

WEB – BASED TASK MANAGEMENT SYSTEM

A PROJECT REPORT

(21CSC205P – Database Management Systems)

Submitted by

PRIYANSHU KUMAR [RA2211030010074]

ANSHIKA GAUTAM [RA2211030010076]

AVIV P JOJI [RA2211030010097]

Under the Guidance of

Dr.R.KAYALVIZHI

Assistant Professor, Department of Networking and Communications

in partial fulfillment of the requirements for the degree of

BACHELOR OF TECHNOLOGY

in

**COMPUTER SCIENCE AND ENGINEERING
specialization in Cyber Security**



DEPARTMENT OF NETWORKING AND COMMUNICATIONS

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR– 603 203

MAY 2024



SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR-603 203
BONAFIDE CERTIFICATE

Certified to be the bonafide work done by Aviv P Joji (RA2211030010097) of II year/IV sem B.Tech Degree Course in the Project Course – **21CSC205P Database Management Systems** in **SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**, Kattankulathur for the academic year 2023-2024.

Date:

Faculty in Charge

Dr.Kayalvizhi R
Assistant Professor
Department of Networking and Communications
SRM Institute of Science and Technology
Kattankulathur,
Chennai – 603 203

HEAD OF THE DEPARTMENT

Dr. Annapurani K
Professor & Head
Department of Networking and Communications
SRM Institute of Science and Technology
Kattankulathur,
Chennai – 603 203

ABSTRACT

Efficiently managing tasks, monitoring workflow among multiple people, and keeping track of the same is a crucial aspect of corporate management work. This project aims to deliver a unified and cost-effective solution that provides real-time task assignment on a web-based software. Our task management system offers users an intuitive interface to navigate, create, edit, delete, and organize their objectives. Simultaneously, these tasks are updated for all other users on the network, empowering them with a comprehensive view of their organization's workflow. The project's standout feature is the web-based application, allowing secure remote server administration and management. Built with security in mind, the application safeguards against common web vulnerabilities while providing a powerful tool for task analysis and management. The project emphasizes robust design and user-friendly interfaces for a seamless user experience. Users can perform various operations with confidence, including task creation, deletion and editing. The project's implementation adheres to best practices, featuring robust testing, comprehensive code documentation, version control, and alignment with coding standards. It serves as a practical and educational example of creating a web-based interface that enhances productivity and task management while remaining cost-effective.

PROBLEM STATEMENT

Purpose:

The purpose of our online task management system project is multifaceted. Primarily, it aims to alleviate the common challenges individuals and teams face in organizing and overseeing tasks and projects. By providing a centralized platform accessible anytime, anywhere, our system enables users to efficiently manage their workload, enhance productivity, and meet deadlines effectively. Furthermore, our project seeks to foster collaboration and communication among team members, promoting synergy and collective achievement of goals. Ultimately, our system aspires to empower users with the tools and capabilities necessary to navigate the complexities of modern work environments with confidence and ease.

Scope:

Our project's scope encompasses the comprehensive development of a web-based application with an array of robust task management features. From task creation and assignment to scheduling, progress tracking, and notification systems, our system offers a holistic solution to meet diverse user needs. Additionally, customizable dashboards, reporting functionalities, and seamless integration with other productivity tools enrich the user experience, ensuring adaptability and scalability. By prioritizing usability and functionality, our system aims to cater to the dynamic requirements of individuals, teams, and organizations across various industries, delivering tangible value and driving operational efficiency.

Target Audience:

The target audience for our online task management system is broad and inclusive, spanning professionals, teams, small businesses, and organizations of all sizes and industries. Freelancers seeking better organization, project managers coordinating complex initiatives, and entire departments collaborating on large-scale projects can benefit from our system's versatile capabilities. By tailoring features and functionalities to address the diverse needs and preferences of users, we aim to serve as a reliable ally in their pursuit of productivity and success. Whether managing personal tasks, team projects, or organizational objectives, our system offers a user-centric solution designed to empower individuals and teams to achieve their goals with confidence and efficiency.

Specific Requirements and Constraints:

In developing our online task management system, several specific requirements and constraints must be considered to ensure optimal performance, usability, and security. Firstly, the system must be accessible across multiple devices, including desktop computers, laptops, tablets, and smartphones, to accommodate diverse user preferences and facilitate seamless workflow management. Security measures, including robust user authentication, data encryption, and access control mechanisms, are imperative to protect sensitive information and uphold user privacy and confidentiality. Additionally, prioritizing intuitive user interface design and user experience optimization is essential to minimize learning curves and maximize user adoption and satisfaction. Lastly, seamless integration with third-party applications, such as calendar tools, email clients, and collaboration platforms, enhances interoperability and workflow efficiency, enriching the overall user experience and utility of our system.

TABLE OF CONTENTS

Chapter No	Chapter Name	Page No
1.	Problem understanding, Identification of Entity and Relationships, Construction of DB using ER Model for the project	7
2.	Design of Relational Schemas, Creation of Database Tables for the project.	15
3.	Complex queries based on the concepts of constraints, sets, joins, views, Triggers and Cursors.	29
4.	Analyzing the pitfalls, identifying the dependencies, and applying normalizations	37
5.	Implementation of concurrency control and recovery mechanisms	44
6.	Code for the project	47
7.	Result and Discussion (Screen shots of the implementation with front end)	52
8.	Online course certificate and GitHub link	59

CHAPTER 1

INRODUCTION

Problem Understanding-

Need for the project:

Implementing a task management system for an organization offers numerous benefits that contribute to improved efficiency, collaboration, and overall productivity. Here are some key reasons why organizations need a task management system.

Organization and Structure

Task management systems provide a centralized platform for organizing and structuring work. This helps prevent chaos, reduces confusion, and ensures that everyone is on the same page regarding their responsibilities.

Efficient Workflow

A task management system streamlines workflows by allowing users to create, assign, and track tasks in a systematic manner. This efficiency minimizes delays, bottlenecks, and the likelihood of tasks falling through the cracks.

Improved Collaboration

Collaboration is enhanced through features such as real-time updates, comments, and file sharing. Team members can communicate within the system, ensuring that everyone is informed and aligned on project progress.

Task Prioritization

The ability to prioritize tasks helps teams focus on the most important and time-sensitive activities. This ensures that resources are allocated efficiently, and critical deadlines are met.

Visibility and Accountability

Task management systems provide visibility into the status of tasks and projects. Team members can see who is responsible for each task, track progress, and identify any potential roadblocks. This transparency promotes accountability within the team.

Resource Optimization

Organizations can optimize resource allocation by assigning tasks based on individual strengths and expertise. This leads to a more efficient use of skills and talents within the team.

Time Management

Task management systems help individuals and teams manage their time effectively. By setting deadlines, receiving reminders, and tracking progress, employees can better prioritize their workload and avoid procrastination.

Adaptability and Flexibility

Modern task management systems often come with features that allow for flexibility and adaptability. Users can make adjustments to tasks, timelines, and priorities as project requirements evolve.

Data and Insights

Task management systems provide valuable data and insights into team and individual performance. Analytics and reporting features help organizations identify trends, assess productivity, and make data-driven decisions.

Remote Work Support

With the rise of remote work, a task management system facilitates collaboration and communication among distributed teams. Team members can access tasks and project information from anywhere, promoting flexibility and remote collaboration.

Reduced Email Overload

By centralizing task-related communication within the task management system, organizations can reduce the reliance on email for task updates and discussions. This can lead to a more organized and focused email communication.

Scalability

As organizations grow, a scalable task management system can adapt to increasing workloads, additional team members, and evolving project complexities. This scalability ensures that the system remains effective in managing organizational tasks over time.

Features:

Real-Time Status Monitoring

Real-Time Status Monitoring stands as a cornerstone of our architecture. This essential component operates by continuously collecting and presenting real-time task status to Project Managers. They benefit from a dynamic dashboard offering insights into their employees' performance and activity. This functionality seems to benefit both project managers and department leaders. Such a functionality requires a comprehensive backend, which has been deployed in MySQL.

Web-Based Application

The Web-Based Application component represents the gateway to remote server administration. Users can securely assign tasks on their servers through this versatile interface. It simplifies employee administration and facilitates job efficiency. This interactive and user-friendly application empowers administrators, project managers, and users to efficiently control and manage their duties and assigned responsibilities, whether performing routine work or working on specialized projects.

Database Management

Database Management is a fundamental component for preserving data consistency and integrity. To achieve this, we utilize MySQL. This robust system serves as the bridge between the application and the database, ensuring the secure storage and retrieval of user data and preferences. Whether it's managing assigned tasks, storing essential workflow information, or tracking employee performance, the Database Management component underpins data persistence and user-specific interactions, fostering a reliable and organized data environment.

The Security Layer

Security is paramount within our architectural framework. The Security Layer encompasses a multi-faceted approach to safeguarding user data and maintaining the confidentiality and integrity of the system. This layer incorporates robust authentication mechanisms, user authorization controls, and advanced data encryption protocols. By fortifying the system against potential threats, it ensures that user data remains protected and that sensitive information remains shielded from unauthorized access. Our unwavering commitment to security spans across the architecture, providing users with the assurance they need when working with sensitive data and executing critical system tasks.

In essence, our system architecture constitutes a harmonious assembly of components, meticulously crafted to fulfill distinct roles. These interwoven components collaboratively create an efficient, user-friendly, and secure web-based task management system interface.

