VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JNANA SANGAMA", BELAGAVI - 590 018

A MINI PROJECT REPORT

on

# 'ENVIRONMENTAL DATA OBSERVATION AND SPECIES PRESERVATION MANAGEMENT SYSTEM'

*submitted by*

| | |
|---|---|
| Nihara | 4SF21CD018 |
| Jaideep N | 4S21FCD009 |

*In partial fulfillment of the requirements for the V semester*

DBMS LABORATORY WITH MINI PROJECT

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE & ENGINEERING (DATA SCIENCE)

*under the Guidance of*

Mr. Vasudeva Rao P V

Assistant Professor

Department of ISE

SAHYADRI

College of Engineering & Management

An Autonomous Institution

Adyar, Mangaluru - 575 007

AY: 2023 - 24

# SAHYADRI

## College of Engineering & Management

**An Autonomous Institution**

**Adyar, Mangaluru - 575 007**

**Department of Computer Science & Engineering (Data Science)**

## CERTIFICATE

This is to certify that the **Mini Project** entitled **"ENVIRONMENTAL DATA OBSERVATION AND SPECIES PRESERVATION MANAGEMENT SYSTEM"** has been carried out by **Nihara (4SF21CD018)** and **Jaideep N (4SF21CD009)**, the bonafide students of Sahyadri College of Engineering & Management in partial fulfillment of the requirements for the V semester **DBMS Laboratory with Mini Project (21CSL55)** of **Computer Science & Engineering (Data Science)** of Visvesvaraya Technological University, Belagavi during the Academic Year 2023 - 24. It is certified that all corrections/suggestions indicated for Continuous Internal Assessment have been incorporated in the report deposited in the departmental library of Information Science & Engineering. The mini project report has been approved as it satisfies the academic requirements in respect of mini project work.

| | |
|---|---|
| **Mr. Vasudeva Rao P V** | **Dr. Rithesh Pakkala P.** |
| Assistant Professor | Assoc.Professor & Head |
| Dept. of ISE, SCEM | Dept. of ISE & CSE(DS), SCEM |

**External Practical Examination:**

| Examiner's Name | Signature with Date |
|---|---|
| 1. .................... | .................... |
| 2. .................... | .................... |

SAHYADRI

College of Engineering & Management

An Autonomous Institution

Adyar, Mangaluru - 575 007

Department of Computer Science & Engineering (Data Science)

# DECLARATION

We hereby declare that the entire work embodied in this Mini Project Report titled **"ENVIRONMENTAL DATA OBSERVATION AND SPECIES PRESERVATION MANAGEMENT SYSTEM"** has been carried out by us at Sahyadri College of Engineering & Management, Mangaluru under the supervision of **Mr. Vasudeva Rao P V** as the part of the V semester **DBMS Laboratory with Mini Project (21CSL55)** of **Bachelor of Engineering** in **Computer Science & Engineering (Data Science)**. This report has not been submitted to this or any other university.

**Nihara (4SF21CD018)**

**Jaideep N (4SF21CD009)**

SCEM, Mangaluru

# Abstract

This project aims to revolutionize wildlife conservation efforts by introducing an integrated system for species management, environmental data collection, and conservation planning within wildlife preserves managed by forest departments. The system offers a comprehensive suite of tools and functionalities designed to streamline the management of diverse species, monitor environmental indicators, and devise effective conservation strategies. Key features include advanced species management capabilities for tracking population trends, habitat preferences, and conservation statuses, alongside robust environmental data collection modules enabling the monitoring of biodiversity indicators and ecosystem health. Additionally, the system facilitates the planning and execution of targeted conservation initiatives, with specialized support for the protection of endangered species and the mitigation of environmental threats. By centralizing data, providing sophisticated analysis tools, and fostering collaboration among stakeholders, the project aims to enhance the efficiency and effectiveness of wildlife conservation efforts. Ultimately, the overarching goal is to contribute to the preservation of biodiversity, the protection of endangered species, and the sustainability of ecosystems, thereby ensuring the long-term health and resilience of our natural world.

# Acknowledgement

It is with great satisfaction and euphoria that we are submitting the Mini Project Report on **“ENVIRONMENTAL DATA OBSERVATION AND SPECIES PRESERVATION MANAGEMENT SYSTEM”**. We have completed it as a part of the V semester **DBMS Laboratory with Mini Project (21CSL55)** of **Bachelor of Engineering** in **Computer Science & Engineering (Data Science)** of Visvesvaraya Technological University, Belagavi.

We are profoundly indebted to our guide, **Mr. Vasudeva Rao P V**, Assistant Professor, Department of Information Science & Engineering (ISE) for innumerable acts of timely advice, encouragement and We sincerely express our gratitude.

We express our sincere gratitude to **Dr. Rithesh Pakkala P.**, Assoc. Professor & Head, Department of ISE & CSE(DS) for his invaluable support and guidance.

We sincerely thank **Dr. S S Injaganeri**, Principal, Sahyadri College of Engineering & Management who have always been a great source of inspiration.

We further thank the non-teaching staff members of the Department of ISE, who have provided necessary support during the course of the work.

Finally, yet importantly, We express our heartfelt thanks to our family & friends for their wishes and encouragement throughout the work.

**Nihara**
4SF21CD018
V Sem, B.E., CSE(DS)
SCEM, Mangaluru

**Jaideep N**
4SF21CD009
V Sem, B.E., CSE(DS)
SCEM, Mangaluru

# Table of Contents

# List of Figures

# Chapter 1

# Introduction

## 1.1 Database

A database is a structured collection of data that is organized and stored electronically. It serves as a central repository for storing and managing information efficiently. Databases play a crucial role in modern information systems by providing a systematic way to store, retrieve, and manipulate data. They are used in various domains, including business, education, healthcare, finance, and more. In a database, data is organized into tables, each containing rows and columns. This tabular structure allows for easy querying and analysis of data, enabling users to extract meaningful insights and make informed decisions. With the advent of digital technologies, databases have become indispensable tools for organizations seeking to streamline their operations, improve decision-making processes, and gain a competitive edge in today's data-driven world.

## 1.2 Database Management System(DBMS)

A Database Management System (DBMS) is a software application that facilitates the creation, management, and manipulation of databases. It serves as an interface between users and the database, providing tools and utilities to efficiently store, retrieve, and update data. DBMSs are designed to handle large volumes of data and ensure data integrity, security, and concurrency control. They offer features such as data definition, data manipulation, query processing, and user access control. DBMSs come in various types, including relational, object-oriented, and NoSQL, each suited for different data models and application requirements. With the proliferation of data in today's digital age, DBMSs have become essential components of modern information systems, empowering organizations to effectively manage their data assets and derive valuable insights for strategic decision-making.

## 1.3 Characteristics of the Database Approach

- **Self-describing nature of a database system:** A database system contains a comprehensive description of the data it stores, known as metadata. This metadata includes information about the structure of the database, such as the types of data stored, relationships between data elements, constraints, and access permissions. This self-describing nature makes it easier for users and applications to understand and work with the database, as they can query the metadata to retrieve information about the database schema and data definitions.

- **Insulation between programs and data, and data abstraction:** In a database system, there is a clear separation between the programs that access the data and the underlying data itself. This insulation ensures that changes to the database structure or implementation details do not require modifications to the applications that use the data. Data abstraction allows users and applications to interact with the database at a high level, using conceptual models and query languages, without needing to understand the internal storage mechanisms or physical organization of the data.

- **Support of multiple views of the data:** A database system supports the creation of multiple views of the same underlying data. Views provide different perspectives or subsets of the data tailored to specific user requirements or application needs. Each view presents a customized representation of the data, hiding irrelevant details and presenting only the information that is relevant to the user's query or task. This flexibility allows different users to access and manipulate the data in ways that are meaningful and relevant to their roles or responsibilities.

- **Sharing of data and multiuser transaction processing:** A key advantage of database systems is their ability to support concurrent access to the same data by multiple users or applications. This concurrency control is essential for enabling collaborative work environments and ensuring data consistency and integrity. Database management systems (DBMS) implement sophisticated transaction processing mechanisms to manage concurrent transactions and maintain the ACID properties (Atomicity, Consistency, Isolation, Durability) of database transactions. This ensures that transactions execute reliably and efficiently, even in a multiuser environment with concurrent access to the data.

## 1.4 Advantages of using the DBMS Approach

Advantages of using a DBMS and the capabilities that a good DBMS should possess. These capabilities are in addition to the four main characteristics discussed in Section 1.3. The DB must utilize these capabilities to accomplish a variety of objectives related to the design, administration, and use of a large multiuser database. The main advantages of using database are:

- Control of Redundancy.
- Restriction of Unauthorized Access.
- Persistent Storage and Program Objects.
- Storage Structures and Search Techniques.
- Backup and Recovery.
- Multiple User Interfaces.
- Representation of Complex Relationships.
- Enforcement of Integrity Constraints.
- Interference and Actions using Rules and Triggers.

## 1.5 Schemas

In the realm of databases, a schema serves as a foundational framework, delineating the structure, organization, and constraints governing stored data. It encompasses various elements such as tables, columns, relationships, and constraints, providing a blueprint for how data is organized within the database. There exist three principal types of schemas:

- Physical Schema: Focuses on storage structures, indexing strategies, and data storage formats to optimize data storage and retrieval efficiency.
- Logical Schema: Defines the conceptual representation of data, including table organization, relationships, and integrity constraints, facilitating data integration and conceptual modeling.
- External Schema: Provides a user-centric view of data, catering to specific user or application needs, while shielding them from underlying complexities. Multiple external schemas can be crafted to offer customized views of data.

