MS SQL Server Database task done:  
Database Name: Bays\_Mountain

Tables Created:  
 CREATE TABLE Keepers (

KeeperID INT PRIMARY KEY,

FirstName VARCHAR(50),

LastName VARCHAR(50),

ContactNumber VARCHAR(15),

Email VARCHAR(100)

);

CREATE TABLE Enclosures (

EnclosureID INT PRIMARY KEY,

EnclosureName VARCHAR(50),

Location VARCHAR(100),

Size FLOAT,

Type VARCHAR(50)

);

CREATE TABLE Mammals (

MammalID INT PRIMARY KEY,

Species VARCHAR(50),

Name VARCHAR(50),

BirthDate DATE,

Gender VARCHAR(10),

EnclosureID INT,

KeeperID INT,

FOREIGN KEY (EnclosureID) REFERENCES Enclosures(EnclosureID),

FOREIGN KEY (KeeperID) REFERENCES Keepers(KeeperID)

);

CREATE TABLE Observations (

ObservationID INT PRIMARY KEY IDENTITY(1,1),

MammalID INT FOREIGN KEY REFERENCES Mammals(MammalID),

Date DATE,

ObservationType NVARCHAR(100),

Finding NVARCHAR(500),

Notes NVARCHAR(500)

);

CREATE TABLE HabitatCleaningRecords (

CleaningID INT PRIMARY KEY IDENTITY(1,1),

MammalID INT FOREIGN KEY REFERENCES Mammals(MammalID),

Date DATE,

CleaningDescription NVARCHAR(500),

Notes NVARCHAR(500)

);

CREATE TABLE FeedingRecords (

FeedingID INT PRIMARY KEY,

MammalID INT,

Date DATE,

FoodType VARCHAR(50),

Quantity FLOAT,

FeedingNotes NVARCHAR(500),

FOREIGN KEY (MammalID) REFERENCES Mammals(MammalID)

);

CREATE TABLE MedicalRecords (

MedicalRecordID INT PRIMARY KEY IDENTITY(1,1),

MammalID INT FOREIGN KEY REFERENCES Mammals(MammalID),

Date DATE,

EncounterType NVARCHAR(100),

Observation NVARCHAR(500),

Intervention NVARCHAR(500),

InjuryLogType NVARCHAR(100),

InjuryDescription NVARCHAR(500),

ExamType NVARCHAR(100),

AnimalSedated BIT,

VetNotified BIT,

VetResponse NVARCHAR(500),

MedicationAdministered BIT,

MedicalNotes NVARCHAR(500)

);

CREATE TABLE SedationRecords (

SedationID INT PRIMARY KEY IDENTITY(1,1),

MammalID INT FOREIGN KEY REFERENCES Mammals(MammalID),

Date DATE,

SedationMedication NVARCHAR(100),

SedationKitUsed NVARCHAR(100),

AdministrationMethod NVARCHAR(100),

Dose FLOAT,

TimeAdministered TIME,

TimeToRespond TIME

);

CREATE TABLE MedicationRecords (

MedicationID INT PRIMARY KEY IDENTITY(1,1),

MammalID INT FOREIGN KEY REFERENCES Mammals(MammalID),

Date DATE,

Type NVARCHAR(100),

Dose FLOAT,

AdministrationRoute NVARCHAR(100),

DidAnimalTakeMedication BIT,

DidAnimalSedate BIT

);

CREATE TABLE VetRecords (

VetRecordID INT PRIMARY KEY IDENTITY(1,1),

MammalID INT FOREIGN KEY REFERENCES Mammals(MammalID),

Date DATE,

Type NVARCHAR(100),

VetName NVARCHAR(100),

Location NVARCHAR(100),

Notes NVARCHAR(500)

);

CREATE TABLE BreedingRecords (

BreedingID INT PRIMARY KEY,

MammalID INT,

PartnerMammalID INT,

BreedingDate DATE,

Outcome VARCHAR(100),

FOREIGN KEY (MammalID) REFERENCES Mammals(MammalID),

FOREIGN KEY (PartnerMammalID) REFERENCES Mammals(MammalID)

);

