Caterpillar System - Detailed Requirements Gathering Document

Team: Advait Vagerwal, Sahil Thakare, Samarth Prajapati

1. System Overview

The Caterpillar system is a database-driven stock brokerage platform designed to simulate real-world operations of a brokerage firm. It allows clients to register, trade stocks, monitor their portfolios, and analyze stock price history. The database stores structured information about users, stocks, market orders, price history, and watchlists. The system is intended for academic and demonstration purposes, but it follows industry best practices in relational database design.

The platform interacts with the backend database via SQL queries and Python scripts, allowing for data manipulation and retrieval. It supports relational integrity through primary and foreign keys, ensuring that data relationships remain consistent. The project fulfills all stages of database development as outlined in the CS4092 course requirements.

2. Project Scope

The scope of the Caterpillar system includes the following:
- Management of user registration and profile data.
- Maintenance of a list of tradable stocks, including company details.
- Tracking of each user's portfolio holdings, including quantities and average buy prices.
- Recording of all market orders (buy/sell) with timestamps, statuses, and prices.
- Storing historical price data for every listed stock.
- Allowing users to maintain a watchlist of selected stocks for monitoring.
- Providing reporting capabilities such as portfolio summaries, active order listings, and price trend analysis.

This scope does not include advanced trading algorithms, real-time price feeds, or high-frequency trading features, as it is intended for educational use.

3. Data Requirements

The system is built on six primary entities, each representing a logical grouping of data. These entities and their key fields are as follows:

Entity	Key Fields
Users	user_id (PK), full_name, email (unique), phone_number, address, date_joined
Stocks	stock_id (PK), ticker_symbol (unique), company_name, sector, market
Portfolios	portfolio_id (PK), user_id (FK), stock_id (FK), quantity, average_buy_price

MarketOrders	order_id (PK), user_id (FK), stock_id (FK), order_type, quantity, price, order_status, timestamp
StockPriceHistory	history_id (PK), stock_id (FK), date, open_price, close_price, high_price, low_price, volume
Watchlist	watchlist_id (PK), user_id (FK), stock_id (FK), added_on

4. Use Cases

Below are the primary use cases supported by the system:

Use Case ID	Title	Description
UC-01	Register New User	Allow new clients to register with their personal details and create an account in the system.
UC-02	View Available Stocks	Retrieve and display a list of all available stocks with their company details, sector, and market.
UC-03	Track Portfolio Holdings	Display a summary of a user's portfolio, including stocks held, quantities, and average purchase prices.
UC-04	Place Market Order	Enable users to submit buy or sell orders with specified quantities and prices.
UC-05	Monitor Stock Prices	Allow users to view historical stock price data including open, close, high, low, and volume.
UC-06	Add Stock to Watchlist	Permit users to add specific stocks to a personal watchlist for monitoring.
UC-07	Generate Trading Reports	Produce queries and reports summarizing user activity, stock trends, and order histories.

5. Functional Requirements

- The system must support creating, reading, updating, and deleting user records.
- The system must allow the insertion and retrieval of stock listings.
- The system must maintain accurate portfolio records for each user.
- The system must record market orders with timestamps and statuses.
- The system must store historical price data for each stock.
- The system must allow users to add and remove stocks from a watchlist.

- The system should execute multi-table queries to join relevant information for analysis.
- The system must support Python-based business logic for database interactions.

6. Non-Functional Requirements

- Use MySQL or PostgreSQL as the relational database management system.
- Implement referential integrity using primary and foreign keys.
- Normalize database design to at least 3NF to reduce redundancy.
- Maintain secure database access with proper authentication.
- Provide efficient query execution and indexing where needed.
- Ensure CLI-based or minimal UI-based interaction for testing and demonstration.
- NSupport scalability to add more tables and features in the future.

7. Assumptions and Constraints

Assumptions:

- All stock and user data is manually inserted or imported from reliable sources.
- The system is intended for academic demonstration and not for production trading.
- User authentication is minimal and implemented only for demonstration.

Constraints:

- Limited to SQL operations supported by MySQL/PostgreSQL.
- No integration with live stock market feeds.
- Frontend implementation is optional and minimal.