## 1.6 DBMS Languages

In a database management system (DBMS), languages and interfaces serve as indispensable tools for users to interact with and manage databases effectively. The Data Definition Language (DDL) holds a pivotal role in defining the structure and organization of databases, encompassing tasks such as schema creation and mapping between different schema levels. Simultaneously, the Data Manipulation Language (DML) empowers users to execute operations like data retrieval, insertion, deletion, and modification, ensuring seamless interaction with the database contents. Furthermore, the Storage Definition Language (SDL) delves into optimizing internal schema specifications to enhance storage and retrieval efficiency, thereby bolstering overall database performance.

As a comprehensive database language, SQL (Structured Query Language) amalgamates DDL, DML, and other functionalities, offering a unified platform for schema definition, view management, and data manipulation. Additionally, user-friendly interfaces play a crucial role in simplifying database interactions, catering to diverse user preferences and requirements. By providing intuitive platforms, these interfaces promote accessibility and usability across a wide spectrum of user groups, ranging from casual end users to database administrators. Together, these languages and interfaces form the backbone of DBMS, facilitating seamless database management and utilization in various domains and applications.

## 1.7 Motivational Challenges for the Project Work

In the context of our project, which involves database management for wildlife preservation, motivational challenges was due to the complexity of data management tasks, uncertainties in achieving conservation goals, solving many errors, and external pressures such as time constraints. To overcome these challenges, fostering a collaborative team environment where we understand the project's significance and our role within it was essential. Clear communication about project goals, regular updates on progress, and recognition of individual contributions helped us maintain team morale and motivation. Additionally, providing adequate resources and support has enhance productivity, ultimately lead to the successful implementation of wildlife preservation initiatives.

## 1.8 Application of the Project

- Efficient Data Management: The project streamlines processes associated with data storage, retrieval, and organization, ensuring quick access to pertinent information and supporting decision-making processes.

- Business Intelligence: It contributes to business intelligence efforts by providing insights gleaned from data analysis and visualization techniques, empowering stakeholders to make informed decisions based on identified trends and patterns.

- Process Automation: The project plays a pivotal role in process automation, automating repetitive tasks and workflows to improve operational efficiency and productivity, thereby reducing manual errors and accelerating task completion.

- Security and Compliance: Ensuring data security through robust access controls and encryption mechanisms, the project aids in regulatory compliance efforts, maintaining data integrity and safeguarding sensitive information from unauthorized access or breaches.

- Collaboration and Integration: Facilitating collaboration and integration, the project provides a platform for seamless data sharing and collaboration among teams, integrating with existing systems and tools to enhance communication and coordination across departments.

# Chapter 2

# Conceptual Data Modeling

Conceptual data modeling is a critical step in the process of designing databases, serving as the initial stage where abstract ideas are translated into structured representations. At its essence, conceptual data modeling involves capturing the essential business concepts and their relationships within an organization, laying the groundwork for subsequent database design decisions. By abstracting away implementation details and focusing on the semantics of the data, conceptual data modeling provides a clear and concise overview of the organization's data requirements. Through the identification of entities, attributes, relationships, and constraints, stakeholders can gain a comprehensive understanding of the structure and behavior of the data, fostering effective communication and collaboration among business analysts, database designers, and end-users. Ultimately, conceptual data modeling sets the stage for the development of robust and scalable databases that align closely with the organization's goals and objectives, making it a crucial component of the database development lifecycle.

## 2.1 Entity Relationship (ER) Model

The Entity-Relationship (ER) model serves as a foundational framework for conceptualizing and representing the structure and relationships within a database. In this model, entities represent real-world objects or concepts, while relationships depict the associations between these entities. Attributes, meanwhile, describe the properties or characteristics of entities. The ER model employs various symbols and notation, such as rectangles for entities, diamonds for relationships, and ovals for attributes, to visually represent the components and connections within a database schema. By providing a clear and intuitive representation of data structures and relationships, the ER model facilitates effective database design, development, and communication among stakeholders involved in database projects. This model helps database designers identify entities and their relationships, ensuring that the database accurately reflects the real-world domain it represents and supporting the development of robust and efficient database systems.

Furthermore, the ER model allows for the visualization and documentation of complex data structures, aiding in the understanding and analysis of database designs. It serves as a basis for designing relational databases, providing a structured approach to defining entities, attributes, and relationships. Additionally, the ER model supports database normalization, helping to eliminate data redundancy and maintain data integrity. By adhering to the principles of the ER model, database designers can create databases that are well-structured, easy to maintain, and scalable to accommodate future changes and requirements.

## 2.2 Entities

Entities in the context of database management refer to distinct objects, concepts, or subjects about which data is stored. In essence, entities represent the nouns or substantive elements within a database schema. These entities possess attributes that describe their characteristics or properties.For example, the ”Species” entity encompasses attributes such as ”SpeciesID,” ”SpeciesName,” ”Classification,” and ”ConservationStatus,” enabling the systematic cataloging and tracking of various species. Similarly, the ”Preserve” entity includes attributes like ”PreserveID,” ”Name,” ”Location,” and ”EcosystemType,” providing essential details about wildlife preserves and their ecological contexts. These entities, along with others like ”Observation” and ”Conservation Plan,” form the foundational elements of our database, facilitating effective species management and conservation efforts.

## 2.3 Attributes

Attributes in a database context refer to the specific properties or characteristics that describe an entity. These attributes provide detailed information about the entity they belong to, helping to define its identity and behavior within the database schema. In essence, attributes represent the various facets or dimensions of an entity.In our project on species management and environmental conservation, attributes are crucial for capturing relevant data. For example, attributes associated with the ”Species” entity might include ”SpeciesID,” ”SpeciesName,” ”Population,” ”HabitatPreferences,” and ”ConservationStatus,” providing essential details about different species and their conservation statuses.

## 2.4 Relationships

In the context of databases, relations refer to the connections or associations between entities. These relationships establish meaningful links between different entities, allowing for the representation of complex data interactions and dependencies within the database schema. Relationships are fundamental for modeling real-world scenarios accurately and facilitating data integrity and consistency.

Similarly, in our project on species management and environmental conservation, various relationships exist between entities such as ”Species,” ”Preserve,” ”EnvironmentalData,” and ”ConservationPlan.” For instance, there might be a one-to-many relationship between the ”Preserve” entity and the ”Species” entity, indicating that a wildlife preserve can host multiple species, while each species can be found in one or more preserves.These relationships enable us to model the interconnectedness of data elements within the ecosystem, facilitating comprehensive analysis and decision-making in species management and conservation efforts.

## 2.5 Notation for ER Diagrams

In Entity-Relationship (ER) diagrams, various symbols and notations are used to represent entities, attributes, relationships, and other elements. Here’s a brief overview of some common symbols and notations:

- **Entity**: Represented by a rectangle with rounded corners, containing the entity name.
- **Attribute**: Depicted as an oval connected to its respective entity. Attributes describe properties or characteristics of entities.
- **Primary Key**: Indicated by underlining the attribute(s) within an entity that uniquely identifies each entity instance.
- **Relationship**: Shown as a diamond shape connecting two or more entities. Relationships illustrate associations between entities. Common relationship types include one-to-one, one-to-many, and many-to-many.
- **Cardinality**: Represented near the endpoints of relationship lines to indicate the minimum and maximum number of instances that can participate in the relationship. Cardinality can be denoted using symbols such as ”1,” ”0..1,” ”0..N,” or ”1..N.”

- **Weak Entity**: Identified by double rectangles and represents an entity that cannot be uniquely identified by its attributes alone. It relies on a related entity, known as the identifying or owner entity, for identification.
- **Participation Constraint**: Indicates whether participation in a relationship is mandatory (total participation) or optional (partial participation) for entities involved in the relationship. This constraint is typically represented using symbols such as a solid line (total participation) or a dashed line (partial participation).

These notations provide a standardized way to visually represent the structure and relationships within a database schema, facilitating communication and understanding among database designers, developers, and stakeholders.

