Final Project Report: Database for Wildlife Conservation

Brody Soedel, Harshitha Komaravelli, Vinh Nguyen, Dorian Sommerfeld

CSS 475

Table of Contents

1 Concept and Application

The proposal of our final project is to create a local wildlife conservation database. This database will hold information about local species, their estimated population, their conservation status, and local habitat information and range.

This data will help conservationists monitor population trends, identify unstable populations, and also can be used by general educators or the public to learn more about local wildlife. For example, a user could pull all the population data from 2018 to the most recent population survey for coyotes in the Bothell area to find out how the population has changed over time.

Another user case could pull all animals listed as endangered or vulnerable by the IUCN in the database to see where the local populations are located. A researcher user case would find all animals belonging to the kingdom *Mammalia* in the database in order to see what mammal populations exist in the local area, to analyze the population distributions of specific groups.

In essence, these are the possible operational tasks the application can perform:

- Local species and their scientific classifications
- Population trends over time
- Conservation status (safe, critical, etc.)
- Geographic and habitat distribution/range
- Survey and research data
- Specific information filter (date range, common name, class id)2 Design

2.1 Assumptions

The requisite assumptions for the database system is as follows:

- One organization can make many population surveys.
- A population survey can record many species, but only one location.
- A location can have many animals and population surveys.
- An animal can only belong to one species and location.
- A species can consist of many animals, be in many population surveys, and live in many different habitats.
- A habitat can contain many different species.

2.2 Entity Relationship (ER) Diagram

We designed our database with an ER diagram (Figure 1) below, by identifying several key entities that we felt were needed to be represented in our diagram. Because our database was a

local conservation database, we needed to track populations of animals and tagged animals, so those became entities.

In terms of the actual data represented by the entities, we are representing individual surveys and dates where an animal was tagged. The population survey entity represents a dated survey of a monitored population of wildlife monitored in the database, which is represented by "population". Species was added because it would make population surveys and tagged animals easier to search and sort by species, including by their specific taxonomy. We included habitat so some idea of a species preferred habitat would be included for background information on the animal's environmental preferences.

Additionally, we separated location into its own entity after having it as a separate attribute for both animal and population survey, and added an organization entity to represent wildlife organizations responsible for surveying the population and contributing the data we are representing. Lastly, it's important to point out that the LOCATION entity only stores the initial tagging location for the animal, and does not accurately track the animal's movement over time. This is a limitation of our current database design, which means we are unable to monitor the animal's location if it migrates or moves to a different location. Incorporating additional movement data would require expanding our data structure to include tracking information, which isn't currently supported in this current iteration so far.

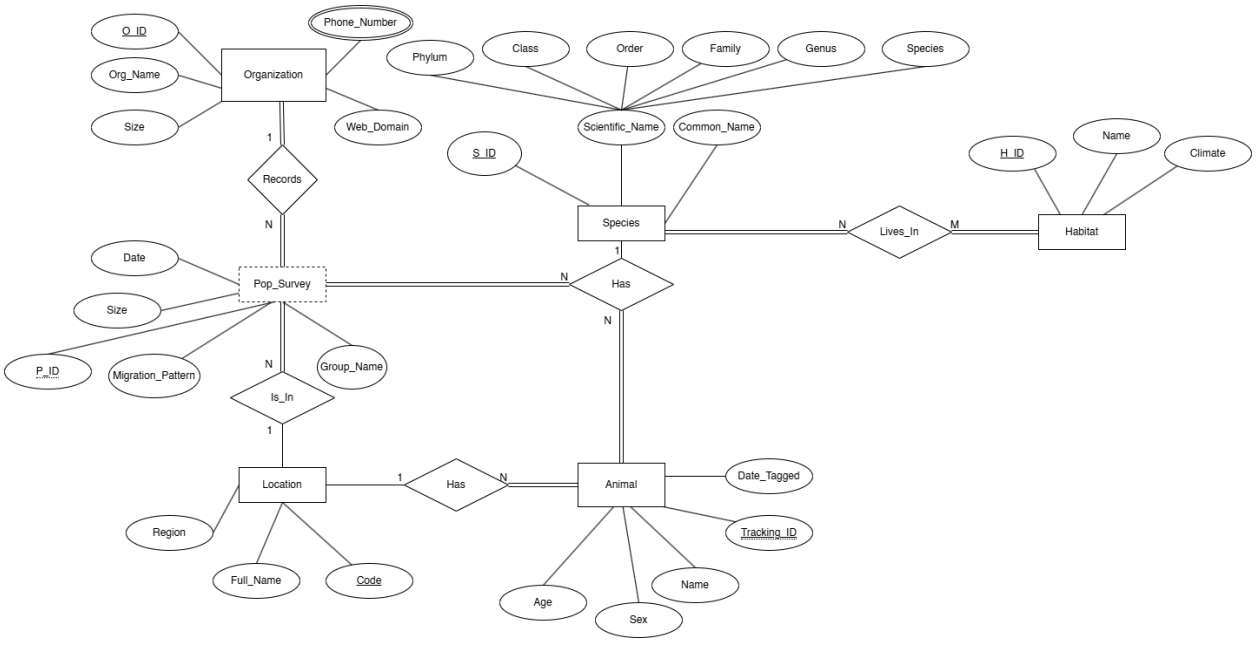

Figure 1: ER Diagram.

2.2.1 Constraints

Organization

- O_ID
    - Primary Key.

    - Cannot be NULL.
    - Series of 8 numbers that form a unique identifier for an organization in the database.
- Org_Name
    - 64-character alphanumeric string.
    - Cannot be NULL.
    - Identifies the name of the organization recording the population survey.
- Size
    - Cannot be NULL.
    - Non-zero integers.
    - Best estimation of an organization's head counts.
- Phone_Number
    - Can be NULL if the organization utilizes other communication means.
    - 10-digit non-negative integers (fixed size).
- Web_Domain
    - Cannot be NULL.
    - 64-character address to the organization's main website page.

