Inventory Management System for B2B SaaS

Backend Engineering Intern (Bynry Inc) – Case Study

-Aryan Sahu

Part 1 : **Code Review & Debugging**

Most of the errors can be identified using given context

1

problem - No atomicity and consistency as db.session.commit() is called twice ,if product commit passes and inventory commit fails then we may have an extra product with no inventory record which is not consistent

Impact – products may be in the underlying in the catalog but missing in inventory which may cause not found errors

Fix – we have to execute all or nothing in one go using for atomic state using a try/except block or a context manager by using rollback and commit

---

2

problem – No input validation for any parameter , If any input is missing or the data is invalid like price being negative etc is not handled

Impact – If the price is negative(-100) it may be depicted as we owe the customer any money or it can be shown like be have negative inventory

Fix – add checks and constraints for the parameters and not trust blindly

---

3

problem – the product model includes warehouse id and the context says that products can exist in multiple warehouses ,but here we are restricting the product to a single warehouse by asking “warehouse_id”

Impact – will result in duplication of products ,if product x exists in 3 warehouses then we will have to make 3 entries for the same product with different warehouse_id which costs us a lot memory if we scale up to a big level

Fix – removing warehouse_id from products and letting it stay in inventory for mapping

4

problem – the price is decimal as stated in context if we use normal computer math we might get slightly diff results for compounded results

Impact – if 1+2 = 3.000001 then for big values ,the 0.1 in the end may hamper our financial tallies

Fix – hence we should use the correct decimal mappings which guarantee that 1+2 = 3 only

---

5

problem – sku's are unique but there is no check for it if it already exists or not

Impact – if there are no duplicates allowed and we forcefully insert a sku matching earlier sku's it may return internal server error 500

Fix – we can use a try/except block here as well ,we try to add the product then commit ,if error pops we rollback saying "the sku already exists"

CORRECTED CODE IS BELOW

|

|

|

|

|

|

|

V

```python
from decimal import Decimal
@app.route('/api/products' , methods = ['POST'])
def create_product():
    data = request.json

    try :
        name = data['name']
        sku = data['sku']
        price = Decimal(str(data['price']))
        qty = int(data.get('initial_quantity' , 0)) #defaulting 0
        warehouse_id = data.get('warehouse_id')

        if price < 0 or qty < 0:
            return {"error": "Price/Quantity cannot be negative"}, 400

        product = Product(
            name = name,
            sku = sku,
            price = price
            #removing the warehouse_id
        )
        db.session.add(product)
        db.session.flush() #temp addition to db ,not commited

        if warehouse_id:
            inv = Inventory(
                product_id = product.id,
                warehouse_id = warehouse_id,
                quantity = qty
            )
            db.session.add(inv)

        #finally commiting
        db.session.commit()
        return {"message": "Success", "product_id": product.id}, 201
    except KeyError:
        db.session.rollback()
        return {"error": "Missing name, sku, or price"}, 400
    except Exception:
        db.session.rollback()
        return {"error": "Invalid data format"}, 400
```

Tried to keep it simple and understandable ,the code is of flask following initial code design

