Database Management System

Project Report

TITLE:

Weather Prediction System Using Past Data
DBMS + Python + Flask Full Stack Project

Student Name: Agrim Yadav

Sap ID: 590012947

Batch: 30

Student Name: Satyam Gond

Sap ID: 590013705

Batch: 30

Submitted to- Kalluri Shareef Babu

Date- 5th December 2025

Abstract:

Weather prediction is one of the most important applications of data analysis, meteorology, and computer science. This project presents a modular, database-driven weather prediction system that stores historical weather records and predicts future temperature using a moving average statistical model. The system integrates MySQL for structured data storage, Python for logic processing, and Flask for a dynamic web-based graphical interface. The model takes the latest three temperature values and computes the next day's forecasted temperature, offering a simple yet effective short-term prediction technique suitable for academic DBMS projects. The system supports full CRUD operations (Create, Read, Update, Delete), ensuring complete data management capabilities. This report explains the architecture, database schema, normalization method, algorithm, GUI, testing results, limitations, and future scope of the project.

Introduction

Weather forecasting has become increasingly essential for climate-sensitive activities such as agriculture, logistics, aviation, and public safety. However, real-world forecasting relies on complex numerical models and massive datasets, which are not always feasible for academic demonstration. This project addresses the need for a simplified, educational weather prediction system using a relational database and basic forecasting. It focuses on demonstrating database organization, CRUD operations, backend processing, and a web-based graphical interface. The project highlights the importance of storing weather data efficiently while enabling easy retrieval and logical processing to generate predictions. By using MySQL, Python, and Flask, the system provides functionality similar to real-world dashboards but in a simplified format suitable for students and teaching. Additionally, the project aims to show how software engineering principles—such as modularity, separation of concerns, and layered architecture—can be applied to create scalable academic applications. Modern DBMS projects require integration with a frontend and backend, and this system exemplifies such integration clearly

PROBLEM STATEMENT

A weather prediction system must meet the following requirements: 1. Store historical weather records in a structured and normalized format. 2. Allow users to create, view, update, and delete records easily. 3. Provide a method to fetch recent weather information quickly. 4. Predict the next day's temperature using statistical or historical analysis. 5. Offer a graphical interface that simplifies interaction with the system. 6. Maintain referential integrity through proper database constraints. Traditional forecasting models are extremely complex, so this project focuses on a simplified but functional approach that is suitable for DBMS-level learning and demonstration

OBJECTIVES OF THE PROJECT

This project aims to accomplish the following technical and functional objectives: • Design a fully normalized relational database to store weather details. • Implement robust CRUD operations using SQL queries. • Build an integrated backend using Python for processing and prediction. • Develop a Flask-based GUI allowing easy interaction without SQL knowledge. • Use a moving average model to predict short-term temperature trends. • Enforce foreign key relationships to maintain structured and consistent data. • Ensure modularity so that future enhancements can be added effortlessly.

EXISTING SYSTEM ANALYSIS

Professional weather prediction systems such as IMD (Indian Meteorological Department), NOAA, or commercial APIs like OpenWeatherMap rely on: • Satellite data • Radar imaging • Temperature-pressure models • Machine learning and statistical analysis • Continuous atmospheric data feeds These systems are expensive, complex, and require high computational resources. For academic environments, such complexity is neither required nor practical. Therefore, this project proposes a lightweight system that operates on historical weather data stored in a database. This makes it ideal for understanding DBMS concepts such as normalization, relational mapping, querying, transactions, and indexing.

PROPOSED SYSTEM

The proposed system is a complete database-driven weather prediction application. It consists of: 1. **Database Layer (MySQL):** Stores weather readings, locations, and prediction history. 2. **Backend Layer (Python):** Contains logic for inserting, processing, and predicting temperature. 3. **Presentation Layer (Flask GUI):** Allows users to interact with the system through a browser without writing SQL. The system is user-friendly and modular and can be enhanced with graphs, real-time APIs, or ML algorithms. Its simplicity helps users understand DBMS fundamentals clearly.

SYSTEM ARCHITECTURE

The system follows a **three-tier architecture**, which separates responsibilities across layers: **1. Presentation Layer (Frontend)** • Implemented using HTML, CSS, and Bootstrap. • Flask templates render dynamic weather and prediction data. **2. Application Layer (Backend)** • Written in Python using the Flask framework. • Handles form submissions, routes, validation, and prediction logic. • Communicates with MySQL using mysql-connector-python. **3. Database Layer (MySQL)** • Stores locations, weather records, and predictions. • Ensures data integrity via primary and foreign keys. This architecture ensures scalability, modularity, and ease of future upgrades.

ER DIAGRAM ANALYSIS

The Entity Relationship (ER) model describes how data is stored and connected. **Entities:** • **Location** – Stores city, state, country. • **Weather_Data** – Stores daily weather readings. • **Prediction** – Stores predicted results (optional table). **Relationships:** • One Location → Many Weather_Data entries • One Location → Many Predictions This ensures that weather from multiple cities can be tracked independently.

DATABASE SCHEMA

The project uses the following tables: **1. Location** • location_id (PK) • city • state • country **2. Weather_Data** • weather_id (PK) • location_id (FK referencing Location) • date • temperature • humidity • rainfall • wind_speed **3. Prediction** • prediction_id (PK) • location_id (FK) • predicted_temperature • prediction_date The schema is designed to maintain referential integrity and support efficient querying.

NORMALIZATION

The database follows **Third Normal Form (3NF)**. **1NF:** • All values are atomic and indivisible. **2NF:** • Non-key attributes fully depend on the primary key. **3NF:** • No transitive dependency exists (attributes depend only on PK). Normalization ensures minimal redundancy, prevents anomalies, and improves maintainability.

SQL IMPLEMENTATION DETAILS

SQL queries implemented include: • **CREATE TABLE** – defines tables and constraints • **INSERT** – adds new weather/location data • **UPDATE** – modifies the details • **DELETE** – removes entries while enforcing FK rules • **SELECT** – retrieves data for GUI display • **ORDER BY + LIMIT** – fetches recent weather entries • **AVG()** – calculates moving average for prediction Example Prediction Query: SELECT AVG(temperature) FROM (SELECT temperature FROM Weather_Data WHERE location_id = X ORDER BY date DESC LIMIT 3) AS last3

PYTHON–MYSQL INTEGRATION

Python interacts with MySQL via the mysql-connector-python library. Steps: 1. Create a connection using get_connection() 2. Open cursor 3. Execute SQL query 4. Fetch results or commit updates 5. Close connection This modular approach ensures clean and reusable backend code.

FLASK WEB INTERFACE

The Flask application provides: • Dynamic webpages • Forms for input • Cards and tables for display • Buttons for CRUD operations • Jinja2 templates for rendering • CSS for styling and responsiveness Users can manage all data from the browser without writing SQL.

PREDICTION MODEL

The prediction uses **Moving Average Model (MA-3)**: Predicted Temp = (T1 + T2 + T3) / 3 This model is suitable because: • Simple to implement • Effective for short-term trends • Low computational cost • Ideal for academic demonstration

TESTING AND RESULTS

Tests performed: • Added locations and weather data • Verified updates and deletions • Confirmed prediction accuracy • Checked GUI responsiveness • Ensured SQL constraints prevented invalid operations Results show that the model works reliably and the system performs all CRUD tasks without issues

CONCLUSION

This project integrates DBMS concepts, backend development, and frontend design into a single weather prediction platform. It meets all academic requirements and provides a base for future enhancements such as real-time APIs, graph visualization, mobile support, or machine learning forecasting.