Population

- P_ID
    - Primary Key
    - Cannot be NULL.
    - Series of 8 numbers that form a unique identifier for the population survey.
- Migration_Pattern
    - Can be NULL if there's no migration pattern.
    - 256 alphanumeric character string that briefly describes migration pattern of population
- Pop_Name
    - Cannot be NULL.

    - 64-character alphanumeric string that identifies the name of population group

Pop_Survey

- Date
    - UNIX timestamp of when population was last tracked
    - Can be NULL if data is missing
- Size
    - Non-negative integer value.
    - Estimated size of population
    - Can be NULL.

Location

- Code
    - Primary Key
    - Cannot be NULL.
    - Unique 8-character alphanumeric string identifier that identifies a certain local location.
- Region
    - Cannot be NULL.
    - 64-character alphabetical string that identifies which region of Washington location is in.
- Full_Name
    - Cannot be NULL.
    - 64-character alphanumeric string that identifies the specific/full legal name of the location.

Species

- S_ID
    - Primary Key
    - Cannot be NULL.
    - Series of 8 numbers that form a unique identifier for a species in the database.

- Scientific_Name
    - Contains the attributes: Phylum, Class, Order, Family, Genus, and Species.
    - Cannot be NULL.
    - Each attribute consists of a 32-character alphabetical string that signifies animal's classification in each taxon.
- Common_Name
    - Cannot be NULL.
    - 64-character alphabetical string that is the common name of the species.

Animal

- Tracking_ID
    - Primary Key
    - Cannot be NULL
    - Unique 8-digit tracking number that identifies a tagged animal.
- Age
    - Can be NULL if the age is not identified/identifiable.
    - Non-zero integers.
    - Best estimation of the animal's age in year at the time of tagging.
- Name
    - 32-character alphanumeric string.
    - Can be NULL if the animal is not named.
    - Identifies the name of the tagged animal, if one is recorded.
- Date Tagged
    - UNIX timestamp of when the animal was tagged to the nearest half hour.
    - Can be NULL if the tagging date was not recorded.
- Sex
    - Can be NULL if sex is not identified/identifiable (for animals that have a binary sex).
    - Consists of three possible strings: {"Male", "Female", "N/A"}

Habitat

- H_ID
    - Primary Key.
    - Cannot be NULL.
    - 8-digit ID number that identifies a certain type of habitat in the database.
- Name
    - Cannot be NULL.
    - 64-character alphabetical string that describes a specific habitat.
- Climate
    - Cannot be NULL.
    - 64-character alphabetical string that describes a specific climate.

2.3 Relational Model (RM) Diagram

The RM diagram below (Figure 2), is converted based on the ER diagram above. Two new tables are created aside from the entities in our ER diagram to portray the M:N cardinality in the Lives_In relationship and also the multi-valued Phone_Numbers attribute. For our domains we came up with two main domains D_{name} and D_{int} which represented a generic string and unsigned integer respectively. We have also implemented a Survey_Animal as a ternary relationship among Population, Species, and Animal if one were to query which animals identified from which survey. From these domains we derived other domains to fit the constraints that we laid out previously in our ER diagram.

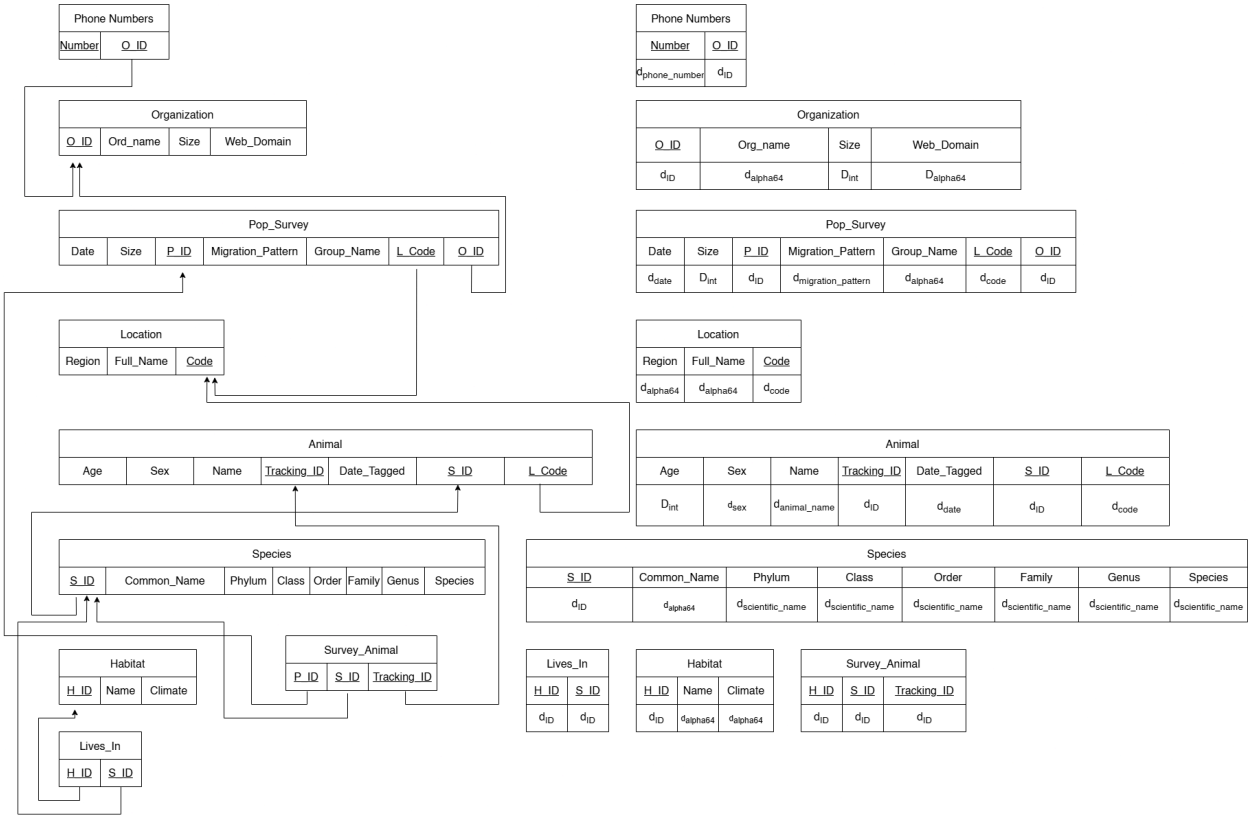

Figure 2: RM Diagram.

