_ThreatName VARCHAR(255),
    IN p_ThreatDescription TEXT,
    IN p_Severity TINYINT,
    IN p_OrganizationName VARCHAR(255),
    IN p_ProjectDescription TEXT,
    IN p_StartDate DATE,
    IN p_EndDate DATE,
    IN p_Cloclatitude DECIMAL(10, 6),
    IN p_Cloclongitude DECIMAL(10, 6),
```

```
    IN p_Cloccountry VARCHAR(255),
    IN p_Clocregion VARCHAR(255),
    IN p_ConservationPark VARCHAR(255)
)
BEGIN
    -- Update or insert into the species table
        INSERT INTO species (ScientificName, CommonName,
PopulationSize, Description, EstimatedDateOfExtinction)
      VALUES (p_ScientificName, p_CommonName, p_PopulationSize,
p_Description, p_EstimatedExtinctionDate)
    ON DUPLICATE KEY UPDATE
        CommonName = VALUES(CommonName),
        PopulationSize = VALUES(PopulationSize),
        Description = VALUES(Description),
                                EstimatedDateOfExtinction =
VALUES(EstimatedDateOfExtinction);

    -- Update or insert into the NaturalHabitat table
       INSERT INTO NaturalHabitat (Country, Region, Latitude,
Longitude)
    VALUES (p_Country, p_Region, p_Latitude, p_Longitude)
    ON DUPLICATE KEY UPDATE
        Latitude = VALUES(Latitude),
        Longitude = VALUES(Longitude);

    -- Update or insert into the FoundAt table
      CALL UpdateRelationTables(p_ScientificName, p_CommonName,
p_Country,    p_Region,    p_ThreatName,    p_OrganizationName,
p_Cloccountry, p_Clocregion);

    -- Update or insert into the Threats table
    INSERT INTO Threats (ThreatName, Description, Severity)
    VALUES (p_ThreatName, p_ThreatDescription, p_Severity)
    ON DUPLICATE KEY UPDATE
        Description = VALUES(Description),
```

```
        Severity = VALUES(Severity);

    -- Update or insert into the ConservationEfforts table
          INSERT INTO ConservationEfforts (OrganizationName,
ProjectDescription, StartDate, EndDate)
            VALUES (p_OrganizationName, p_ProjectDescription,
p_StartDate, p_EndDate)
    ON DUPLICATE KEY UPDATE
        ProjectDescription = VALUES(ProjectDescription),
        StartDate = VALUES(StartDate),
        EndDate = VALUES(EndDate);

    -- Update or insert into the ConservationLocation table
     INSERT INTO ConservationLocation (Country, Region, Latitude,
Longitude, ConservationPark)
         VALUES (p_Cloccountry, p_Clocregion, p_Cloclatitude,
p_Cloclongitude, p_ConservationPark)
    ON DUPLICATE KEY UPDATE
        Latitude = VALUES(Latitude),
        Longitude = VALUES(Longitude),
        ConservationPark = VALUES(ConservationPark);

      -- Check if the updated population size is zero and call
DeleteSpeciesEntries if true
    IF p_PopulationSize = 0 THEN
        CALL ExtinctSpecies(p_CommonName, p_ScientificName);
    END IF;

END //

DELIMITER ;
```

- **Procedure to delete a species from the database, and insert into the deletedSpecies table:**

```
DELIMITER //

CREATE PROCEDURE DeleteSpeciesEntries (
    IN p_CommonName VARCHAR(255),
    IN p_ScientificName VARCHAR(255)
)
BEGIN
      -- Delete entries from FoundAt table for the specified species
    DELETE FROM FoundAt
    WHERE SpeciesScientificName = p_ScientificName;

     -- Delete entries from ConservedAt table for the specified species
    DELETE FROM ConservedAt
    WHERE OrganizationName IN (
        SELECT OrganizationName FROM ConservedBy
        WHERE SpeciesScientificName = p_ScientificName
    );

     -- Delete entries from ConservedBy table for the specified species
    DELETE FROM ConservedBy
    WHERE SpeciesScientificName = p_ScientificName;

     -- Delete entries from ThreatenedBy table for the specified species
    DELETE FROM ThreatenedBy
    WHERE SpeciesScientificName = p_ScientificName;
```

```
    -- Insert the deleted species into the deletedSpecies table
    INSERT INTO deletedSpecies(ScientificName, CommonName)
        SELECT ScientificName, CommonName FROM Species WHERE
ScientificName = p_ScientificName;

    -- Delete the specified species from the Species table
    DELETE FROM Species
    WHERE ScientificName = p_ScientificName;
END //

DELIMITER ;
```

- **Procedure to move species to extinctSpecies table when its population gets updated to 0:**

```
DELIMITER //

CREATE PROCEDURE ExtinctSpecies (
    IN p_CommonName VARCHAR(255),
    IN p_ScientificName VARCHAR(255)
)
BEGIN
      -- Delete entries from FoundAt table for the specified
species
    DELETE FROM FoundAt
    WHERE SpeciesScientificName = p_ScientificName;

     -- Delete entries from ConservedAt table for the specified
species
    DELETE FROM ConservedAt
    WHERE OrganizationName IN (
        SELECT OrganizationName FROM ConservedBy
        WHERE SpeciesScientificName = p_ScientificName
    );
```

```
     -- Delete entries from ConservedBy table for the specified
species
    DELETE FROM ConservedBy
    WHERE SpeciesScientificName = p_ScientificName;

     -- Delete entries from ThreatenedBy table for the specified
species
    DELETE FROM ThreatenedBy
    WHERE SpeciesScientificName = p_ScientificName;

    -- Insert the deleted species into the deletedSpecies table
    INSERT INTO extinctSpecies(ScientificName, CommonName)
        SELECT ScientificName, CommonName FROM Species WHERE
ScientificName = p_ScientificName;

    -- Delete the specified species from the Species table
    DELETE FROM Species
    WHERE ScientificName = p_ScientificName;
END //

DELIMITER ;
```

7. Create Operations

All the tables as per our original schema have been created:

```
-- Species Table:
CREATE DATABASE heritage_in_peril;
USE heritage_in_peril;

CREATE TABLE species (
    ScientificName VARCHAR(255) PRIMARY KEY,
    CommonName VARCHAR(255) NOT NULL,
    PopulationSize INT NOT NULL,
    Description TEXT NOT NULL,
    EstimatedDateOfExtinction DATE
);

-- Closest Neighbour Table:
CREATE TABLE ClosestNeighbour (
    CommonName VARCHAR(255) NOT NULL,
    ScientificName VARCHAR(255) NOT NULL,
    PRIMARY KEY (CommonName, ScientificName),
                FOREIGN    KEY    (ScientificName)    REFERENCES
species(ScientificName) ON UPDATE CASCADE
);