ER Diagram

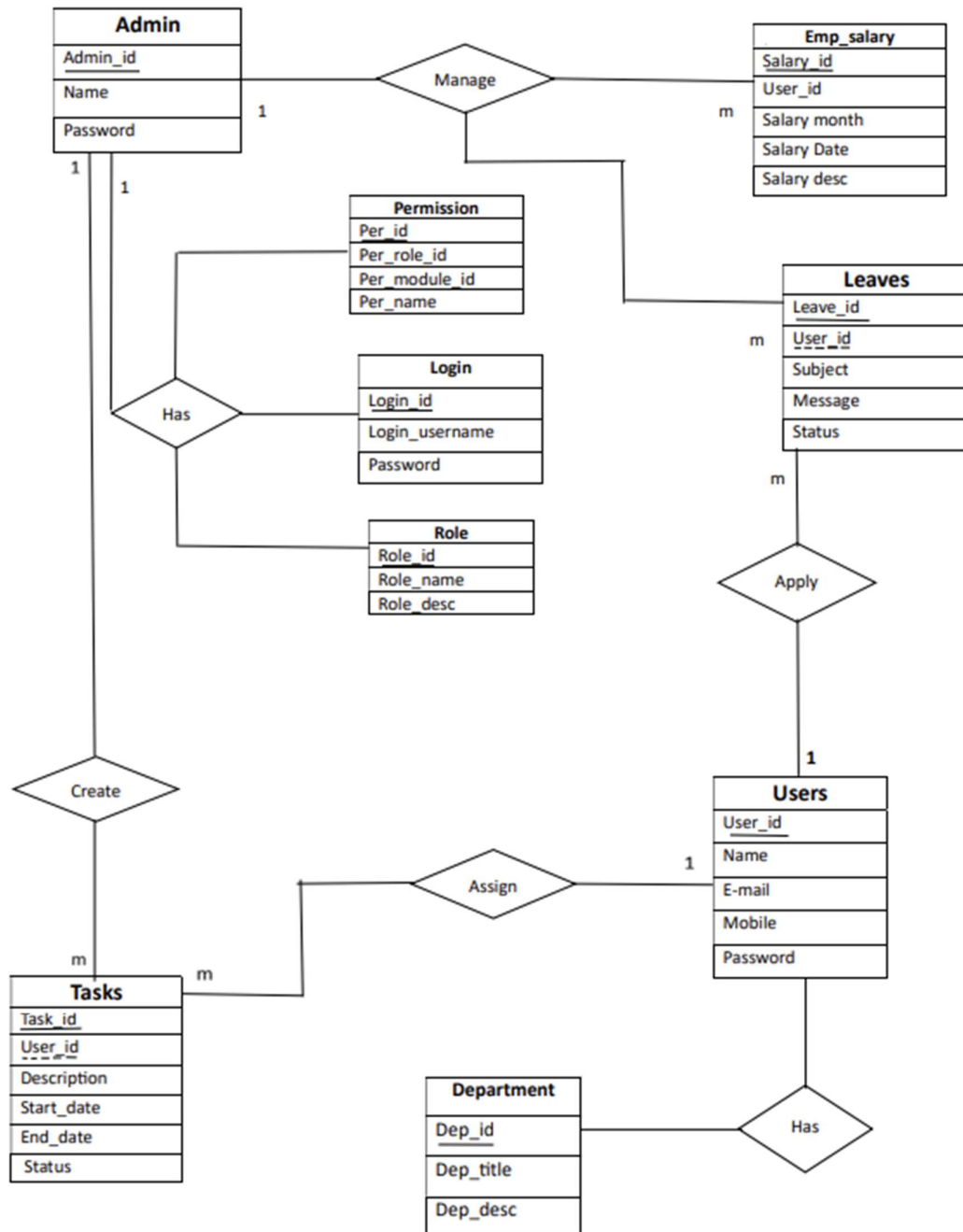


Fig 1.1: Entity Relationship diagram for Task management system

Explanation-

Admin Entity:

Attributes: AdminID (Primary Key), Name, Password

Relationships: Have relationships with other entities for activities like managing users, tasks, and permissions.

User Entity:

Attributes: UserID (Primary Key), Name, E-mail, Password, Mobile

Relationships: Associated with tasks, leave, login, permissions, departments, role, and emp salary.

Tasks Entity:

Attributes: TaskID (Primary Key), Description, StartDate, EndDate, Status, AssignedTo (Foreign Key referencing UserID)

Relationships: Linked to User (AssignedTo and TaskCreator)

Leave Entity:

Attributes: LeaveID (Primary Key), UserID (Foreign Key referencing UserID), LeaveStartDate, LeaveEndDate, LeaveStatus

Relationships: Connected to User (UserID).

Login Entity:

Attributes: LoginID (Primary Key), UserName, Password.

Relationships: Linked to Admin(LoginID)

Permission Entity:

Attributes: PermissionID (Primary Key), RoleID, ModuleID, PermissionName.

Relationships: Linked to Admin.

Departments Entity:

Attributes: DepartmentID (Primary Key), Title, Description.

Relationships: Associated with tasks, users.

Role Entity:

Attributes: RoleID (Primary Key), RoleName, Description.

Relationships: Associated with users, Admin.

Emp Salary Entity:

Attributes: SalaryID (Primary Key), UserID (Foreign Key referencing UserID), SalaryAmount, SalaryDate, Description.

Relationships: Linked to User (UserID), Admin.

Cardinality**Admin – User Relationship:**

Admin can manage multiple users (one-to-many).

User – Tasks Relationship:

A user can be assigned multiple tasks (one-to-many).

User – Leave Relationship:

A user can have multiple leave records (one-to-many).

Admin- Login Relationship:

An admin can have multiple login records (one-to-many).

Admin – Permission Relationship:

An admin can grant multiple permissions (one-to-many).

Tasks – Departments Relationship:

A task can be associated with one department (one-to-one).

User – Departments Relationship:

A user belongs to one department (one-to-one).

User – Role Relationship:

A user can have many role (one-to-many).

User – Emp Salary Relationship:

A user can have multiple salary records (one-to-many).

CHAPTER 2

DESIGN AND CREATION OF RELATIONAL TABLE

User Table :

Attributes	Data Type	Constraints
U_id	int	Primary Key
firstname	Varchar(30)	NOT NULL
lastname	Varchar(30)	NOT NULL
email	Varchar(20)	NOT NULL
password	Varchar(20)	NOT NULL
Mobile_no	Int(10)	NOT NULL
date_created	datetime	NOT NULL DEFAULT

Table 2.1: Overview of User table

Table name: USERTABLE

This is the name of the table where user information will be stored.

Attributes:

U_ID (USER ID):

Data Type: int

Constraints: Primary Key

Explanation: This column serves as the primary key for the table, which means each value must be unique and identifies a specific user in the table.

FIRSTNAME:

Data Type: Varchar(30)

Constraints: NOT NULL

Explanation: This column stores the first name of the user. The Varchar(30) data type allows for variable-length character strings with a maximum length of 30 characters. The NOT NULL constraint ensures that every record must have a value for this column.

LASTNAME:

Data Type: Varchar(30)

Constraints: NOT NULL

Explanation: Similar to the firstname column, this column stores the last name of the user.

EMAIL:

Data Type: Varchar(20)

Constraints: NOT NULL

Explanation: This column stores the email address of the user. The Varchar(20) data type is used for variable-length character strings with a maximum length of 20 characters.

PASSWORD:

Data Type: Varchar(20)

Constraints: NOT NULL

Explanation: This column stores the password associated with the user's account. The Varchar(20) data type is used for variable-length character strings with a maximum length of 20 characters.

MOBILE_NO (MOBILE NUMBER):

Data Type: Int(10)

Constraints: NOT NULL

Explanation: This column stores the user's mobile number as an integer with a maximum length of 10 digits. The NOT NULL constraint ensures that every record must have a value for this column.

DATE_CREATED:

Data Type: datetime

Constraints: NOT NULL DEFAULT

Explanation: This column stores the date and time when the user account was created. The datetime data type is used to represent date and time values. The NOT NULL constraint ensures that every record must have a value for this column, and the DEFAULT keyword is used to set a default value if none is provided.

Admin Table :

Attributes	Data Type	Constraints
A_Id	Int	Primary Key
firstname	Varchar(30)	NOT NULL
lastname	Varchar(30)	NOT NULL
email	Varchar(20)	NOT NULL
password	Varchar(20)	NOT NULL
Mobile_no	Int(10)	NOT NULL
date_created	date	NOT NULL DEFAULT

Table 2.2: Overview of Admin table

Table name: ADMIN

This is the name of the table where information about administrators will be stored.

ATTRIBUTES:

A_ID (ADMIN ID):

Data Type: Int

Constraints: Primary Key

Explanation: This column serves as the primary key for the table, ensuring that each value is unique and identifies a specific administrator in the table.

FIRSTNAME:

Data Type: Varchar(30)

Constraints: NOT NULL

Explanation: This column stores the first name of the administrator. The Varchar(30) data type allows for variable-length character strings with a maximum length of 30 characters. The NOT NULL constraint ensures that every record must have a value for this column.

LASTNAME:

Data Type: Varchar(30)

Constraints: NOT NULL

Explanation: Similar to the firstname column, this column stores the last name of the administrator.

EMAIL:

Data Type: Varchar(20)

Constraints: NOT NULL

Explanation: This column stores the email address of the administrator. The Varchar(20) data type is used for variable-length character strings with a maximum length of 20 characters.

PASSWORD:

Data Type: Varchar(20)

Constraints: NOT NULL

Explanation: This column stores the password associated with the administrator's account. The Varchar(20) data type is used for variable-length character strings with a maximum length of 20 characters.

MOBILE_NO (MOBILE NUMBER):

Data Type: Int(10)

Constraints: NOT NULL

Explanation: This column stores the mobile number of the administrator as an integer with a maximum length of 10 digits. The NOT NULL constraint ensures that every record must have a value for this column.

DATE_CREATED:

Data Type: date

Constraints: NOT NULL DEFAULT

Explanation: This column stores the date when the administrator account was created. The date data type is used to represent date values. The NOT NULL constraint ensures that every record must have a value for this column, and the DEFAULT keyword is used to set a default value if none is provided.

Task Table:

Attributes	Data Type	Constraints
t_id	int	Primary Key
U_id	int	Foreign Key
description	Text	NOT NULL
end_date	date	NOT NULL
start_date	date	NOT NULL
status	Varchar(200)	NOT NULL
date_created	date	NOT NULL DEFAULT

Table 2.3: Overview of Task table

Table name: TASK

This is the name of the table where information about tasks will be stored.