2.3.1 RM Assumptions

Lives_In primary key is a combination of both its foreign keys.

Survey_Animal primary key is a combination of all 3 of its foreign keys.

Pop_Survey primary key is a combination of date and the foreign key P_ID

2.4 Domains:

D_{name}: String, Can be NULL

D_{int}: Unsigned Integer, Can be NULL

2.5 Subdomains:

$D_{int} \subset d_{date}$: UNIX timestamp, can be NULL

$D_{name} \subset d_{webdomain}$: 64 character string, cannot be NULL

$D_{int} \subset d_{ID}$: Integer ranging from 0 to 99,999,999 (8 digits), cannot be NULL

$D_{name} \subset d_{alpha\_numeric64}$: 64 character alphanumeric string, cannot be NULL

$D_{int} \subset d_{phone\_number}$: Ranging from 1000000000 to 9999999999 (10 digits) can be NULL

$D_{name} \subset d_{migration\_pattern}$: 256 character length string, Can be NULL

$D_{name} \subset d_{code}$:8 character alpha-numeric string, cannot be NULL

$D_{name} \subset d_{scientific\_name}$: 32 alphabet character string, cannot be NULL

$D_{name} \subset d_{animal\_name}$: 32 alphabet character string, can be NULL

$d_{name} \subset d_{sex}$: Consisting of {“Male”, “Female”, “N/A”}, can be NULL

$D_{name} \subset d_{alpha64}$: 64 alphabet character string, cannot be NULL

2.6 Possible Concerns & Limitations:

There is a concern with the ternary relationship between Animal, Population Survey, and Species; Does the three aforementioned variables not convey relations well enough?

One limitation of our database is that Location only tracks the initial tagging location of an animal. There are possibilities that an animal will move to different locations throughout its lifespan, thus additional tracking systems would be required, and somehow be implemented in the future iterations.

Another limitation is that the accuracy of the population survey (and the database as a whole) depends on external organization, meaning there will naturally be inconsistencies in data collection methods across different sources that we will be using.

3 Normalization

After establishing the ER diagram and Relational Model, we proceed with normalization to identify and address any sub optimality in the design. The objective of normalization is to achieve the third normal form (3NF), which ensures that the database structure is efficient and free from redundancy. To achieve this, we must satisfy the conditions of 1NF (i.e., atomic attributes, and all relation must have a Primary Key), 2NF (i.e., no partial dependency), and no transitive dependency (non-prime attribute should depend only on the Primary Key).

ORGANIZATION

- O_ID -> Org_Name
- O_ID -> Size
- O_ID -> Web_Domain

Organization is in third normal form because every attribute is dependent on the key and there are no transitive dependencies.

PHONE_NUMBERS

- Number -> O_ID

PHONE_NUMBERS is already in 3rd normal form because every attribute relies on the primary key and there are no transitive dependencies.

LOCATION

- Code -> Full_Name
- Code -> Region

Location is in Third normal form because every attribute relies on the full primary key, which is Code.

POPULATION

- P_ID -> Migration_Pattern
- P_ID -> Size
- P_ID -> L_Code

Population is in Third Normal Form because every attribute relies on the full primary key, which is P_ID.

POP_SURVEY

- P_ID, Date -> Size
- P_ID, Date -> O_ID

POP_SURVEY is in Third normal form because every attribute relies on the full primary key, which is P_ID and Date.

SPECIES

- S_ID -> Common_Name
- S_ID -> Phylum
- S_ID -> Class
- S_ID -> Order
- S_ID -> Family
- S_ID -> Genus
- S_ID -> Species

POP_SURVEY is in third normal form because of its lack of transitive dependencies and every attribute depending on the primary key.

ANIMAL

- Tracking_ID -> Age

- Tracking_ID -> Name
- Tracking_ID -> Date_Tagged
- Tracking_ID -> Sex

Animal is in third normal form because every attribute is dependent on the primary key, which is Tracking_ID.

HABITAT

- H_ID -> Name
- H_ID -> Climate

Habitat is already in third normal form because every attribute depends on the primary key, and there are no transitive dependencies.

SURVEY_ANIMAL

- P_ID -> S_ID
- S_ID -> Tracking_ID

Survey animal is in second normal form because the Species id could be dependent on the tracking id, which creates a transitive dependency.

LIVES_IN

- H_ID -> S_ID

LIVES_IN is already in third normal form because every attribute relies on the primary key, and every attribute in here is a candidate key.

4 Tooling Assessment:

For our DBMS, MySQL is shown to be the best for small projects, which compared to other larger projects, is great for what we are doing. The community version is free to download. Another free program is GitHub, which we will be using for Version Control; many of us are already familiar with Github, and it is very handy when it comes to tracking different versions of our project. For hosting, Azure is free for students and is a fairly popular software among organizations, and we believe that it will synergize well with our SQL workflow. For UI, we will use REACT as some of us have had prior experience with the program, and we have found it to be very user-friendly (an important factor in our selection), making the integration of it into our project seamless.