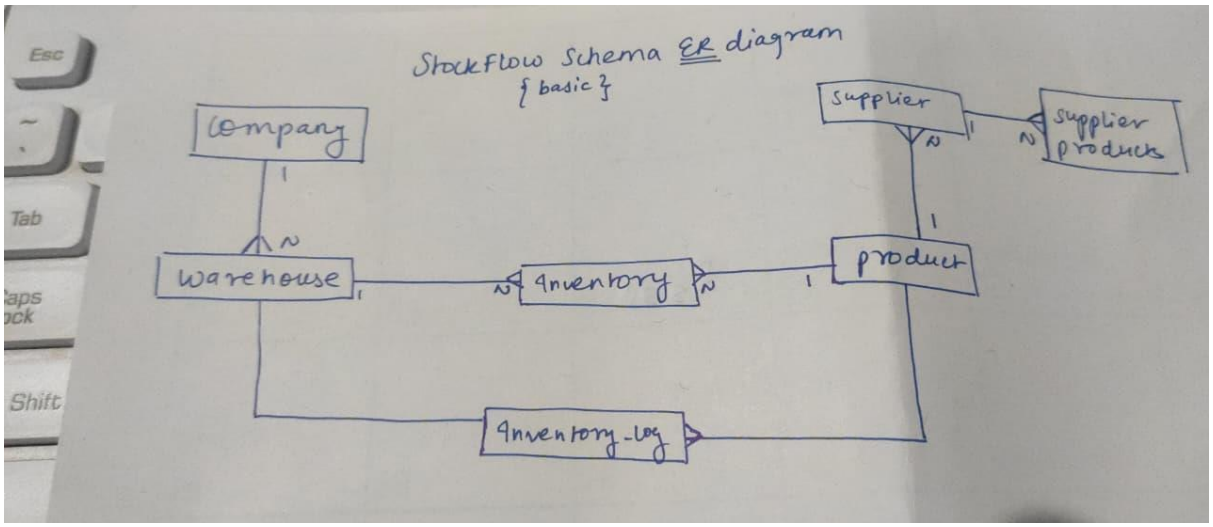
The codefiles are in repo link as well

Part 2 : **Database Design**

First I implemented a basic er diagram on paper and created table according to it itself ,

All the tables are referenced accordingly to maintain referential integrity

ER DIAGRAM



part2_database.sql

```sql
create table companies (
    id primary key,
    name text not null
);
create table warehouses(
    id primary key,
    name text not null,
    company_id integer ,
    foreign key company_id references companies(id)
);
create table products(
    id primary key,sku text unique not null,
    name text not null,
    price decimal(10,2),
    is_bundle boolean default false
);
create table inventory(
    product_id int ,
    warehouse_id int,
    foreign key product_id references products(id),
    foreign key warehouse_id_id references warehouses(id),
    primary key (product_id,warehouse_id)
);
```

```sql
create table inventory_log(
    -- to track changes everytime
    id primary key,
    product_id int ,
    warehouse_id int ,
    foreign key product_id references products(id),
    foreign key warehouse_id_id references warehouses(id),
    change_amount int,
    reason text,
    created_at timestamp default now()
);
create table suppliers(
    id primary key,
    name text not null,
    phone int
);

create table supplier_products(
    supplier_id int,
    foreign key supplier_id references suppliers(id),
    product_id int ,
    foreign key product_id references products(id),
    cost decimal(10,2)
);
```

indexes

```sql

create index idx_warehouse_company on warehouses(id);
create index idx_inventory_warehouese on inventory(warehouse_id);
-- this will help us to look inventory by warehouse

--finally
create index idx_date on inventory_log(created_at);
--this will help us to sort log history by date fastly
```

Assumptions : 1) assumed the skus are unique across all companies globally

2) assuming supplier x sells product y to everyone

3) the price mentioned is the selling price

Methodology:

- According to er diagram ,I mapped all the tables with respected parent tables for mapping and relations
- I created indexes on selected parts only rather than every id as primary key is also present which helps us in fast searches
- There are more tables possible for eg sales , product bundles ,product_categories, employees etc but these tables are sufficient for the start .We can expand eventually
- I tried to include all the requirements given ,I have made a separate inventory log to track changes everytime a product is added (+10) or subtracted/sold (-10)
- I have also showed the relationship between tables that is 1:1 or 1:N depicting the nature of relations

More info I could use:

- Does a supplier sell to a specific company or to everyone?
- Can inventory go negative? like we sell products even if we don't have it currently
- How are the bundles working ? Do we track the bundle stock separately
- More info about sales would help me a lot because I was getting confused in the sales part

