Part 1: Code Review & Debugging

Issues Identified:

- - No SKU uniqueness check (may result in duplicate SKUs)
- - No error handling or input validation
- - Products wrongly tied to a single warehouse
- - Price might not support decimals
- - No rollback if inventory creation fails

Fixed Code with Explanation:

- - Added input validation and SKU uniqueness check
- - Used float for price
- - Used transaction safety with rollback
- - Removed warehouse_id from Product (belongs in Inventory)

Code :

```
from flask import request, jsonify
from sqlalchemy.exc import IntegrityError, DataError
from decimal import Decimal, InvalidOperation
import re

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

```
    if not data:
        return jsonify({"error": "No data provided"}), 400

    # Validate required fields
    required_fields = ['name', 'sku', 'price', 'warehouse_id', 'initial_quantity']
    missing_fields = [field for field in required_fields if field not in data]

    if missing_fields:
        return jsonify({
            "error": "Missing required fields",
            "missing_fields": missing_fields
        }), 400

    # Validate data types and formats
    validation_errors = []

    # Validate name (non-empty string)
    if not isinstance(data['name'], str) or not data['name'].strip():
        validation_errors.append("name must be a non-empty string")

    # Validate SKU (alphanumeric, unique)
    if not isinstance(data['sku'], str) or not data['sku'].strip():
        validation_errors.append("sku must be a non-empty string")
    elif not re.match(r'^[A-Za-z0-9\-_] $', data['sku']):
        validation_errors.append("sku must contain only alphanumeric characters, hyphens, and underscores")

    # Check SKU uniqueness
```

```
existing_product = Product.query.filter_by(sku=data['sku']).first()
if existing_product:
    return jsonify({"error": "SKU already exists"}), 409

# Validate price (positive decimal)
try:
    price = Decimal(str(data['price']))
    if price <= 0:
        validation_errors.append("price must be a positive number")
except (InvalidOperation, ValueError, TypeError):
    validation_errors.append("price must be a valid decimal number")

# Validate warehouse_id (positive integer)
try:
    warehouse_id = int(data['warehouse_id'])
    if warehouse_id <= 0:
        validation_errors.append("warehouse_id must be a positive integer")
except (ValueError, TypeError):
    validation_errors.append("warehouse_id must be a valid integer")

# Validate initial_quantity (non-negative integer)
try:
    initial_quantity = int(data['initial_quantity'])
    if initial_quantity < 0:
        validation_errors.append("initial_quantity must be a non-negative integer")
except (ValueError, TypeError):
    validation_errors.append("initial_quantity must be a valid integer")
```

```
# Check if warehouse exists
warehouse = Warehouse.query.get(warehouse_id)
if not warehouse:
    return jsonify({"error": "Warehouse not found"}), 404


if validation_errors:
    return jsonify({
        "error": "Validation errors",
        "details": validation_errors
    }), 400

# Start database transaction
try:
    # Create new product (without warehouse_id in Product model)
    product = Product(
        name=data['name'].strip(),
        sku=data['sku'].strip(),
        price=price
    )

    db.session.add(product)
    db.session.flush()  # Get the product ID without committing

    # Create inventory record
    inventory = Inventory(
        product_id=product.id,
        warehouse_id=warehouse_id,
        quantity=initial_quantity
```

```
        )

        db.session.add(inventory)
        db.session.commit()

        return jsonify({
            "message": "Product created successfully",
            "product_id": product.id,
            "inventory_id": inventory.id
        }), 201

    except IntegrityError as e:
        db.session.rollback()
        return jsonify({"error": "Database integrity error", "details": str(e)}), 400
    except DataError as e:
        db.session.rollback()
        return jsonify({"error": "Data error", "details": str(e)}), 400
    except Exception as e:
        db.session.rollback()
        return jsonify({"error": "Internal server error"}), 500

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

Part 2: Database Design

Tables Designed:

- - companies(id, name)
- - warehouses(id, company_id, name)
- - products(id, name, sku UNIQUE, price DECIMAL, is_bundle)
- - product_bundles(bundle_id, component_id, quantity)
- - inventory(id, product_id, warehouse_id, quantity)
- - inventory_logs(id, product_id, warehouse_id, quantity_change, timestamp)
- - suppliers(id, name, contact_email)
- - supplier_products(supplier_id, product_id)

Questions for Product Team:

- - Do warehouses need to track more details (address, contact, capacity, etc.)?
- - Should we track who made the change (user/accountability)?
- - What are the possible reasons for inventory changes (returns, damages, etc.)?
- - Should we track who made the change (user/accountability)?
- - Should bundles be tracked as inventory themselves, or only their components?
- - Do products belong to a company, or are they global?
- - Do we need to track expiration dates?

Design Justification:

Indexes:

• Primary keys on all tables for fast lookup.

• Unique constraints on (warehouse_id, product_id) in inventory for quick inventory checks.

• Foreign keys for referential integrity.

• Indexes on foreign keys (e.g., company_id in warehouses, product_id in inventory) for join performance.

Constraints:

• Foreign key constraints to ensure data integrity.

• Unique constraints to prevent duplicate records (e.g., product-supplier, warehouse-product).

• Not null constraints on required fields.

Bundle Handling:

• Used a join table (product_bundles) to allow bundles to contain multiple products and specify quantities.

• Used a boolean is_bundle in products for easy identification.

Inventory Change Tracking:

• Separate inventory_changes table for audit/history, linked to inventory for traceability.

• Includes timestamp and reason for change.

Flexibility:

• Many-to-many relationships (e.g., products-suppliers, bundles-products) for flexibility.

• Schema can be extended for additional attributes (e.g., product categories, warehouse details).