5 Updates to ER & RM Model

Nearing the database creation stage of the project, we have made revisions to the ER diagram and RM assumptions to better aid the goal of tracking populations of animals over time. This change involved adding another table called population that made population survey a weak entity in relation to it. We then distributed the attributes from population survey to population

and population survey. This change was done because in the previous model, multiple population surveys were not able to be linked to the same populations of animals, only to the same locations. This proves to be a problem, if for example there were multiple bald eagle populations being tracked in the location of “Skagit River Valley”. So we made it so populations were defined in the database as their own table and population surveys were taken of populations, meaning over time population surveys of different sizes can be retrieved from the database to examine how a population has changed over time.

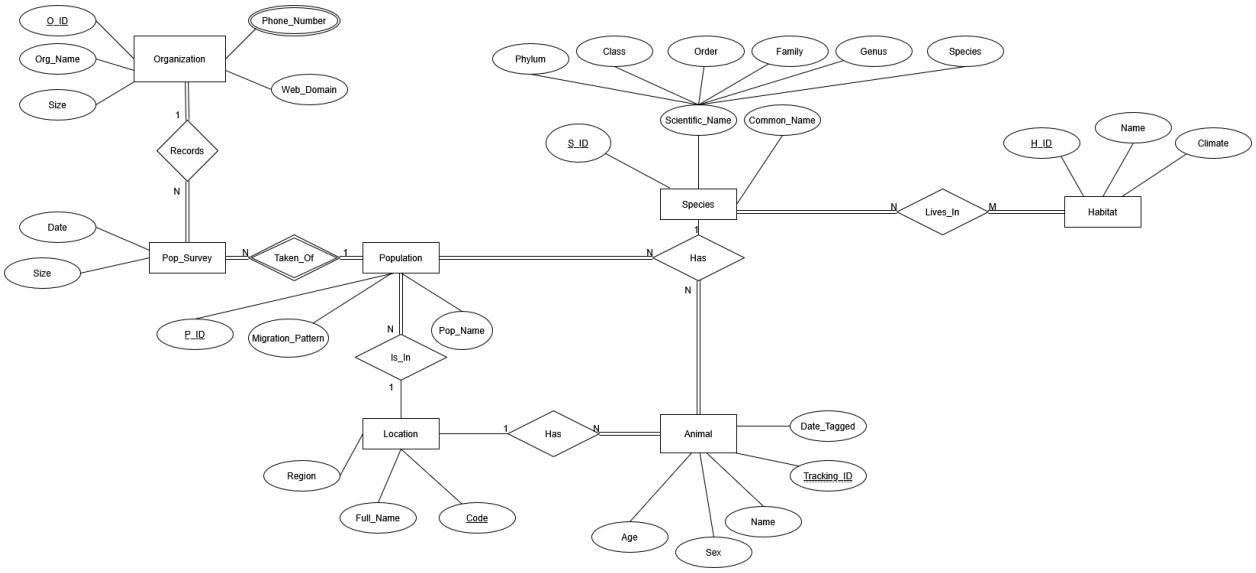

Figure 3: ER Diagram, revised.

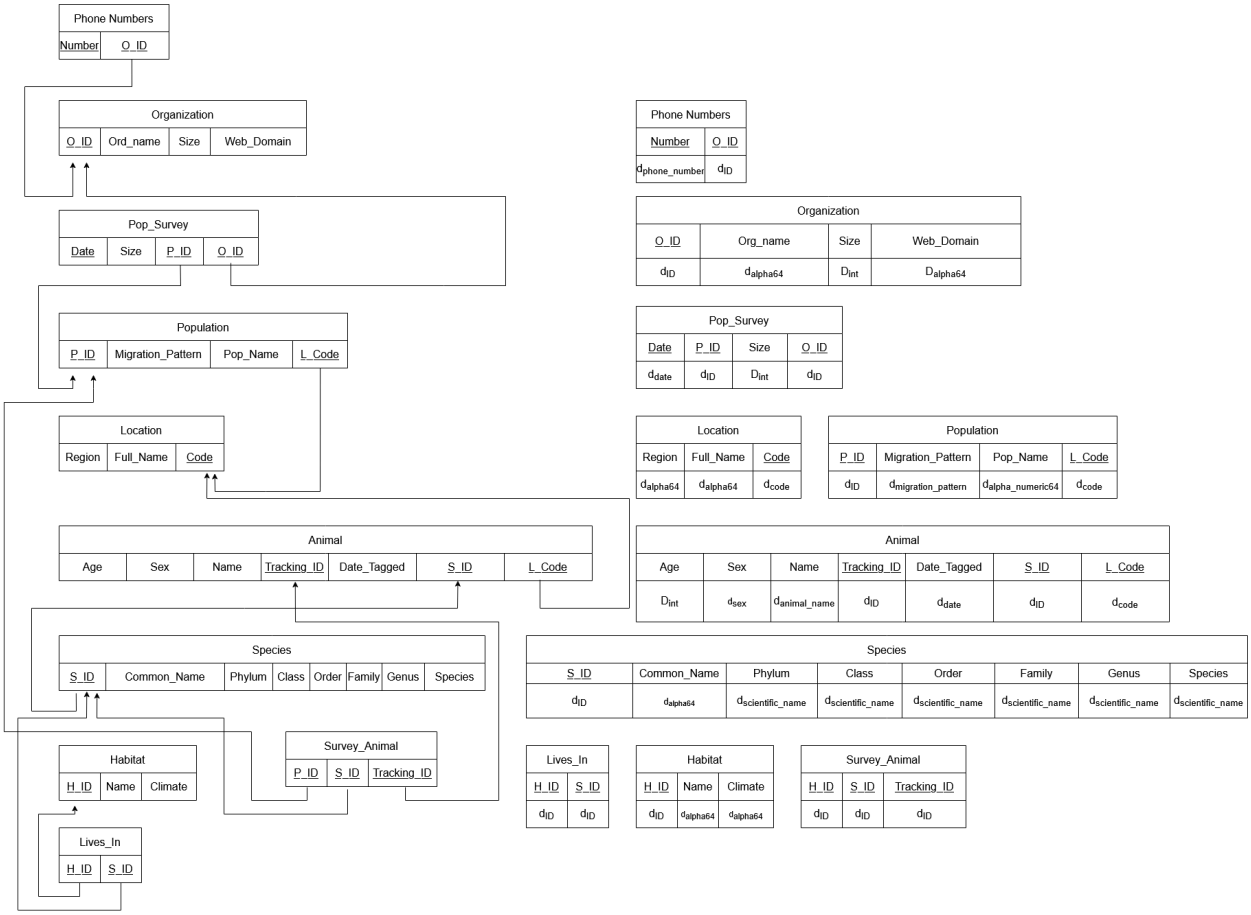

Figure 4: RM Diagram, revised.