ATTRIBUTES

T_ID (TASK ID):

Data Type: int

Constraints: Primary Key

Explanation: This column serves as the primary key for the table, ensuring that each value is unique and identifies a specific task in the table.

U_ID (USER ID):

Data Type: int

Constraints: Foreign Key

Explanation: This column is a foreign key that references the U_id (User ID) column in another table. It establishes a relationship between the "task" table and the table containing user information.

DESCRIPTION:

Data Type: Text

Constraints: NOT NULL

Explanation: This column stores the description or details of the task. The Text data type allows for variable-length text storage.

END_DATE:

Data Type: date

Constraints: NOT NULL

Explanation: This column stores the end date of the task. The date data type is used to represent date values, and the NOT NULL constraint ensures that every record must have a value for this column.

START_DATE:

Data Type: date

Constraints: NOT NULL

Explanation: This column stores the start date of the task. Similar to the end_date column, the date data type is used, and the NOT NULL constraint ensures data integrity.

STATUS:

Data Type: Varchar(200)

Constraints: NOT NULL

Explanation: This column stores the status or current state of the task. The Varchar(200) data type allows for variable-length character strings with a maximum length of 200 characters.

DATE_CREATED:

Data Type: date

Constraints: NOT NULL DEFAULT

Explanation: This column stores the date when the task was created. The date data type is used, and the NOT NULL constraint ensures that every record must have a value for this column. The DEFAULT keyword is used to set a default value if none is provided.

Leave Table:

Attributes	Data Type	Constraints
L_Id	int	Primary Key
U_id	int	Foreign Key
subject	text	NOT NULL
message	text	NOT NULL
status	Varchar(200)	NOT NULL
date_created	date	NOT NULL DEFAULT

Table 2.4: Overview of Leave table

Table name: LEAVE

This is the name of the table where information about leave requests will be stored.

ATTRIBUTES:

L_ID (LEAVE ID):

Data Type: int

Constraints: Primary Key

Explanation: This column serves as the primary key for the table, ensuring that each value is unique and identifies a specific leave request in the table.

U_ID (USER ID):

Data Type: int

Constraints: Foreign Key

Explanation: This column is a foreign key that references the U_id (User ID) column in another table. It establishes a relationship between the "leave" table and the table containing user information, indicating which user the leave request belongs to.

SUBJECT:

Data Type: Text

Constraints: NOT NULL

Explanation: This column stores the subject or reason for the leave request. The Text data type allows for variable-length text storage.

MESSAGE:

Data Type: Text

Constraints: NOT NULL

Explanation: This column stores the detailed message or description of the leave request. Similar to the subject column, the Text data type is used, and the NOT NULL constraint ensures that every record must have a value for this column.

STATUS:

Data Type: Varchar(200)

Constraints: NOT NULL

Explanation: This column stores the status or current state of the leave request. The Varchar(200) data type allows for variable-length character strings with a maximum length of 200 characters.

DATE_CREATED:

Data Type: date

Constraints: NOT NULL DEFAULT

Explanation: This column stores the date when the leave request was created. The date data type is used, and the NOT NULL constraint ensures that every record must have a value for this column. The DEFAULT keyword is used to set a default value if none is provided.

Employee salary:

Attributes	Data Type	Constraints
Salary_id	int	Primary Key
U_id	int	Foreign Key
Salary_date	date	NOT NULL
Salary_Month	varchar	NOT NULL
Salary_desc	text	NOT NULL

Table 2.5: Overview of Employee Salary table

Table name: EMPLOYEE SALARY

Salary_id (int, Primary Key): This attribute serves as a unique identifier for each salary record in the table. It is of integer data type, typically auto-incremented, ensuring that each entry has a distinct identifier. Being designated as the primary key, it uniquely identifies each row in the table and ensures data integrity and efficient indexing.

U_id (int, Foreign Key): This attribute represents the user ID of the employee associated with the salary record. It is of integer data type and serves as a reference to the primary key of another table where employee details are stored. By establishing a foreign key relationship, it ensures referential integrity, meaning that the value in this attribute must correspond to a valid user ID in the related table.

Salary_date (date, NOT NULL): This attribute stores the date on which the salary was issued or processed. It is of date data type and is marked as NOT NULL, meaning that it must have a valid date value for each salary record. This constraint ensures that the date information is always present and accurate.

Salary_Month (varchar, NOT NULL): This attribute stores the month for which the salary is being processed or paid. It is of variable character (varchar) data type, suitable for storing month names or abbreviations. Similar to the salary_date attribute, it is marked as NOT NULL to ensure that the month information is always provided for each salary record.

Salary_desc (text, NOT NULL): This attribute stores a description or additional details related to the salary payment, such as bonuses, deductions, or any relevant notes. It is of text data type, allowing for the storage of large amounts of text. Like the previous attributes, it is marked as NOT NULL to ensure that descriptive information is always provided for each salary record, promoting completeness and clarity in the data.

Department Table :

Attributes	Data Type	Constraints
Dep_Id	int	Primary Key
Dep_title	Varchar(100)	NOT NULL
Dep_desc	text	NOT NULL

Table 2.6: Overview of department table

Table name: DEPARTMENT

Dep_Id (int, Primary Key): This attribute serves as the primary key for the table. It is of integer data type, indicating it stores whole numbers. The primary key uniquely identifies each department record in the table, ensuring that no two departments have the same Dep_Id. Typically, this is an auto-incrementing value assigned by the database management system.

Dep_title (Varchar(100), NOT NULL): This attribute stores the title or name of the department. It is of variable character (varchar) data type with a maximum length of 100 characters. The NOT NULL constraint ensures that this attribute cannot be left empty for any department record, meaning each department must have a title.

Dep_desc (text, NOT NULL): This attribute stores a description or details about the department. It is of text data type, allowing for the storage of large amounts of text. The NOT NULL constraint ensures that there must be a description provided for every department record, and it cannot be left empty.

Login Table :

Attributes	Data Type	Constraints
Login_Id	int	Primary Key
Login_username	Varchar(20)	NOT NULL
Password	Varchar(20)	NOT NULL

Table 2.7: Overview of login table

Table name: LOGIN

Login_Id (int, Primary Key): This attribute serves as the primary key for the table. It is of integer data type, meaning it stores whole numbers. The primary key uniquely identifies each login record in the table, ensuring that no two logins have the same Login_Id. Typically, this is an auto-incrementing value assigned by the database management system.

Login_username (Varchar(20), NOT NULL): This attribute stores the username associated with the login. It is of variable character (varchar) data type with a maximum length of 20 characters. The NOT NULL constraint ensures that this attribute cannot be left empty for any login record, meaning each login must have a username.

Password (Varchar(20), NOT NULL): This attribute stores the password associated with the login. It is of variable character (varchar) data type with a maximum length of 20 characters. The NOT NULL constraint ensures that this attribute cannot be left empty for any login record, meaning each login must have a password.

Permission Table:

Attributes	Data Type	Constraints
Per_Id	int	Primary Key
U_id	int	Foreign Key
Per_name	Varchar(20)	NOT NULL
status	Varchar(200)	NOT NULL
date_created	date	NOT NULL DEFAULT

Table 2.8: Overview of Permission table

Table name: PERMISSION

Per_Id (int, Primary Key): This attribute serves as the primary key for the table. It is of integer data type, meaning it stores whole numbers. The primary key uniquely identifies each permission record in the table, ensuring that no two permissions have the same Per_Id. Typically, this is an auto-incrementing value assigned by the database management system.

U_id (int, Foreign Key): This attribute is a foreign key referencing another table, possibly a table containing user information. It is of integer data type. Foreign keys establish a relationship between tables. In this case, U_id likely refers to the unique identifier of a user in another table. It ensures referential integrity, meaning that values in this column must correspond to values in the referenced table's primary key column.

Per_name (Varchar(20), NOT NULL): This attribute stores the name or title of the permission. It is of variable character (varchar) data type with a maximum length of 20 characters. The NOT NULL constraint ensures that this attribute cannot be left empty for any permission record, meaning each permission must have a name.

status (Varchar(200), NOT NULL): This attribute stores the status or description of the permission. It is of variable character (varchar) data type with a maximum length of 200 characters. The NOT NULL constraint ensures that this attribute cannot be left empty for any permission record, meaning each permission must have a status.

date_created (date, NOT NULL, DEFAULT): This attribute stores the date when the permission record was created. It is of date data type, representing calendar dates. The NOT NULL constraint ensures that this attribute cannot contain null values, meaning every record must have a valid date. The DEFAULT constraint likely assigns a default value for this attribute if one is not explicitly provided during insertion.

Role Table :

Attributes	Data Type	Constraints
Role_Id	int	Primary Key
A_id	int	Foreign Key
Role_name	Varchar(100)	NOT NULL
Role_desc	text	NOT NULL
date_created	date	NOT NULL DEFAULT

Table 2.9: Overview of Role table

Table name: ROLE

Role_Id (int, Primary Key): This attribute serves as the primary key for the table. It is of integer data type, meaning it stores whole numbers. The primary key uniquely identifies each role record in the table, ensuring that no two roles have the same Role_Id. Typically, this is an auto-incrementing value assigned by the database management system.

A_id (int, Foreign Key): This attribute is a foreign key referencing another table, likely a table containing information about users or accounts. It is of integer data type. Foreign keys establish a relationship between tables. In this case, A_id likely refers to the unique identifier of an account or user associated with the role in another table. It ensures referential integrity, meaning that values in this column must correspond to values in the referenced table's primary key column.

Role_name (Varchar(100), NOT NULL): This attribute stores the name or title of the role. It is of variable character (varchar) data type with a maximum length of 100 characters. The NOT NULL constraint ensures that this attribute cannot be left empty for any role record, meaning each role must have a name.