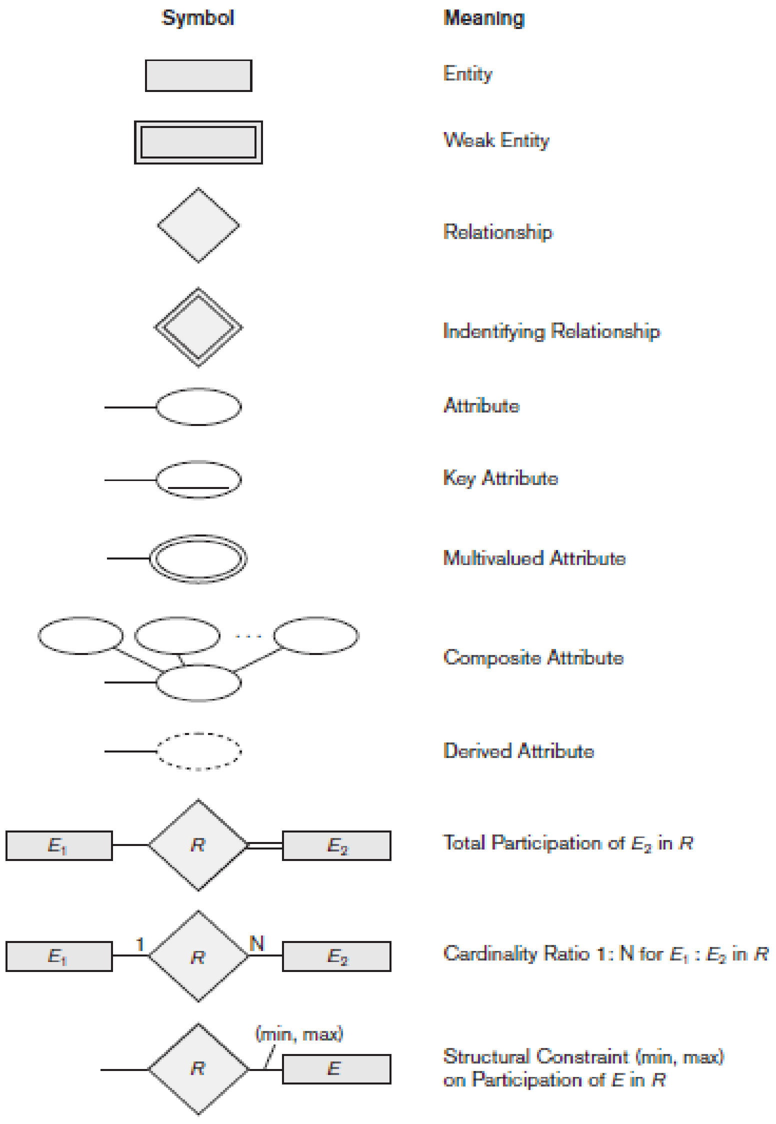

Symbol
Meaning
Entity
Weak Entity
Relationship
Indentifying Relationship
Attribute
Key Attribute
Multivalued Attribute
. . .
Composite Attribute
Derived Attribute
$E_1$
R
$E_2$
Total Participation of $E_2$ in $R$
1
N
Cardinality Ratio 1: N for $E_1$ : $E_2$ in $R$
(min, max)
E
Structural Constraint (min, max) on Participation of $E$ in $R$

Figure 2.1: Summary of the Notation for ER Diagrams

# Chapter 3

# Relational Data Model

## 3.1 Relational Model Constraints

- **Entity Integrity**: Ensures that each tuple in a relation has a unique primary key value, preventing duplicate or null entries.
- **Referential Integrity**: Ensures that foreign key values in one relation match primary key values in another relation, maintaining consistency across related tables.
- **Domain Integrity**: Ensures that all values in a column satisfy specific domain constraints, such as data type, range, or format requirements.
- **Null Constraint**: Specifies whether a column can contain null values or must have a non-null value for each tuple.
- **Check Constraint**: Defines conditions that must be satisfied for values in a column, ensuring data integrity based on specific business rules or criteria.
- **Unique Constraint**: Ensures that all values in a column, or combination of columns, are unique across all tuples in the relation.
- **Key Constraint**: Defines one or more columns as a candidate key or primary key for the relation, enforcing uniqueness and providing a reference for establishing relationships with other tables.
- **Default Constraint**: Specifies a default value for a column if no explicit value is provided during tuple insertion, ensuring data consistency and completeness.

# Chapter 4

# Structured Query Language(SQL) Programming

## 4.1 SQL Data Definition and Data Types

In database management systems (DBMS), SQL (Structured Query Language) is commonly used for data definition, including creating and modifying database structures. Here are some SQL commands for data definition along with commonly used data types:

- Creating a Table

  To create a new table in a database, use the `CREATE TABLE` statement followed by the table name and a list of columns with their data types and constraints.

- Altering a Table

  To modify an existing table, you can use the `ALTER TABLE` statement to add, modify, or drop columns, as well as add constraints.

  - **INTEGER**: Integer numeric data type for storing whole numbers.
  - **VARCHAR(length)**: Variable-length character string with a specified maximum length.
  - **DATE**: Date data type for storing dates without time information.
  - **BOOLEAN**: Boolean data type for representing true or false values.
  - **DECIMAL(precision, scale)**: Decimal data type for storing fixed-point numbers with specified precision and scale.

## 4.2 Assertions

Assertions in SQL enforce data integrity rules or conditions on the database. They are used for enforcing business rules or constraints that cannot be expressed using primary key or foreign key constraints. To create an assertion, the ‘CREATE ASSERTION‘ statement is used, followed by the assertion name and the condition that must be satisfied. ensures that all employee salaries are greater than 0. To remove an assertion, the ‘DROP ASSERTION‘ statement is used, followed by the assertion name, like ‘DROP ASSERTION SalaryCheck;‘. Once created, assertions automatically enforce their specified conditions whenever data is inserted, updated, or deleted. If the condition is not met, the operation is rejected, and an error is raised. Assertions are crucial for enforcing complex business rules and constraints in the database.

## 4.3 Triggers

Triggers in SQL are specialized procedures that are automatically executed in response to certain events or actions within a database. These events typically include data manipulation operations like INSERT, UPDATE, and DELETE statements on specific tables, as well as changes to the database schema. Triggers serve various purposes, such as enforcing referential integrity, implementing complex business rules, maintaining audit trails, and performing data validation or transformation.

A trigger is defined using the ‘CREATE TRIGGER‘ statement, specifying the trigger name, the triggering event, the affected table, and the trigger body containing SQL statements to execute when the trigger fires. Triggers can be set to execute before or after the triggering event and can operate on each affected row (‘FOR EACH ROW‘) or once per statement (‘FOR EACH STATEMENT‘). Careful consideration should be given to the use of triggers, as they can introduce complexity and overhead to database operations. Overuse or poorly designed triggers can negatively impact database performance and maintainability. Therefore, triggers should be employed judiciously for tasks that cannot be accomplished using other database features or constraints.

## 4.4 Database Programming

Database programming involves writing code to interact with a database management system (DBMS), enabling various operations like querying, inserting, updating, and deleting data. Typically, developers utilize programming languages such as SQL or languages with built-in support for database access, like Python or Java. These languages facilitate tasks such as connecting to a database, executing SQL queries, handling transactions, processing result sets, and implementing error handling mechanisms. Security measures, including parameterized queries and access control, are also essential to prevent vulnerabilities like SQL injection attacks. Proficiency in database programming is crucial for building dynamic web applications, desktop software, and mobile apps that rely on efficient and secure data storage and retrieval.

In database programming, developers must understand database concepts, SQL syntax, and best practices for database interactions. Effective database programming skills enable the creation of robust and scalable software applications capable of handling large volumes of data while ensuring data consistency, integrity, and security. By mastering database programming techniques, developers can build reliable database-driven applications tailored to meet the needs of various domains and industries.

## 4.5 Database Connection

Establishing a database connection is vital for database programming, allowing applications to interact with databases. Developers use connection libraries or APIs provided by the DBMS to programmatically set up connections, specifying parameters like host, port, username, password, and database name. Once established, applications can execute SQL queries and perform database operations. Proper connection management, including opening, closing, and pooling connections, is essential to ensure efficient resource usage and prevent leaks. Error handling mechanisms should also be implemented to manage connection failures gracefully and provide users with informative feedback.

## 4.6 Stored Procedures

Stored procedures play a pivotal role in database programming, offering a convenient way to encapsulate and execute frequently used database operations. These procedures are precompiled SQL statements stored in the database server, providing several benefits such as improved performance, enhanced security, and simplified maintenance. Developers can invoke stored procedures from application code by calling their names and passing parameters, enabling seamless integration of business logic within database operations. Stored procedures can perform a wide range of tasks, including data manipulation, validation, and complex computations, reducing the need for repetitive SQL statements in application code. Moreover, stored procedures promote code reusability and maintainability, as changes made to the procedure logic can be applied universally across all applications that utilize them. Effective use of stored procedures enhances application performance, scalability, and security while promoting modular and maintainable database-centric application architectures.

# Chapter 5

# Requirements Specification

## 5.1 Hardware Requirements

- Processor : Any Processor more than 500 MHz
- RAM : 2GB
- Hard Disk : 500GB
- Input Device : Standard Keyboard and Mouse
- Output Device : Monitor

## 5.2 Software Requirements

- Database :MySQL 8.0 or higher
- Programming Language : Python
- IDE : Apache Netbeans 12.0 or higher, PyCharm 2021.3 or higher
- Operating System : Windows 10