6 Database Creation and Sample Data

The sample data was generated using Mockaroo and Fabricate, which produced an SQL file that was then executed in MySQL Workbench, hosted on Azure. Additionally, manually created data was inserted to test system integrity, ensuring compliance with constraints such as entity integrity and domain constraints.

The database is hosted on an Azure server under the name:
datalibrarian-475.mysql.database.azure.com

While the server must be started and maintained using Azure, it remains accessible online once up, allowing external users to connect as needed.

The database includes the following tables:

- animal
- habitat
- lives_in

- location
- organization
- phone_numbers
- pop_survey
- population
- species
- survey_animal

6.1 Queries

The following list gives some examples of queries that can be ran with the database:

- Retrieve all animals in the database.
- Retrieve all organizations.
- Get all species and their conservation status.
- Find all habitats recorded in the database.
- Find all animals that live in a specific habitat (e.g., “Forest”).
- Find all endangered species.
- Get all animals found in a specific location (“e.g., “Washington”).
- Get the total population recorded for each species.
- Find the most surveyed species.
- Find all surveys conducted after a certain date (e.g., after 2005).
- Retrieve the phone number associated with each organization.
- Find the organizations that manage critical species.

These are some example queries that management roles would use to monitor the operations of the conservation efforts. The information returned helps give more insight into what species require more attention, which organizations should be more trusted with, the accommodations that most endangered animals prefer, and more.

6.2 Database Creation Process

The creation of the database is through the SQL script below, which was run through MySQL hosted with Azure for Appsmith to access.

```
DROP TABLE IF EXISTS wildlife.LIVES_IN;
DROP TABLE IF EXISTS wildlife.SURVEY_ANIMAL;
DROP TABLE IF EXISTS wildlife.PHONE_NUMBERS;
DROP TABLE IF EXISTS wildlife.POP_SURVEY;
DROP TABLE IF EXISTS wildlife.POPULATION;
DROP TABLE IF EXISTS wildlife.ANIMAL;
DROP TABLE IF EXISTS wildlife.SPECIES;
DROP TABLE IF EXISTS wildlife.HABITAT;
DROP TABLE IF EXISTS wildlife.LOCATION;
DROP TABLE IF EXISTS wildlife.ORGANIZATION;

CREATE TABLE wildlife.ORGANIZATION (
    O_ID INT NOT NULL,
    Org_Name VARCHAR(64) NOT NULL UNIQUE CHECK (Org_Name <> ''),
    Size INT NOT NULL CHECK (Size > 0),
    Web_Domain VARCHAR(64) NOT NULL UNIQUE CHECK (Web_Domain LIKE
'%.%'),
    PRIMARY KEY (O_ID)
);

CREATE TABLE wildlife.PHONE_NUMBERS (
    Phone_Number BIGINT CHECK (Phone_Number IS NULL OR (Phone_Number
BETWEEN 1000000000 AND 9999999999)),
    O_ID INT NOT NULL,
    PRIMARY KEY (Phone_Number),
    FOREIGN KEY (O_ID) REFERENCES wildlife.ORGANIZATION(O_ID)
);

CREATE TABLE wildlife.LOCATION (
    CODE INT(8) NOT NULL,
    Region VARCHAR(64) NOT NULL,
    Full_Name VARCHAR(64) NOT NULL,
    PRIMARY KEY (CODE)
);

CREATE TABLE wildlife.POPULATION (
    P_ID INT(8) NOT NULL,
    Migration_Pattern VARCHAR(256) CHECK (LENGTH(Migration_Pattern) >
0),
```

```
    Pop_Name VARCHAR(64) NOT NULL CHECK (Pop_Name <> ''),
    L_CODE INT(8) NOT NULL,
    PRIMARY KEY (P_ID),
    FOREIGN KEY (L_CODE) REFERENCES wildlife.LOCATION(CODE) ON DELETE
CASCADE
);

CREATE TABLE wildlife.POP_SURVEY (
    P_ID INT(8) NOT NULL,
    Date Datetime NOT NULL DEFAULT CURRENT_TIMESTAMP,
    Size INT NOT NULL CHECK (Size > 0) DEFAULT 1,
    O_ID INT NOT NULL,
    PRIMARY KEY (P_ID, Date),
    FOREIGN KEY (O_ID) REFERENCES wildlife.ORGANIZATION(O_ID) ON
DELETE CASCADE,
    FOREIGN KEY (P_ID) REFERENCES wildlife.POPULATION(P_ID) ON DELETE
CASCADE
);

CREATE TABLE wildlife.SPECIES (
    S_ID INT(8) NOT NULL,
    Scientific_Name VARCHAR(32) NOT NULL UNIQUE CHECK
(Scientific_Name <> ''),
    Common_Name VARCHAR(64) NOT NULL UNIQUE CHECK (Common_Name <>
''),
    PRIMARY KEY (S_ID)
);

CREATE TABLE wildlife.ANIMAL (
    TRACKING_ID INT(8) NOT NULL,
    Age INT NOT NULL CHECK (Age >= 0),
    NAME VARCHAR(32),
    Date_Tagged Datetime DEFAULT CURRENT_TIMESTAMP,
    Sex VARCHAR(6) NOT NULL CHECK (Sex IN ('Male', 'Female',
'Unknown')),
    PRIMARY KEY (TRACKING_ID)
);

CREATE TABLE wildlife.HABITAT (
```

```
    H_ID INT(8) NOT NULL,
    Name VARCHAR(64) NOT NULL CHECK (Name <> ''),
    Climate VARCHAR(64) NOT NULL CHECK (Climate <> ''),
    PRIMARY KEY (H_ID)
);

CREATE TABLE wildlife.SURVEY_ANIMAL (
    P_ID INT(8) NOT NULL,
    S_ID INT(8) NOT NULL,
    TRACKING_ID INT(8) NOT NULL,
    FOREIGN KEY (P_ID) REFERENCES wildlife.POP_SURVEY(P_ID) ON DELETE
CASCADE,
    FOREIGN KEY (S_ID) REFERENCES wildlife.SPECIES(S_ID) ON DELETE
CASCADE,
    FOREIGN KEY (TRACKING_ID) REFERENCES wildlife.ANIMAL(TRACKING_ID)
ON DELETE CASCADE
);

CREATE TABLE wildlife.LIVES_IN (
    H_ID INT(8) NOT NULL,
    S_ID INT(8) NOT NULL,
    FOREIGN KEY (H_ID) REFERENCES wildlife.HABITAT(H_ID) ON DELETE
CASCADE,
    FOREIGN KEY (S_ID) REFERENCES wildlife.SPECIES(S_ID) ON DELETE
CASCADE
);
```