Role_desc (text, NOT NULL): This attribute stores a description or details about the role. It is of text data type, allowing for the storage of large amounts of text. The NOT NULL constraint ensures that there must be a description provided for every role record, and it cannot be left empty.

date_created (date, NOT NULL, DEFAULT): This attribute stores the date when the role record was created. It is of date data type, representing calendar dates. The NOT NULL constraint ensures that this attribute cannot contain null values, meaning every record must have a valid date. The DEFAULT constraint likely assigns a default value for this attribute if one is not explicitly provided during insertion.

CHAPTER 3

COMPLEX QUERIES BASED ON SETS, VIEWS, JOINS AND CURSORS

Select all admins:

```
SELECT * FROM admins;
```

Explanation:

This query retrieves all information from the 'admins' table, presumably containing details about users who have administrative privileges in the task management system.

Select all users:

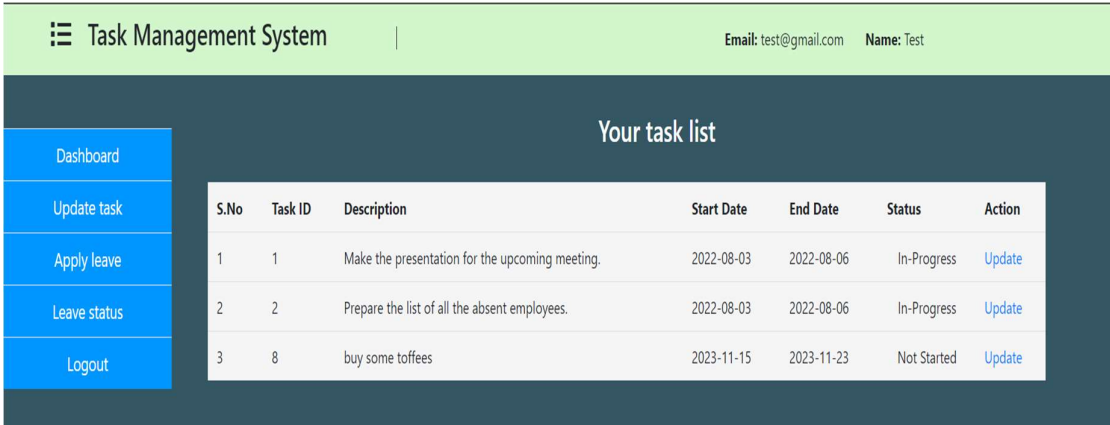
```
SELECT * FROM users;
```

Explanation:

Similarly, this query fetches all records from the 'users' table, which likely stores information about all users registered in the system.

Select all tasks assigned to a specific user ('Test'):

```
SELECT * FROM tasks WHERE uid = (SELECT uid FROM users WHERE name = 'Test');
```



The screenshot displays a web application titled 'Task Management System'. At the top right, it shows the user's email as 'test@gmail.com' and name as 'Test'. On the left, there is a sidebar with navigation links: 'Dashboard', 'Update task', 'Apply leave', 'Leave status', and 'Logout'. The main content area is titled 'Your task list' and contains a table with the following data:

S.No	Task ID	Description	Start Date	End Date	Status	Action
1	1	Make the presentation for the upcoming meeting.	2022-08-03	2022-08-06	In-Progress	Update
2	2	Prepare the list of all the absent employees.	2022-08-03	2022-08-06	In-Progress	Update
3	8	buy some toffees	2023-11-15	2023-11-23	Not Started	Update

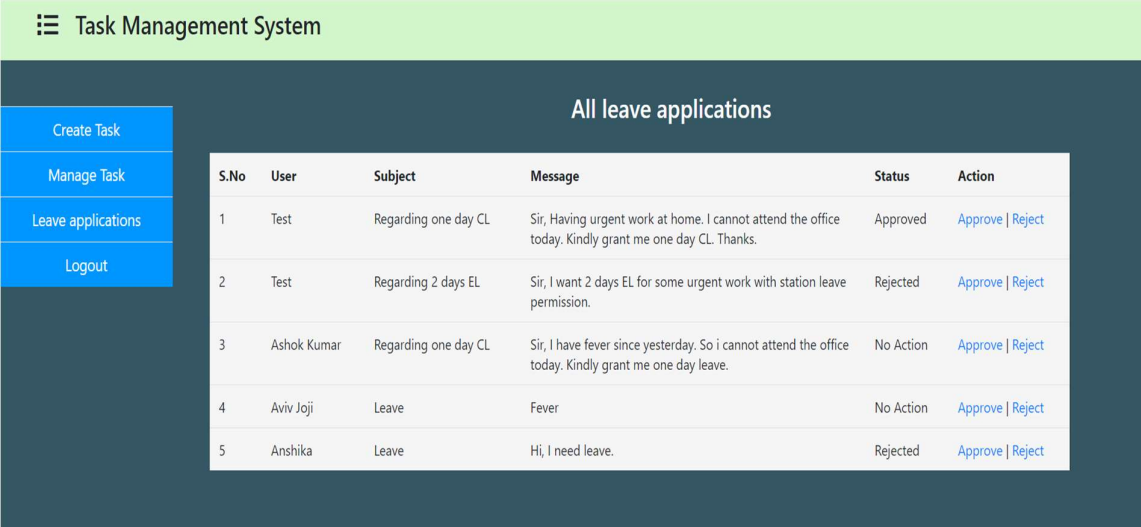
Fig 3.1: Task list page

Explanation:

This query retrieves tasks from the 'tasks' table that are assigned to a user with the name 'Test'. It uses a subquery to find the user's ID based on the name.

Count the total number of leaves requested:

```
SELECT COUNT(*) AS total_leaves FROM leaves;
```



The screenshot shows a web application titled "Task Management System". On the left is a sidebar with four blue buttons: "Create Task", "Manage Task", "Leave applications", and "Logout". The main content area has a dark blue header with the text "All leave applications". Below this is a table with 6 columns: "S.No", "User", "Subject", "Message", "Status", and "Action". The table contains 5 rows of leave requests.

S.No	User	Subject	Message	Status	Action
1	Test	Regarding one day CL	Sir, Having urgent work at home. I cannot attend the office today. Kindly grant me one day CL. Thanks.	Approved	Approve Reject
2	Test	Regarding 2 days EL	Sir, I want 2 days EL for some urgent work with station leave permission.	Rejected	Approve Reject
3	Ashok Kumar	Regarding one day CL	Sir, I have fever since yesterday. So i cannot attend the office today. Kindly grant me one day leave.	No Action	Approve Reject
4	Aviv Joji	Leave	Fever	No Action	Approve Reject
5	Anshika	Leave	Hi, I need leave.	Rejected	Approve Reject

Fig 3.2: List of leave requests

Explanation:

This query calculates the total number of leave requests in the system by counting all the rows in the 'leaves' table. The COUNT(*) function counts all rows, and the result is aliased as total_leaves for clarity.

Find users who haven't requested any leave:

```
SELECT * FROM users WHERE uid NOT IN (SELECT DISTINCT uid FROM leaves);
```

Explanation: Using a subquery, this query identifies users from the 'users' table who do not have any corresponding records in the 'leaves' table, indicating they haven't requested any leave.

Select leave requests along with user details:

```
SELECT leaves.*, users.name AS user_name, users.email AS user_email
FROM leaves
JOIN users ON leaves.uid = users.uid;
```

Explanation:

This query retrieves leave requests from the 'leaves' table along with details of the users who made those requests, such as their names and emails. It joins the 'leaves' table with the 'users' table based on the user ID.

Select tasks that are yet to start:

```
SELECT * FROM tasks WHERE status = 'Not Started';
```

Explanation:

This query fetches tasks from the 'tasks' table that have a status of 'Not Started', indicating they have not commenced yet.

Find the user who has the maximum number of tasks:

```
SELECT u.name, COUNT(t.tid) AS task_count
FROM users u
JOIN tasks t ON u.uid = t.uid
GROUP BY u.name
ORDER BY task_count DESC
LIMIT 1;
```

Explanation:

By joining the 'users' table with the 'tasks' table and using aggregation functions, this query identifies the user with the highest count of tasks assigned to them.

Calculate the average number of leaves requested per user:

```
SELECT AVG(leaves_per_user) AS avg_leaves_per_user  
FROM (SELECT COUNT(*) AS leaves_per_user FROM leaves GROUP BY  
uid) AS subquery;
```

Explanation:

This query calculates the average number of leave requests made by each user by first counting the number of leaves per user and then finding the average of these counts.

Select all users along with the count of tasks assigned to each:

```
SELECT u., IFNULL(task_count, 0) AS task_count  
FROM users u  
LEFT JOIN (SELECT uid, COUNT() AS task_count FROM tasks GROUP BY  
uid) AS subquery ON u.uid = subquery.uid;
```

Explanation:

This query retrieves all users along with the count of tasks assigned to each user. It uses a LEFT JOIN to ensure all users are included in the result, even if they have no tasks assigned.

Insert a new task:

```
INSERT INTO tasks (task_name, description, status, due_date)  
VALUES ('Task Name 1', 'Description of Task 1', 'Pending', '2024-04-30');
```


Task Management System

Create a new Task

Select user:

Description:

Start date:

End date:

Fig 3.3: Creating task dashboard

Explanation:

This query adds a new task to the 'tasks' table with specified details such as task name, description, status, and due date.

