# A Case Study on

# Material Procurement Management for Construction Projects

**Prepared by**

**Chinedu Patrick Amagwara**

Table of Contents

# Introduction

- Material procurement involves acquiring the necessary materials and supplies for construction projects.
- Efficient procurement is crucial for project success, impacting timelines, costs, and overall quality

Challenges

- Complexity of supply chains
- Managing multiple suppliers and contracts
- Tracking inventory levels and material costs
- Ensuring timely deliveries to avoid project delays

**Mission Statement**

The mission of this Procurement Management System is to enhance decision-making, ensure accuracy, and optimize resource allocation for all construction projects.

**Objectives**

1. Build an efficient database system
2. Optimize Inventory Management
3. Improve procurement Tracking
4. Enhanced Reporting

**Building an Efficient Database System: A Step-by-Step Approach**

In today's data-driven world, the backbone of any successful IT system is a well-designed and efficient database. Whether you are managing customer information, tracking inventory, or storing transactional data, the database is key to ensuring that data is organized, accessible, and secure. In this article, I will walk you through the process of building a robust database system, drawing from my own experience in database design.

## 1. Understanding the Business Objectives

The construction industry relies heavily on precise procurement processes to avoid delays and cost overruns. The primary objectives for the database design include:

- Efficient Data Storage: Storing and organizing information about clients, departments, employees, suppliers, materials, warehouses, projects, delivery, orders and inventory.
- Optimized Inventory Management: Tracking materials in real-time across multiple warehouses.
- Procurement Tracking: Monitoring orders and deliveries from suppliers to ensure project deadlines are met.
- Enhanced Reporting: Generating accurate reports for better decision-making on procurement and resource management.

## 2. Understanding the Business Requirements

The first and most crucial step in building a database is understanding the business requirements. Before diving into tables, relationships, or SQL code, it is essential to have a clear understanding of what the database is meant to accomplish. For example, if you are building a database for a retail business, you will need to store data about products, customers, orders, and inventory.

In my case study projects, I worked with a procurement company for a construction client. The goal was to create a system that stored and tracked supplier data, materials, employee details, orders, and deliveries. The business requirements included:

- Storing supplier and client information.
- Managing inventory across multiple warehouses.
- Tracking orders and deliveries from suppliers to project sites.

Having these requirements laid out, clearly helped guide the structure of the database.

## 3. Entity Identification and Structuring the Data

Once you have a solid understanding of the business goals, the next step is identifying the subject/noun and entities in the system. Entities are the things or objects that your database will track. In the case of the procurement database, the main entities were:

- Suppliers: Information about the external vendors or companies that provide materials and services to the organization.
- Client:  Holds data about the organization's customers or clients for whom projects are being executed
- Department: Represents the various departments within the organization, such as procurement, operations, finance, and so on. It is linked to employees to categorize them based on their roles.
- Employees: Employees managing procurement, deliveries, and inventory.
- Materials: Stores data about the raw materials, goods, or products required for projects.

- Warehouse: Tracks the physical locations where materials are stored.
- Projects: represents specific tasks or jobs being carried out for a client.
- Inventory: Tracks the materials stored in each warehouse and their current stock levels.
- SupplierMaterial: Serves as a junction table to manage the many-to-many relationship between suppliers and materials and tracks which suppliers provide which material.
- Orders: Represents the procurement orders placed by the organization for materials. It tracks the details of what materials were ordered, for which project, and from which supplier.
- Deliveries: Tracks the shipments of materials from suppliers to the warehouse or project site.

Each of these entities becomes a table in the database, and the next step is to define the specific data (attributes) for each table. For example, the Suppliers table might include fields like SupplierID, SupplierName, ContactPerson, and Email. Similarly, the Orders table would store OrderID, ProjectID, SupplierID, Quantity, and OrderDate.