# Chapter 6

# Relational Database Design

## 6.1 Entity Relation(ER) Diagram

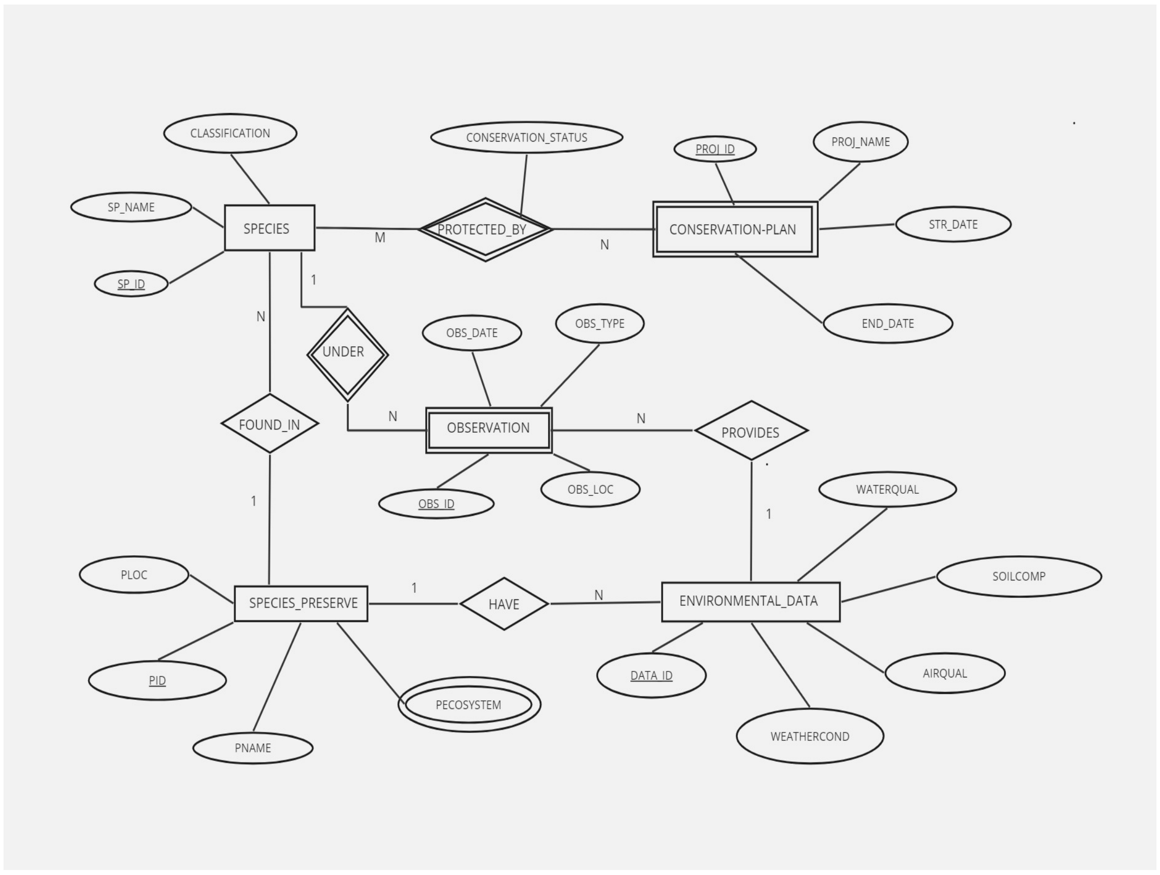


Figure 6.1: Schema Diagram of Management System

## 6.2 Mapping From ER Diagram to Relational Schema Diagram

**1.Mapping of Regular Entities**:This step involves mapping all the regular entity types to tabular format by identifying their primary keys.
**2.Mapping of 1:1 Relation:**In this step foreign keys are assigned using foreign key approach.The primary key of the participating relation R or S is added as primary key to second entity types by looking at the participating constraints.
**3.Mapping of 1:N Relation:**Foreign key approach is used to add one sided primary key to the n sided entity at foreign key.
**4.Mapping of M:N Relation:**Here we use the cross reference approach where the relationship is converted to a new relation within attributes on primary keys of both participating relation.
**5.Mapping of Weak Entity:**When mapping weak entity types along with other attributes the partial key and primary key of parent entity together will form their primary key of the new relation.
**6.Mapping of N-ary Relation:**For mapping N array relationship we create a new relation with a relationship name in its attribute and primary keys of all participating entity types.
**7.Mapping of Multivalued Relation:**For multivalued attributes a separate relation has to be created along with primary key of parent relation.

In our database we have the following mappings:
**Step – 1 : Mapping of Regular Entities.**
From the ER diagram we identify all the strong entities E and create a relation R that includes all it's simple attributes and primary keys.
The following are the strong entities from our schema diagram :
1.SPECIES(SP-ID,CLASSIFICATION,SP-NAME)
2.ENVIRONMENTAL-DATA(DATA-ID,WATERQUAL,SOILCOMP,AIRQUAL,WEATHER
3.SPECIESPRESERVE(PID,PLOC,PNAME,PECOSYTEM)

**Step – 2 : Mapping of binary 1:1 Relation Types.**

There is no 1:1 relation

**Step – 3 : Mapping of binary 1:N Relation Types.**

The SPECIES and the SPECIES-PRESERVE entities are participating in the 1:N relation type. similarly OBSERVATION and SPECIES , OBSERVATION and ENVIRONMENTAL-DATA,SPECIES-PRESERVE and ENVIRONMENTAL-DATA are in 1:1 relations

**Step – 4 : Mapping of binary M:N Relation Types.**

The relationship between the SPECIES and the CONSERVATION-PLAN is M:N .So we create a new relation PROTECTED-BY which includes the primary key of SPECIES and CONSERVATION-PLAN entity. The combination of the two primary keys will form the primary key of the PURCHASES relation.

## 6.3 Relational Schema Diagram

A Schema is a pictorial representation of the relationship between the database tables in the database that is created. The database schema of a database system is its structure described in a formal language sup ported by the database management system (DBMS). The term "schema" refers to the organization of data as a blueprint of how the database is constructed (divided into database tables in the case of relational databases). The formal definition of a database schema is a set of formulas (sentences) called integrity constraints imposed on a database. These integrity constraints ensure compatibility between parts of the schema. All constraints are expressible in the same language. A database can be considered a structure in realization of the database language.The states of a created conceptual schema are transformed into an explicit mapping, the database schema. This describes how real-world entities are modelled in the database.

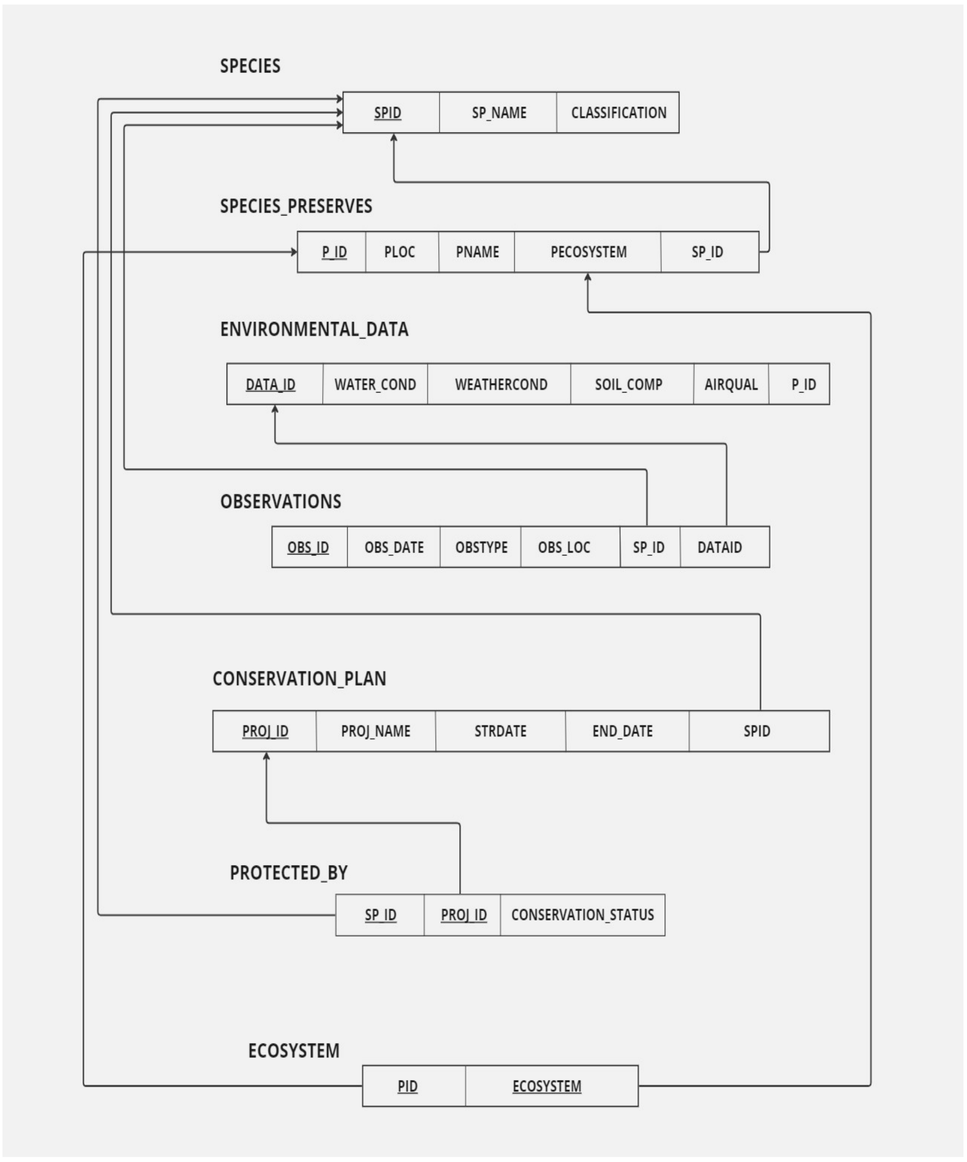


Figure 6.2: Schema Diagram of Species Preservation Management System

# Chapter 7

# Implementation

## 7.1 Table Structure

**SPECIES**

CREATE TABLE SPECIES(

SP ID VARCHAR(30) NOT NULL PRIMARY KEY,

SP NAME VARCHAR(50) NOT NULL,

SP CLASSIFICATION VARCHAR(50));