Update an existing task's status:

UPDATE tasks

SET status = 'Completed'

WHERE task_id = 1;

Task Management System Email: test@gmail.com Name: Test

Your task list

S.No	Task ID	Description	Start Date	End Date	Status	Action
1	1	Make the presentation for the upcoming meeting.	2022-08-03	2022-08-06	In-Progress	Update
2	2	Prepare the list of all the absent employees.	2022-08-03	2022-08-06	In-Progress	Update
3	8	buy some toffees	2023-11-15	2023-11-23	Not Started	Update

Fig 3.4: Updating task dashboard

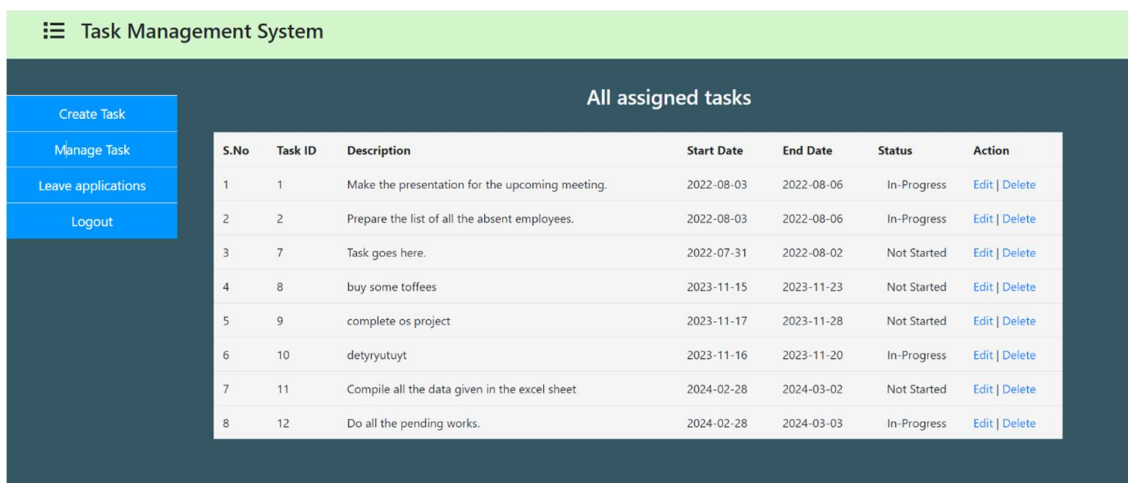
Explanation:

This query updates the status of an existing task in the 'tasks' table to 'Completed' based on the task ID provided.

Delete a task by its ID:

```
DELETE FROM tasks
```

```
WHERE task_id = 1;
```



The screenshot shows a web application titled "Task Management System". On the left is a sidebar with four blue buttons: "Create Task", "Manage Task", "Leave applications", and "Logout". The main area is titled "All assigned tasks" and contains a table with 8 rows of task data. Each row includes a serial number, task ID, description, start and end dates, status, and action links for editing or deleting the task.

S.No	Task ID	Description	Start Date	End Date	Status	Action
1	1	Make the presentation for the upcoming meeting.	2022-08-03	2022-08-06	In-Progress	Edit Delete
2	2	Prepare the list of all the absent employees.	2022-08-03	2022-08-06	In-Progress	Edit Delete
3	7	Task goes here.	2022-07-31	2022-08-02	Not Started	Edit Delete
4	8	buy some toffees	2023-11-15	2023-11-23	Not Started	Edit Delete
5	9	complete os project	2023-11-17	2023-11-28	Not Started	Edit Delete
6	10	detyryutyut	2023-11-16	2023-11-20	In-Progress	Edit Delete
7	11	Compile all the data given in the excel sheet	2024-02-28	2024-03-02	Not Started	Edit Delete
8	12	Do all the pending works.	2024-02-28	2024-03-03	In-Progress	Edit Delete

Fig 3.5: Modifying the task

Explanation:

This query modifies a task from the 'tasks' table based on the provided task ID.

Find the latest leave request for each user:

```
SELECT l.*
```

```
FROM leaves l
```

```
JOIN (
```

```
SELECT MAX(lid) AS latest_leave, uid
```

```
FROM leaves
```

```
GROUP BY uid
```

```
) AS latest_leave_per_user ON l.lid = latest_leave_per_user.latest_leave;
```

Explanation:

This query retrieves the latest leave request for each user by joining the 'leaves' table with a subquery that finds the maximum leave ID for each user.

Select users who have pending leaves (status = 'No Action'):

```
SELECT DISTINCT u.*  
FROM users u  
JOIN leaves l ON u.uid = l.uid  
WHERE l.status = 'No Action';
```

Explanation:

This query identifies users who have pending leave requests by joining the 'users' table with the 'leaves' table based on the status of the leave requests.

Calculate the total number of tasks for all users:

```
SELECT SUM(task_count) AS total_tasks  
FROM (SELECT uid, COUNT(*) AS task_count FROM tasks GROUP BY uid)  
AS subquery;
```

Explanation:

This query calculates the total number of tasks across all users by first counting the number of tasks for each user and then summing these counts.

Find the user with the highest number of approved leaves:

```
SELECT u.name, COUNT(*) AS approved_leaves_count  
FROM users u  
JOIN leaves l ON u.uid = l.uid  
WHERE l.status = 'Approved'  
GROUP BY u.uid  
ORDER BY approved_leaves_count DESC  
LIMIT 1;
```

Explanation:

By joining the 'users' table with the 'leaves' table and filtering for approved leave requests, this query identifies the user with the highest count of approved leaves.

CHAPTER 4

ANALYZING THE PITFALLS, IDENTIFYING THE DEPENDENCIES AND APPLYING NORMALIZATIONS

In the design and development of the Task Management System, careful consideration was given to analyzing potential pitfalls, identifying dependencies, and applying normalization techniques. These processes are crucial for creating a well-structured and efficient database schema that supports data integrity, minimizes redundancy, and ensures the overall reliability and performance of the system.

Analyzing the Pitfalls:

One of the critical steps in the database design process was to analyze potential pitfalls that could arise due to improper data modeling or inadequate normalization. These pitfalls, if left unaddressed, can lead to various issues that can undermine the system's reliability and performance.

Repeating groups, where a single entity contains multiple instances of the same type of data, were identified as a common pitfall. For example, storing multiple phone numbers or email addresses for a user within the same table can lead to data redundancy, making updates and deletions cumbersome and error-prone. To mitigate this issue, separate tables were introduced to store repeating groups, such as a "User Contact Details" table, which maintained a one-to-many relationship with the main "User" table.

Identifying Dependencies:

The identification and understanding of dependencies played a crucial role in the design and normalization of the healthcare information system's database schema. By recognizing these dependencies, the design team was able to create a well-structured and efficient database that minimized data redundancy and update anomalies.

Dependencies:

User table:

User(U_id, firstname, lastname, email, password, Mobile_no, date_created)

$A \rightarrow B C D E F G$

Admin table:

Admin(A_id, firstname, lastname, email, password, Mobile_no, date_created)

$G \rightarrow H I J K$

Task table:

Task(t_id, U_id, description, end_date, start_date, status, date_created)

$L \rightarrow A M N O P Q$

Leaves table:

Leave(L_id, U_id, subject, message, status, date_created)

$R \rightarrow A S T U Q$

Permission table:

Permission(Per_id, U_id, Per_name, status, date_created)

$W \rightarrow A X Y Z$

Role table:

Role(Role_id, A_id, Role_name, Role_desc, status, date_created)

BA → G BB BC BD DE

Department table:

Department(Dep_id, Dep_title, Dep_desc)

AA → AB AC

Login table:

Login(Login_id, Login_username, Password)

CA → CB CC

Employee salary table:

Employee Salary(Salary_id, U_id, Salary_date, Salary_Month, Salary_desc)

DA → A DB DC DB

Categories:

Categories(category_id, name)

FA → FB

Task Categories:

Task Categories(task_category_id, task_id, category_id)

GA → L FA

Comments:

Comments(comment_id , task_id, U_id, comment_text, comment_date)

EA → L A EB EC

Applying Normalization Techniques:

Based on the analysis of pitfalls and dependencies, normalization techniques were applied to the database schema of the healthcare information system. Normalization involves organizing data into tables and defining relationships between them to minimize redundancy and ensure data integrity. The following normalization techniques were applied:

First Normal Form (1NF): Each table in the database was designed to satisfy the requirements of 1NF, which ensures that each column contains atomic values, and there are no repeating groups. For example, patient details like name, date of birth, and gender were stored in separate columns, avoiding repeating groups.

Second Normal Form (2NF): Tables were further normalized to satisfy 2NF, which eliminates partial dependencies by ensuring that each non-key attribute is fully functionally dependent on the primary key.

Third Normal Form (3NF): To achieve 3NF, tables were designed to eliminate transitive dependencies, where non-key attributes depend on other non-key attributes.

Boyce-Codd Normal Form (BCNF): In some cases, tables were further normalized to BCNF, which ensures that every determinant is a candidate key. This was achieved by identifying and resolving dependencies to ensure that each non-trivial functional dependency is a superkey.

User Table (1NF,2NF,3NF,4NF,5NF)

All atomic value (1NF)

There is no partial dependency (2NF)

There is no transitive dependency (3NF)

All candidate keys are in LHS (BCNF)

No multivalued dependency (4NF)

Lossless decomposition (5NF)

Admin Table (1NF,2NF,3NF,4NF,5NF)

All atomic value (1NF)

There is no partial dependency (2NF)

There is no transitive dependency (3NF)

No non trivial functional dependency violation

No multivalued dependency (4NF)

No redundancy arise from join operation (5NF)

Task Table (1NF,2NF,3NF,4NF,5NF)

All atomic value (1NF)

There is no partial dependency (2NF)

There is no transitive dependency (3NF)

Satisfies BCNF

No multivalued dependency (4NF)

Lossless decomposition (5NF)

Leave Table (1NF,2NF,3NF,4NF,5NF)

All atomic value (1NF)

There is no partial dependency (2NF)

There is no transitive dependency (3NF)

All candidate keys are in LHS (BCNF)

No multivalued dependency (4NF)

Lossless decomposition (5NF)

Department Table (1NF,2NF,3NF,4NF,5NF)

All atomic value (1NF)

There is no partial dependency (2NF)

There is no transitive dependency (3NF)

All candidate keys are in LHS (BCNF)

No multivalued dependency (4NF)

Lossless decomposition (5NF)

Comment Table (1NF,2NF,3NF,4NF,5NF)

All atomic value (1NF)

There is no partial dependency (2NF)

There is no transitive dependency (3NF)

All candidate keys are in LHS (BCNF)

No multivalued dependency (4NF)

Lossless decomposition (5NF)

Category Table (1NF,2NF,3NF,4NF,5NF)

All atomic value (1NF)

There is no partial dependency (2NF)

There is no transitive dependency (3NF)

All candidate keys are in LHS (BCNF)

No multivalued dependency (4NF)

Lossless decomposition (5NF)

Task Category Table (1NF,2NF,3NF,4NF,5NF)

All atomic value (1NF)

There is no partial dependency (2NF)

There is no transitive dependency (3NF)

All candidate keys are in LHS (BCNF)

No multivalued dependency (4NF)

Lossless decomposition (5NF)

CHAPTER 5

IMPLEMENTATION OF CONCURRENCY CONTROL AND RECOVERY MECHANISMS

Concurrency Control:

Ensuring data consistency amidst concurrent access is paramount. One effective mechanism is locking. Users acquire locks on tasks they intend to modify, preventing others from concurrently accessing or altering them. We can utilize exclusive locks for strict control, ensuring only one user modifies a task at a time, or shared locks to permit multiple users to read simultaneously while barring modifications. Additionally, we can employ timestamp ordering to sequence transactions based on unique timestamps, guaranteeing serializability. Multi-Version Concurrency Control (MVCC) maintains multiple data versions, enabling consistent snapshots for reading even during modifications by other users. These mechanisms ensure orderly access and modification, preserving data integrity in our system.