-- NaturalHabitat Table:
CREATE TABLE NaturalHabitat (
    Country VARCHAR(255),
    Region VARCHAR(255),
    PRIMARY KEY (Country, Region),
    Latitude VARCHAR(50),
    Longitude VARCHAR(50)
);
```

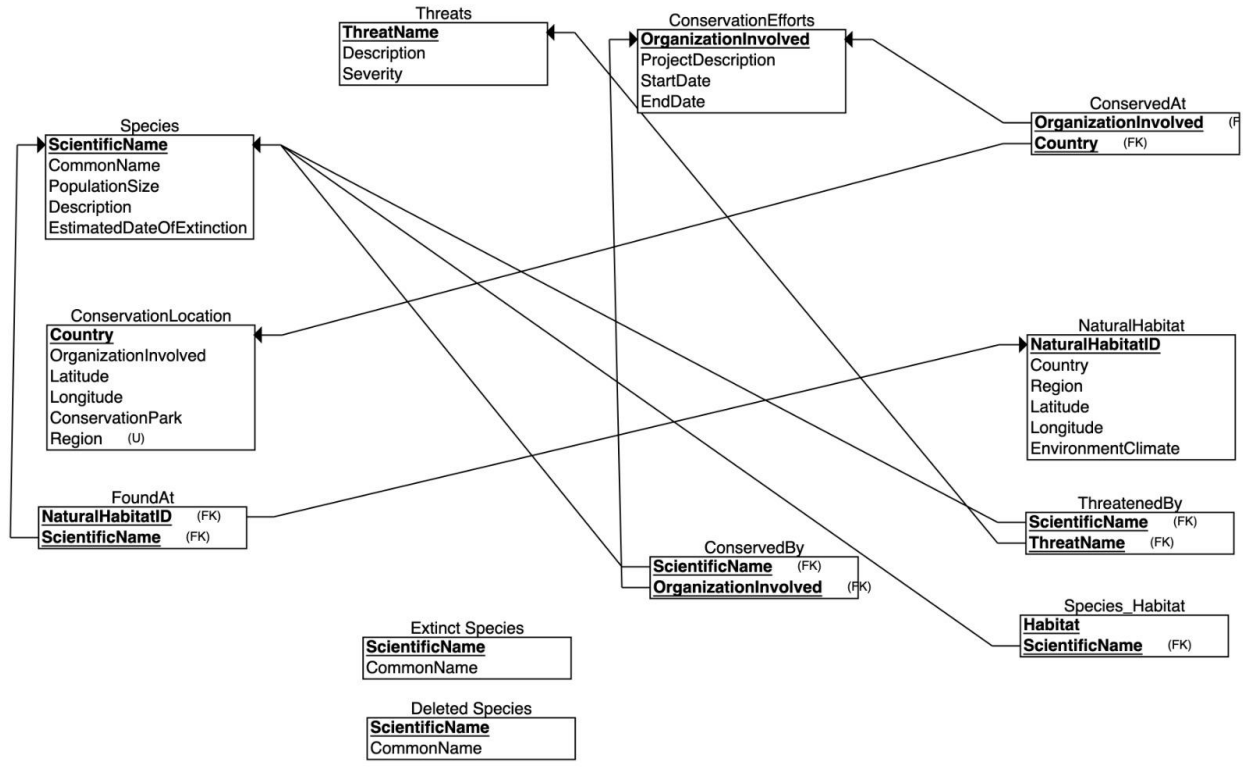
```
-- FoundAt Table:
CREATE TABLE FoundAt (
    NaturalHabitatCountry VARCHAR(255),
    NaturalHabitatRegion VARCHAR(255),
    SpeciesScientificName VARCHAR(255),
      PRIMARY KEY (NaturalHabitatCountry, NaturalHabitatRegion,
SpeciesScientificName),
      FOREIGN KEY (NaturalHabitatCountry, NaturalHabitatRegion)
REFERENCES NaturalHabitat(Country, Region) ON UPDATE CASCADE,
            FOREIGN KEY (SpeciesScientificName) REFERENCES
species(ScientificName) ON UPDATE CASCADE
);

-- Threats Table:
CREATE TABLE Threats (
    ThreatName VARCHAR(255) PRIMARY KEY,
    Description TEXT NOT NULL,
     Severity TINYINT NOT NULL CHECK (Severity >= 1 AND Severity
<= 5)
);

-- ThreatenedBy table:
CREATE TABLE ThreatenedBy (
    SpeciesScientificName VARCHAR(255),
    ThreatName VARCHAR(255),
    PRIMARY KEY (SpeciesScientificName, ThreatName),
            FOREIGN KEY (SpeciesScientificName) REFERENCES
species(ScientificName) ON UPDATE CASCADE,
     FOREIGN KEY (ThreatName) REFERENCES Threats(ThreatName) ON
UPDATE CASCADE
);

-- ConservationEfforts Table:
CREATE TABLE ConservationEfforts (
    OrganizationName VARCHAR(255) PRIMARY KEY,
```

```
    ProjectDescription TEXT NOT NULL,
    StartDate DATE NOT NULL,
    EndDate DATE NOT NULL
);

-- Conservation Location Table:
CREATE TABLE ConservationLocation (
    Country VARCHAR(255),
    Region VARCHAR(255),
    PRIMARY KEY (Country, Region),
    Latitude DECIMAL(10, 6) NOT NULL,
    Longitude DECIMAL(10, 6) NOT NULL,
    ConservationPark VARCHAR(255)
);

-- ConservedBy:
CREATE TABLE ConservedBy (
    OrganizationName VARCHAR(255),
    SpeciesScientificName VARCHAR(255),
    PRIMARY KEY (OrganizationName, SpeciesScientificName),
                FOREIGN    KEY    (OrganizationName)    REFERENCES
ConservationEfforts(OrganizationName) ON UPDATE CASCADE,
             FOREIGN   KEY   (SpeciesScientificName)   REFERENCES
species(ScientificName) ON UPDATE CASCADE
);

-- ConservedAt Table:
CREATE TABLE ConservedAt (
    OrganizationName VARCHAR(255),
    ConservationLocationCountry VARCHAR(255),
    ConservationLocationRegion VARCHAR(255),
     PRIMARY KEY (OrganizationName, ConservationLocationCountry,
ConservationLocationRegion),
                FOREIGN    KEY    (OrganizationName)    REFERENCES
ConservationEfforts(OrganizationName) ON UPDATE CASCADE,
```

```
                FOREIGN    KEY    (ConservationLocationCountry,
ConservationLocationRegion)                              REFERENCES
ConservationLocation(Country, Region) ON UPDATE CASCADE
);

-- User table (for people who subscribe to the page and for
monthly newsletters, etc.)
CREATE TABLE User (
    User_Email VARCHAR(50),
    User_Password VARCHAR(50),
    User_Type VARCHAR(50),
    PRIMARY KEY (User_Email)
);

CREATE TABLE extinctSpecies (
    ScientificName VARCHAR(255) PRIMARY KEY,
    CommonName VARCHAR(255) NOT NULL
);

CREATE TABLE deletedSpecies (
    ScientificName VARCHAR(255) PRIMARY KEY,
    CommonName VARCHAR(255) NOT NULL
);

CREATE TABLE SpeciesChangeLog (
    LogID INT AUTO_INCREMENT PRIMARY KEY,
    ScientificName VARCHAR(255) NOT NULL,
    CommonName VARCHAR(255) NOT NULL,
    ChangeInfo TEXT NOT NULL,
    ChangeDateTime TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

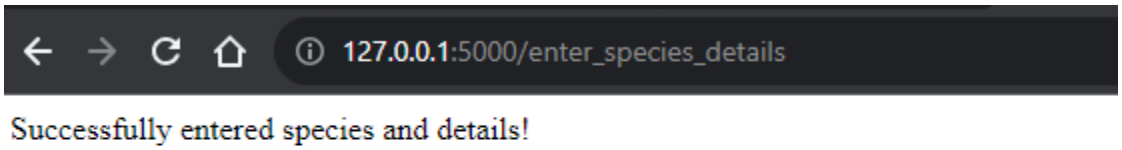
```
mysql> SHOW TABLES;
+-----------------------------+
| Tables_in_heritage_in_peril |
+-----------------------------+
| closestneighbour            |
| conservationefforts         |
| conservationlocation        |
| conservedat                 |
| conservedby                 |
| deletedspecies              |
| extinctspecies              |
| foundat                     |
| naturalhabitat              |
| species                     |
| specieschangelog            |
| threatenedby                |
| threats                     |
| user                        |
+-----------------------------+
14 rows in set (0.00 sec)

mysql>
```

Looking at the schema:



The tables match

With these fields, collaborators can enter values into the database by clicking the submit button



We can see the entries in the relevant tables:

```
mysql> SELECT * FROM SPECIES;
+-------------------------+----------------------+----------------+-----------------------------------------------------+-------------------------+
| ScientificName          | CommonName           | PopulationSize | Description                                         | EstimatedDateOfExtinction |
+-------------------------+----------------------+----------------+-----------------------------------------------------+-------------------------+
| Pseudocheirus occidentalis | Western Ringtail Possum |          15000 | A critically endangered possum species in Western Australia. | 2044-01-01              |
+-------------------------+----------------------+----------------+-----------------------------------------------------+-------------------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT * FROM NATURALHABITAT;
+-----------+------------------+----------+-----------+
| Country   | Region           | Latitude | Longitude |
+-----------+------------------+----------+-----------+
| Australia | WesternAustralia | -31.8229 | 116.0037  |
+-----------+------------------+----------+-----------+
1 row in set (0.00 sec)

mysql>
```

```
mysql> SELECT * FROM FOUNDAT;
+------------------------+-----------------------+-----------------------------+
| NaturalHabitatCountry | NaturalHabitatRegion | SpeciesScientificName      |
+------------------------+-----------------------+-----------------------------+
| Australia              | WesternAustralia     | Pseudocheirus occidentalis |
+------------------------+-----------------------+-----------------------------+
1 row in set (0.00 sec)

mysql>
```

```
mysql> SELECT * FROM ConservationEfforts;
+-----------------------------------------+-----------------------------------------------------+------------+------------+
| OrganizationName                        | ProjectDescription                                  | StartDate  | EndDate    |
+-----------------------------------------+-----------------------------------------------------+------------+------------+
| Western Australian Possum Conservation Society | Protection of remaining habitats and breeding programs. | 2023-07-01 | 2027-07-01 |
+-----------------------------------------+-----------------------------------------------------+------------+------------+
1 row in set (0.00 sec)

mysql>
```

And so on….