### 4. Establishing Relationships Between Entities

The power of a relational database lies in the relationships between tables. This step involves determining how the entities interact with each other. Below are the relationships in procurement database:

Entity Relationships

1. Client ↔ Project (One-to-Many): One client can be associated with multiple projects, but each project is linked to only one client.
2. Material ↔ SupplierMaterial (Many-to-Many via SupplierMaterial): Multiple suppliers can provide the same material, and a supplier can provide multiple materials. This is a many-to-many relationship handled by the SupplierMaterial table.
3. Supplier ↔ SupplierMaterial (One-to-Many): Each supplier can supply multiple materials, but a given supplier-material pair is unique.
4. Material ↔ SupplierMaterial (One-to-Many): Each material can be supplied by multiple suppliers.
5. Employee ↔ Department (Many-to-One): Many employees can belong to a single department, but an employee belongs to only one department.
6. Warehouse ↔ Inventory (One-to-Many): A warehouse can contain multiple inventory items, but each inventory record is stored in one warehouse.
7. SupplierMaterial ↔ Inventory (One-to-Many): One supplier-material combination can have multiple inventory records in different warehouses.
8. Project ↔ Order (One-to-Many): A project can have multiple orders placed for materials, but each order is linked to one project.
9. SupplierMaterial ↔ Order (One-to-Many): Each order corresponds to a specific supplier-material pair.
10. Employee ↔ Order (One-to-Many): An employee can place multiple orders, but each order is placed by a single employee.

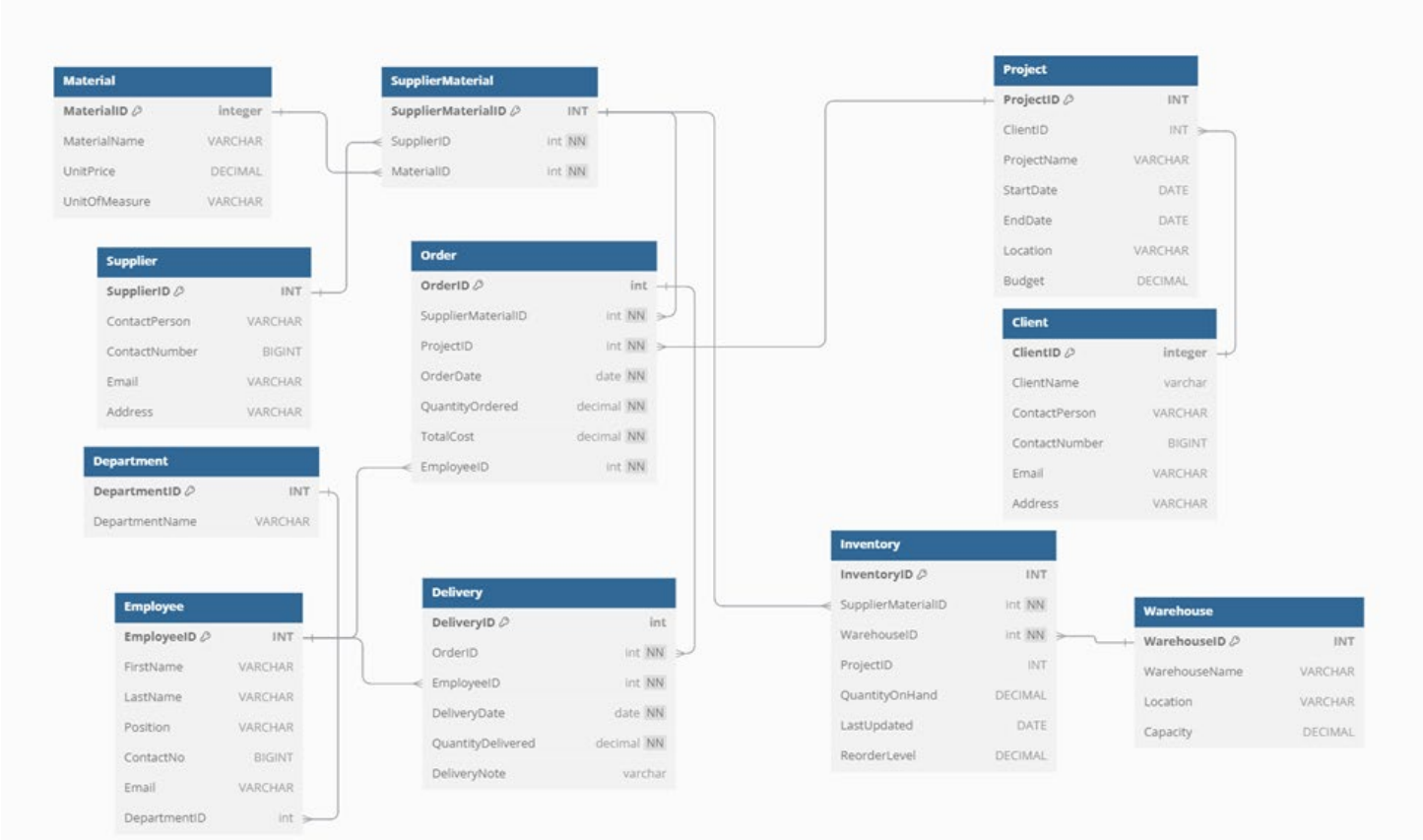
11. Order ↔ Delivery (One-to-Many): An order can have multiple deliveries, but each delivery is related to a single order.
12. Employee ↔ Delivery (One-to-Many): An employee can be responsible for multiple deliveries, but each delivery is handled by a single employee.

