Data Mining and Discovery

Access more content here: [GitHub - Data Mining and Discovery]

SQL Assignment
Inventory Management System

By
Awais Mumtaz ul Haq
ID: 23096809

Project Need and Overview:

To improve supply chain procedures, cut waste, and simplify operations, modern firms need to have excellent inventory control. The following advantages are offered by an inventory management system (IMS):

- **Accuracy and Efficiency:** Reduces human mistake by automating the tracking of orders, shipments, and stock levels.
- **Better Decision-Making:** Offers up-to-date information for demand forecasting, procurement planning, and effective warehouse space management.
- **Cost reduction:** Assures the best possible use of resources by preventing overstocking or stockouts.
- **Improved Supplier Coordination:** Keeps thorough records of suppliers to guarantee dependable and timely delivery.
- **Scalability:** By interacting with other business operations, including sales and logistics, a well-designed system facilitates expansion.

By producing accurate data for the following important entities—Items, Suppliers, Warehouses, and Orders—this initiative aims to create a strong IMS. Before deploying the system in production, we test and validate it by simulating real-world circumstances using tools like pandas and Python's Faker.

Database Schema:

There are four primary tables in the database structure. A written explanation of each table's main characteristics and connections can be found below:

Items Table:

- sku (VARCHAR): Unique identifier (Nominal data) for each product.
- description (VARCHAR): A brief description of the product (Nominal data).
- quality_rating (TINYINT): An ordinal measure (1–5 scale) representing product quality.
- manufacture_date (DATE): The production date (Interval data) for the product.
- stock_level (INT): Current quantity in inventory (Ratio data).
  unit_price (DECIMAL): Price per unit (Ratio data).

SQL 1

```sql
-- Create Items table
CREATE TABLE Items (
    sku VARCHAR(50) PRIMARY KEY,              -- Nominal: Unique product code
    description VARCHAR(255),                 -- Nominal: Product description
    quality_rating TINYINT CHECK (quality_rating BETWEEN 1 AND 5), -- Ordinal: 1 (lowest) to 5 (highest) rating
    manufacture_date DATE,                    -- Interval: Dates can be subtracted to find durations
    stock_level INT,                          -- Ratio: Stock levels (true zero applies)
    unit_price DECIMAL(10,2)                  -- Ratio: Price per unit (price 0 means free)
);
```

Suppliers Table:

- supplier_id (INT): Each supplier's unique identification number.
- vendor_name (VARCHAR): The supplier's name (Nominal data).
- contact_info (VARCHAR): Email or other contact information (Nominal data) is stored in.

SQL 1

```sql
CREATE TABLE Suppliers (
    supplier_id INT PRIMARY KEY,
    supplier_name VARCHAR(255),              -- Nominal: Supplier name
    contact_info VARCHAR(255),
    supplier_priority VARCHAR(255)  -- Ordinal: Ranking supplier priority
);
```

Warehouses Table:

- warehouse_id (INT): Unique warehouse identifier.
- location (VARCHAR): Address or description of the warehouse (Nominal data).
- capacity (INT): Storage capacity (Ratio data).

SQL 1

```sql
CREATE TABLE Warehouses (
    warehouse_id INT PRIMARY KEY,
    location VARCHAR(255),                  -- Nominal: Warehouse location name/address
    capacity INT                            -- Ratio: Maximum capacity (e.g., square feet or units)
);
```

Orders Table:

- **order_id (INT):** Unique identifier for each order.
- **order_date (DATE):** Date of order placement (Interval data).
- **order_type (ENUM):** Type of order (‘in’ for incoming or ‘out’ for outgoing stock; Nominal data).
- **sku (VARCHAR):** References the Items table (Foreign key).

- **quantity (INT):** The number of items ordered (Ratio data).
- **warehouse_id (INT):** References the Warehouses table (Foreign key).

SQL 1

```
CREATE TABLE Orders (
    order_id INT PRIMARY KEY,
    order_date DATE,                    -- Interval: Order date (interval differences are meaningful)
    order_type VARCHAR(255),         -- Nominal: Type of order ('in' for incoming stock, 'out' for outgoing shipment)
    sku VARCHAR(50),
    quantity INT,                       -- Ratio: Quantity ordered (zero is a meaningful value)
    warehouse_id INT,
    FOREIGN KEY (sku) REFERENCES Items(sku),
    FOREIGN KEY (warehouse_id) REFERENCES Warehouses(warehouse_id)
);
```

Justification of Separate Tables and Ethical/Data Privacy Discussion:

Separation of Concerns:

- **Items:** Contains details on a certain product. Without being hampered by order or supplier data, item isolation enables comprehensive product management (such as quality rating, price, and stock levels).
- **Suppliers:** Contains information on suppliers that may be updated or audited separately from the products. Multiple vendors for various goods are also supported by this division.
- **Warehouses:** Facilitate effective inventory monitoring and allocation by managing physical storage locations and capacity data.
- **Orders:** Keeps track of all incoming and departing transaction events connected to products and storage facilities. This table facilitates stock movement traceability and is essential to operations.