This is all enabled through the flask route that retrieves the information from the fields and inputs it into the database.

```
@app.route('/enter_species_details', methods=['GET', 'POST'])
def enter_species_details():
```

```
        if "user" not in session or session["user"] !=
"Collaborator":
        abort(403)  # Return a forbidden response if the user is
not a collaborator
    if request.method == 'POST':
        new_species_details = request.form

        scientific_name = new_species_details['scientificName']
        common_name = new_species_details['commonName']
        population_size = new_species_details['populationSize']
        description = new_species_details['description']
                                            extinction_date     =
new_species_details['estimatedExtinctionDate']
        country = new_species_details['country']
        region = new_species_details['region']
        latitude = new_species_details['latitude']
        longitude = new_species_details['longitude']
        threat_name = new_species_details['threatName']
                                          threat_description    =
new_species_details['threatDescription']
        severity = new_species_details['severity']
                                           organization_name    =
new_species_details['organizationName']
                                         project_description    =
new_species_details['projectDescription']
        start_date = new_species_details['startDate']
        end_date = new_species_details['endDate']
        cloclatitude = new_species_details['cloclatitude']
        cloclongitude = new_species_details['cloclongitude']
        cloccountry = new_species_details['cloccountry']
        clocregion = new_species_details['clocregion']
                                            conservationPark    =
new_species_details['conservationPark']

        cur = mysql.connection.cursor()
```

```
          # Check if the species already exists in the 'species'
table
           cur.execute("SELECT ScientificName FROM species WHERE
ScientificName = %s", (scientific_name,))
        existing_species = cur.fetchone()

        if existing_species:
            return "Species already exists!"

        # Insert into 'species' table
        cur.execute("""
                INSERT INTO species (ScientificName, CommonName,
PopulationSize, Description, EstimatedDateOfExtinction)
            VALUES (%s, %s, %s, %s, %s)
            """, (scientific_name, common_name, population_size,
description, extinction_date))


        # NaturalHabitat table
        cur.execute("""
                SELECT Country, Region FROM naturalhabitat WHERE
Country = %s AND Region = %s
        """, (country, region))
        existing_natural_habitat = cur.fetchone()
        if not existing_natural_habitat:
             cur.execute(""" INSERT INTO NaturalHabitat (Country,
Region, Latitude, Longitude)
                VALUES (%s, %s, %s, %s)
            """, (country, region, latitude, longitude))


        # Threats table
        cur.execute("""
            SELECT ThreatName FROM threats WHERE ThreatName = %s
```

```
        """, (threat_name,))
        existing_threat = cur.fetchone()
        if not existing_threat:
            cur.execute("""
                    INSERT INTO Threats (ThreatName, Description,
Severity)
                VALUES (%s, %s, %s)
            """, (threat_name, threat_description, severity))

        # ConservationEfforts table
        cur.execute("""
                SELECT OrganizationName FROM conservationefforts
WHERE OrganizationName = %s
        """, (organization_name,))
        existing_conservation_effort = cur.fetchone()
        if not existing_conservation_effort:
            cur.execute("""
                                INSERT INTO ConservationEfforts
(OrganizationName, ProjectDescription, StartDate, EndDate)
                VALUES (%s, %s, %s, %s)
                   """, (organization_name, project_description,
start_date, end_date))

        # ConservationLocation table
        cur.execute("""
                SELECT Country, Region FROM conservationlocation
WHERE Country = %s AND Region = %s
        """, (cloccountry, clocregion))
        existing_conservation_location = cur.fetchone()
        if not existing_conservation_location:
            cur.execute("""
                    INSERT INTO ConservationLocation (Latitude,
Longitude, Country, Region, ConservationPark)
                VALUES (%s, %s, %s, %s, %s)
```

```
                """, (cloclatitude, cloclongitude, cloccountry,
clocregion, conservationPark))


            cur.execute("CALL UpdateRelationTables(%s, %s, %s, %s,
%s, %s, %s, %s)",
                    (scientific_name, common_name, country, region,
threat_name,
                organization_name, cloccountry, clocregion))

            mysql.connection.commit()
            cur.close()
            return "Successfully entered species and details!"

    if "user" in session:
            dummy_data = get_random_dummy_data()
                return  render_template('enter_species_details.html',
dummy_data = dummy_data)
```

To make all of the many-to-many relationship tables in our database, a call is made to the `UpdateRelationTables()` procedure who's functionality has been described in an earlier section.

8. Read Operations

The read operations in our database is done through an interactive map of Australia that let's the user specify filters which include Country, Severity Level and a limit(that limits the number of species being displayed) which then calls a SELECT query in MySQL through Flask to obtain the relevant data

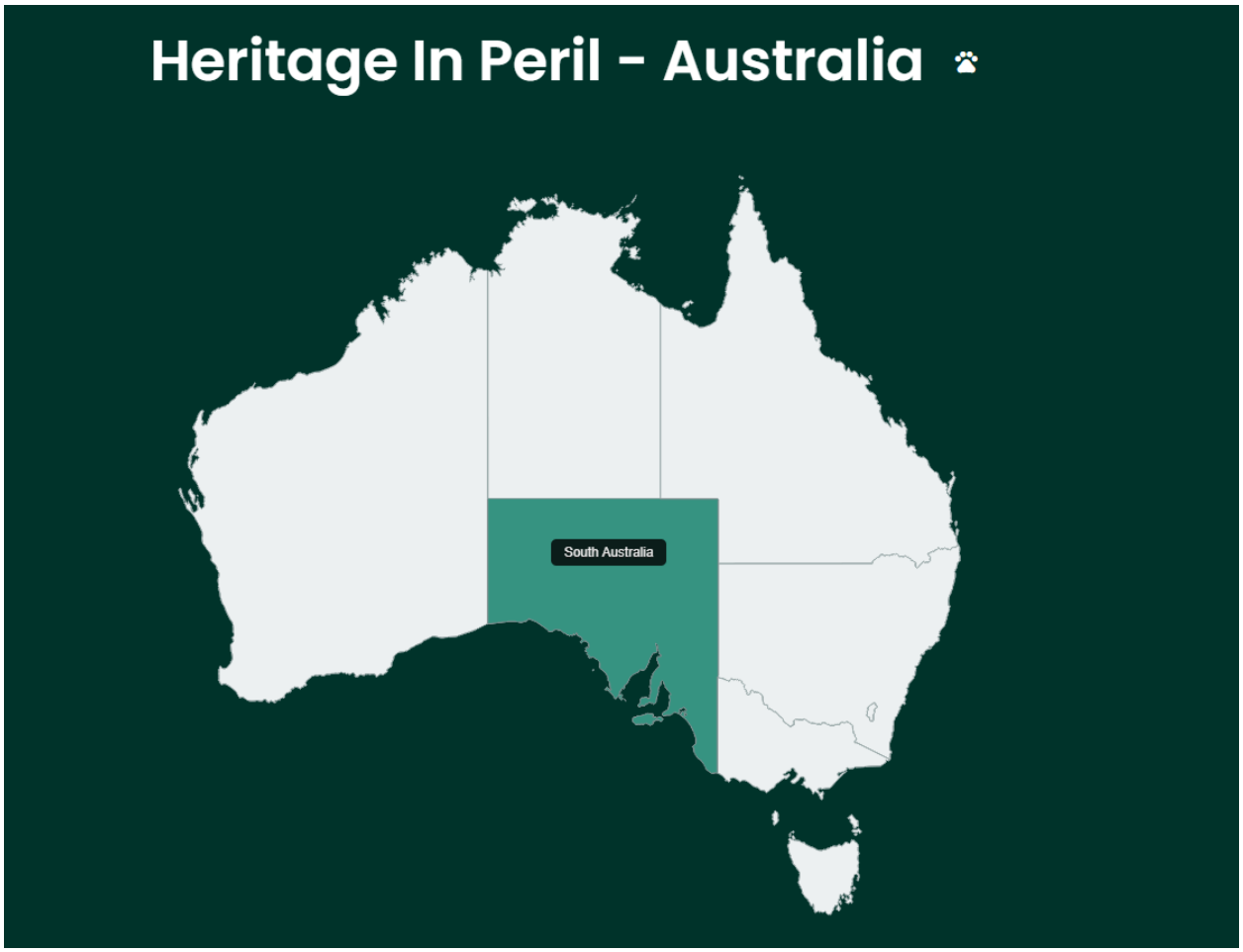
```
@app.route('/map_page', methods=['GET', 'POST'])
def map_page():
    if request.method == 'POST':
        new_species_details = request.form

        # Extract filter values from the form
        region_filter = new_species_details.get('region')
                                                severity_filter      =
int(new_species_details.get('conservation-status'))
        limit_filter = int(new_species_details.get('limit'))

        # Construct the WHERE clause based on filters
        conditions = []
        parameters = []

        if region_filter:
            conditions.append("nh.Country = %s")
            parameters.append(region_filter)
        if severity_filter:
            conditions.append("t.Severity = %s")
            parameters.append(severity_filter)

         # Construct the SQL query with dynamic WHERE clause and
placeholders
        filter_query = f"""
            SELECT
                s.ScientificName,
                s.CommonName,
```

```
                s.PopulationSize,
                s.Description,
                s.EstimatedDateOfExtinction,
                nh.Country AS HabitatCountry,
                nh.Region AS HabitatRegion,
                nh.Latitude AS HabitatLatitude,
                nh.Longitude AS HabitatLongitude,
                t.ThreatName,
                t.Description AS ThreatDescription,
                t.Severity AS ThreatSeverity
            FROM
                species s
                    JOIN FoundAt fa ON s.ScientificName =
fa.SpeciesScientificName
             JOIN NaturalHabitat nh ON fa.NaturalHabitatCountry =
nh.Country AND fa.NaturalHabitatRegion = nh.Region
                  JOIN ThreatenedBy tb ON s.ScientificName =
tb.SpeciesScientificName
            JOIN Threats t ON tb.ThreatName = t.ThreatName
             WHERE nh.Region = \"{region_filter}\" AND t.Severity
= {str(severity_filter)};
        """


        print("filter query:", filter_query)
        # Execute the query with the filter values
        cur = mysql.connection.cursor()
        cur.execute(filter_query)
        filtered_species = cur.fetchall()

        print("filtered species", filtered_species)
        cur.close()

        output_dict_list = [{
                    'ScientificName': data[0],
```

```
                    'CommonName': data[1],
                    'PopulationSize': data[2],
                    'Description': data[3],
                    'EstimatedDateOfExtinction': data[4],
                    'Country': data[5],
                    'Region': data[6],
                    'Latitude': data[7],
                    'Longitude': data[8],
                    'ThreatName': data[9],
                    'ThreatDescription': data[10],
                    'Severity': data[11]
                } for data in filtered_species]

        # Render the template with the filtered species data
                            return    render_template('map.html',
filtered_species=output_dict_list,      css_url=url_for('static',
filename='cssmap-australia/cssmap-australia.css'),
get_unsplash_image_url=get_unsplash_image_url)

    # If not a POST request, render the initial map.html
     return render_template('map.html', css_url=url_for('static',
filename='cssmap-australia/cssmap-australia.css'),
get_unsplash_image_url=get_unsplash_image_url)
```