Overall Relationship Overview:

- Client ↔ Project: One client can have many projects.
- Supplier ↔ SupplierMaterial ↔ Material: A supplier can supply multiple materials, and materials can come from multiple suppliers (many-to-many).
- Department ↔ Employee: Each department has multiple employees.
- Warehouse ↔ Inventory: A warehouse can store many inventory items.
- Project ↔ Order: Each project can have multiple material orders.
- SupplierMaterial ↔ Order: An order is for a specific supplier-material combination.
- Order ↔ Delivery: An order can have multiple deliveries.
- Employee ↔ Order/Delivery: Employees manage both orders and deliveries.

To represent these relationships, I used primary keys (to uniquely identify records) and foreign keys (to establish relationships between tables). For example, the SupplierID in the SupplierMaterials table acts as a foreign key that links each material to a supplier.

# Entity Relationship Diagram




## 5. Defining Constraints and Data Integrity

After defining the entities and relationships, the next step is ensuring data integrity. This means setting rules that prevent errors and inconsistencies in the database. Some common data integrity rules include:

- Unique Constraints: Fields like email addresses should be unique to prevent duplicates.
- Not Null Constraints: Critical fields like OrderDate or Quantity should always have a value and cannot be left blank.
- Foreign Key Constraints: Ensure that foreign keys only reference valid records in related tables, maintaining referential integrity.

For instance, in the procurement database, if an order references a SupplierID, that supplier must already exist in the Suppliers table. Without this rule, the database might allow an order to be created for a nonexistent supplier, leading to inconsistent data.

## 6. Implementing the Database Schema

Once the design is ready, it is time to implement the database schema. This involves writing the SQL code to create the tables, relationships, and constraints in the database management system (DBMS) of your choice—such as MySQL, PostgreSQL, or SQL Server.

Here is an example of a simple SQL query to create the Suppliers table in MySQL:

```sql
CREATE TABLE Supplier (
    SupplierID INT AUTO_INCREMENT PRIMARY KEY,
    SupplierName VARCHAR(100) NOT NULL,
    ContactPerson VARCHAR(100),
    ContactNumber BIGINT,
    Email VARCHAR(100) UNIQUE NOT NULL,
    Address VARCHAR(255)
);
```

After creating all the tables and relationships, the next step is to populate the database with data and test it. Test queries can help ensure the database structure supports the business operations and allows for efficient data retrieval.

## 7. Optimizing Performance

A well-structured database is not just about having the right tables and relationships—it also needs to be optimized for performance. This means ensuring that the system can handle a large number of queries without slowing down, especially as the dataset grows.

Some optimization techniques include:

- Indexes: Creating indexes on frequently queried fields can speed up data retrieval. For example, indexing the SupplierID in the Orders table can make searching for all orders from a specific supplier much faster.
- Normalization: Normalization helps eliminate redundancy by dividing the data into smaller tables. However, there's a balance between too much normalization (which can complicate queries) and too little (which can result in data duplication).
- Efficient Queries: Writing optimized SQL queries is critical to ensure that large datasets are queried efficiently without unnecessary processing.

### 8. Ensuring Scalability and Security

As the system grows, the database must be scalable to handle increasing data and user load. This can involve setting up distributed databases, sharding, or partitioning data. Additionally, security measures such as encryption, user authentication, and access control should be in place to protect sensitive data.