MongoDB Database task done:

pip install pymongo streamlit

**Set Up MongoDB**

**Option 1: Use MongoDB Atlas (Cloud)**

1. **Create a MongoDB Atlas Account**:
   * Go to [MongoDB Atlas](https://www.mongodb.com/atlas).
   * Sign up for a free account and create a new cluster.
2. **Set Up a Cluster**:
   * Create a new cluster (use the free tier).
   * Wait for the cluster to initialize (may take a few minutes).
3. **Create a Database and Collections**:
   * Open the **Atlas Dashboard**.
   * Go to **Database > Collections** and create a new database, e.g., bays\_mountain\_db.
   * Add two collections: notes and events.
4. **Set Up a User**:
   * Go to **Database Access** and create a new database user.
   * Assign the user a username and password.
   * Choose Read and Write to any database as the role.
5. **Allow Connections**:
   * Go to **Network Access** and allow connections from 0.0.0.0/0 (all IPs) for testing purposes.
   * Once ready for production, restrict access to specific IPs.
6. **Get Connection String**:
   * In the **Clusters** tab, click **Connect > Connect Your Application**.
   * Copy the connection string (e.g., mongodb+srv://<username>:<password>@cluster0.mongodb.net/<dbname>?retryWrites=true&w=majority).

**Option 2: Use a Local MongoDB Instance**

1. **Download and Install MongoDB**:
   * Download MongoDB from the [official website](https://www.mongodb.com/try/download/community).
   * Follow the installation guide for your OS.
2. **Run the MongoDB Server**:
   * Start the MongoDB service or run the command:

bash

Copy code

mongod

* + By default, MongoDB listens on localhost:27017.

1. **Create a Database and Collections**:
   * Use the mongo shell or a GUI tool like **MongoDB Compass**.
   * Create a database (bays\_mountain\_db) and collections (notes and events).

If you're designing a relational database for an application similar to **Microsoft Notes** and **Google Calendar**, your database schema should address the following functionalities:

1. **Notes Application**:
   * Allow users to create, read, update, and delete notes.
   * Support shared or private notes.
   * Enable tags, categories, or folders for organization.
2. **Calendar Application**:
   * Allow users to create, edit, and delete events.
   * Support recurring events, reminders, and shared events.
   * Handle multiple users accessing and managing events.

**Relational Database Schema**

**1. Tables for Notes Functionality**

**Table: Users**

* user\_id (Primary Key): Unique identifier for each user.
* username: Name of the user.
* email: Email address of the user.
* password: Hashed password for authentication.
* created\_at: Timestamp when the user account was created.

**Table: Notes**

* note\_id (Primary Key): Unique identifier for each note.
* user\_id (Foreign Key): Links the note to a specific user.
* title: Title of the note.
* content: Content of the note (can use TEXT or VARCHAR(MAX) depending on the DBMS).
* is\_shared: Boolean flag indicating if the note is shared.
* created\_at: Timestamp when the note was created.
* updated\_at: Timestamp for the last modification.

**Table: Shared\_Notes**

* shared\_note\_id (Primary Key): Unique identifier for shared notes.
* note\_id (Foreign Key): Links to the Notes table.
* shared\_with\_user\_id (Foreign Key): Links to the Users table for the user the note is shared with.

**Table: Tags**

* tag\_id (Primary Key): Unique identifier for each tag.
* name: Name of the tag.

**Table: Note\_Tags**

* note\_tag\_id (Primary Key): Unique identifier for the relationship.
* note\_id (Foreign Key): Links to the Notes table.
* tag\_id (Foreign Key): Links to the Tags table.

**2. Tables for Calendar Functionality**

**Table: Events**

* event\_id (Primary Key): Unique identifier for each event.
* user\_id (Foreign Key): Links the event to a specific user.
* title: Title or name of the event.
* description: Details about the event.
* start\_time: Start time of the event.
* end\_time: End time of the event.
* is\_recurring: Boolean flag indicating if the event is recurring.
* recurrence\_pattern: Defines recurrence details (e.g., daily, weekly).
* created\_at: Timestamp when the event was created.
* updated\_at: Timestamp for the last modification.

**Table: Shared\_Events**

* shared\_event\_id (Primary Key): Unique identifier for shared events.
* event\_id (Foreign Key): Links to the Events table.
* shared\_with\_user\_id (Foreign Key): Links to the Users table for the user the event is shared with.

**Table: Reminders**

* reminder\_id (Primary Key): Unique identifier for each reminder.
* event\_id (Foreign Key): Links to the Events table.
* reminder\_time: Time to trigger the reminder (e.g., 30 minutes before the event).

**ER Diagram Representation**

Here's a conceptual overview of the relationships:

1. **Users** can create **Notes** and **Events**.
2. **Notes** can be tagged using **Tags** via the **Note\_Tags** table.
3. **Notes** and **Events** can be shared with other users via **Shared\_Notes** and **Shared\_Events**, respectively.
4. **Events** can have **Reminders** and optionally be recurring.