The `filter_query` variable here handles the retrieval who's output is then passed to the HTML file for display:

A screenshot of the interactive map is shown here: You can highlight over any region and select that specific region



Apply Filters To Search Through Our Database:

Select Region:

SouthAustralia

Conservation Status:

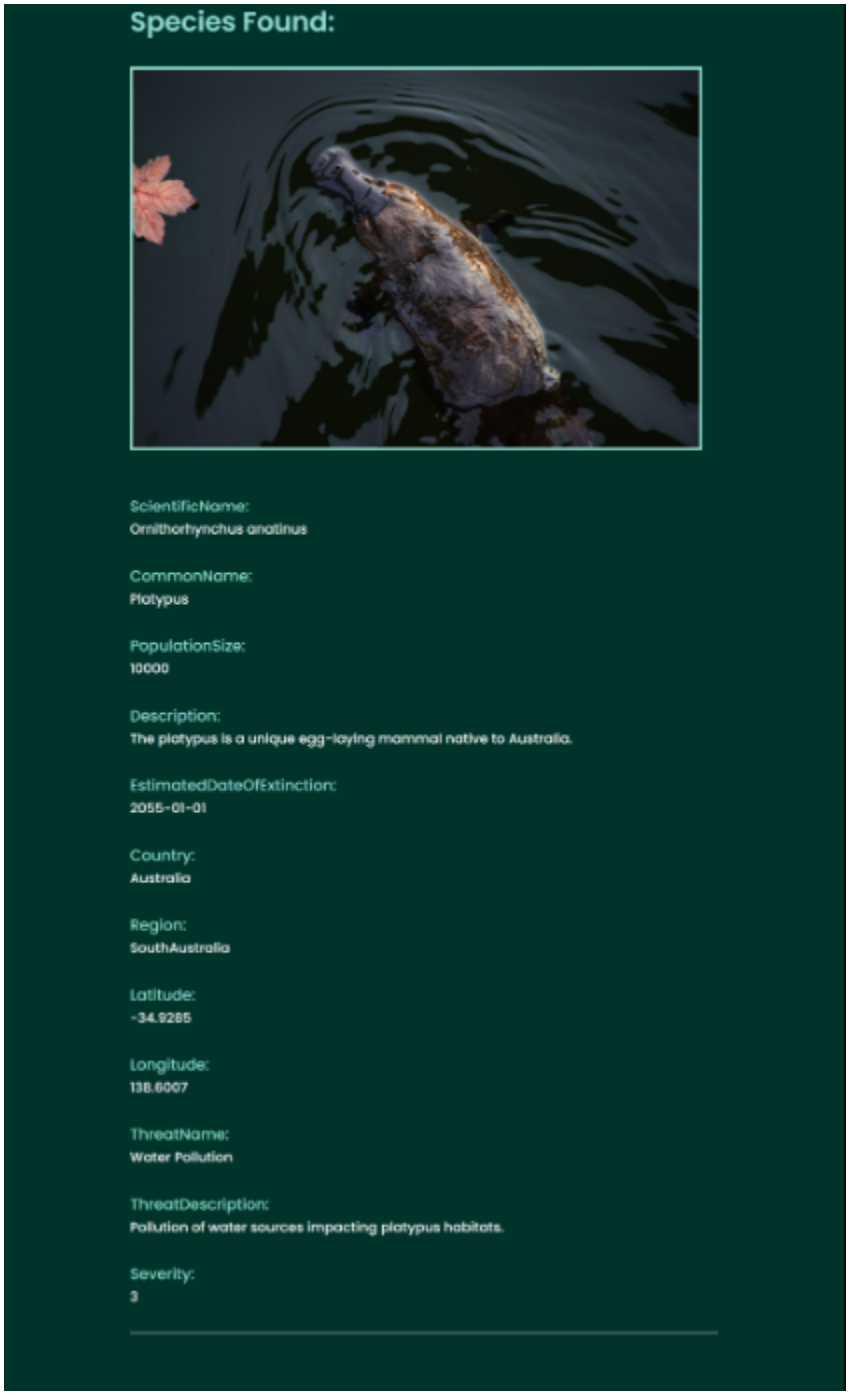
Vulnerable

Limit:

5

Submit

The region is then automatically entered into the filter and the user can adjust the other as necessary. On clicking submit, it displays the species within that region:

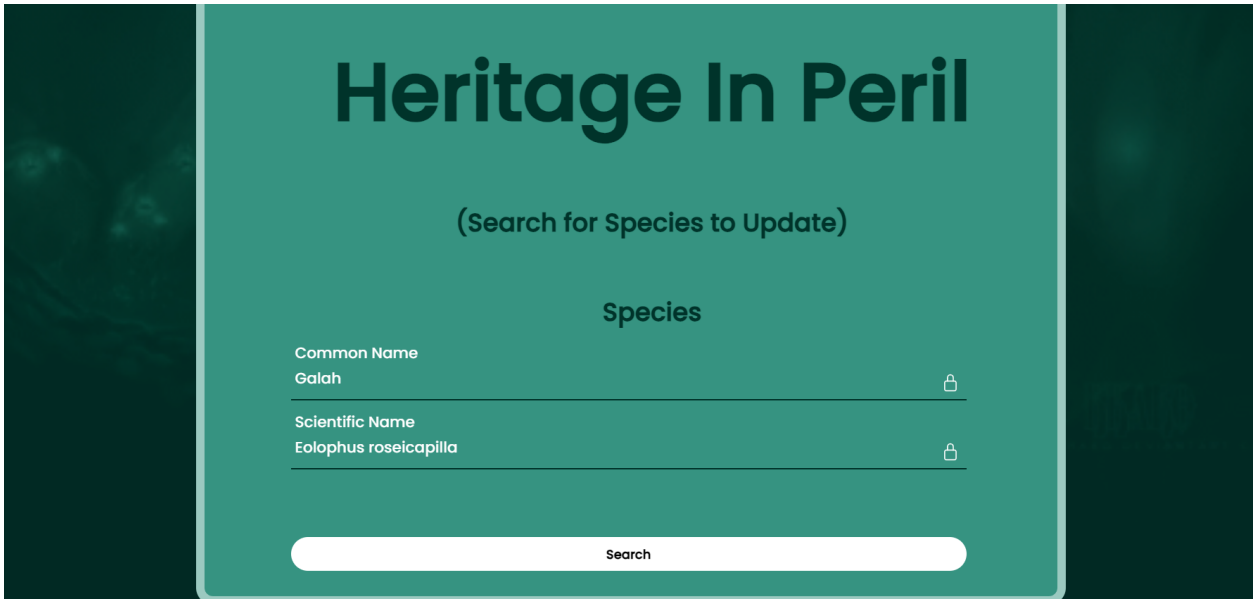


The images are obtained through an API that works with unsplash.