6.3 Database Populating Process

The database will then be populated with a sample data SQL file. This was done through Mockaroo/Fabricate, by first importing the database creation file above into the table/relationship input. This should automatically convert into corresponding tables populated with 100 randomly-generated data by default. We've tweaked certain constraints and parameters unique to the Fabricate website itself, to ensure that the data aligns with the proper criterias. Then, we export the data as an SQL script, sending it back to MySQL for populating the local database. To access it through Appsmith, permissions and access will need to be granted by the database host.

6.4 Query Listing

This section details the queries implemented for this project. The format will be the source code first, followed by a brief description of the purpose and its intention.

Listing 1: Group Species' Endanger level based on parameters

```
SELECT
   CASE
      WHEN species_count BETWEEN 0 AND 250 THEN 'Critical'
      WHEN species_count BETWEEN 251 AND 2500 THEN 'Endangered'
      WHEN species_count BETWEEN 2501 AND 10000 THEN 'Vulnerable'
      ELSE 'Safe'
   END AS endangerLevel,
   COUNT(*) AS species_count
FROM (
         SELECT S_ID, COUNT(TRACKING_ID) AS species_count
         FROM SURVEY_ANIMAL
         GROUP BY S_ID
) AS species_count_table
GROUP BY endangerLevel;
```

The query in listing 1 can be used by conservationists or researchers to identify which species are at high risk and to monitor their conservation status based on different parameters. This specific parameter groups species according to their endangerment level; in this case, the parameters are based on National Geographic's categorical standards, but other criterias such as IUCN status can be used.

Listing 2: Retrieve all Endangered Species

```
SELECT
   s.S_ID,
   s.Scientific_Name,
   s.Common_Name,
   COUNT(sa.S_ID) AS species_count,
   CASE
      WHEN COUNT(sa.TRACKING_ID) BETWEEN 0 AND 250 THEN 'Critical'
      WHEN COUNT(sa.TRACKING_ID) BETWEEN 251 AND 2500 THEN 'Endangered'
      WHEN COUNT(sa.TRACKING_ID) BETWEEN 2501 AND 10000 THEN 'Vulnerable'
      ELSE 'Safe'
   END AS endangerLevel
FROM SPECIES s
LEFT JOIN SURVEY_ANIMAL sa ON s.S_ID = sa.S_ID
GROUP BY s.S_ID, s.Scientific_Name, s.Common_Name
HAVING endangerLevel IN ('Critical', 'Endangered')
ORDER BY species_count ASC;
```

The query in listing 2 can be used by conservationists to identify from a list of all species that need the most critical attention. This query retrieves a list of all species that are classified as Critical or Endangered status (based on the National Geographic's categorical standards), ordered from the lowest to highest population count.

Listing 3: Retrieve all Organizations and their Websites

```
SELECT O_ID, Org_Name, Web_Domain
FROM ORGANIZATION
ORDER BY Org_Name;
```

The query in listing 3 can be used by the general users (researchers or the public) to easily find organizations that are partnered with the conservation efforts. This query lists all organizations involved in the system's database, along with their website details.

Listing 4: Get Population Surveys conducted by each Organizations

```
SELECT p.P_ID, p.Date, p.Size, o.Org_Name
FROM POP_SURVEY p
JOIN ORGANIZATION o ON p.O_ID = o.O_ID
ORDER BY p.Date DESC;
```

The query in listing 4 can be used by conservationists to understand how different organizations are tracking wildlife populations, and to track trends over different entities. This query retrieves the population survey data conducted by each organization (i.e., date and size), ordered by the most recent date.

Listing 5: Retrieve Population data for a specific Region

```
SELECT pop.P_ID, pop.Pop_Name, loc.Region, loc.Full_Name
FROM POPULATION pop
JOIN LOCATION loc ON pop.L_CODE = loc.CODE
WHERE loc.Region = 'Pacific Northwest';
```

The query in listing 5 can be used by researchers to focus on particular regions for tracking/comparison of wildlife populations, to make observations such as on population trends. This query retrieves the population data from the "Pacific Northwest" region.