**SQL Scripts to Create Tables**

Here’s an example SQL script to create the Users and Notes functionality:

sql

CREATE TABLE Users (

user\_id INT IDENTITY(1,1) PRIMARY KEY,

username NVARCHAR(255) NOT NULL,

email NVARCHAR(255) UNIQUE NOT NULL,

password NVARCHAR(255) NOT NULL,

created\_at DATETIME2 DEFAULT GETDATE()

);

CREATE TABLE Notes (

note\_id INT IDENTITY(1,1) PRIMARY KEY,

user\_id INT NOT NULL,

title NVARCHAR(255) NOT NULL,

content NVARCHAR(MAX),

category NVARCHAR(255),

is\_shared BIT DEFAULT 0,

created\_at DATETIME2 DEFAULT GETDATE(),

updated\_at DATETIME2 DEFAULT GETDATE(),

CONSTRAINT FK\_Notes\_Users FOREIGN KEY (user\_id) REFERENCES Users(user\_id)

);

CREATE TABLE Shared\_Notes (

shared\_note\_id INT IDENTITY(1,1) PRIMARY KEY,

note\_id INT NOT NULL,

shared\_with\_user\_id INT NOT NULL,

CONSTRAINT FK\_Shared\_Notes\_Note FOREIGN KEY (note\_id) REFERENCES Notes(note\_id),

CONSTRAINT FK\_Shared\_Notes\_User FOREIGN KEY (shared\_with\_user\_id) REFERENCES Users(user\_id)

);

CREATE TABLE Tags ( id INT PRIMARY KEY IDENTITY(1,1), note\_id INT, tag\_name VARCHAR(255), FOREIGN KEY (note\_id) REFERENCES Notes(id) );

CREATE INDEX idx\_notes\_search ON Notes(title, content);

CREATE TABLE Note\_Tags (

note\_tag\_id INT IDENTITY(1,1) PRIMARY KEY,

note\_id INT NOT NULL,

tag\_id INT NOT NULL,

CONSTRAINT FK\_Note\_Tags\_Note FOREIGN KEY (note\_id) REFERENCES Notes(note\_id),

CONSTRAINT FK\_Note\_Tags\_Tag FOREIGN KEY (tag\_id) REFERENCES Tags(tag\_id)

);

CREATE TABLE Events (

event\_id INT IDENTITY(1,1) PRIMARY KEY,

user\_id INT NOT NULL,

title NVARCHAR(255) NOT NULL,

description NVARCHAR(MAX),

start\_time DATETIME2 NOT NULL,

end\_time DATETIME2 NOT NULL,

is\_recurring BIT DEFAULT 0,

recurrence\_pattern NVARCHAR(255),

created\_at DATETIME2 DEFAULT GETDATE(),

updated\_at DATETIME2 DEFAULT GETDATE(),

CONSTRAINT FK\_Events\_Users FOREIGN KEY (user\_id) REFERENCES Users(user\_id)

);

CREATE TABLE Shared\_Events (

shared\_event\_id INT IDENTITY(1,1) PRIMARY KEY,

event\_id INT NOT NULL,

shared\_with\_user\_id INT NOT NULL,

CONSTRAINT FK\_Shared\_Events\_Event FOREIGN KEY (event\_id) REFERENCES Events(event\_id),

CONSTRAINT FK\_Shared\_Events\_User FOREIGN KEY (shared\_with\_user\_id) REFERENCES Users(user\_id)

);

CREATE TABLE Reminders (

reminder\_id INT IDENTITY(1,1) PRIMARY KEY,

event\_id INT NOT NULL,

reminder\_time DATETIME2 NOT NULL,

CONSTRAINT FK\_Reminders\_Event FOREIGN KEY (event\_id) REFERENCES Events(event\_id)

);

**Key Points**

* A relational database is ideal for this use case due to structured data and clear relationships.
* The schema includes modular design with normalization to avoid redundancy.
* For scaling, consider indexing frequently queried columns (e.g., user\_id, event\_id, start\_time).