Database Schema :

```
CREATE TABLE companies (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL UNIQUE
);
```

```
CREATE TABLE warehouses (
    id SERIAL PRIMARY KEY,
    company_id INT NOT NULL REFERENCES companies(id),
    name VARCHAR(255) NOT NULL,
    location VARCHAR(255)
);

CREATE TABLE suppliers (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL UNIQUE
);

CREATE TABLE products (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    sku VARCHAR(100) UNIQUE,
    is_bundle BOOLEAN NOT NULL DEFAULT FALSE
);

CREATE TABLE product_suppliers (
    product_id INT REFERENCES products(id),
    supplier_id INT REFERENCES suppliers(id),
    PRIMARY KEY (product_id, supplier_id)
);

CREATE TABLE inventory (
    id SERIAL PRIMARY KEY,
```

```
    warehouse_id INT REFERENCES warehouses(id),
    product_id INT REFERENCES products(id),
    quantity INT NOT NULL DEFAULT 0,
    UNIQUE (warehouse_id, product_id)
);

CREATE TABLE inventory_changes (
    id SERIAL PRIMARY KEY,
    inventory_id INT REFERENCES inventory(id),
    change_type VARCHAR(50) NOT NULL,
    quantity_delta INT NOT NULL,
    changed_at TIMESTAMP NOT NULL DEFAULT now(),
    reason TEXT
);

CREATE TABLE product_bundles (
    bundle_id INT REFERENCES products(id),
    product_id INT REFERENCES products(id),
    quantity INT NOT NULL DEFAULT 1,
    PRIMARY KEY (bundle_id, product_id)
);
```

Part 3: API Implementation

Endpoint: GET /api/companies/<company_id>/alerts/low-stock

Assumption:

- - SQLAlchemy models: Product, Inventory, Warehouse, Supplier, Sale, LowStockThreshold
- - Product has: id, name, sku, type, supplier_id
- - Inventory has: product_id, warehouse_id, quantity
- - Warehouse has: id, name, company_id
- - Supplier has: id, name, contact_email
- - Sale has: id, product_id, warehouse_id, date, quantity
- - LowStock Threshold has: product_type, threshold
- - db is SQLAlchemy session
- - Flask app is named 'app'
- - All models are imported

Business Rules:

- - Only alert for products with recent sales (last 30 days)
- - Low stock threshold varies by product type
- - Multiple warehouses per company
- - Include supplier info

Edge Cases:

- - Company not found: return 404
- - No products/warehouses: return empty alerts
- - No supplier: supplier field is null

code :

```
from flask import request, jsonify
from datetime import datetime, timedelta

@app.route('/api/companies/<int:company_id>/alerts/low-stock', methods=['GET'])
def low_stock_alerts(company_id):
    # 1. Get all warehouses for the company
    warehouses = Warehouse.query.filter_by(company_id=company_id).all()
    if not warehouses:
        return jsonify({"alerts": [], "total_alerts": 0}), 200
    warehouse_ids = [w.id for w in warehouses]
    warehouse_map = {w.id: w.name for w in warehouses}

    # 2. Get all products in these warehouses (via Inventory)
    inventories = Inventory.query.filter(Inventory.warehouse_id.in_(warehouse_ids)).all()
    if not inventories:
        return jsonify({"alerts": [], "total_alerts": 0}), 200

    # 3. Get recent sales (last 30 days) for these products/warehouses
    thirty_days_ago = datetime.utcnow() - timedelta(days=30)
    sales = Sale.query.filter(
        Sale.product_id.in_([inv.product_id for inv in inventories]),
        Sale.warehouse_id.in_(warehouse_ids),
        Sale.date >= thirty_days_ago
    ).all()
    recent_product_ids = set(s.product_id for s in sales)
    if not recent_product_ids:
```

```
        return jsonify({"alerts": [], "total_alerts": 0}), 200


    # 4. Get thresholds by product type
    # (Assume all product types in this set)
    product_ids = [inv.product_id for inv in inventories if inv.product_id in
recent_product_ids]
    products = Product.query.filter(Product.id.in_(product_ids)).all()
    product_map = {p.id: p for p in products}
    product_types = set(p.type for p in products)
    thresholds =
LowStockThreshold.query.filter(LowStockThreshold.product_type.in_(product_types)).all()
    threshold_map = {t.product_type: t.threshold for t in thresholds}


    # 5. Build alerts
    alerts = []
    for inv in inventories:
        if inv.product_id not in recent_product_ids:
            continue
        product = product_map.get(inv.product_id)
        if not product:
            continue
        threshold = threshold_map.get(product.type, 10)  # Default threshold if missing
        if inv.quantity < threshold:
            # Estimate days until stockout: avg daily sales in last 30 days
            sales_for_product = [s for s in sales if s.product_id == inv.product_id and
s.warehouse_id == inv.warehouse_id]
            total_sold = sum(s.quantity for s in sales_for_product)
            avg_daily_sales = total_sold / 30 if total_sold else 0.1  # Avoid div by zero
            days_until_stockout = int(inv.quantity / avg_daily_sales) if avg_daily_sales else None
```

```
        # Supplier info
        supplier = Supplier.query.get(product.supplier_id) if product.supplier_id else None
        supplier_info = {
            "id": supplier.id,
            "name": supplier.name,
            "contact_email": supplier.contact_email
        } if supplier else None
        alerts.append({
            "product_id": product.id,
            "product_name": product.name,
            "sku": product.sku,
            "warehouse_id": inv.warehouse_id,
            "warehouse_name": warehouse_map.get(inv.warehouse_id, "Unknown"),
            "current_stock": inv.quantity,
            "threshold": threshold,
            "days_until_stockout": days_until_stockout,
            "supplier": supplier_info
        })
    return jsonify({"alerts": alerts, "total_alerts": len(alerts)})
```