In the procurement system, for example, sensitive data such as supplier and employee contact details need to be protected from unauthorized access. This was achieved by enforcing role-based access controls (RBAC), where only authorized personnel could access or modify specific data.

## Conclusion

Building an efficient database system requires careful planning, a thorough understanding of business needs, and attention to detail. From identifying the right entities and relationships to enforcing data integrity and optimizing performance, each step contributes to creating a system that supports the organization's goals.

For me, the key to success in database design is balancing structure and flexibility—ensuring that the database is robust enough to manage current data needs while also being scalable for future growth. In the end, a well-designed database serves as the foundation for reliable, efficient, and secure IT systems.

# Appendix

## Tables

**Preliminary Tables:**

1. Supplier
2. Client
3. Material
4. Employee
5. Department
6. Warehouse
7. Project
8. Inventory
9. Delivery
10. ~~Procurement~~
11. ~~Budget~~
12. ~~Finance~~

**Final Table:**

1. Supplier
2. Client
3. Employee
4. Department
5. Material
6. Warehouse
7. Project
8. Inventory
9. SupplierMaterial
10. Delivery
11. Order

## Dimension Tables

Client Table (Dimension Table)

| Field | Data Type | Constraint |
|---|---|---|
| ClientID | INT AUTO_INCREMENT | PRIMARY KEY |
| ClientName | VARCHAR(255) | NOT NULL |
| ContactPerson | VARCHAR(255 | |
| ContactNumber | BIGINT | CHECK (ContactNumber BETWEEN 1000000000 AND 9999999999) |
| Email | VARCHAR(255) | UNIQUE NOT NULL CHECK, (Email LIKE '%_@__%.__%'), |
| Address | VARCHAR(255) | |

Material Table (Dimension Table)

| Field | Data Type | Constraint |
|---|---|---|
| MaterialID | INT AUTO_INCREMENT | PRIMARY KEY |
| MaterialName | VARCHAR(255) | NOT NULL |
| UnitPrice | DECIMAL(10, 2) | NOT NULL CHECK (UnitPrice > 0) |
| UnitOfMeasure | VARCHAR(50) | NOT NULL |

Supplier Table (Dimension Table)

| Field | Data Type | Constraint |
|---|---|---|
| SupplierID | INT AUTO_INCREMENT | PRIMARY KEY |
| SupplierName | VARCHAR(255) | NOT NULL |
| ContactPerson | VARCHAR(255 | |
| ContactNumber | BIGINT | CHECK (ContactNumber BETWEEN 1000000000 AND 9999999999) |
| Email | VARCHAR(255) | UNIQUE NOT NULL CHECK, (Email LIKE '%_@__%.__%'), |

Department Table (Dimension Table)

| Field | Data Type | Constraint |
|---|---|---|
| DepartmentID | INT AUTO_INCREMENT | PRIMARY KEY |
| DepartmentName | VARCHAR(255) | NOT NULL |

Employee Table (Dimension Table)

| Field | Data Type | Constraint |
|---|---|---|
| EmployeeID | INT AUTO_INCREMENT | PRIMARY KEY |
| FirstName | VARCHAR(255) | NOT NULL |
| LastName | VARCHAR(255 | NOT NULL |
| Position | VARCHAR(256 | NOT NULL |
| ContactNumber | BIGINT | CHECK (ContactNumber BETWEEN 1000000000 AND 9999999999) |
| Email | VARCHAR(255) | UNIQUE NOT NULL CHECK, (Email LIKE '%_@__%.__%'), |

Warehouse Table (Dimension Table)

| Field | Data Type | Constraint |
|---|---|---|
| WarehouseID | INT AUTO_INCREMENT | PRIMARY KEY |
| WarehouseName | VARCHAR(255) | NOT NULL |
| Location | VARCHAR(255) | NOT NULL |
| Capacity | DECIMAL(10, 2) | CHECK (Capacity > 0 |