Normalization:

- By standardizing data across many tables, the architecture prevents data redundancy. To guarantee that product data are only kept once, the SKU in the Orders record, for instance, is a foreign key to the Items table.
- Data integrity and relational consistency are enforced via constraints like primary keys, foreign keys, and check constraints (such as quality_rating between 1 and 5).

Constraints and Data Integrity:

- Primary Keys: Make sure every record can be identified by its unique number.
- Maintain referential integrity between Orders and associated tables (Warehouses and Items) by using foreign keys.
- Enumerated Fields: ENUM types guarantee that only legitimate categories are maintained for characteristics such as supplier_priority and order_type.

- Range Restrictions: Invalid data entry is avoided by the quality_rating check (between 1 and 5).

Ethical and Data Privacy Considerations:

Data Minimization:

Only the information that is required is kept. For instance, suppliers' contact details are restricted to an email address instead of more private information, which lowers the danger of exposure.

Access Controls:

Only authorized workers should have access to sensitive data (such as supplier contacts) in a production setting. Only people with a valid business requirement should be able to see or alter the data, hence role-based access control (RBAC) systems should be put in place.

Data Protection and Encryption:

- To protect against breaches or unwanted access, sensitive fields should be encrypted while they are in transit and at rest.
- To keep a safe environment, regular audits and security protocol changes are necessary.

Ethical Use of Data:

- Making sure the data is only utilized for operational effectiveness and commercial enhancement is one ethical aspect. Repurposing it for unexpected purposes without the proper authorization is not advised.
- Trust is increased with suppliers and internal stakeholders when data collection and use are transparent.

Data Generation:

```
fake = Faker()

# Generate Items Data (50 records)
def generate_items(n=50):
    items = []
    for _ in range(n):
        sku = fake.unique.bothify(text='??-#####')  # Nominal unique SKU
        description = fake.sentence(nb_words=6)       # Nominal description
        quality_rating = random.randint(1, 5)          # Ordinal: rating from 1 to 5
        manufacture_date = fake.date_between(start_date='-5y', end_date='today')  # Interval: date
        stock_level = random.randint(0, 1000)          # Ratio: stock level
        unit_price = round(random.uniform(1, 500), 2)    # Ratio: unit price
        items.append({
            "sku": sku,
            "description": description,
            "quality_rating": quality_rating,
            "manufacture_date": manufacture_date,
            "stock_level": stock_level,
            "unit_price": unit_price
        })
    return pd.DataFrame(items)
```

```python
# Generate Suppliers Data (20 records)
def generate_suppliers(n=20):
    suppliers = []
    for i in range(1, n + 1):
        supplier_name = fake.company()                # Nominal: company name
        contact_info = fake.email()                     # Nominal: contact info
        supplier_priority = random.choice(['High', 'Medium', 'Low'])  # Ordinal: supplier ranking
        suppliers.append({
            "supplier_id": i,
            "supplier_name": supplier_name,
            "contact_info": contact_info,
            "supplier_priority": supplier_priority
        })
    return pd.DataFrame(suppliers)
```

```python
# Generate Warehouses Data (10 records)
def generate_warehouses(n=10):
    warehouses = []
    for i in range(1, n + 1):
        location = fake.address().replace("\n", ", ")   # Nominal: location address
        capacity = random.randint(500, 5000)              # Ratio: capacity
        warehouses.append({
            "warehouse_id": i,
            "location": location,
            "capacity": capacity
        })
    return pd.DataFrame(warehouses)
```

```python
# Generate Orders Data (1000 records)
def generate_orders(n=1000, items_df=None, warehouses_df=None):
    orders = []
    if items_df is None or warehouses_df is None:
        raise ValueError("Items and Warehouses dataframes must be provided")

    skus = items_df['sku'].tolist()
    warehouse_ids = warehouses_df['warehouse_id'].tolist()

    for i in range(1, n + 1):
        order_date = fake.date_between(start_date='-1y', end_date='today')  # Interval: order date
        order_type = random.choice(['in', 'out'])         # Nominal: order type
        sku = random.choice(skus)                           # Reference to an Item's SKU
        quantity = random.randint(1, 200)                   # Ratio: quantity ordered
        warehouse_id = random.choice(warehouse_ids)         # Reference to a Warehouse
        orders.append({
            "order_id": i,
            "order_date": order_date,
            "order_type": order_type,
            "sku": sku,
            "quantity": quantity,
            "warehouse_id": warehouse_id
        })
    return pd.DataFrame(orders)
```

```python
# Generate DataFrames
items_df = generate_items(50)
suppliers_df = generate_suppliers(20)
warehouses_df = generate_warehouses(10)
orders_df = generate_orders(1000, items_df, warehouses_df)

# Save DataFrames to CSV files
items_df.to_csv("items.csv", index=False)
suppliers_df.to_csv("suppliers.csv", index=False)
warehouses_df.to_csv("warehouses.csv", index=False)
orders_df.to_csv("orders.csv", index=False)

print("CSV files generated: items.csv, suppliers.csv, warehouses.csv, orders.csv")
```

