Task Management System

Task Management System implemented in Python using SQLite as the database. Let's break down the organization of the database structure and explain why this particular structure was chosen:

Database Structure:

The database contains a single table named 'tasks', which is used to store information about tasks. Here's the structure of the 'tasks' table:

id (INTEGER, PRIMARY KEY, AUTOINCREMENT): This field is used as the primary key for the table, and it automatically increments its value for each new task added. It ensures that each task has a unique identifier.

task_name (TEXT): This field stores the name or title of the task. It is of type TEXT, which can hold strings of variable length.

task_description (TEXT): This field stores the description or details of the task. It is also of type TEXT.

task_status (TEXT): This field stores the status of the task, which can be 'Pending' or any other status you define. It is of type TEXT to allow for flexibility in defining task statuses.

Why This Structure:

Separation of Concerns: The chosen structure separates different attributes of a task (name, description, and status) into individual fields, making it easy to manage and query tasks based on specific criteria. This separation also ensures that each task has a unique identifier (the id field).

Primary Key: Using an auto-incremented primary key (id) is a common practice in relational databases. It ensures that each task has a unique identifier, simplifying the retrieval and updating of specific tasks.

Flexibility: Storing task name, description, and status as TEXT fields provides flexibility. You can store tasks with varying lengths of names and descriptions, and you can define custom task statuses as needed.

Normalization: This database structure doesn't perform extensive normalization, as it's a relatively simple application. However, it achieves a basic level of normalization by storing task attributes in separate fields. If the application were more complex, you might consider additional tables to reduce data redundancy and improve data integrity.

Ease of Use: The structure is straightforward and easy to work with. It allows for common operations like adding, listing, updating, and deleting tasks, which are essential for a task management system.

User Interface Integration: This structure is suitable for integration with a user interface, as it maps well to how tasks are typically displayed and manipulated in a task management application.

In summary, the chosen database structure strikes a balance between simplicity and functionality for a basic task management system. It allows for efficient storage and retrieval of task data while remaining easy to understand and work within the provided Python code.