9. Update Operations

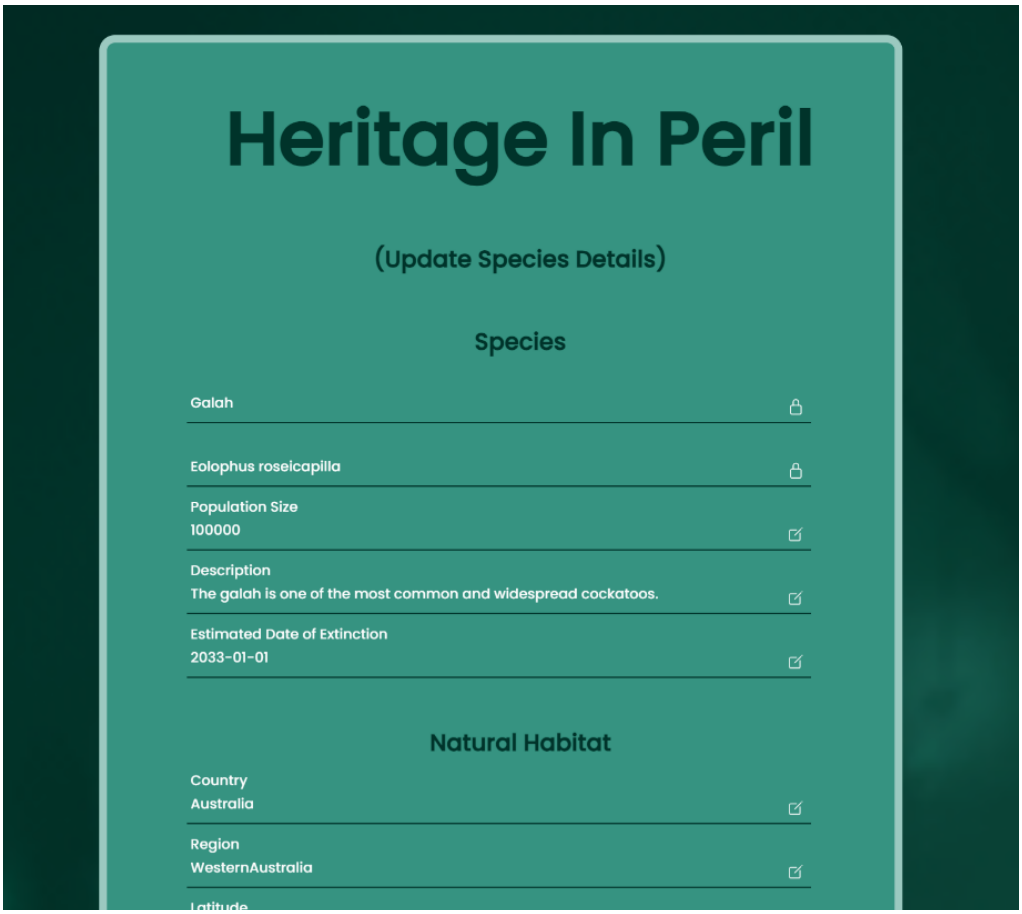
Through our website, we can edit the details of species that already exist in the database. We first look through the database to see if the species already exists. On finding a match in the database, we can update the details of that species.

We first search for the species (to see if it exists in the database)



We can then view the details of the species, as in the database. We can edit anything but the common name and the scientific name of the animal.

For example:

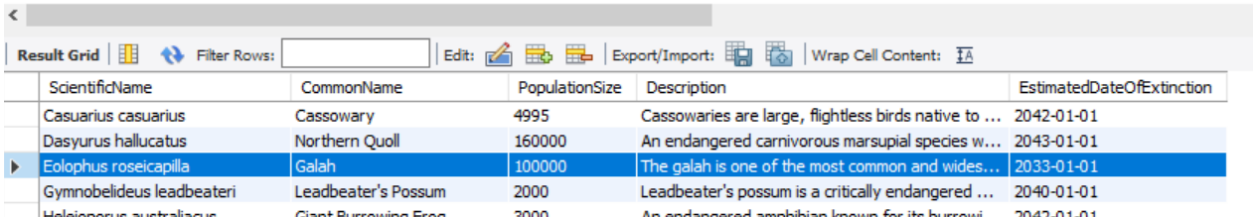


For the sake of this example, we are going to edit the description:

As we can see, the database is updated accordingly:

Before:

```
1 •  SELECT * FROM Species;
```

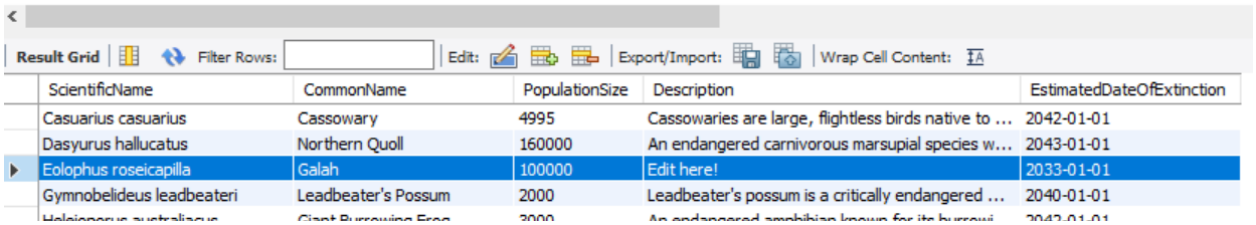


Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

ScientificName	CommonName	PopulationSize	Description	EstimatedDateOfExtinction
Casuarius casuarius	Cassowary	4995	Cassowaries are large, flightless birds native to ...	2042-01-01
Dasyurus hallucatus	Northern Quoll	160000	An endangered carnivorous marsupial species w...	2043-01-01
Eolophus roseicapilla	Galah	100000	The galah is one of the most common and wides...	2033-01-01
Gymnobelideus leadbeateri	Leadbeater's Possum	2000	Leadbeater's possum is a critically endangered ...	2040-01-01

After:

```
1 •  SELECT * FROM Species;
```



Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

ScientificName	CommonName	PopulationSize	Description	EstimatedDateOfExtinction
Casuarius casuarius	Cassowary	4995	Cassowaries are large, flightless birds native to ...	2042-01-01
Dasyurus hallucatus	Northern Quoll	160000	An endangered carnivorous marsupial species w...	2043-01-01
Eolophus roseicapilla	Galah	100000	Edit here!	2033-01-01
Gymnobelideus leadbeateri	Leadbeater's Possum	2000	Leadbeater's possum is a critically endangered ...	2040-01-01