Project Table (Dimension Table)

| Field | Data Type | Constraint |
|---|---|---|
| ProjectID | INT AUTO_INCREMENT | PRIMARY KEY |
| ClientID | INT | NOT NULL, FOREIGN KEY |
| ProjectName | VARCHAR(255 | |
| StartDate | DATE | NOT NULL |
| EndDate | DATE | NOT NULL |
| Location | VARCHAR(255) | |

## Fact Tables

SupplierMaterial Table

| Field | Data Type | Constraint |
|---|---|---|
| SupplierMaterialID | INT AUTO_INCREMENT | PRIMARY KEY |
| SupplierID | INT | NOT NULL, FOREIGN KEY |
| MaterialID | INT | NOT NULL, FOREIGN KEY |

Inventory Table (Fact Table)

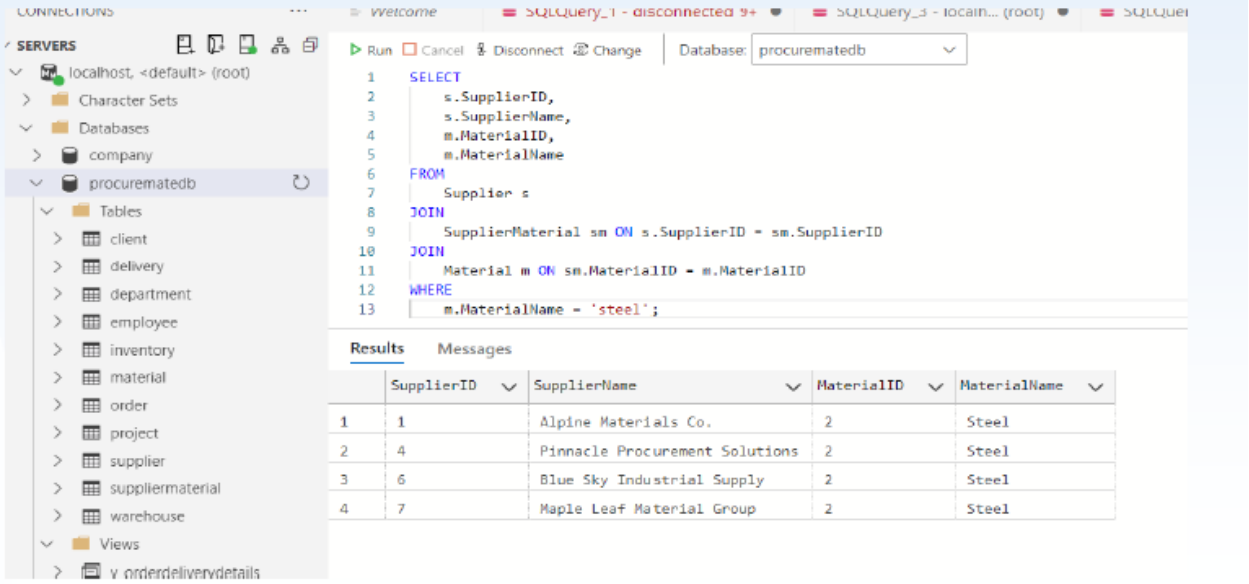
| Field | Data Type | Constraint |
|---|---|---|
| InventoryID | INT AUTO_INCREMENT | PRIMARY KEY |
| SupplierMaterialID | INT | NOT NULL, FOREIGN KEY |
| ProjectID | INT | NOT NULL, FOREIGN KEY |
| WarehouseID | INT | NOT NULL, FOREIGN KEY |
| QuantityOnHand | DECIMAL(10, 2) | CHECK (QuantityOnHand >= 0) |
| LastUpdated | DATE | NOT NULL |
| ReorderLevel | DECIMAL(10, 2) | CHECK (QuantityOnHand >= 0) |

Order Table (Fact Table)

| Field | Data Type | Constraint |
|---|---|---|
| OrderID | INT AUTO_INCREMENT | PRIMARY KEY |
| SupplierMaterialID | INT | NOT NULL, FOREIGN KEY |
| ProjectID | INT | NOT NULL, FOREIGN KEY |
| OrderDate | DATE | NOT NULL |
| QuantityOnHand | DECIMAL(10, 2) | CHECK (QuantityOnHand >= 0) |
| TotalCost | DECIMAL(10, 2) | CHECK (TotalCost > 0) |
| EmployeeID | INT | NOT NULL, FOREIGN KEY |