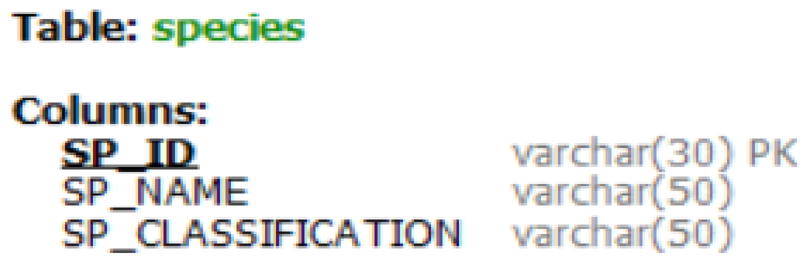


Table: species

Columns:

| | |
|---|---|
| SP_ID | varchar(30) PK |
| SP_NAME | varchar(50) |
| SP_CLASSIFICATION | varchar(50) |

Figure 7.1: Structure of Species table

**SPECIES PRESERVES**

CREATE TABLE SPECIES PRESERVES (

PID VARCHAR(30) NOT NULL PRIMARY KEY,

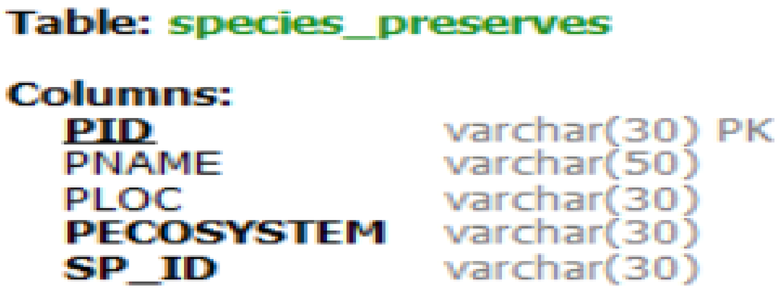
PNAME VARCHAR(50) NOT NULL,

PLOC VARCHAR(30),

PECOSYSTEM VARCHAR(50),

SP ID VARCHAR(30),

FOREIGN KEY (SP ID) REFERENCES SPECIES(SP ID) );



```
Table: species_preserves

Columns:
   PID          varchar(30) PK
   PNAME        varchar(50)
   PLOC         varchar(30)
   PECOSYSTEM   varchar(30)
   SP_ID        varchar(30)
```

Figure 7.2: Structure of Species preserves table

**ENVIRONMENTAL DATA**

CREATE TABLE ENVIRONMENTAL DATA (

D ID VARCHAR(30) PRIMARY KEY,

WATER QUAL VARCHAR(30),

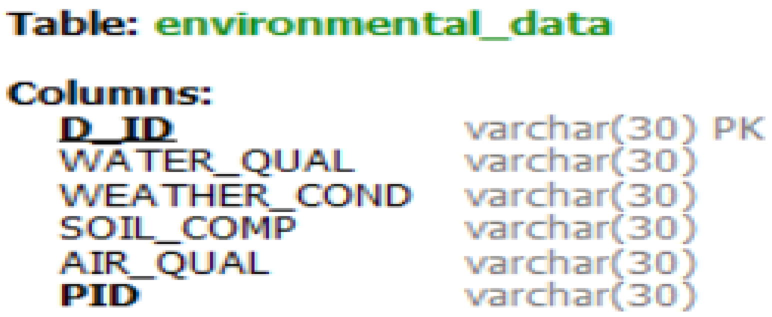
WEATHER COND VARCHAR(30),

SOIL COMP VARCHAR(30),

AIR QUAL VARCHAR(30),

PID VARCHAR(30),

FOREIGN KEY (PID) REFERENCES SPECIES PRESERVES(PID) );



```
Table: environmental_data

Columns:
   D_ID           varchar(30) PK
   WATER_QUAL     varchar(30)
   WEATHER_COND   varchar(30)
   SOIL_COMP      varchar(30)
   AIR_QUAL       varchar(30)
   PID            varchar(30)
```

Figure 7.3: Structure of Environmental data table

**OBSERVATIONS**

CREATE TABLE OBSERVATIONS (

OB ID VARCHAR(30) PRIMARY KEY,

OB DATE DATE,

OB LOC VARCHAR(30),

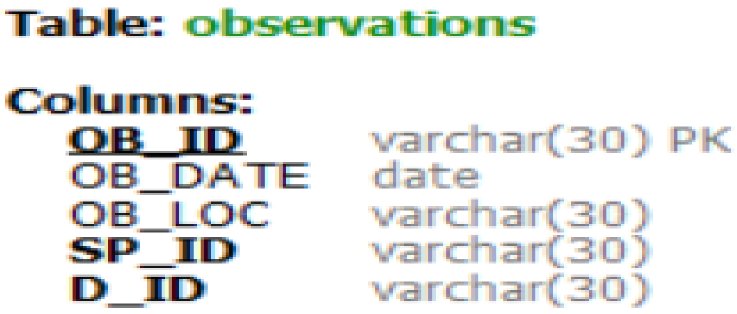


Table: observations

Columns:

| | |
|---|---|
| OB_ID | varchar(30) PK |
| OB_DATE | date |
| OB_LOC | varchar(30) |
| SP_ID | varchar(30) |
| D_ID | varchar(30) |

Figure 7.4: Structure of Observations table

SP ID VARCHAR(30),

D ID VARCHAR(30),

FOREIGN KEY (SP ID) REFERENCES SPECIES(SP ID),

FOREIGN KEY (D ID) REFERENCES ENVIRONMENTAL DATA(D ID) );

**CONSERVATION PLAN**

CREATE TABLE CONSERVATION PLAN (

PROJ ID VARCHAR(30) PRIMARY KEY,

PROJ NAME VARCHAR(70),

STR DATE DATE,

END DATE DATE,

SP ID VARCHAR(30),

FOREIGN KEY (SP ID) REFERENCES SPECIES(SP ID) );

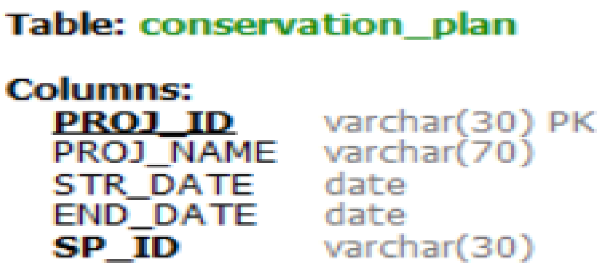


Table: conservation_plan

Columns:

| | |
|---|---|
| PROJ_ID | varchar(30) PK |
| PROJ_NAME | varchar(70) |
| STR_DATE | date |
| END_DATE | date |
| SP_ID | varchar(30) |

Figure 7.5: Structure of Conservations table

**PROTECTED BY**

CREATE TABLE PROTECTED BY (
CONSERVATION STATUS VARCHAR(30),
SP ID VARCHAR(30),
PROJ ID VARCHAR(30),
PRIMARY KEY (SP ID, PROJ ID),
FOREIGN KEY (SP ID) REFERENCES SPECIES(SP ID) ON DELETE CASCADE,
FOREIGN KEY (PROJ ID) REFERENCES CONSERVATIONPLAN(PROJ ID) ON DELETE CASCADE );

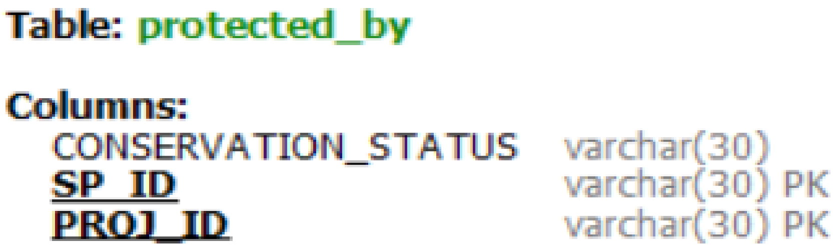


Table: protected_by

Columns:

| | |
|---|---|
| CONSERVATION_STATUS | varchar(30) |
| SP_ID | varchar(30) PK |
| PROJ_ID | varchar(30) PK |

Figure 7.6: Structure of Protected By table

**ECOSYSTEM**

CREATE TABLE ECOSYSTEM (
PID VARCHAR(30),
PECOSYSTEM VARCHAR(30),
PRIMARY KEY (PID, PECOSYSTEM),
FOREIGN KEY (PID) REFERENCES SPECIESPRESERVES(PID) ON DELETE CASCADE,
FOREIGN KEY (PECOSYSTEM) REFERENCES SPECIESPRESERVES(PECOSYSTEM) ON DELETE CASCADE );

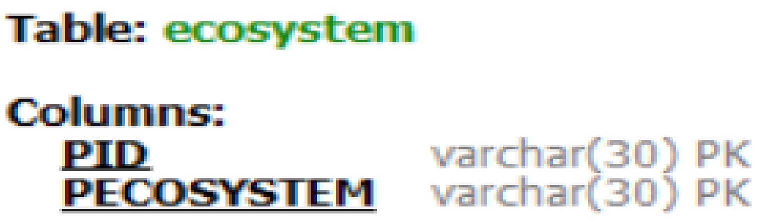


Table: ecosystem

Columns:

| | |
|---|---|
| PID | varchar(30) PK |
| PECOSYSTEM | varchar(30) PK |

Figure 7.7: Structure of Ecosystem table

**MANAGEMENTUSERS**

```
CREATE TABLE MANAGEMENTUSERS(
ID INT,
USERNAME VARCHAR(30),
EMAIL VARCHAR(30),
PASSWORD VARCHAR(30),
DEPARTMENT VARCHAR(30));
```