The searching for the species in the database, and the updating of the species according to the new data, are done by the following routes in our flask applications (/search_for_species and /update_species_details):

```
@app.route('/search_for_update', methods=['GET', 'POST'])
def search_for_update():
        if "user" not in session or session["user"] != "Collaborator":
        abort(403)  # Return a forbidden response if the user is not a collaborator

    if request.method == 'POST':
```

```
        # Assuming form data is being submitted, process it here
        species_details = request.form
        commonName = species_details['commonName']
        scientificName = species_details['scientificName']

        print(commonName, scientificName)

        cur = mysql.connection.cursor()

          # Check if the species already exists in the 'species'
table
            cur.execute("SELECT ScientificName, CommonName FROM
species WHERE ScientificName = %s AND CommonName = %s",
(scientificName, commonName))
        existing_species = cur.fetchone()

        if existing_species:
                    global scientificNameOfTheSpeciesToUpdate,
commonNameOfTheSpeciesToUpdate
            scientificNameOfTheSpeciesToUpdate = scientificName
            commonNameOfTheSpeciesToUpdate = commonName
            return redirect(url_for('update_species_details'))
        else:
             return "Not editable! Species does not exist in the
database"

    if "user" in session:
        return render_template('search_for_update.html')


@app.route('/update_species_details', methods=['GET', 'POST'])
def update_species_details():
         if "user" not in session or session["user"] !=
"Collaborator":
```

```
        abort(403)  # Return a forbidden response if the user is
not a collaborator

    if request.method == 'POST':
        # Assuming form data is being submitted, process it here
        species_details = request.form

                    print('------species_details:  -------  \n',
species_details)

        # Extracting form data
        scientific_name = species_details['scientificName']
        common_name = species_details['commonName']
        population_size = species_details['populationSize']
        description = species_details['description']
                                    estimated_extinction_date    =
species_details['estimatedExtinctionDate']
        country = species_details['country']
        region = species_details['region']
        latitude = species_details['latitude']
        longitude = species_details['longitude']
        threat_name = species_details['threatName']
                                        threat_description     =
species_details['threatDescription']
        severity = species_details['severity']
        organization_name = species_details['organizationName']
                                       project_description     =
species_details['projectDescription']
        start_date = species_details['startDate']
        end_date = species_details['endDate']
        cloclatitude = species_details['cloclatitude']
        cloclongitude = species_details['cloclongitude']
        cloccountry = species_details['cloccountry']
        clocregion = species_details['clocregion']
        conservation_park = species_details['conservationPark']
```

```
        # Prepare the parameters for the stored procedure
        params = (
            common_name,
            scientific_name,
            int(population_size),
            description,
            estimated_extinction_date,
            country,
            region,
            float(latitude),
            float(longitude),
            threat_name,
            threat_description,
            int(severity),
            organization_name,
            project_description,
            start_date,
            end_date,
            float(cloclatitude),
            float(cloclongitude),
            cloccountry,
            clocregion,
            conservation_park
        )

        # Execute the stored procedure
        cur = mysql.connection.cursor()
        cur.callproc('UpdateSpeciesDetails', params)
        mysql.connection.commit()
        cur.close()
          return 'update successful!'   # Return a response after
processing

    if "user" in session:
```

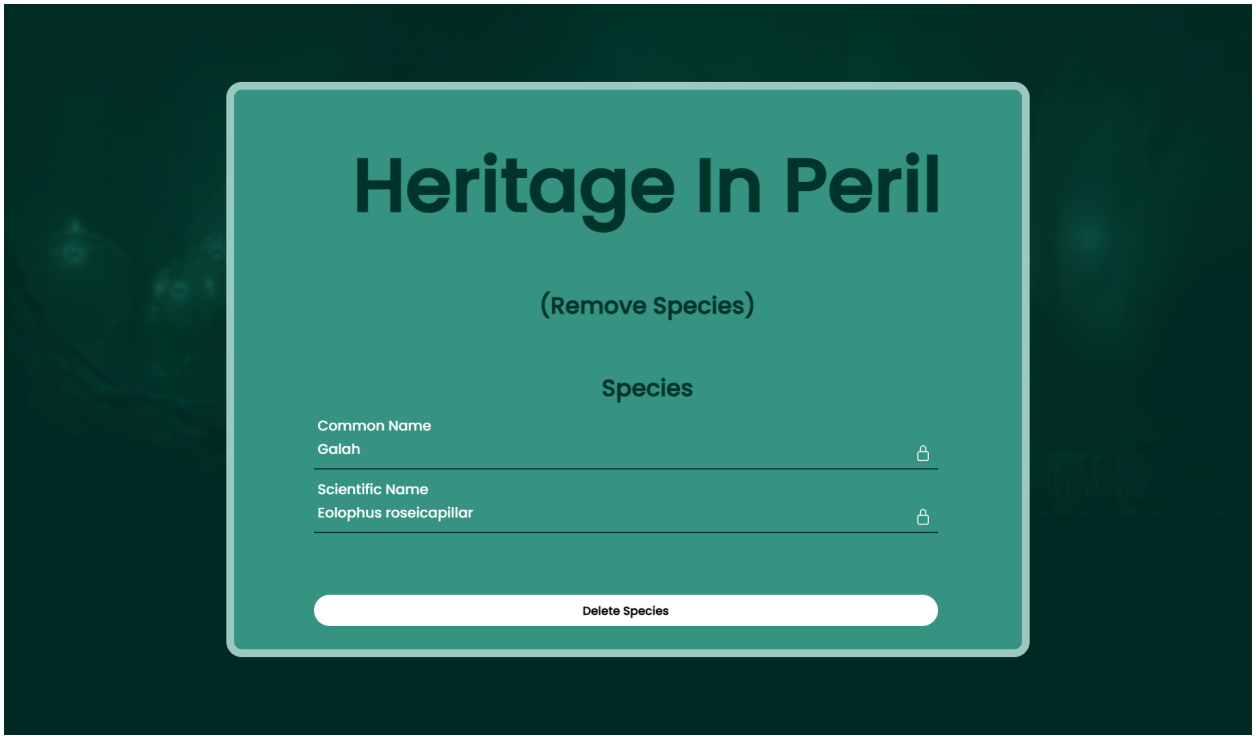
```
        scientific_name = scientificNameOfTheSpeciesToUpdate
        common_name = commonNameOfTheSpeciesToUpdate

        cur = mysql.connection.cursor()

        cur.execute(
            """
                        SELECT s.CommonName, s.ScientificName,
s.PopulationSize, s.Description, s.EstimatedDateOfExtinction,
               nh.Country, nh.Region, nh.Latitude AS nh_latitude,
nh.Longitude AS nh_longitude,
               t.ThreatName, t.Description AS threat_description,
t.Severity,
                     ce.OrganizationName, ce.ProjectDescription,
ce.StartDate, ce.EndDate,
                   cl.Latitude AS cl_latitude, cl.Longitude AS
cl_longitude,   cl.Country   AS   cloccountry,   cl.Region   AS
clocregion,
            cl.ConservationPark
            FROM species s
                   LEFT JOIN FoundAt fa ON s.ScientificName =
fa.SpeciesScientificName
                              LEFT JOIN NaturalHabitat nh ON
fa.NaturalHabitatCountry = nh.Country AND
fa.NaturalHabitatRegion = nh.Region
                LEFT JOIN ThreatenedBy tb ON s.ScientificName =
tb.SpeciesScientificName
            LEFT JOIN Threats t ON tb.ThreatName = t.ThreatName
                 LEFT JOIN ConservedBy cb ON s.ScientificName =
cb.SpeciesScientificName
                          LEFT JOIN ConservationEfforts ce ON
cb.OrganizationName = ce.OrganizationName
               LEFT JOIN ConservedAt ca ON ce.OrganizationName =
ca.OrganizationName
```

```
                    LEFT JOIN ConservationLocation cl ON
ca.ConservationLocationCountry = cl.Country AND
ca.ConservationLocationRegion = cl.Region
            WHERE s.ScientificName = %s AND s.CommonName = %s
            """, (scientific_name, common_name)
        )

        species_data = cur.fetchone()

        print(species_data)

        output_dict = {
            'commonName': species_data[0],
            'scientificName': species_data[1],
            'populationSize': species_data[2],
            'description': species_data[3],
            'estimatedExtinctionDate': species_data[4],
            'country': species_data[5],
            'region': species_data[6],
            'latitude': species_data[7],
            'longitude': species_data[8],
            'threatName': species_data[9],
            'threatDescription': species_data[10],
            'severity': species_data[11],
            'organizationName': species_data[12],
            'projectDescription': species_data[13],
            'startDate': species_data[14],
            'endDate': species_data[15],
            'cloclatitude': species_data[16],
            'cloclongitude': species_data[17],
            'cloccountry': species_data[18],
            'clocregion': species_data[19],
            'conservationPark': species_data[20]
        }
```

```
        print(output_dict)
            return  render_template('update_species_details.html',
species_data = output_dict)
```

10. Delete Operations

We can delete a species from the database, too. This is done by entering the common name and the scientific name of the species to be eliminated from the database. The eliminated species are added into the deletedSpecies table (in case we need them for future reference).