Next part

|

|

|

|

|

|

|

|

|

|

V

Part 3: API Implementation

Correct code

part3_api.py > get_low_stock_alerts

```python
from flask import Flask,jsonify
from sqlalchemy import text

@app.route('/api/companies/<int:company_id>/alersts/low-stock',methods = ['GET'])
def get_low_stock_alerts(company_id):
    #getting inventory list first

    sql = text(f"""
        SELECT
                p.id as pid, p.name, p.sku ,p.price,
                w.id as wid , w.name as w_name,
                i.quantity as stock,
                s.id as sid,s.name as sname ,s.email
        FROM inventory i
        JOIN products p on i.product_id = p.id
        JOIN warehouses w on i.warehouse_id = w.id
        left join supplier_products sp on p.id = sp.product_id
        left join supplier s on sp.supplier_id = s.id
        where w.company_id = {company_id}
    """)

    inventory_items = db.session.execute(sql)
    list_for_alerts = []

    for item in inventory_items:
        #1-selecting threshold manually
        if item.price and item.price > 1000:
            threshold = 5 #high price items get a lower warn
        else :
            threshold = 20
        if item.stock >= threshold:
            continue
            #nothing needed as stock is adequate

        #2-checking recent sales
        sales_sql = text(f"""
            select sum(quantity) from sales_order_items s
            join sales_orders so on s.order_id = so.id
            where s.product_id = {item.pid}
```

```
            where s.product_id = {item.pid}
            and so.created_at >= NOW() - INTERVAL '30 days'
        """)

        recent_sales = db.session.execute(sales_sql).scalar() or 0

        if recent_sales == 0:
            continue
        #this is the dead stock which no one is buying hence we dont need to alert it

        avg_daily_sales = recent_sales/30.0
        days_left = int(item.stock/avg_daily_sales)

        #3 add everything to return listt
        list_for_alerts.append({
            "product_id" : item.pid,
            "product_name" : item.name,
            "sku" : item.sku,
            "warehouse_id" : item.wid,
            "warehouse_name" : item.w_name,
            "current_stock" : item.stock,
            "threshold" : threshold,
            "days_until_stockout":days_left,
            "supplier":{
                "id" : item.sid,
                "name":item.sname,
                "mobile":item.phone
                #i used phone number instead of email in data definition
            }
        })
    return jsonify({
        "alerts":list_for_alerts,
        "total_alerts" : len(list_for_alerts)
    })


```

APPROACH :

1 - WE first fetch all the inventory of the company in one single query ,we join the product ,warehouse and supplier tables to get raw data and stock

2 - We process the rules for each row using for loop ,we are checking for thresholds and dead stock

3 - Finally we calculate days_until_stockout using the formula (stock/avg sales per day) and create the final response that is the alert

Then I successfully return the jsonified answer with 2 params ,Hence this api endpoint works perfectly according to the business logic and stock awareness for any company present in databasE

ASSUMPTIONS:

A - I assumed the sales schema to be like this :

sales.sql

```sql
create table sales_order(
    id primary key,
    creaed_at timestamp default NOW()
);

create table sales_order_items(
    id primary key,
    order_id int,
    foreign key order_id references sales_order(id),
    product_id int ,
    foreign key product_id references products(id),
    quantity int
```

The sales_order tracks when the sale happened and the id of sales , the sales_order_items tracks what is inside that order and is referencing the id from sales_order table

B - I didn't group by the suppliers and products so assuming that product has one supplier only

C -the price currency I am handling is rupees only but can vary in real world (dollars , yen, pound etc)

Edge cases:

1 – I am handling the dead stock edge case here gracefully ,if the item is not purchased in last 30 days it can be said as dead

2 – price based thresholds – I am placing checks for threshold wrt price ,lower price products should be warned even If they have comparatively high numbered products

3 – handling division by zero error ,for days_left we require avg_sales which can we 0 ,but the case is already handled in edge case(1)

Hence I successfully implemented all the 3 tasks in the case study ,all of this is handwritten.

I hope I get a positive response as I have given a lot of efforts for the case study

Thank you for the opportunity

ARYAN SAHU

aryansahu2705@gmail.com

(PICT ENTC) third year student

06/01/2026