QUESTION : 2

**1. Data Flow Diagram**

The Data Flow Diagram (DFD) provides a high-level overview of how data moves through the inventory management system.

**Entities:**

* **User**: Interacts with the system to input data and view reports.
* **Sales System**: Provides real-time sales data.
* **Inventory Database**: Stores current stock levels, reorder points, and historical sales data.
* **Reorder Algorithm**: Calculates optimal reorder points and quantities.
* **Reporting Module**: Generates reports on inventory turnover, stockouts, and overstock situations.

**Process Flow:**

1. **Sales Data Input**: Sales data is fed into the system in real-time.
2. **Inventory Update**: The system updates stock levels based on sales data.
3. **Threshold Check**: The system checks if any product stock falls below the reorder threshold.
4. **Reorder Calculation**: If the threshold is breached, the Reorder Algorithm calculates the optimal reorder quantity.
5. **Alerts**: The system generates an alert if a reorder is needed.
6. **Reporting**: The Reporting Module generates periodic reports on various inventory metrics.

**Diagram Components:**

* **User Input** → **Inventory Database** ↔ **Reorder Algorithm**
* **Sales System** → **Inventory Update Process** → **Threshold Check**
* **Threshold Check** → **Reorder Alert** → **User**
* **Reporting Module** ↔ **Inventory Database**

**2. Pseudocode and Implementation :**

**Inventory Tracking Pseudocode**

**Initialize inventory levels from database**

**While sales data is incoming:**

**For each product in sales data:**

**Deduct sold quantity from inventory levels**

**If inventory level < reorder threshold:**

**Calculate reorder quantity using reorder algorithm**

**Generate reorder alert**

**Update reorder information in the database**

**Update inventory levels in the database**

**Reorder Calculation Pseudocode**

**Function calculate\_reorder\_quantity(product\_id, current\_stock, lead\_time, demand\_forecast):**

**reorder\_point = (lead\_time \* average\_daily\_demand) + safety\_stock**

**reorder\_quantity = reorder\_point - current\_stock**

**If reorder\_quantity < minimum\_order\_quantity:**

**reorder\_quantity = minimum\_order\_quantity**

**Return reorder\_quantity**

**3. Python Implementation**

**import datetime**

**# Example inventory database structure**

**inventory = {**

**'product\_id': {**

**'name': 'Product Name',**

**'stock\_level': 100,**

**'reorder\_threshold': 20,**

**'average\_daily\_demand': 5,**

**'lead\_time': 7, # in days**

**'safety\_stock': 10**

**}**

**}**

**# Function to update inventory levels**

**def update\_inventory(product\_id, quantity\_sold):**

**if product\_id in inventory:**

**inventory[product\_id]['stock\_level'] -= quantity\_sold**

**check\_reorder(product\_id)**

**# Function to calculate reorder quantity**

**def calculate\_reorder\_quantity(product\_id):**

**product = inventory[product\_id]**

**reorder\_point = (product['lead\_time'] \* product['average\_daily\_demand']) + product['safety\_stock']**

**reorder\_quantity = reorder\_point - product['stock\_level']**

**reorder\_quantity = max(reorder\_quantity, 0) # Ensure non-negative**

**return reorder\_quantity**

**# Function to check if reorder is needed**

**def check\_reorder(product\_id):**

**product = inventory[product\_id]**

**if product['stock\_level'] < product['reorder\_threshold']:**

**reorder\_quantity = calculate\_reorder\_quantity(product\_id)**

**if reorder\_quantity > 0:**

**generate\_reorder\_alert(product\_id, reorder\_quantity)**

**# Function to generate reorder alert**

**def generate\_reorder\_alert(product\_id, reorder\_quantity):**

**print(f"Reorder Alert: Order {reorder\_quantity} units of {inventory[product\_id]['name']}")**

**# Simulate sales data**

**sales\_data = [**

**{'product\_id': 'product\_id', 'quantity\_sold': 85}**

**]**

**# Process sales data**

**for sale in sales\_data:**

**update\_inventory(sale['product\_id'], sale['quantity\_sold'])**

**4. Documentation**

**Reorder Algorithm**

* **Reorder Point Calculation: Based on the formula:  
  Reorder Point=(Lead Time×Average Daily Demand)+Safety StockReorder Point=(Lead Time×Average Daily Demand)+Safety Stock**
* **Reorder Quantity: The difference between the Reorder Point and current stock level. Ensures the stock is replenished to avoid stockouts.**

**Historical Data Influence**

* **Average Daily Demand: Calculated using historical sales data. It’s crucial for estimating future demand and determining the reorder point.**
* **Safety Stock: A buffer to account for variations in demand or delays in supply, calculated based on demand variability.**

**Assumptions**

* **Constant Lead Times: Assumes lead times from suppliers are constant.**
* **Stable Demand: Assumes that demand is relatively stable with predictable fluctuations.**

**5. User Interface**

**The user interface can be implemented using a simple command-line interface (CLI) or a graphical user interface (GUI). The CLI can allow users to input product IDs or names to view current stock levels, reorder recommendations, and historical data.**

**Example CLI Interaction:**

**bash**

**Copy code**

**$ python inventory\_management.py Enter Product ID: product\_id Current Stock Level: 15 Reorder Recommended: Yes Reorder Quantity: 45**

**6. Assumptions and Improvements**

* **Demand Patterns: The system assumes stable demand patterns, which might not be accurate in highly volatile markets. Implementing machine learning models to predict demand more accurately can improve reorder calculations.**
* **Supplier Reliability: Assumes suppliers are reliable with constant lead times. Introducing variability in lead times and building a more resilient system can reduce the risk of stockouts.**
* **Inventory Costs: The system doesn’t currently account for holding costs, ordering costs, or stockout costs. Incorporating these factors into the reorder algorithm can optimize inventory levels further.**