Listing 6: Get all Animals tagged in the past year

```
SELECT TRACKING_ID, NAME, Age, Sex, Date_Tagged
FROM ANIMAL
WHERE Date_Tagged >= DATE_SUB(CURRENT_DATE, INTERVAL 12 MONTH);
```

The query in listing 6 can be used by conservationists or researchers to monitor the tracking/tagging activities that have recently occurred, providing insights into the latest conservation efforts and newest addition to the wildlife database. This query retrieves animals that have been tagged within the past 1 year.

Listing 7: Find all species within a specific Habitat

```
SELECT s.S_ID, s.Scientific_Name, s.Common_Name, h.Name AS Habitat
FROM SPECIES s
JOIN LIVES_IN li ON s.S_ID = li.S_ID
JOIN HABITAT h ON li.H_ID = h.H_ID
WHERE h.Name = 'Wetlands';
```

The query in listing 7 can be used by researchers or conservationists to analyze species that live in particular habitats for conservation efforts and habitat management. This query identifies all species that inhabit the “Wetlands” areas.

Listing 8: Find the most recent Population Survey for each Population

```
SELECT ps.P_ID,
    MAX(ps.Date) AS Last_Survey_Date,
    ps.Size
FROM POP_SURVEY ps
GROUP BY ps.P_ID, ps.Size
ORDER BY Last_Survey_Date DESC;
```

The query in listing 8 can be used by researchers to analyze the latest surveys and better understand the current population status and trends. This query finds the most recent population survey for each population.

Listing 9: Find all species within a specific Region

```
SELECT Pop.Pop_Name, SUM(PS.Size) AS Total_Population
FROM POPULATION Pop
JOIN POP_SURVEY PS ON Pop.P_ID = PS.P_ID
JOIN LOCATION Loc ON Pop.L_CODE = Loc.CODE
WHERE Loc.Region = 'Africa'
```

```
GROUP BY Pop.Pop_Name;
```

The query in listing 9 can be used by researchers to focus on species within a particular geographic area, to better understand regional species biodiversity. This query retrieves all species that are found within the “Africa” region.

Listing 10: List the common and scientific name of all Species

```
SELECT Sp.Common_Name, Sp.Scientific_Name
FROM SPECIES Sp;
```

The query in listing 10 can be used by the general users to compare common and scientific names of species for identification and/or study purposes. This query lists both the common and scientific names of species in the wildlife database.

7 Testing and Deployment

The development process for the wildlife population monitoring app uses GitHub for version control and Appsmith for UI deployment. The code includes SQL queries and JavaScript functions, managed through GitHub, and can be accessed online for collaboration purposes.

7.1 Faulty Testing

We’ve also done some faulty testing, with some examples listed below.

Test 1: Expected result is to fail due to CHECK (Size >0).

```
INSERT INTO wildlife.ORGANIZATION (O_ID, Org_Name, Size, Web_Domain)
VALUES (1, 'Invalid Org', -100, 'invalid.org');
```

Test 2: Expected to fail due to CHECK (Phone_Number BETWEEN 1000000000 AND 9999999999).

```
INSERT INTO wildlife.PHONE_NUMBERS (Phone_Number, O_ID)
VALUES (123456, 1);
```

Testing 3: Expected to fail due to NOT NULL constraint on Pop_Name.

```
INSERT INTO wildlife.POPULATION (P_ID, Migration_Pattern, Pop_Name,
L_CODE)
VALUES (100, 'North to South', NULL, 200);
```

7.2 Deployment

Appsmith is used to create an interactive user interface that connects to the app's Azure-hosted MySQL database. The deployment process is automated through GitHub, and the app itself is deployed to Appsmith from the repository.

In terms of testing, the application went through multiple integration and testing within the Appsmith platform. The UI elements were tested for responsiveness and performance, and ensuring that the data on the app matches that of the database's. The visual editor of the app helps in creating and debugging queries to make sure it works for the users.

The deployment process is through Appsmith, where the front-end is instantly accessible for users upon each update and rollout, without any need for manual installation or configuration for the user. This is one of the key advantages for the app, as it allows conservationists, educators, and researchers to access the application effortlessly and interact with real-time wildlife population data.

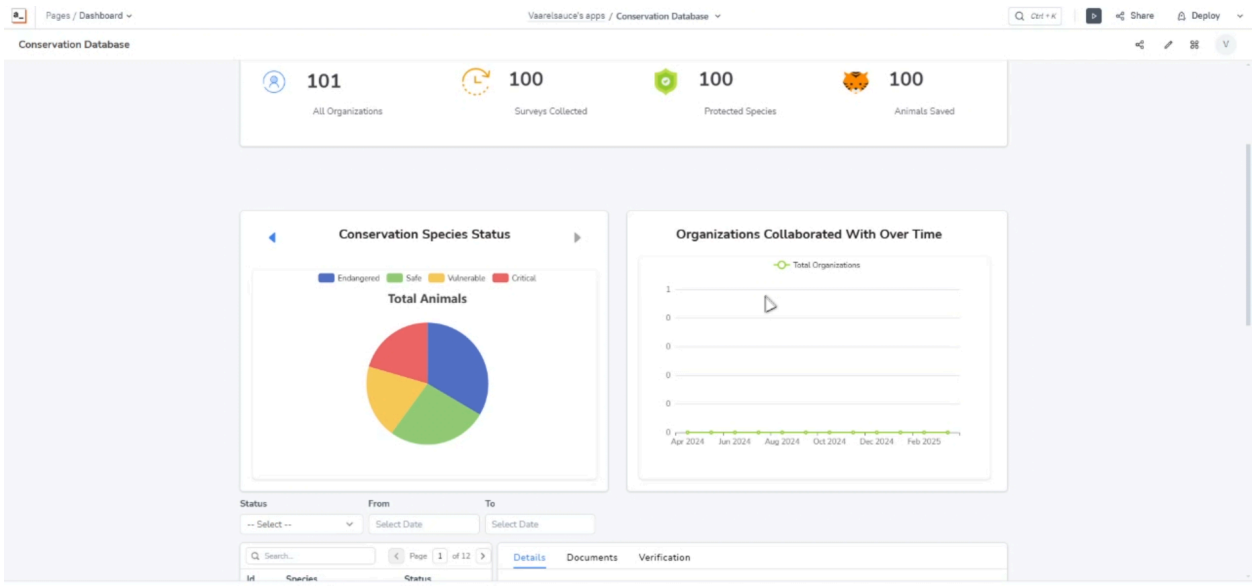

Figure 5: ConservationDB Dashboard (Appsmith)