The species is removed from the Species table: (we cannot see the Galah here, anymore)

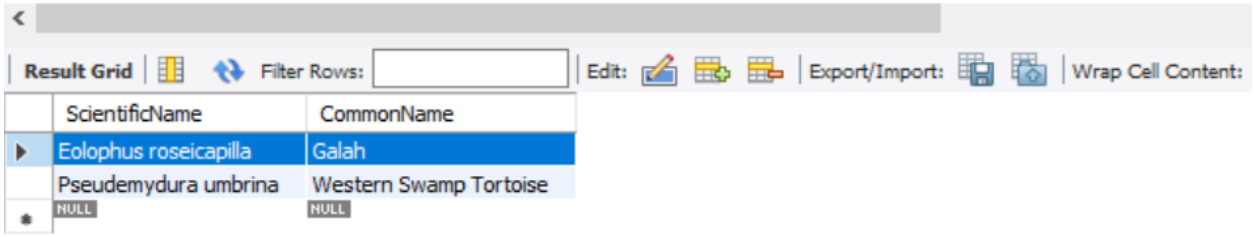
```
1 •  SELECT * FROM Species;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

ScientificName	CommonName	PopulationSize	Description	EstimatedDateOfExtinction
Casuarius casuarius	Cassowary	4995	Cassowaries are large, flightless birds native to ...	2042-01-01
Dasyurus hallucatus	Northern Quoll	160000	An endangered carnivorous marsupial species w...	2043-01-01
Gymnobelideus leadbeateri	Leadbeater's Possum	2000	Leadbeater's possum is a critically endangered ...	2040-01-01
Heleioporus australiacus	Giant Burrowing Frog	3000	An endangered amphibian known for its burrowi...	2042-01-01
Megascolides australis	Giant Gippsland Earthworm	2000	The giant Gippsland earthworm is one of the lar...	2043-01-01
Miniopterus schreibersii bassanii	Southern Bent-Winged Bat	78000	An endangered microbat species with distinctive...	2045-01-01

The species is entered into the deletedSpecies table:

```
1 •  SELECT * FROM deletedSpecies;
```



Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

ScientificName	CommonName
Eolophus roseicapilla	Galah
Pseudemydura umbrina	Western Swamp Tortoise
NULL	NULL

The /delete_species route in the flask application enables this delete operation.

```
@app.route('/delete_species', methods=['GET', 'POST'])
def delete_species():
         if  "user"  not  in  session  or  session["user"]  !=
"Collaborator":
         abort(403)  # Return a forbidden response if the user is
not a collaborator

    if request.method == 'POST':
        # Assuming form data is being submitted, process it here
        species_details = request.form
        commonName = species_details['commonName']
        scientificName = species_details['scientificName']

        print(commonName, scientificName)

        cur = mysql.connection.cursor()

          # Check if the species already exists in the 'species'
table
```

```
            cur.execute("SELECT ScientificName, CommonName FROM
species WHERE ScientificName = %s AND CommonName = %s",
(scientificName, commonName))
        existing_species = cur.fetchone()

        if existing_species:
            # Call the stored procedure to delete the species
                cur.callproc('DeleteSpeciesEntries', (commonName,
scientificName))
            mysql.connection.commit()
            return "Species deleted successfully!"
        else:
                    return "Cannot delete. Species not found in
database."

    if "user" in session:
        return render_template('delete_species.html')
```

11. Queries

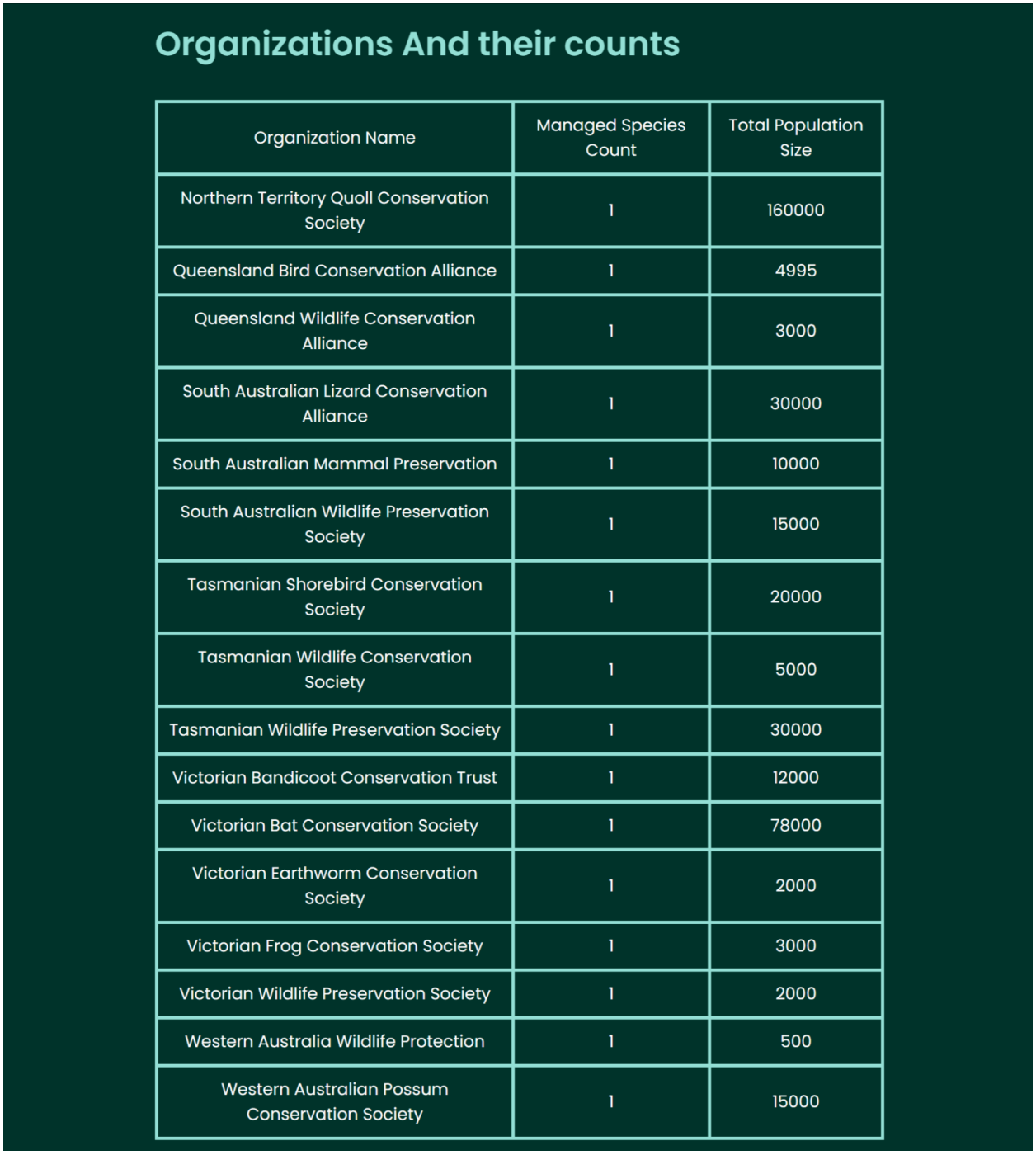
All of our application queries are done on a separate webpage /analytics where we display an example of aggregate, nested and one join query.
Similar to the read operations page(map.html), they can select a few filters to display some analytics of the data within our database:

a. Aggregate Queries:

The below query executed will obtain a table of all conservation organizations, the number of species they conserve and the total population of all species they manage:

```
query = """
SELECT
    c.OrganizationName,
                COUNT(DISTINCT c.SpeciesScientificName) AS
ManagedSpeciesCount,
    SUM(s.PopulationSize) AS TotalPopulationSize
FROM
    ConservedBy c
JOIN
                species s ON c.SpeciesScientificName =
s.ScientificName
GROUP BY
    c.OrganizationName;
 """
```

GUI Screenshot:



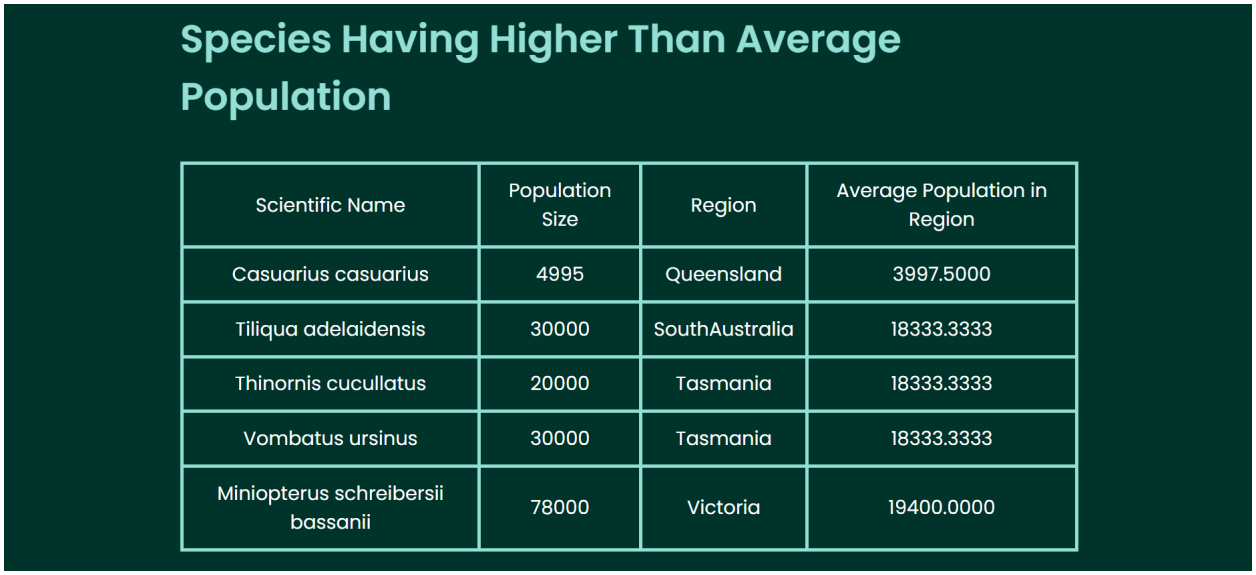
b. Correlated Subquery:

This is an example of a correlated subquery that fetches the species that have higher than average population within their respective regions