Concurrency Control Mechanisms:

Locking:

Exclusive Locks: These locks ensure that only one user can modify a task at a time. When a user acquires an exclusive lock on a task, it prevents other users from reading or modifying it until the lock is released.

Shared Locks:

Shared locks allow multiple users to read a task simultaneously but prevent any user from modifying it. These locks are compatible with each other, meaning multiple users can hold shared locks simultaneously, but an exclusive lock cannot be acquired as long as any shared locks are held.

Timestamp Ordering:

Each transaction is assigned a unique timestamp. Transactions are then ordered based on their timestamps to ensure serializability. This mechanism ensures that transactions are executed in a consistent order, preventing conflicts and maintaining data consistency.

Multi-Version Concurrency Control (MVCC):

MVCC maintains multiple versions of data. When a user modifies a task, a new version of the task is created, leaving the original version intact. This allows other transactions to read a consistent snapshot of the data even if other transactions are modifying it.

Recovery Mechanism:

Alongside concurrency control, a robust recovery mechanism is indispensable for maintaining system integrity post-failures. Logging serves as a pivotal tool. Before committing any changes to task data, we diligently log these modifications. Employing Write-Ahead Logging (WAL), log records are persistently stored on disk before corresponding data alterations, assuring durability. During recovery, these logs facilitate redo operations, reapplying

changes to restore system consistency. Periodic checkpointing supplements logging by capturing snapshots of system states at intervals. In the event of a failure, we can revert to the latest checkpoint and reapply logged changes to reconstruct the system's state. By amalgamating logging, checkpointing, and recovery techniques, our system maintains resilience and data integrity, ensuring seamless operations for our online task management platform.

Logging:

Write-Ahead Logging (WAL): With WAL, log records are written to disk before the corresponding data modifications are made. This ensures durability, as changes are logged before being applied to the database. In case of a failure, the logged changes can be redone to bring the system back to a consistent state.

Redo Logging: In redo logging, only the changes made to the data are logged. During recovery, these changes are reapplied (redone) to bring the system to a consistent state.

Checkpointing:

Periodic snapshots of the system's state are taken, and these snapshots are recorded in the log. During recovery, the system can be restored to the latest checkpoint, and any changes since the checkpoint can be replayed from the log.

CHAPTER 6

CODE FOR THE PROJECT

Admin page:

```
<?php
session_start();
if(isset($_SESSION['email'])){
    include("../includes/connection.php");
    if(isset($_POST['submit_leave'])){
        $query = "insert into leaves values(null,1,'$_POST[subject]', '$_POST[message]')";
        $query_run = mysqli_query($connection,$query);
        if($query_run){
            echo "<script type='text/javascript'>
                alert('Form submitted successfully...');
                window.location.href = 'user_dashboard.php';
            </script>";
        }
        else{
            echo "<script type='text/javascript'>
                alert('Error...Plz try again. ');
                window.location.href = 'user_dashboard.php';
            </script>";
        }
    }
}

if(isset($_POST['create_task'])){
    $query = "insert into tasks values(null,$_POST[id], '$_POST[description]', '$_POST[start_date]', '$_POST[end_date]', 'Not Started')";
    $query_run = mysqli_query($connection,$query);
    if($query_run){
        echo "<script type='text/javascript'>
            alert('Task created successfully...');
            window.location.href = 'admin_dashboard.php';
        </script>";
    }
    else{
        echo "<script type='text/javascript'>
            alert('Error...Plz try again. ');
            window.location.href = 'admin_dashboard.php';
        </script>";
    }
}
}
}

<?php
<!-- jQuery file -->
<script src="..../includes/jquery_latest.js"></script>
<!-- Bootstrap files -->
<link rel="stylesheet" href="..../bootstrap/css/bootstrap.min.css">
<script src="..../bootstrap/js/bootstrap.min.js"></script>
<!-- External CSS file -->
<link rel="stylesheet" href="..../css/style.css">
<!-- Font Awesome -->
<script src="https://kit.fontawesome.com/527a10858c.js" crossorigin="anonymous"></script>
<script type="text/javascript">
    $(document).ready(function(){
        $('#create_task').click(function(){
            $('#right_sidebar').load("create_task.php");
        });
    });
    $(document).ready(function(){
        $('#manage_task').click(function(){
            $('#right_sidebar').load("manage_task.php");
        });
    });
    $(document).ready(function(){
        $('#view_leave').click(function(){
            $('#right_sidebar').load("view_leave.php");
        });
    });
</script>
</head>
<body>
<!-- Header code starts here -->
<div class="row" id="header">
<div class="col-md-12">
<h3><i class="fa fa-solid fa-list" style="padding-right: 15px;"> Task Management System</h3>
</div>
</div>
<!-- Header code ends -->
<div class="row">
<div id="left_sidebar" class="col-md-2">
<table class="table" style="width:100%; text-align:center">
<tr><td style="text-align:center"><a type="button" id="create_task">Create Task</a></td></tr>
<tr><td style="text-align:center"><a type="button" id="manage_task">Manage Task</a></td></tr>
<tr><td style="text-align:center"><a type="button" id="view_leave">Leave applications</a></td></tr>
<tr><td style="text-align:center"><a type="button" href="..../logout.php" id="logout_link">Logout</a></td></tr>
</table>
</div>
<div id="right_sidebar" class="col-md-10">
<div id="Instructions for Employees">
<ol style="list-style-type: none;">
<li>1. All the employee should mark their attendance daily.</li>
<li>2. Everyone must complete the tasks assigned to them.</li>
<li>3. Kindly maintain decorum of the office.</li>
<li>4. Keep office and your area neat and clean.</li>
</ol>
</div>
</div>
</div>
</body>
</html>
<?php
```

Fig 6.1: Code of Admin page

Create task:

```
<?php
session_start();
if(isset($_SESSION['email'])){
    include('../includes/connection.php');
}
?>
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
</head>
<body>
    <h3>Create a new Task</h3><br>
    <div class="row">
        <div class="col-md-6">
            <form action="" method="post">
                <div class="form-group">
                    <label>Select user:</label>
                    <select name="id" class="form-control">
                        <option>--Select--</option>
                        <?php
                            $query = "select uid,name from users";
                            $query_run = mysqli_query($connection,$query);
                            if(mysqli_num_rows($query_run)){
                                while($row = mysqli_fetch_assoc($query_run)){
                                    <?php
                                        <option value="<?php echo $row['uid']; ?>"><?php echo $row['name']; ?></option>
                                    </php>
                                }
                            }
                        </?php>
                    </select>
                </div>
                <div class="form-group">
                    <label>Description:</label>
                    <textarea class="form-control" rows="3" cols="50" name="description" placeholder="Mention the task"></textarea>
                </div>
                <div class="form-group">
                    <label>Start date:</label>
                    <input type="date" class="form-control" name="start_date" />
                </div>
                <div class="form-group">
                    <label>End date:</label>
                    <input type="date" class="form-control" name="end_date" placeholder="Start date" />
                </div>
                <div>
                    <input type="submit" class="btn btn-warning" name="create_task" value="Create" placeholder="End date" />
                </div>
            </form>
        </div>
    </div>
</body>
</html>
<?php
}
else{
    header('Location:admin_login.php');
}
?>
```

Fig 6.2: Code for creating tasks

Leave Status:

```
<?php
session_start();
if(isset($_SESSION['email'])){
    include('../includes/connection.php');
}
?>
<html>
<body>
    <center><h3>Your leave applications</h3></center><br>
    <table class="table" style="background-color: whitesmoke;width: 75vw;">
        <tr>
            <th>S.No</th>
            <th>Subject</th>
            <th style="width:40%;">Message</th>
            <th>Status</th>
        </tr>
        <?php
            $sno = 1;
            $query = "select * from leaves where uid = $_SESSION[uid]";
            $query_run = mysqli_query($connection,$query);
            while($row = mysqli_fetch_assoc($query_run)){
                <?php
                    <tr>
                        <td><?php echo $sno; ?></td>
                        <td><?php echo $row['subject']; ?></td>
                        <td><?php echo $row['message']; ?></td>
                        <td><?php echo $row['status']; ?></td>
                    </tr>
                    <?php
                        $sno = $sno + 1;
                    </?php>
                </?php>
            }
        </?php>
    </table>
</body>
</html>
<?php
}
else{
    header('Location:user_login.php');
}
?>
```