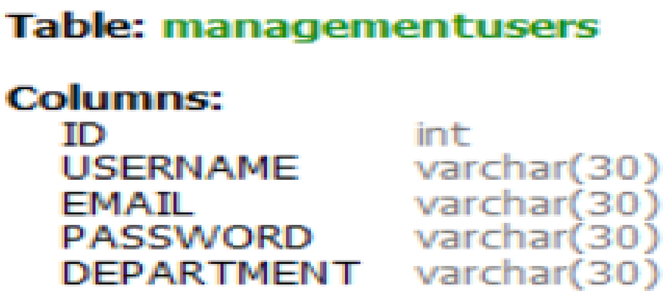


Table: managementusers

Columns:

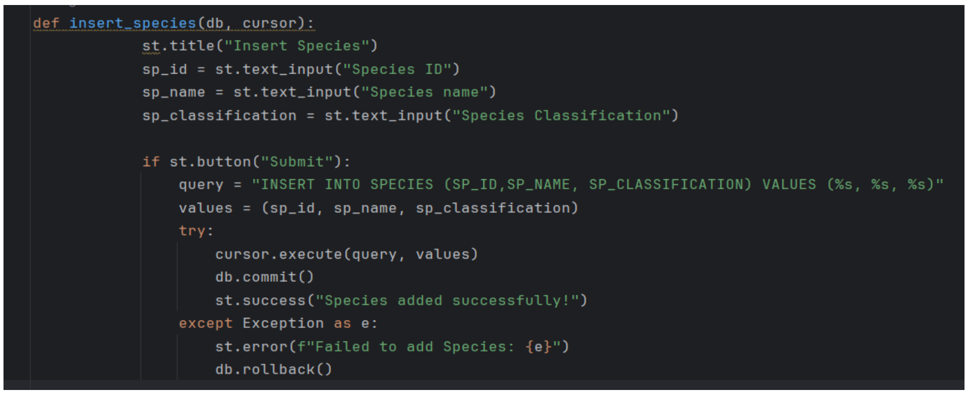
| | |
|---|---|
| ID | int |
| USERNAME | varchar(30) |
| EMAIL | varchar(30) |
| PASSWORD | varchar(30) |
| DEPARTMENT | varchar(30) |

Figure 7.8: Structure of Managementusers table

## 7.2 Codes Used For Modules:

### Insert

The code snippet from Figure 7.9 facilitates the insertion of species data into a database. It prompts users to input species details such as ID, name, and classification through text input fields. Upon clicking the submit button, the data is inserted into the 'SPECIES' table in the database. If successful, a confirmation message is displayed.

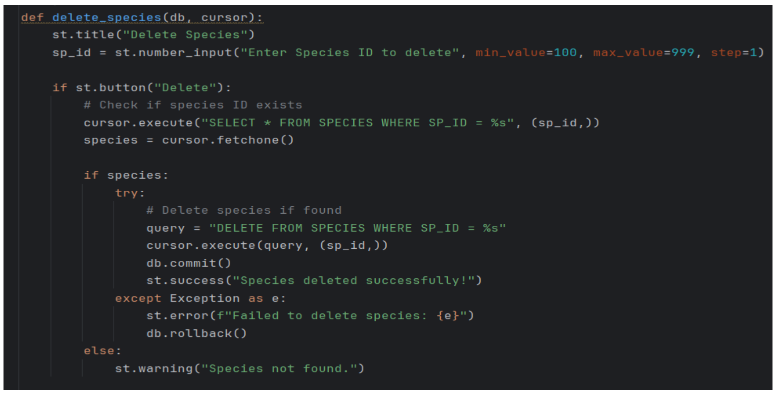


```
def insert_species(db, cursor):
            st.title("Insert Species")
            sp_id = st.text_input("Species ID")
            sp_name = st.text_input("Species name")
            sp_classification = st.text_input("Species Classification")

            if st.button("Submit"):
                query = "INSERT INTO SPECIES (SP_ID,SP_NAME, SP_CLASSIFICATION) VALUES (%s, %s, %s)"
                values = (sp_id, sp_name, sp_classification)
                try:
                    cursor.execute(query, values)
                    db.commit()
                    st.success("Species added successfully!")
                except Exception as e:
                    st.error(f"Failed to add Species: {e}")
                    db.rollback()
```

Figure 7.9: Insert for species table

### Delete

The provided function from fig 7.10 enables the deletion of species records from a database. It presents users with a title indicating the delete operation and prompts for the species ID to be deleted using a numeric input field. Upon clicking the delete button, the function verifies if the specified species ID exists in the database. If found, the corresponding species record is deleted, and a success message is displayed.

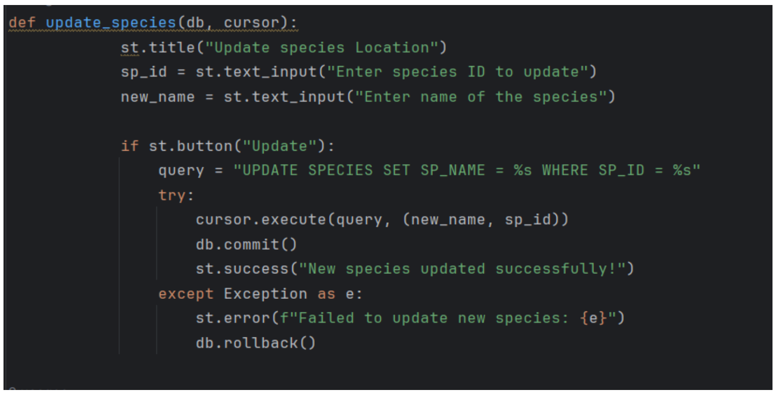


```
def delete_species(db, cursor):
    st.title("Delete Species")
    sp_id = st.number_input("Enter Species ID to delete", min_value=100, max_value=999, step=1)

    if st.button("Delete"):
        # Check if species ID exists
        cursor.execute("SELECT * FROM SPECIES WHERE SP_ID = %s", (sp_id,))
        species = cursor.fetchone()

        if species:
            try:
                # Delete species if found
                query = "DELETE FROM SPECIES WHERE SP_ID = %s"
                cursor.execute(query, (sp_id,))
                db.commit()
                st.success("Species deleted successfully!")
            except Exception as e:
                st.error(f"Failed to delete species: {e}")
                db.rollback()
        else:
            st.warning("Species not found.")
```

Figure 7.10: Delete for species table

## Update

The provided function from fig 7.11 facilitates the update of species records in the database, specifically allowing users to modify the name of a species identified by its unique ID. Upon accessing the update functionality, users are prompted to input the species ID and the new name for the species through text input fields. Upon clicking the update button, the function executes an SQL query to update the species name in the database.

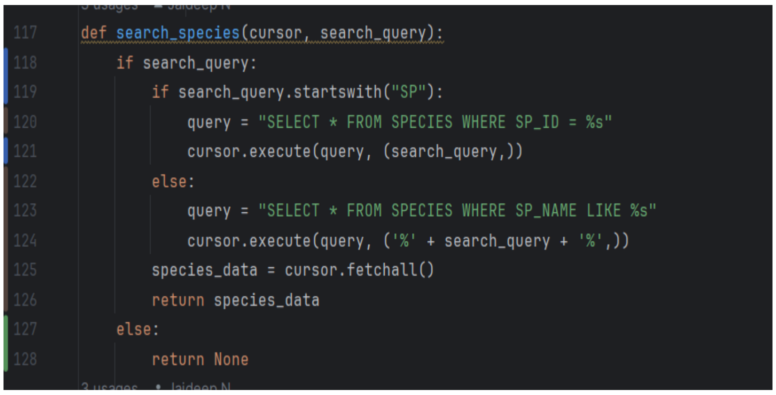


```python
def update_species(db, cursor):
            st.title("Update species Location")
            sp_id = st.text_input("Enter species ID to update")
            new_name = st.text_input("Enter name of the species")

            if st.button("Update"):
                query = "UPDATE SPECIES SET SP_NAME = %s WHERE SP_ID = %s"
                try:
                    cursor.execute(query, (new_name, sp_id))
                    db.commit()
                    st.success("New species updated successfully!")
                except Exception as e:
                    st.error(f"Failed to update new species: {e}")
                    db.rollback()
```

Figure 7.11: Update for species table

## Search

The function from fig 7.12 retrieves habitat information from the database based on a user-provided search query. It handles both habitat ID and name searches, constructing SQL queries accordingly. After executing the query, the function returns the retrieved habitat data for further processing.