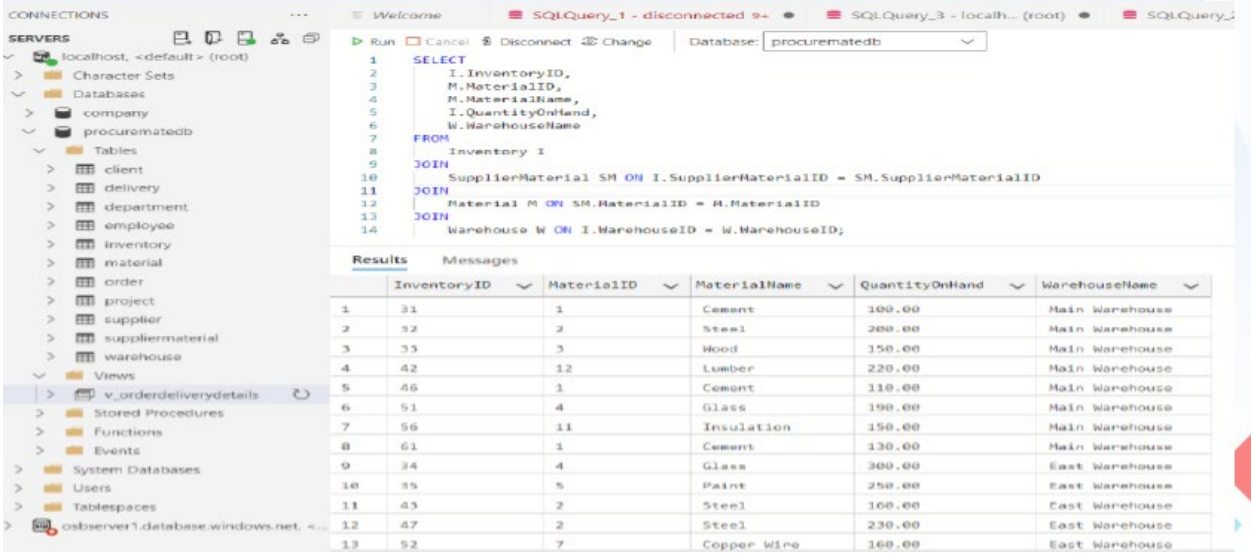
# Queries

Query 1:
Identify Suppliers for Specific Material (ex :steel)

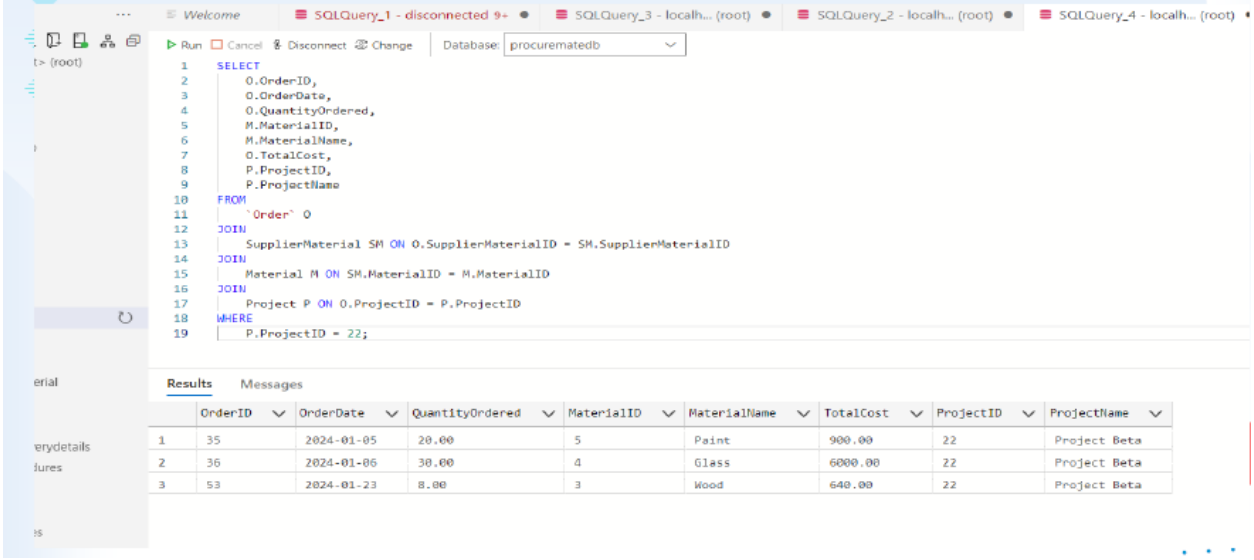



Query 2 /view

provides valuable insights into the inventory status of various materials across different warehouse locations.(facts from Inventory table)