8 Schedule of Iteration Planning

ITERATIONS	OBJECTIVES	Brody	Vinh	Harshitha	Dorian
1: Research & Initial Documentati	1/22: Research and gather	Research about the database and	Draft the initial ER diagram.	Identify any constraints/requirements	Review and clean up the shared

on (1/22 - 2/9)	requirements for the project. 1/29: Draft initial design documentation & review early drafts. 2/9: Finalize the design documentation.	take notes.		of the database.	documentation files.
1.5: Presentation Preparation (2/10 - 2/18)	2/10: Research tools & write scripts. 2/17: Create slides & practice presentation. 2/18: Finalize slides & submit presentations.	Tool researching.	Tool researching.	Create the presentation script.	Create slides and templates.
2: Tooling & Refinement (2/19 - 2/22)	2/?: Revise database schema based on feedback from iteration 1. 2/19: Research	Writing sections.	Writing sections.	Researching tooling & revision.	Research tooling & revision.

	database tools based on prior feedback for 1.5, collaborative findings. 2/20: Write Section 2. 2/22: Finalize report for submission.				
3: Database Creation & Testing (2/23 - ⅗)	2/?: Incorporate changes based on feedback. 2/23: Initial database creation (tables, relationships & constraints). 3/2: Independent testing of database functionality, and begin Section 3 creation.	Database setup and indexing.	Do query optimizations.	Sample data insertion and validation.	Documentation and compile a feedback/report.

	⅗: Team review and bug-fixing.				
4: Final Revisions (3/6 - 3/20)	3/6: Address all remaining feedback and finalize the database. 3/15: Prepare for the final presentation. 3/20: Conduct final review and prepare for submission.	Review and finalize the final report.	Run multiple tests and check for flaws.	Finalize presentation materials.	Complete and proofread all documentation.

9 Discussion

This section serves as a retrospective analysis of the creation and development of the database for the final project. The goal of this project was to design and implement a relational database that tracks and monitors wildlife populations, their habitats, and related conservation data. The project spanned multiple weeks and involved several stages, including requirements gathering, data modeling, database creation, integration with external tools, and deployment. In this retrospective, we will reflect on the successes, challenges, potential improvements, and of course if we had achieved the original goal that we had set out at the start of the project.

9.1 Successes

The database design was one of the major successes of the project, and only minor adjustments were made long after its implementation. The relational model ensures the relationship between entities in a nominal and logical manner, which can be attributed to our thorough planning during

the ER/RM brainstorming. We are also very proactive with making changes and feedback, which helps in refinement of future iterations.

Building the database through MySQL Azure had some hiccups in the beginning phase, but as time went by its performance and robustness really shined through. The infrastructure was able to handle multiple concurrent users and store extensive population and species data without any issues.

The integration of the database with MySQL/Azure, and deployment of the GitHub/Appsmith interface went smoothly. The user interface was developed using a combination of SQL and JavaScript, with SQL being an accumulative knowledge learned throughout the course. Meanwhile, JavaScript was syntactically similar to Python, which was a language that we were already familiar with, and was relatively easy to adopt for building the interactive elements within the Appsmith environment.

9.2 Issues

One of the challenges we faced was performance issues during query execution. Certain complex queries had slower performance than others, with optimization and limited server bandwidth (we used free trial accounts) hampering down the fetch speed. Additionally, as we tested and modified data using the Appsmith interface, we sometimes inadvertently altered or deleted data in the mock database. This required us to restore data manually, and is a noticeable flaw that should be resolved before scaling the project further.

Time is also an important limitation that prevents us from doing further comprehensive testing. Combining that with lack of communications and urgency amongst some members in the group, this led to some minor bugs and issues (for example, with retrieval optimization) that could have been addressed with additional testing time. Thorough checking of edge cases would have also been tested before deployment, to prevent users from exploiting or encountering edge case issues.

9.3 Improvements

Both the database system and Appsmith UI could be more robust with stronger validation rules and checks. This includes extensive triggers/constraints on the database, retrieval optimization, and data cleaning routines (on the UI layer) to ensure that incorrect records are noted off before it’s too late. We can also add more options to data filtering (e.g., region, species, survey date on the graphs), to better improve usability. An option to create a custom visualization, based on the user’s filtering, could be very effective in helping users interpret the data for their specific needs. In case of error, implementing some sort of error handling to show detailed error messages will benefit both the tester and user in their overall user experience.

On the database level, performance optimization strategies would've been beneficial to our system had we given more time to research and implement. This includes indexing frequently queried columns, query refactoring, and partitioning the database by region or species (which ties into refining our Relational Model).

Conclusion

Ultimately, the creation of the wildlife conservation database for the final project was a significant accomplishment, and despite some challenges, the project has proven to be a valuable learning experience. Through MySQL on Azure and Appsmith, we successfully created an interactive system that enables conservationists, researchers, and the general public to access and analyze wildlife population data.

Throughout this process, we broadened and refined our understanding of database management, relational model optimization, and utilized real-world practices like normalization. Moving forward, there are ways that we can further enhance the system by incorporating additional visualization tools, improve query efficiency, and expanding the database further to include more ecological datasets for complex analysis.Overall, this project has provided valuable hands-on experience in database development and deployment, reinforcing our technical skills while also contributing to a meaningful conservation initiative.