CREATE TABLE Categories (

category\_id INT IDENTITY(1,1) PRIMARY KEY,

name NVARCHAR(100) NOT NULL UNIQUE

);

CREATE TABLE Note\_Categories (

note\_category\_id INT IDENTITY(1,1) PRIMARY KEY,

note\_id INT NOT NULL,

category\_id INT NOT NULL,

CONSTRAINT FK\_Note\_Categories\_Note FOREIGN KEY (note\_id) REFERENCES Notes(note\_id),

CONSTRAINT FK\_Note\_Categories\_Category FOREIGN KEY (category\_id) REFERENCES Categories(category\_id)

);

**Enhancements to the notes\_page Function**

1. **User Authentication:**
   * Currently, the user\_id is a text input, which is not secure or scalable. Consider integrating a user authentication system to validate users and associate notes with their accounts.
   * You could use third-party libraries like auth0 or implement a simple login system.
2. **Error Handling:**
   * Add try-except blocks around database operations to handle potential errors gracefully and provide meaningful error messages to the user.
3. **Improved Note Display:**
   * Paginate notes if there are many. Use st.experimental\_memo to cache data and improve performance.
   * Add sorting options (e.g., sort by creation date or pinned status).
4. **Rich Text Editor Improvements:**
   * Use the st\_quill editor for better formatting and visual appeal.
   * Allow users to preview notes in rendered format.
5. **Pinning Notes:**
   * Replace the Pin button with a toggle switch or a star icon for better UI/UX.
6. **Search Functionality:**
   * Highlight search terms in the displayed notes for better visibility.

CREATE TABLE Tasks (

task\_id INT IDENTITY(1,1) PRIMARY KEY,

title NVARCHAR(255) NOT NULL,

description NVARCHAR(MAX),

date DATE NOT NULL,

assigned\_to INT NULL, -- Foreign key to Users table, if assigned

is\_reminder BIT NOT NULL DEFAULT 0, -- 1 for reminders, 0 for assigned tasks

created\_at DATETIME DEFAULT GETDATE(),

updated\_at DATETIME DEFAULT GETDATE()

);

ALTER TABLE Tasks

ADD CONSTRAINT FK\_Tasks\_Users

FOREIGN KEY (assigned\_to) REFERENCES Users(user\_id);

**Initialize Git (if not already initialized)**

If your project isn't already a Git repository:

git init

**Step 4: Configure Git (if not configured)**

Set your GitHub credentials (only needed once):

git config --global user.name "Your Name"

git config --global user.email "your-email@example.com"

**Step 5: Stage, Commit, and Push**

1. **Stage All Files**:

git add .

1. **Commit the Files**:

git commit -m "Initial commit: Added calendar and notes functionalities"

1. **Add the Remote Repository**: If you haven't added the remote yet:

git remote add origin https://github.com/<your-username>/bays-mountain-project.git

1. **Push to GitHub**:

git branch -M main

git push -u origin main

Changed the database to PostgreSQL:

The main reason being the difficulty in installing MSSQL in a new device. Faced too many issues regarding corrupt and incomplete installation. Therefore, we are choosing PostgreSQL for the project. With easy installation, there are various benefits to using PostgreSQL for this project like:  
Given the nature of your project (calendar, notes, and ethogram form data), **PostgreSQL** is the best choice:

1. **Advanced Features**: PostgreSQL offers better support for handling structured and unstructured data (e.g., JSON for calendar tasks).
2. **Scalability**: If your ethogram data grows, PostgreSQL can scale well.
3. **Performance**: PostgreSQL excels in handling complex queries and time-series data, which aligns with your calendar functionality.
4. **Integration**: It integrates seamlessly with Python and Streamlit through libraries like psycopg2 or SQLAlchemy.

Before setting up the database, make sure that you have turned off the firewall protection and windows defender. These anti-virus protection software might confuse some of the installation files as threat and remove them causing issues while installing the database server among other things.

**Create role bays\_owner with LOGIN PASSWORD 'B@V5';**

**GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO bays\_owner;**

**GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA pg\_catalog TO bays\_owner;**

**GRANT ALL PRIVILEGES ON ALL SEQUENCES IN SCHEMA public TO bays\_owner;**

**ALTER TABLE notes ADD COLUMN is\_pinned BOOLEAN DEFAULT true;**

If PostgreSQL Is Installed Locally:

If PostgreSQL is running on the same computer where your Python script will run:

The host will usually be:

localhost

Or 127.0.0.1 (loopback IP address)

Steps to Verify

Check postgresql.conf File:

Locate the postgresql.conf file. Its location depends on your operating system:

Windows: C:\Program Files\PostgreSQL\<version>\data\postgresql.conf

Look for the listen\_addresses setting:

listen\_addresses = 'localhost'

If it's set to 'localhost', then the database is only accessible locally. If it’s set to '\*' or an IP address, it accepts connections from those addresses.

Ensure that PostgreSQL is running:

* **Windows**: Check PostgreSQL service status in the **Services** application.