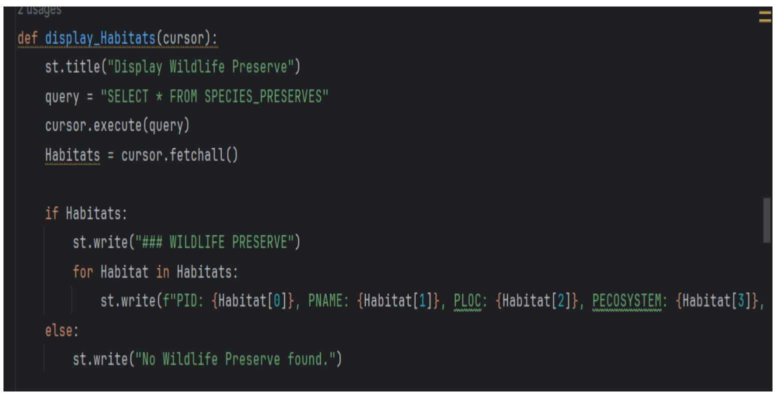


```python
def search_species(cursor, search_query):
    if search_query:
        if search_query.startswith("SP"):
            query = "SELECT * FROM SPECIES WHERE SP_ID = %s"
            cursor.execute(query, (search_query,))
        else:
            query = "SELECT * FROM SPECIES WHERE SP_NAME LIKE %s"
            cursor.execute(query, ('%' + search_query + '%',))
        species_data = cursor.fetchall()
        return species_data
    else:
        return None
```

Figure 7.12: search for Habitat table

## Display

The function from fig 7.13 retrieves and displays wildlife preserve information from the database. It executes a SQL query to fetch all records from the table and then iterates over the retrieved data to display details about each wildlife preserve. If no wildlife preserves are found in the database, it shows a corresponding message.

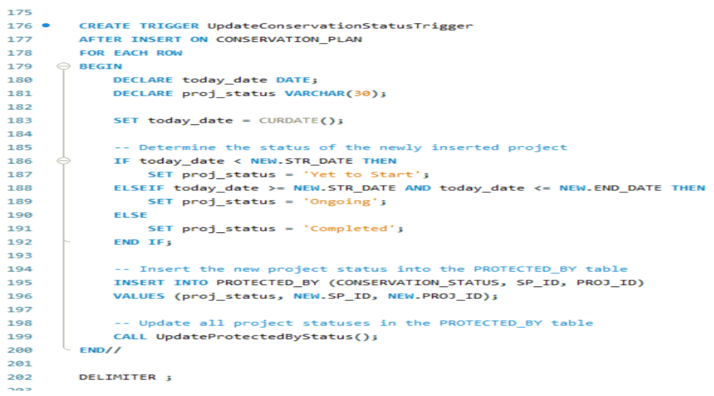


```python
def display_Habitats(cursor):
    st.title("Display Wildlife Preserve")
    query = "SELECT * FROM SPECIES_PRESERVES"
    cursor.execute(query)
    Habitats = cursor.fetchall()

    if Habitats:
        st.write("### WILDLIFE PRESERVE")
        for Habitat in Habitats:
            st.write(f"PID: {Habitat[0]}, PNAME: {Habitat[1]}, PLOC: {Habitat[2]}, PECOSYSTEM: {Habitat[3]},
    else:
        st.write("No Wildlife Preserve found.")
```

Figure 7.13: Display for wildlife preserve table

## Trigger

The provided trigger from fig 7.14 is designed to automatically update the conservation status of a project in the table based on the insertion of a new record into the CONSERVATIONPLAN table. It calculates the current status of the project based on the start and end dates of the conservation plan. If the current date is before the start date, the status is set to Yet to Start. If the current date is between the start and end dates, the status is set to Ongoing. Otherwise, if the current date is after the end date, the status is set to Completed. After determining the status, the trigger inserts a new record into the PROTECTEDBY table, including the project status along with the species ID and project ID.

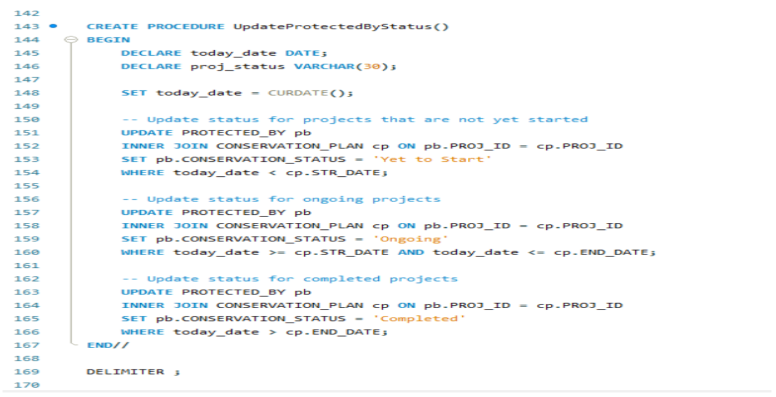


```sql
CREATE TRIGGER UpdateConservationStatusTrigger
AFTER INSERT ON CONSERVATION_PLAN
FOR EACH ROW
BEGIN
    DECLARE today_date DATE;
    DECLARE proj_status VARCHAR(30);

    SET today_date = CURDATE();

    -- Determine the status of the newly inserted project
    IF today_date < NEW.STR_DATE THEN
        SET proj_status = 'Yet to Start';
    ELSEIF today_date >= NEW.STR_DATE AND today_date <= NEW.END_DATE THEN
        SET proj_status = 'Ongoing';
    ELSE
        SET proj_status = 'Completed';
    END IF;

    -- Insert the new project status into the PROTECTED_BY table
    INSERT INTO PROTECTED_BY (CONSERVATION_STATUS, SP_ID, PROJ_ID)
    VALUES (proj_status, NEW.SP_ID, NEW.PROJ_ID);

    -- Update all project statuses in the PROTECTED_BY table
    CALL UpdateProtectedByStatus();
END//

DELIMITER ;
```

Figure 7.14: Trigger for SpeciesPreserve table

### Stored Procedure

The stored procedure from 7.15 updates the conservation status of projects in the PROTECTEDBY table based on their start and end dates. It declares variables for the current date and project status before updating the status of projects by joining with the CONSERVATIONPLAN table. It sets the status to Yet to Start for projects not yet started, Ongoing for those in progress, and Completed for finished projects, ensuring accurate reflection of project statuses in the PROTECTED$_BYtable.$



```
CREATE PROCEDURE UpdateProtectedByStatus()
BEGIN
    DECLARE today_date DATE;
    DECLARE proj_status VARCHAR(30);

    SET today_date = CURDATE();

    -- Update status for projects that are not yet started
    UPDATE PROTECTED_BY pb
    INNER JOIN CONSERVATION_PLAN cp ON pb.PROJ_ID = cp.PROJ_ID
    SET pb.CONSERVATION_STATUS = 'Yet to Start'
    WHERE today_date < cp.STR_DATE;

    -- Update status for ongoing projects
    UPDATE PROTECTED_BY pb
    INNER JOIN CONSERVATION_PLAN cp ON pb.PROJ_ID = cp.PROJ_ID
    SET pb.CONSERVATION_STATUS = 'Ongoing'
    WHERE today_date >= cp.STR_DATE AND today_date <= cp.END_DATE;

    -- Update status for completed projects
    UPDATE PROTECTED_BY pb
    INNER JOIN CONSERVATION_PLAN cp ON pb.PROJ_ID = cp.PROJ_ID
    SET pb.CONSERVATION_STATUS = 'Completed'
    WHERE today_date > cp.END_DATE;
END//

DELIMITER ;
```

Figure 7.15: Stored Procedure for protected by table

# Chapter 8

# Results and Discussion

**Login Page:**

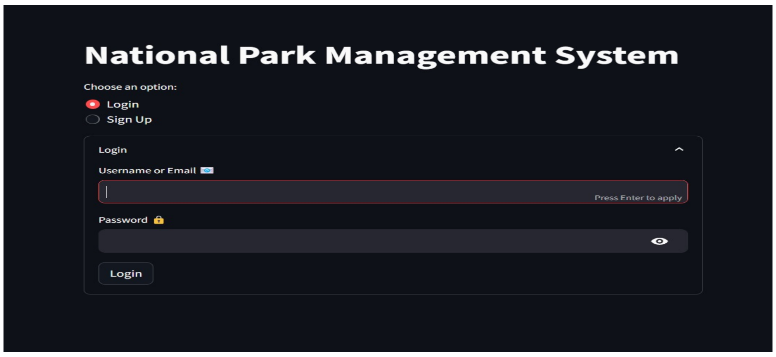
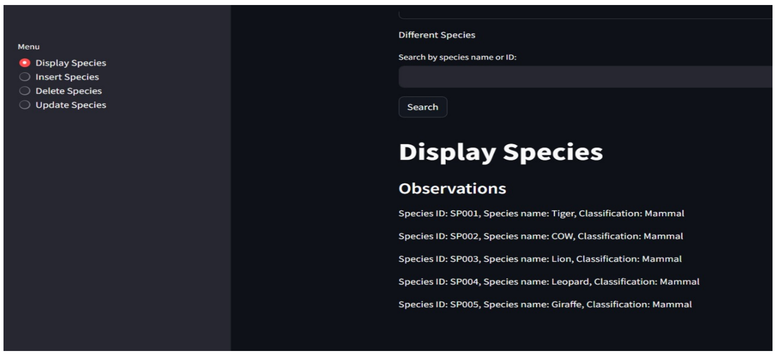


Figure 8.1: Login Page

This is our login page.Here the National Park Management or the Department can login.