Queries on the Data:

1- List All Items with Their Details:

SQL 1

```sql
SELECT sku, description, quality_rating, manufacture_date, stock_level, unit_price
FROM Items;
```

	sku	description	quality_rating	manufacture_date	stock_level	unit_price
1	Ge-88375	Rise employee once drive film.	2	2021-10-27	406	59.82
2	EM-41084	Condition least push property.	3	2024-12-27	639	151.79
3	Hx-86918	Forget only character I during ...	3	2024-05-22	855	189.5
4	bO-09408	Article race stay enjoy scientis...	5	2022-02-25	917	292.29
5	VS-76755	Well pay industry measure ...	3	2022-08-28	599	309.67
6	Yh-44283	Rock skin writer check later.	4	2021-04-26	956	290.01

```
Execution finished without errors.
Result: 50 rows returned in 71ms
At line 1:
SELECT sku, description, quality_rating, manufacture_date, stock_level, unit_price
FROM Items;
```

2- Find Items with Low Stock (e.g., Stock Level Below 100):

SQL 1

```sql
SELECT sku, description, stock_level
FROM Items
WHERE stock_level < 100;
```

	sku	description	stock_level
1	LL-62793	Third nice film music.	44
2	Zd-53896	Tend land simple defense sell.	76
3	aw-23156	Remain involve last pay many ...	75
4	Df-85807	Defense continue soon of happ...	69
5	Tl-44345	Dinner level high third your ...	27
6	Lu-16805	Some individual address place ...	88

```
Execution finished without errors.
Result: 6 rows returned in 19ms
At line 1:
SELECT sku, description, stock_level
FROM Items
WHERE stock_level < 100;
```

3- Get Detailed Order Information (Joining Items and Warehouses):

SQL 1

```sql
SELECT o.order_id,
       o.order_date,
       o.order_type,
       o.quantity,
       i.sku,
       i.description,
       w.location AS warehouse_location
FROM Orders o
JOIN Items i ON o.sku = i.sku
JOIN Warehouses w ON o.warehouse_id = w.warehouse_id;
```

	order_id	order_date	order_type	quantity	sku	description	warehouse_location
1	1	2024-10-27	out	9	Lu-16805	Some individual address place ...	9038 Porter Divide Apt. 766, ...
2	2	2024-10-11	in	49	VS-76755	Well pay industry measure ...	80662 Allen Views, West ...
3	3	2024-03-10	out	173	dm-54929	Hope international activity ...	USNS Sanchez, FPO AA 76830
4	4	2024-09-28	out	72	Zd-53896	Tend land simple defense sell.	9038 Porter Divide Apt. 766, ...
5	5	2024-04-27	out	5	YT-50687	Of reveal record market hope ...	80662 Allen Views, West ...
6	6	2024-05-28	in	122	ZH-66147	Yard side different six pay voice.	PSC 9779, Box 0489, APO AA ...

```
Execution finished without errors.
Result: 1000 rows returned in 324ms
```

4- Calculate Total Quantity Ordered per Item:

SQL 1

```sql
SELECT i.sku, i.description, SUM(o.quantity) AS total_ordered
FROM Orders o
JOIN Items i ON o.sku = i.sku
GROUP BY i.sku, i.description
ORDER BY total_ordered DESC;
```

	sku	description	total_ordered
1	Df-55103	Rise ball hot modern next drug...	3219
2	Ge-88375	Rise employee once drive film.	2936
3	BH-71385	Meet fear specific himself do ...	2804
4	EM-41084	Condition least push property.	2773
5	Gy-19731	Government enter now of ...	2759
6	Ws-30160	Consumer decide time game.	2701

```
Execution finished without errors.
Result: 50 rows returned in 77ms
```