Inventory Management System for B2B SaaS

Case Study

Part 1: Code Review & Debugging

Step 1: Identify Issues

Issue	Impact
1. warehouse_id in Product may not make sense	Product can exist in multiple warehouses, so this violates normalization
2. No uniqueness check for sku	Could lead to duplicate SKUs, breaking business logic
3. No validation for required/optional fields	May throw KeyError or save invalid data
4. price should support decimals	If float, may cause precision errors
5. No error handling	On failure, system might crash or commit incomplete state
6. No transaction wrap	Risk of inconsistent state if the second commit fails
7. No inventory check for existing product in the same warehouse	May duplicate inventory rows
8. Hard-coded assumptions (e.g. initial_quantity)	Lacks defaults or null checks

Step 2: Fix the Code (With Comments)

```
@app.route('/api/products', methods=['POST'])
def create_product():
    data = request.get_json()


    try:
        # Validate required fields
```

```
required_fields = ['name', 'sku', 'price', 'initial_quantity', 'warehouse_id']
for field in required_fields:
    if field not in data:
        return {"error": f"{field} is required."}, 400

# Check SKU uniqueness
if Product.query.filter_by(sku=data['sku']).first():
    return {"error": "SKU must be unique."}, 400

# Start DB transaction
with db.session.begin_nested():
    # Create product
    product = Product(
        name=data['name'],
        sku=data['sku'],
        price=Decimal(data['price'])  # ensure decimal precision
    )
    db.session.add(product)
    db.session.flush()  # get product.id

    # Add inventory entry
    inventory = Inventory(
        product_id=product.id,
        warehouse_id=data['warehouse_id'],
        quantity=data['initial_quantity']
    )
    db.session.add(inventory)
```

```
        db.session.commit()
        return {"message": "Product created", "product_id": product.id}, 201

    except Exception as e:
        db.session.rollback()
        return {"error": str(e)}, 500
```

Part 2: Database Design

Step 1: Schema Design

```
-- 1. Company Table
CREATE TABLE Company (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    email VARCHAR(255) UNIQUE
);

-- 2. Warehouse Table
CREATE TABLE Warehouse (
    id SERIAL PRIMARY KEY,
    company_id INT NOT NULL,
    name VARCHAR(255) NOT NULL,
    location VARCHAR(255),
    FOREIGN KEY (company_id) REFERENCES Company(id) ON DELETE CASCADE
);

-- 3. Product Table
```

```
CREATE TABLE Product (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    sku VARCHAR(100) UNIQUE NOT NULL,
    price DECIMAL(10, 2) NOT NULL
    -- You may add 'low_stock_threshold INT' here if required
);

-- 4. Inventory Table
CREATE TABLE Inventory (
    id SERIAL PRIMARY KEY,
    product_id INT NOT NULL,
    warehouse_id INT NOT NULL,
    quantity INT NOT NULL DEFAULT 0,
    UNIQUE(product_id, warehouse_id),
    FOREIGN KEY (product_id) REFERENCES Product(id) ON DELETE CASCADE,
    FOREIGN KEY (warehouse_id) REFERENCES Warehouse(id) ON DELETE CASCADE
);

-- 5. InventoryChangeLog Table
CREATE TABLE InventoryChangeLog (
    id SERIAL PRIMARY KEY,
    inventory_id INT NOT NULL,
    change_amount INT NOT NULL,
    reason TEXT,
    changed_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (inventory_id) REFERENCES Inventory(id) ON DELETE CASCADE
```

```
);

-- 6. Supplier Table
CREATE TABLE Supplier (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    contact_email VARCHAR(255)
);

-- 7. ProductSupplier (many-to-many) Table
CREATE TABLE ProductSupplier (
    product_id INT NOT NULL,
    supplier_id INT NOT NULL,
    PRIMARY KEY (product_id, supplier_id),
    FOREIGN KEY (product_id) REFERENCES Product(id) ON DELETE CASCADE,
    FOREIGN KEY (supplier_id) REFERENCES Supplier(id) ON DELETE CASCADE
);

-- 8. Bundle Table
CREATE TABLE Bundle (
    id SERIAL PRIMARY KEY,
    bundle_name VARCHAR(255) NOT NULL
);

-- 9. BundleItems Table
CREATE TABLE BundleItems (
    bundle_id INT NOT NULL,
    product_id INT NOT NULL,
    quantity INT NOT NULL DEFAULT 1,
    PRIMARY KEY (bundle_id, product_id),
    FOREIGN KEY (bundle_id) REFERENCES Bundle(id) ON DELETE CASCADE,
    FOREIGN KEY (product_id) REFERENCES Product(id) ON DELETE CASCADE
);
```

Step 2: Identify Gaps (Ask these in final doc)

- How is stock level updated (sales, returns, etc.)?
- Are bundles treated as inventory units or just logical groups?
- Can a product have multiple suppliers?
- Is pricing per warehouse or global?

Step 3: Explain Decisions

- **Indexes**: SKU (unique), foreign keys indexed
- **Constraints**: Unique inventory per (product, warehouse)
- **Normalization**: Products abstracted from inventory; suppliers separated
- **Scalability**: Supports multiple warehouses, multiple suppliers, and bundles

Part 3: API Implementation

Implement:

GET /api/companies/{company_id}/alerts/low-stock

Return low-stock products that:

- Have sales activity
- Varying thresholds
- Are aggregated per warehouse
- Include supplier info

Example Python (Flask + SQLAlchemy-style pseudocode):

```
@app.route('/api/companies/<int:company_id>/alerts/low-stock', methods=['GET'])
```

```
def low_stock_alerts(company_id):
    try:
        alerts = []

        # Step 1: Get all warehouses for company
        warehouses = Warehouse.query.filter_by(company_id=company_id).all()

        for warehouse in warehouses:
            # Step 2: Get inventory items in this warehouse with recent sales
            inventory_items = db.session.query(
                Inventory, Product, Supplier
            ).join(Product).outerjoin(ProductSupplier).outerjoin(Supplier).filter(
                Inventory.warehouse_id == warehouse.id,
                Inventory.quantity < Product.low_stock_threshold,  # Assuming threshold stored in Product
                Product.has_recent_sales == True  # Boolean field or derived logic
            ).all()

            for inventory, product, supplier in inventory_items:
                alerts.append({
                    "product_id": product.id,
                    "product_name": product.name,
                    "sku": product.sku,
                    "warehouse_id": warehouse.id,
                    "warehouse_name": warehouse.name,
                    "current_stock": inventory.quantity,
                    "threshold": product.low_stock_threshold,
                    "days_until_stockout": estimate_days_until_stockout(product.id),  # custom logic
                    "supplier": {
                        "id": supplier.id,
                        "name": supplier.name,
                        "contact_email": supplier.contact_email
                    } if supplier else None
                })

        return {"alerts": alerts, "total_alerts": len(alerts)}, 200

    except Exception as e:
        return {"error": str(e)}, 500
```

Edge Cases to Handle :

- Products with no supplier
- Multiple suppliers (pick preferred one or show all?)
- No recent sales activity → skip
- Threshold missing → set default
- Missing inventory record → 0 stock?