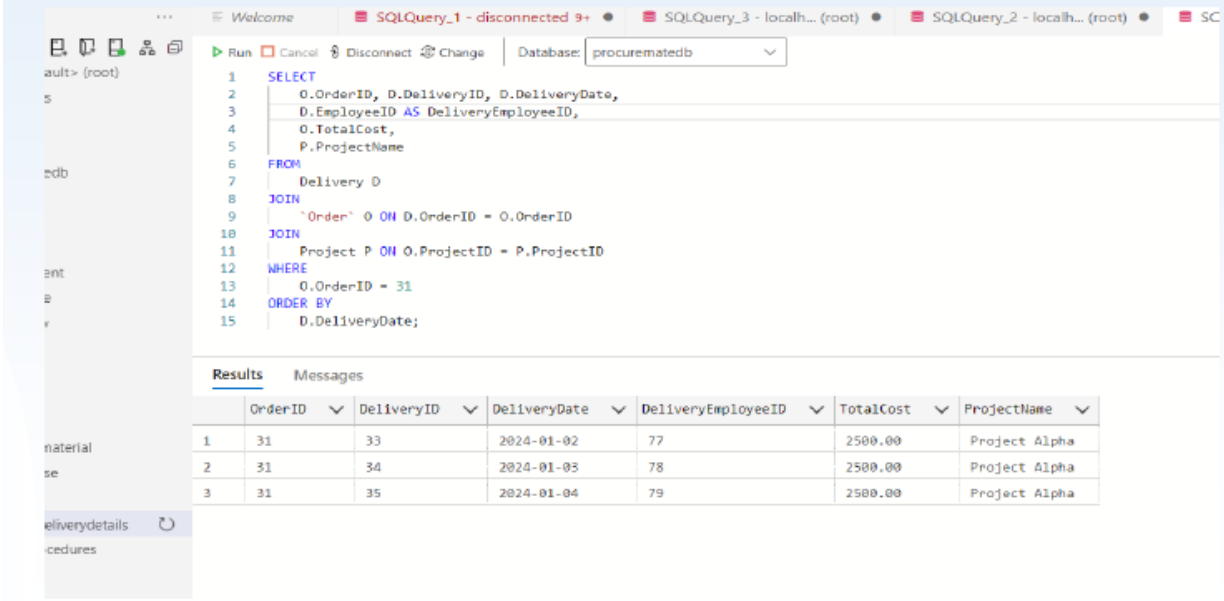


## Query 3:
## materials ordered for a specific project(ex projectid:22)




| | OrderID | OrderDate | QuantityOrdered | MaterialID | MaterialName | TotalCost | ProjectID | ProjectName |
|---|---|---|---|---|---|---|---|---|
| 1 | 35 | 2024-01-05 | 20.00 | 5 | Paint | 900.00 | 22 | Project Beta |
| 2 | 36 | 2024-01-06 | 30.00 | 4 | Glass | 6000.00 | 22 | Project Beta |
| 3 | 53 | 2024-01-23 | 8.00 | 3 | Wood | 640.00 | 22 | Project Beta |

## Query 4:
## tracking all deliveries made for a particular order(ex orderID: 31)




| | OrderID | DeliveryID | DeliveryDate | DeliveryEmployeeID | TotalCost | ProjectName |
|---|---|---|---|---|---|---|
| 1 | 31 | 33 | 2024-01-02 | 77 | 2500.00 | Project Alpha |
| 2 | 31 | 34 | 2024-01-03 | 78 | 2500.00 | Project Alpha |
| 3 | 31 | 35 | 2024-01-04 | 79 | 2500.00 | Project Alpha |