Fig 6.3: Code of leave status

Manage Task:

```
<?php
session_start();
if(isset($_SESSION['email'])){
    include('../includes/connection.php');
}
?>
<html>
<body>
<center><h3>All assigned tasks</h3></center><br>
<table class="table" style="background-color: whitesmoke; width: 75vw;">
    <tr>
        <th>S.No</th>
        <th>Task ID</th>
        <th>Description</th>
        <th>Start Date</th>
        <th>End Date</th>
        <th>Status</th>
        <th>Action</th>
    </tr>
    <tr>
        <?php
        $sno = 1;
        $query = "select * from tasks";
        $query_run = mysqli_query($connection, $query);
        while($row = mysqli_fetch_assoc($query_run)){
            ?>
            <tr>
                <td><?php echo $sno; ?></td>
                <td><?php echo $row['tid']; ?></td>
                <td><?php echo $row['description']; ?></td>
                <td><?php echo $row['start_date']; ?></td>
                <td><?php echo $row['end_date']; ?></td>
                <td><center><?php echo $row['status']; ?></center></td>
                <td><a href="edit_task.php?id=<?php echo $row['tid']; ?>">Edit</a> | <a href="delete_task.php?id=<?php echo $row['tid']; ?>">Delete</a></td>
            </tr>
            <?php
            $sno = $sno + 1;
        }
    </tr>
</table>
</body>
</html>
<?php
}
else{
    header('Location:admin_login.php');
}
?>
```

Fig 6.4: Code for managing tasks

Leave View:

```
<?php
session_start();
if(isset($_SESSION['email'])){
    include('../includes/connection.php');
}
?>
<html>
<body>
<center><h3>All leave applications</h3></center><br>
<table class="table" style="background-color: whitesmoke; width: 75vw;">
    <tr>
        <th>S.No</th>
        <th>User</th>
        <th>Subject</th>
        <th style="width: 40%;">Message</th>
        <th>Status</th>
        <th>Action</th>
    </tr>
    <tr>
        <?php
        $sno = 1;
        $query = "select * from leaves";
        $query_run = mysqli_query($connection, $query);
        while($row = mysqli_fetch_assoc($query_run)){
            ?>
            <tr>
                <td><?php echo $sno; ?></td>
                <?php
                $query1 = "select name from users where uid = $row[uid]";
                $query_run1 = mysqli_query($connection, $query1);
                while($row1 = mysqli_fetch_assoc($query_run1)){
                    ?>
                    <td><?php echo $row1['name']; ?></td>
                    <?php
                }
                <td><?php echo $row['subject']; ?></td>
                <td><?php echo $row['message']; ?></td>
                <td><?php echo $row['status']; ?></td>
                <td><a href="approve_leave.php?id=<?php echo $row['lid']; ?>">Approve</a> | <a href="reject_leave.php?id=<?php echo $row['lid']; ?>">Reject</a></td>
            </tr>
            <?php
            $sno = $sno + 1;
        }
    </tr>
</table>
</body>
</html>
<?php
}
else{
    header('Location:admin_login.php');
}
?>
```

Fig 6.5: Code for viewing leave

Assign Task:

```
<?php
    session_start();
    if(isset($_SESSION['email'])){
        include('includes/connection.php');
    }
?>
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <!-- Bootstrap files -->
    <link rel="stylesheet" href="bootstrap/css/bootstrap.min.css">
    <script src="bootstrap/js/bootstrap.min.js"></script>
    <!-- External CSS file -->
    <link rel="stylesheet" href="css/style.css">
</head>
<body>
    <center><h3>Your task list</h3></center><br>
    <table class="table" style="background-color: whitesmoke;width: 75vw;">
        <tr>
            <th>$sno</th>
            <th>Task ID</th>
            <th>Description</th>
            <th>Start Date</th>
            <th>End Date</th>
            <th>Status</th>
            <th>Action</th>
        </tr>
        <tr>
            <?php
                $sno = 1;
                $query = "select * from tasks where uid = $_SESSION[uid]";
                $query_run = mysqli_query($connection,$query);
                while($row = mysqli_fetch_assoc($query_run)){
                    ?>
                    <tr>
                        <td><?php echo $sno; ?></td>
                        <td><?php echo $row['tid']; ?></td>
                        <td><?php echo $row['description']; ?></td>
                        <td><?php echo $row['start_date']; ?></td>
                        <td><?php echo $row['end_date']; ?></td>
                        <td><center><?php echo $row['status']; ?></center></td>
                        <td><a href="update_status.php?id=<?php echo $row['tid']; ?>">Update</a></td>
                    </tr>
                    <?php
                        $sno = $sno + 1;
                    }
                ?>
            </table>
        </body>
    </html>
    <?php
    }
    else{
        header('Location:user_login.php');
    }
?>
```

Fig 6.6: Code for assigning tasks

Updating Task Status:

```
<?php
session_start();
if(isset($_SESSION['email'])){
include('includes/connection.php');
if(isset($_POST['update'])){
    $query = "update tasks set status = '$_POST[status]' where tid = $_GET[id]";
    $query_run = mysqli_query($connection,$query);
    if($query_run){
        echo "<script type='text/javascript'>
            alert('Status updated successfully...');
            window.location.href = 'user_dashboard.php';
        </script>";
    }
    else{
        echo "<script type='text/javascript'>
            alert('Error...Plz try again. ');
            window.location.href = 'user_dashboard.php';
        </script>";
    }
}
}
?>
<html>
<head>
<title>ETMS</title>
<!-- jQuery file -->
<script src="includes/jquery_latest.js"></script>
<!-- Bootstrap files -->
<link rel="stylesheet" href="bootstrap/css/bootstrap.min.css">
<script src="bootstrap/js/bootstrap.min.js"></script>
<!-- External CSS file -->
<link rel="stylesheet" href="css/style.css">
</head>
<body>
<div class="row">
<div class="col-md-3 m-auto" id="home_page">
<center>
<h3>Update task status</h3>
<?php
$query = "select * from tasks where tid = $_GET[id]";
$query_run = mysqli_query($connection,$query);
while($row = mysqli_fetch_assoc($query_run)){
    ?>
    <form action="" method="post">
    <div class="form-group">
        <input type="hidden" name="id" class="form-control" />
    </div>
    <div class="form-group">
        <select class="form-control" name="status">
        <option>Select</option>
        <option>Complete</option>
        <option>In-Progress</option>
        </select>
    </div>
    <button type="submit" class="btn btn-danger" name="update">Update</button>
    <a href="user_dashboard.php" class="btn btn-primary">Dashboard</a>
    </form>
    <?php
}
?>
</center>
</div>
</div>
</body>
</html>
<?php
```

Fig 6.7: Code for updating task status

CHAPTER 7

RESULT AND DISCUSSION

Our project has resulted in the development of a comprehensive web-based task management system interface, seamlessly integrating task management, real-time status monitoring, and a secure web-based application for user and admin access. The outcomes of our implementation and testing are detailed below.

Efficient Task Management: The core of our system is a user-friendly task management component, enabling effortless task creation, editing, updating, deletion, and assignment to designated employees. This intuitive interface simplifies task management, significantly enhancing user productivity.

Real-Time Status Monitoring: The status monitoring component presents task completion status in real time. Admins have access to critical data, including user assignments and project overview. This real-time feature ensures users are consistently well-informed about their assignments and the overall workflow of the organization.

Web-Based Application: Our web-based application streamlines task management and real-time status monitoring. Users securely log-in and on their machines and get an overview of their jobs via a unified interface, simplifying organization management. Additionally, the terminal supports updating tasks, enabling users to inform project managers of their work almost instantly, boosting efficiency.

Secure and Controlled User Access: Security is a paramount focus of our project. We have implemented robust mechanisms for user authentication and authorization, guaranteeing that only authorized individuals can access and interact with the web-based application. This access control layer ensures the confidentiality of user data and maintains the system's integrity.


Flexible Multi-Platform Compatibility: Our web-based interface is designed to work seamlessly across a variety of platforms and devices. This adaptability allows users to access and manage their tasks from desktop computers, laptops, and even mobile devices, providing accessibility and flexibility.

Code Organization and Documentation: Our codebase adheres to efficient organization principles. It features modular components, and comprehensive code documentation. This meticulous approach ensures that our project remains maintainable, extensible, and easy to comprehend. We have successfully hosted our database in AWS server.

Our project results underscore the successful creation of an efficient, user-friendly, and secure web-based task management system interface. Users can seamlessly manage tasks, monitor employee progress etc. through a single, integrated platform. Robust security measures and comprehensive testing procedures further enhance the system's reliability and performance, rendering it a valuable tool for users in diverse scenarios.

Front end Implementation:

User Registration:

A screenshot of a user registration form. The form is titled "User Registration" in a dark green box. It contains four input fields: "Enter Username", "Enter Email", "Enter Password", and "Mobile No.". Below the input fields is a yellow "Register" button and a red "Go to Home" button. The form is centered on a dark blue background.

User Registration

Enter Username

Enter Email

Enter Password

Mobile No.


Register

Go to Home

Fig 7.1: User registration page

Explanation: The new users can register for online Task Management System by filling all the required credentials.

User Login:

A screenshot of a user login form. The form is titled "User login" in a dark green box. It contains two input fields: "Enter Email" and "Enter Password". Below the input fields is a yellow "Login" button and a red "Go to Home" button. The form is centered on a dark blue background.

User login

Enter Email

Enter Password

Login

Go to Home

Fig 7.2: User Login page

Explanation: All the registered users can login to the website by filling all the given form fields. If the given credentials does not match from the database, the user can not login to the website.

Update Task:

S.No	Task ID	Description	Start Date	End Date	Status	Action
1	1	Make the presentation for the upcoming meeting.	2022-08-03	2022-08-06	Not Started	Update
2	2	Prepare the list of all the absent employees.	2022-08-03	2022-08-06	Complete	Update

Fig 7.3: Update task page

Explanation: This is the user dashboard page, where on clicking update task button, all the assigned task to the particular user can be viewed. Here, user can also update the status of the assigned task.