```
        query = f"""
        SELECT
            s.ScientificName,
            s.PopulationSize,
            nh.Region,
            (
                SELECT AVG(s2.PopulationSize)
                FROM species s2
                        JOIN FoundAt f2 ON s2.ScientificName =
f2.SpeciesScientificName
                                    JOIN NaturalHabitat nh2 ON
f2.NaturalHabitatCountry = nh2.Country AND
f2.NaturalHabitatRegion = nh2.Region
                WHERE nh2.Region = nh.Region
            ) AS AvgPopulationInRegion
        FROM
            species s
        JOIN
                            FoundAt f ON s.ScientificName =
f.SpeciesScientificName
        JOIN
                 NaturalHabitat nh ON f.NaturalHabitatCountry =
nh.Country AND f.NaturalHabitatRegion = nh.Region
        WHERE
            s.PopulationSize > (
                SELECT AVG(s2.PopulationSize)
                FROM species s2
                        JOIN FoundAt f2 ON s2.ScientificName =
f2.SpeciesScientificName
                                    JOIN NaturalHabitat nh2 ON
f2.NaturalHabitatCountry = nh2.Country AND
f2.NaturalHabitatRegion = nh2.Region
```

```
            WHERE nh2.Region = nh.Region
        );
    """
```

GUI Screenshot:



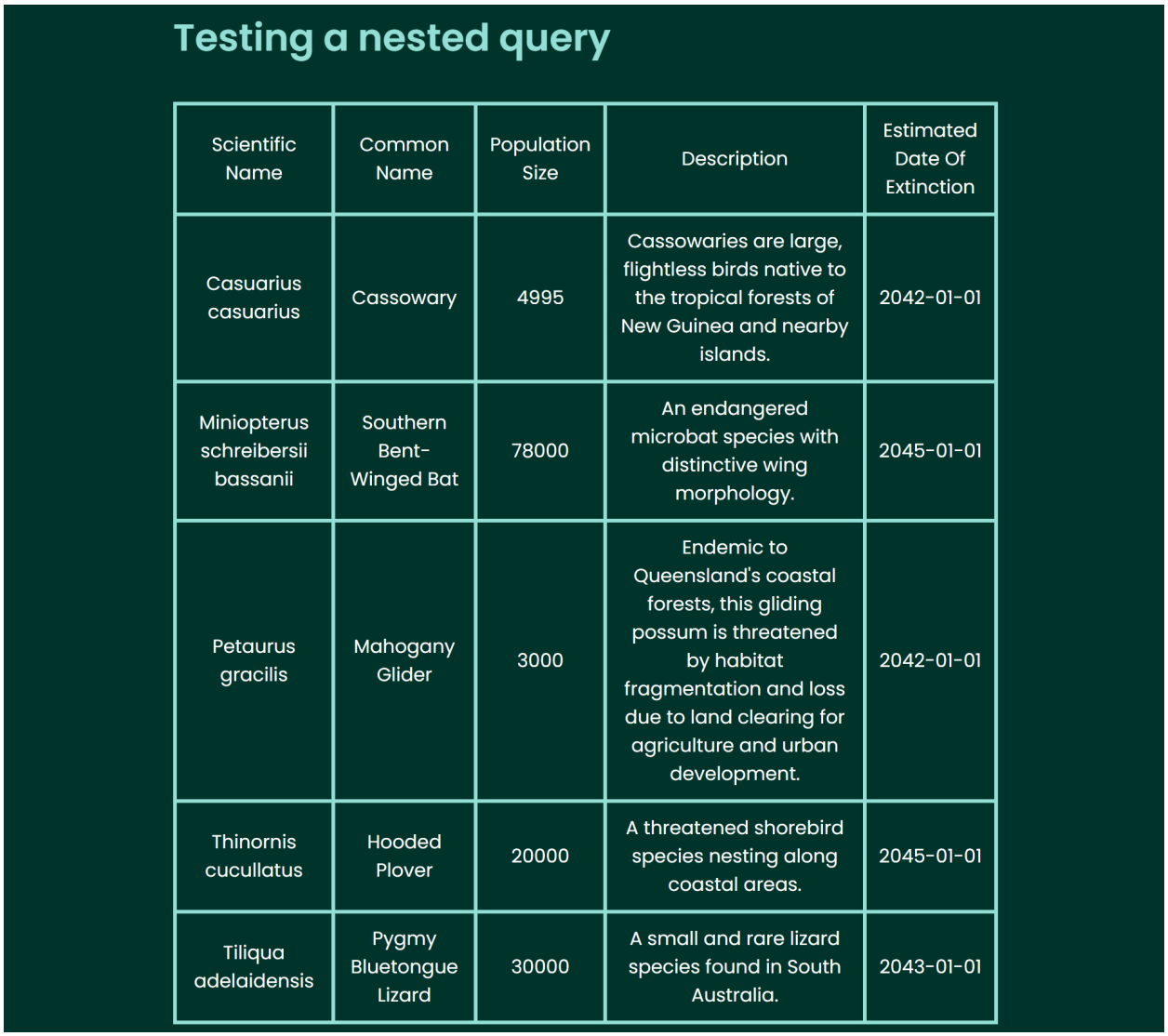
c. Nested Query

The below SQL query retrieves information about species meeting specific criteria from the species table. The main query selects species that are not conserved in a designated region, possess a positive population size, face a threat with a specified severity, and have an estimated date of extinction within a specified range. Two subqueries are employed: one checks for the absence of conservation records for the species in the specified region, while the other ensures the species is threatened by a specified severity of threat. These conditions collectively filter the results to include only those species meeting the outlined conservation, population, and threat criteria.

```
query = f"""
    SELECT
        s.ScientificName,
        s.CommonName,
```

```
                s.PopulationSize,
                s.Description,
                s.EstimatedDateOfExtinction
            FROM
                species s
            WHERE
                NOT EXISTS (
                    SELECT 1
                    FROM ConservedBy c
                                JOIN ConservationLocation cl ON
c.OrganizationName = cl.ConservationPark
                    WHERE
                                        c.SpeciesScientificName =
s.ScientificName
                        AND cl.Region = \"{region_filter}\"
                )
                AND s.PopulationSize > 0 -- Adjust as needed
                     AND s.EstimatedDateOfExtinction BETWEEN
CURDATE() AND '{date_filter}'
                AND s.ScientificName IN (
                    SELECT tb.SpeciesScientificName
                    FROM ThreatenedBy tb
                             JOIN Threats th ON tb.ThreatName =
th.ThreatName
                    WHERE
                        th.Severity = {severity_filter}
                );
        """
```

GUI screenshot:



Conclusion

In essence, “Heritage In Peril” serves as a fun interactive website with a robust database integration to help discover many endangered and extinct wildlife within Australia’s vast natural ecosystem. As urbanization continues to grow exponentially, necessary steps need to be taken in order to raise awareness within our communities to take preventive measure against the cruel fate of elimination that many exotic species in our beguiling world are threatened by and the best way to start is to aggregate the available data and centralize it in order to establish an informative knowledge-pool. Furthermore, the scalability of this database can be justified by both its benignity and the ability to work with Zoology departments in universities to obtain data that might not be publicly disclosed to prevent human intervention in their survival.

The scability of our project can include:

1. Database Optimization: Ensure that database queries are optimized for performance. Indexing, proper use of keys, and efficient query design can significantly impact the system's scalability.
2. Horizontal and Vertical Scaling: The chosen DBMS, MySQL, should support both horizontal and vertical scaling. Horizontal scaling involves adding more servers to distribute the database load, while vertical scaling involves upgrading the server hardware. The chosen solution should allow flexibility in scaling based on the increasing demand for data storage and processing.
3. Sharding and Partitioning: Implementing sharding or partitioning techniques within the MySQL database can enhance scalability. This involves breaking down the database into smaller, manageable parts (shards or partitions) to distribute data across multiple servers. It helps alleviate the load on individual servers and enhances overall performance.
4. Connection Pooling: Implementing connection pooling techniques helps manage and reuse database connections, reducing the overhead of establishing new connections for each user request. This improves scalability by efficiently utilizing available resources.