Part 1: Code Review & Debugging

```python
@app.route('/api/products', methods=['POST'])
def create_product():
    data = request.json
    # Create new product
    product = Product(
        name=data['name'],
        sku=data['sku'],
        price=data['price'],
        warehouse_id=data['warehouse_id']
    )
    db.session.add(product)
    db.session.commit()

    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}
```

1. No data validation / no error handling

- If a required field is missing (name, sku, price, warehouse_id, or initial_quantity), the code will throw a error and crash.
- No validation for:
    - Empty strings
    - Invalid data types (e.g., price as a string)
    - Negative quantities or prices
- No try/except block → any DB error leaves the API returning a generic 500 without rollback.

2. Redundant session commits

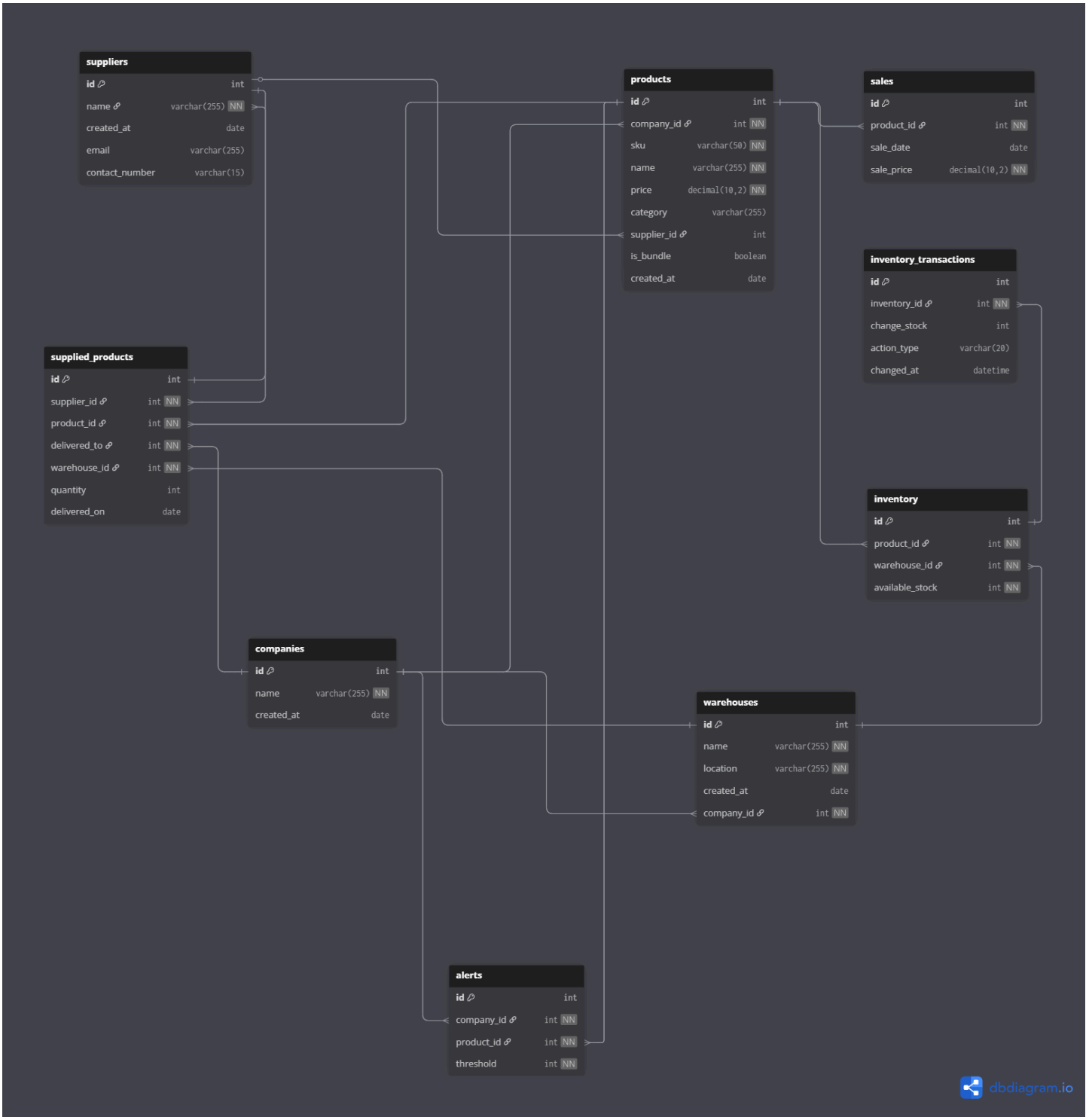
- create **two separate database transactions**.
- In SQLAlchemy, each commit() ends the current transaction and starts a new one.
- If the DB is on a different server, that's **two TCP connections** for one logical action, which:
    - Wastes bandwidth
    - Adds latency
    - Increases failure risk (e.g., first commit succeeds, second fails → inconsistent DB state)
- In production, this could leave orphan Product rows without an Inventory record.