**Species Details:**



Display Species

Observations

Species ID: SP001, Species name: Tiger, Classification: Mammal

Species ID: SP002, Species name: COW, Classification: Mammal

Species ID: SP003, Species name: Lion, Classification: Mammal

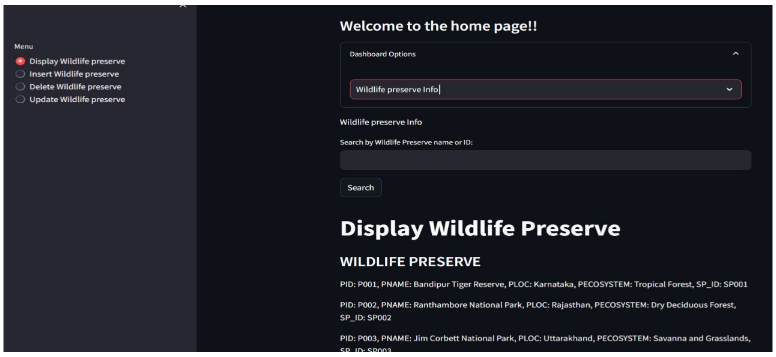
Species ID: SP004, Species name: Leopard, Classification: Mammal

Species ID: SP005, Species name: Giraffe, Classification: Mammal

Figure 8.2: Adding Species Details

This page allows department to add species details i.e name, id and classification.

**Wildlife preserve details:**



Welcome to the home page!!

Wildlife preserve Info

Wildlife preserve Info

Search by Wildlife Preserve name or ID:

Display Wildlife Preserve

WILDLIFE PRESERVE

PID: P001, PNAME: Bandipur Tiger Reserve, PLOC: Karnataka, PECOSYSTEM: Tropical Forest, SP_ID: SP001

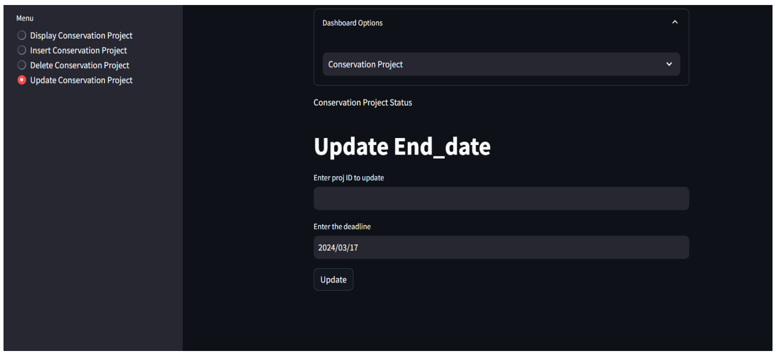
PID: P002, PNAME: Ranthambore National Park, PLOC: Rajasthan, PECOSYSTEM: Dry Deciduous Forest, SP_ID: SP002

PID: P003, PNAME: Jim Corbett National Park, PLOC: Uttarakhand, PECOSYSTEM: Savanna and Grasslands,

Figure 8.3: Add wildlife preserves details

Here department can add wildlife reserves of the species by providing the species id.

**Update conservation project details:**


Menu
Display Conservation Project
Insert Conservation Project
Delete Conservation Project
Update Conservation Project
Dashboard Options
Update

Conservation Project

Conservation Project Status

Update End_date

Enter proj ID to update

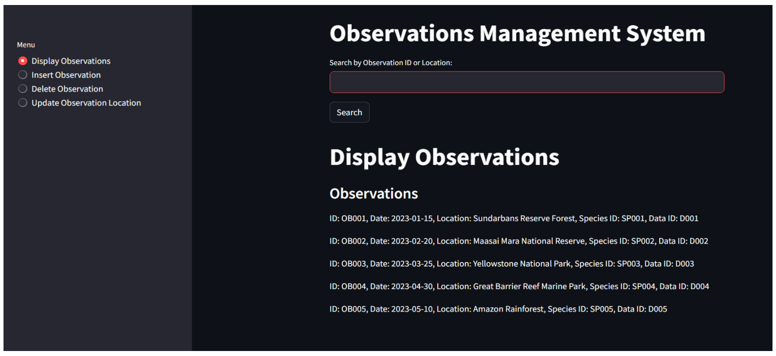
Enter the deadline

2024/03/17

Figure 8.4: Updating Conservation project Details:

Here department can update the end date or deadline of the conservation project.

**Search Observations:**



Observations Management System

Search by Observation ID or Location:

Display Observations

Observations

ID: OB001, Date: 2023-01-15, Location: Sundarbans Reserve Forest, Species ID: SP001, Data ID: D001

ID: OB002, Date: 2023-02-20, Location: Maasai Mara National Reserve, Species ID: SP002, Data ID: D002

ID: OB003, Date: 2023-03-25, Location: Yellowstone National Park, Species ID: SP003, Data ID: D003

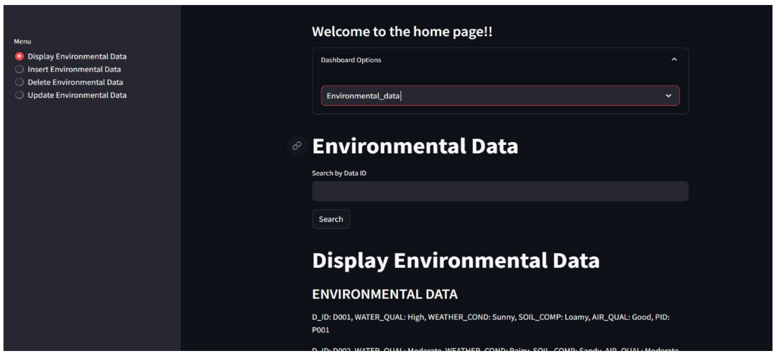
ID: OB004, Date: 2023-04-30, Location: Great Barrier Reef Marine Park, Species ID: SP004, Data ID: D004

ID: OB005, Date: 2023-05-10, Location: Amazon Rainforest, Species ID: SP005, Data ID: D005

Figure 8.5: Searching species by Observation id or location

Here department can search for species by entering the observation id or location.

**Environmental data:**


Menu
Display Environmental Data
Insert Environmental Data
Delete Environmental Data
Update Environmental Data
Dashboard Options
Search

Welcome to the home page!!

Environmental_data

Environmental Data

Search by Data ID

Display Environmental Data

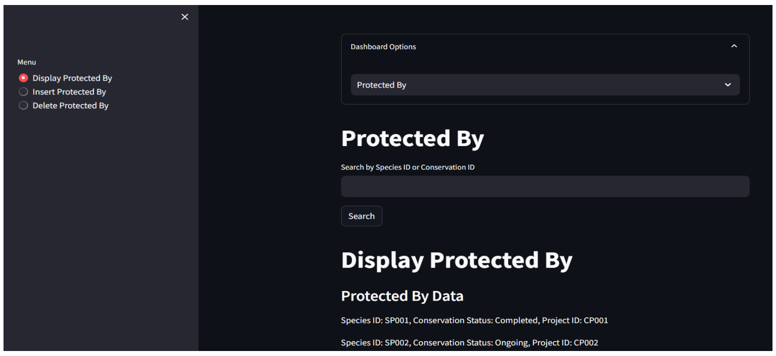
ENVIRONMENTAL DATA

D_ID: D001, WATER_QUAL: High, WEATHER_COND: Sunny, SOIL_COMP: Loamy, AIR_QUAL: Good, PID: P001

Figure 8.6: Information on environmental data

Environmental Data of the particular wildlife preserve is displayed and also can be updated, inserted or deleted.

**Trigger and Stored procedure:**



Protected By

Protected By

Search by Species ID or Conservation ID

Display Protected By

Protected By Data

Species ID: SP001, Conservation Status: Completed, Project ID: CP001

Species ID: SP002, Conservation Status: Ongoing, Project ID: CP002

Figure 8.7: Trigger for updating conservation status information

The provided trigger is designed to automatically update the conservation status of a species based on the conservation project in the table protected by.he trigger inserts a new record into the PROTECTEDBY table, including the project status along with the species ID and project ID. Finally, it calls the UpdateProtected-ByStatus stored procedure to update all project statuses in the PROTECTEDBY tableThe stored procedure updates the conservation status of projects in the PROTECTEDBY table based on their start and end dates

# Chapter 9

# Conclusion and Future work

In conclusion, the project has successfully implemented various functionalities for managing species and wildlife preserves. Through the development of database triggers, stored procedures, and user interfaces, efficient data manipulation and retrieval have been achieved. The integration of these components has enhanced the overall functionality and usability of the system, providing users with a comprehensive platform for wildlife conservation management.

Looking ahead, future work could focus on expanding the system's capabilities to include advanced features such as data analysis tools, reporting functionalities, and integration with external databases or APIs. Additionally, enhancements to the user interface, including improved visualization and interactivity, could further enhance user experience and productivity. Moreover, incorporating machine learning algorithms for species prediction or habitat modeling could provide valuable insights for conservation efforts. Overall, continuous development and refinement of the system would contribute to its effectiveness in supporting wildlife conservation initiatives.

# References

[1] Database systems Models, Languages, Design and Application Programming, Ramez Elmasri and Shamkant B. Navathe, 6th Edition, Pearson.

[2] Database management systems, Ramakrishnan, and Gehrke, 3rd Edition, 2014, McGraw Hill.

[3] Silberschatz Korth and Sudharshan: Database System Concepts, 6th Edition, Mc-Graw Hill, 2013.

[4] Coronel, Morris, and Rob, Database Principles Fundamentals of Design, Implementation and Management, Cengage Learning 2012.