Apply leave:

Enter Subject

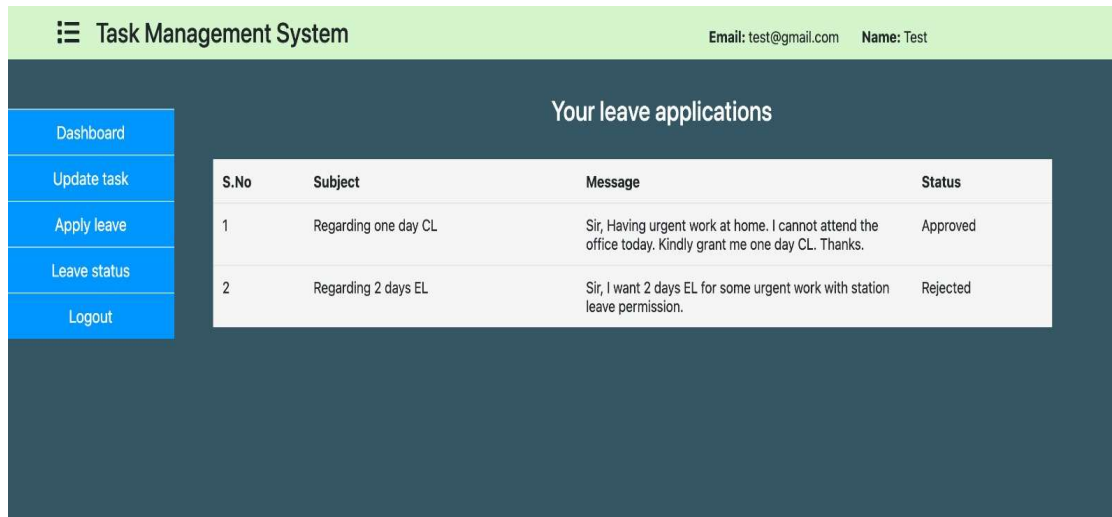
Type Message

Submit

Fig 7.4: Apply leave page

Explanation: This is the user dashboard page, where on clicking apply leave button, the users can apply for leave by providing the leave subject and the reason for taking the leave.

Leave Status:



Task Management System		Email: test@gmail.com	Name: Test
Dashboard	Your leave applications		
Update task			
Apply leave			
Leave status			
Logout			
S.No	Subject	Message	Status
1	Regarding one day CL	Sir, Having urgent work at home. I cannot attend the office today. Kindly grant me one day CL. Thanks.	Approved
2	Regarding 2 days EL	Sir, I want 2 days EL for some urgent work with station leave permission.	Rejected

Fig 7.5: Leave status page

Explanation: This is the user dashboard page, where on clicking leave status button, user can view the status of leave, whether the leave application is approved or denied.

Admin Login:

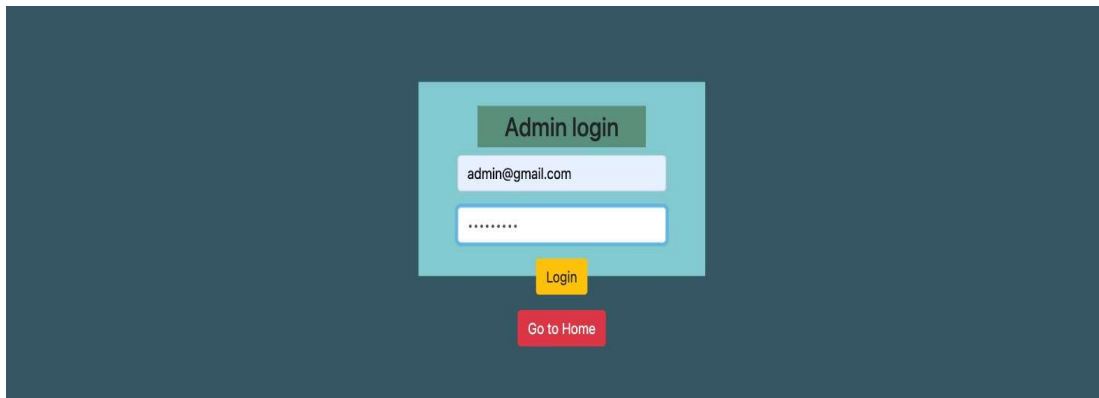
The image shows a dark blue background with a light blue rectangular box in the center. Inside this box, there is a green header with the text "Admin login". Below the header, there is a white input field containing the email address "admin@gmail.com". Underneath the email field is another white input field with a series of dots, representing a password field. Below the password field, there is a yellow button with the text "Login". At the bottom of the light blue box, there is a red button with the text "Go to Home".

Fig 7.6: Admin Login page

Explanation: All the registered admins can login to the website by filling all the given form fields. If the given credentials do not match from the database, the admin cannot login to the website.

Create Task:

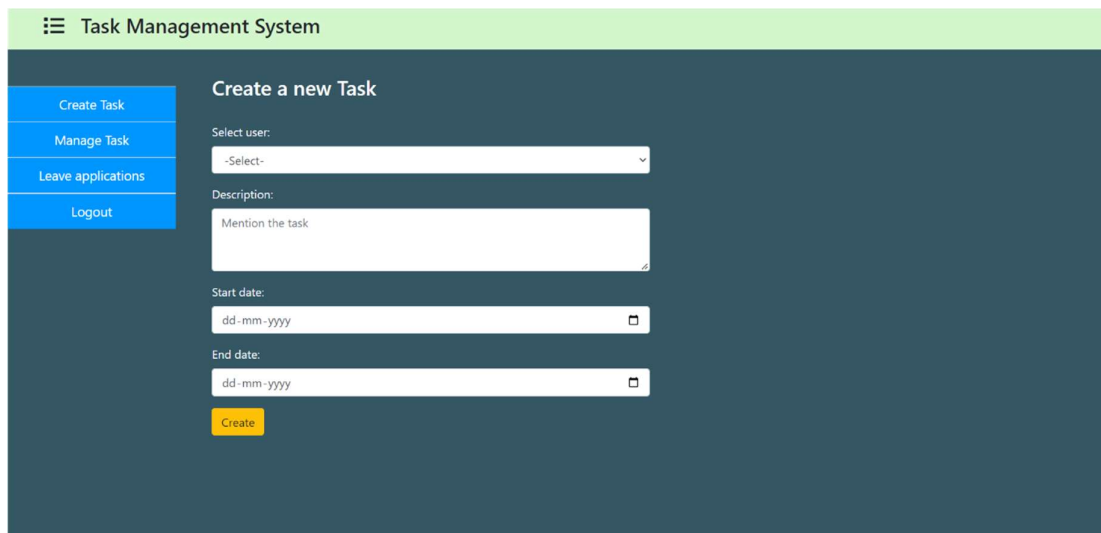
The image shows a dark blue background with a light green header bar at the top containing a hamburger menu icon and the text "Task Management System". On the left side, there is a blue sidebar with four buttons: "Create Task", "Manage Task", "Leave applications", and "Logout". The main content area is titled "Create a new Task". It contains a "Select user:" label followed by a dropdown menu with "-Select-" as the selected option. Below this is a "Description:" label followed by a text input field with the placeholder text "Mention the task". Underneath the description field are two date input fields: "Start date:" and "End date:", both with a placeholder "dd-mm-yyyy" and a calendar icon. At the bottom of the form is a yellow "Create" button.

Fig 7.7: Create Task page

Explanation: This is the admin dashboard page, where on clicking create task button, admin can assign tasks to the users. First of all, admin selects the user's name for whom he is going to assign the task. He provides task description along with start date and end date.

Manage Task:

Task Management System						
Create Task	All assigned tasks					
Manage Task	S.No	Task ID	Description	Start Date	End Date	Status Action
Leave applications	1	1	Make the presentation for the upcoming meeting.	2022-08-03	2022-08-06	Complete Edit view
Logout	2	2	Prepare the list of all the absent employees.	2022-08-03	2022-08-06	Complete Edit view
	3	7	Task goes here.	2022-07-31	2022-08-02	Not Started Edit view
	4	8	complete dbms project	2024-02-29	2024-03-09	Not Started Edit view
	5	9	complete the analysis	2024-03-26	2024-03-28	Not Started Edit view
	6	10	do pushups and squats	2024-03-27	2024-03-29	Not Started Edit view
	7	11	make a project based on AI and ML	2024-04-17	2024-05-11	Not Started Edit view
	8	12	Prepare dbms report as soon as possible	2024-04-28	2024-05-11	In-Progress Edit view

Fig 7.8: Manage Task page

Explanation: This is the admin dashboard page, where on clicking manage task button, admin can view the progress of tasks and at the same time, he can edit the tasks assigned according to the requirements.

Leave Applications:

Task Management System						
Create Task	All leave applications					
Manage Task	S.No	User	Subject	Message	Status	Action
Leave applications	1	Test	Regarding one day CL	Sir, Having urgent work at home. I cannot attend the office today. Kindly grant me one day CL. Thanks.	Approved	Approve Reject
Logout	2	Test	Regarding 2 days EL	Sir, I want 2 days EL for some urgent work with station leave permission.	Rejected	Approve Reject
	3	Ashok Kumar	Regarding one day CL	Sir, I have fever since yesterday. So i cannot attend the office today. Kindly grant me one day leave.	No Action	Approve Reject

Fig 7.9: List of leave applications

Explanation: This is the admin dashboard page, where on clicking leave application button, admin can view all the leave application. On clicking the approve button or reject button, the admin can grant or deny leave.

CHAPTER 8

ONLINE COURSE CERTIFICATE



NPTEL Online Certification

(Funded by the MoE, Govt. of India)



This certificate is awarded to

AVIV P JOJI

for successfully completing the course

Data Base Management System

with a consolidated score of **57** %

Online Assignments	20.83/25	Proctored Exam	36/75
--------------------	----------	----------------	-------

Total number of candidates certified in this course: **6225**

Jan-Mar 2024

(8 week course)

Prof. Haimanti Banerji
Coordinator, NPTEL
IIT Kharagpur



Indian Institute of Technology Kharagpur



Roll No: NPTEL24CS21S544100224

To verify the certificate



No. of credits recommended: 2 or 3

GitHub Link: <https://github.com/AvivJoji/Task-Management-System>