Corrected Code

```
# Pydantic model for validation

class Product(BaseModel):

    name: Optional[str]

    sku: str

    price: int

    warehouse_id: int

    quantity: int

@app.post("/api/products")

def save_product(product: Product):

    data = product.model_dump()

    try:

        with Session(engine) as session:

            # Create new product

            new_product = Products(

                name=data['name'],

                sku=data['sku'],

                price=data['price'],

                warehouse_id=data['warehouse_id']

            )

            session.add(new_product)

            session.flush()  # ensures new_product.id is available


            # Create inventory entry

            new_inventory = Inventory(

                product_id=new_product.id,

                warehouse_id=data['warehouse_id'],

                quantity=data['quantity']

            )

            session.add(new_inventory)
```

```
        # Commit transaction (both inserts at once)
        session.commit()


        return {
            "message": "Product created successfully",
            "product_id": new_product.id
        }
    except Exception as e:
        return {
            "error": str(e),
            "message": "Failed to create product"
        }
```

Part 2 : Database Design



Part 3: API Implementation

```
@app.get("/api/companies/{company_id}/alerts/low-stock")
def give_alerts(company_id: int):
    sql_query = """
        WITH recent_sold_product_dates AS (
            SELECT
                ps.id AS product_id,
                ps.name,
                MAX(sa.sale_date) AS recent_date
            FROM products ps
            JOIN sales sa ON sa.product_id = ps.id
            WHERE ps.company_id = :company_id
            GROUP BY ps.id, ps.name
        ),
        recently_sold_products AS (
            SELECT name
            FROM recent_sold_product_dates
            WHERE recent_date >= DATE('now', '-90 day')
        ),
        products_sold_each_month AS (
            SELECT
                p.id AS product_id,
                strftime('%m', sa.sale_date) AS sale_month,
                COUNT(*) AS total_sold
            FROM sales sa
            JOIN products p ON p.id = sa.product_id
            GROUP BY p.id, sale_month
        ),
        products_sold_each_day AS (
            SELECT
                product_id,
                SUM(total_sold) / 365.0 AS sold_per_day
            FROM products_sold_each_month
            GROUP BY product_id
        )
        SELECT
            ps.id AS product_id,
            ps.name AS product_name,
            ps.sku,
            wa.id AS warehouse_id,
            wa.name AS warehouse_name,
            inven.available_stock AS current_stock,
            al.threshold,
            (inven.available_stock / NULLIF(psed.sold_per_day, 0)) AS
days_until_stockout,
            sup.id AS supplier_id,
```

```
            sup.name AS supplier_name,
            sup.email AS supplier_email
        FROM products ps
        JOIN inventory inven ON ps.id = inven.product_id
        JOIN warehouses wa ON wa.id = inven.warehouse_id
        JOIN alerts al ON al.product_id = ps.id
        JOIN suppliers sup ON sup.id = ps.supplier_id
        LEFT JOIN products_sold_each_day psed ON psed.product_id = ps.id
        WHERE ps.name IN (SELECT name FROM recently_sold_products)
        AND inven.available_stock < al.threshold;
    """

    with engine.connect() as conn:
        results = conn.execute(text(sql_query), {"company_id": company_id})

        alerts = [
            {
                "product_id": row.product_id,
                "product_name": row.product_name,
                "sku": row.sku,
                "warehouse_id": row.warehouse_id,
                "warehouse_name": row.warehouse_name,
                "current_stock": row.current_stock,
                "threshold": row.threshold,
                "days_until_stockout": row.days_until_stockout,
                "supplier": {
                    "id": row.supplier_id,
                    "name": row.supplier_name,
                    "contact_email": row.supplier_email
                }
            }
            for row in results
        ]

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