

MINI – PROJECT: SCM

(Supply Chain Management)

In this SCM, they are six modules on this project. There are:

1. Supplier Module
2. Product module
3. Inventory module
4. Warehouse module
5. Order module
6. Transportation module

In this project, I have implemented database normalization to ensure efficient data organization and minimize redundancy. As a result of this approach, I have designed and created 14 distinct tables, each serving a specific purpose to support the project's requirements. The tables are as follows

1. Supplier Module: supplier table.
2. Product Module: Products, industrial_goods, consumer_goods, products_information tables.
3. Inventory Module: inventory table
4. Warehouse Module: warehouse, warehouse_storage tables
5. Order Module: orders, order_details, customer tables
6. Transportation Module: carriers, transport, shipment tables

	name	object_id	principal_id	schema_id	parent_object_id	type	type_desc	create_date	modify_date	is_ms_shipped	is_published	is_sch
1	supplier	1205579333	NULL	1	0	U	USER_TABLE	2024-07-30 23:35:39.330	2024-07-30 23:35:39.443	0	0	0
2	industrial_goods	1269579561	NULL	1	0	U	USER_TABLE	2024-07-30 23:35:39.367	2024-07-30 23:35:39.443	0	0	0
3	consumer_goods	1317579732	NULL	1	0	U	USER_TABLE	2024-07-30 23:35:39.403	2024-07-30 23:35:39.443	0	0	0
4	products	1365579903	NULL	1	0	U	USER_TABLE	2024-07-30 23:35:39.427	2024-07-30 23:35:39.493	0	0	0
5	products_information	1413580074	NULL	1	0	U	USER_TABLE	2024-07-30 23:35:39.430	2024-07-30 23:35:39.430	0	0	0
6	inventory	1509580416	NULL	1	0	U	USER_TABLE	2024-07-30 23:35:39.447	2024-07-30 23:35:39.467	0	0	0
7	warehouse	1589580701	NULL	1	0	U	USER_TABLE	2024-07-30 23:35:39.450	2024-07-30 23:35:39.467	0	0	0
8	warehouse_storage	1621580815	NULL	1	0	U	USER_TABLE	2024-07-30 23:35:39.450	2024-07-30 23:35:39.450	0	0	0
9	customer	1669580986	NULL	1	0	U	USER_TABLE	2024-07-30 23:35:39.467	2024-07-30 23:35:39.490	0	0	0
10	orders	1733581214	NULL	1	0	U	USER_TABLE	2024-07-30 23:35:39.470	2024-07-30 23:35:39.503	0	0	0
11	order_details	1813581499	NULL	1	0	U	USER_TABLE	2024-07-30 23:35:39.490	2024-07-30 23:35:39.490	0	0	0
12	carriers	1861581670	NULL	1	0	U	USER_TABLE	2024-07-30 23:35:39.493	2024-07-30 23:35:39.500	0	0	0
13	transport	1925581898	NULL	1	0	U	USER_TABLE	2024-07-30 23:35:39.500	2024-07-30 23:35:39.503	0	0	0
14	shipment	1973582069	NULL	1	0	U	USER_TABLE	2024-07-30 23:35:39.500	2024-07-30 23:35:39.500	0	0	0

This involves defining the structure and relationships of each table to ensure that the database supports the required functionalities effectively.

1. Supplier

```
-- Create supplier table
CREATE TABLE supplier(
supplierId varchar(25) PRIMARY KEY,
supplierName varchar(50) NOT NULL,
contactPerson varchar(50),
email varchar(50) UNIQUE,
phone bigint UNIQUE
);
```

2. industrial_goods

```
-- Create industrial_goods table
CREATE TABLE industrial_goods(
industryId varchar(25) PRIMARY KEY,
industry varchar(50) UNIQUE,
industry_description varchar(255)
);
```

3. consumer_goods

```
-- Create industrial_goods tableZ
CREATE TABLE industrial_goods(
industryId varchar(25) PRIMARY KEY,
industry varchar(50) UNIQUE,
industry_description varchar(255)
);
```

4. Products

```
-- Create products table
CREATE TABLE products(
productId varchar(25) PRIMARY KEY,
productName varchar(50) NOT NULL,
productDescription varchar(255),
unitPrice FLOAT DEFAULT 0
);
```

5. products_information

```
-- Create product information
CREATE TABLE products_information(
productInfoId varchar(25) PRIMARY KEY,
productId varchar(25),
supplierId varchar(25),
industryId varchar(25),
consumerId varchar(25),
CONSTRAINT FK_productId FOREIGN KEY (productId) REFERENCES products(productId) ON DELETE CASCADE,
CONSTRAINT FK_supplier FOREIGN KEY (supplierId) REFERENCES supplier(supplierId) ON DELETE CASCADE,
CONSTRAINT FK_industrialGoods FOREIGN KEY (industryId) REFERENCES industrial_goods(industryId) ON DELETE CASCADE,
CONSTRAINT FK_consumerGoods FOREIGN KEY (consumerId) REFERENCES consumer_goods(consumerId) ON DELETE CASCADE
);
```

6. Inventory

```
-- Create Inventory table
CREATE TABLE inventory(
inventoryId varchar(25) PRIMARY KEY,
productId varchar(25),
quantityInStock bigint DEFAULT 0,
lastStockUpdate datetime DEFAULT GETDATE(),
CONSTRAINT Fk_inventory_productId FOREIGN KEY (productId) REFERENCES products(productId) ON DELETE CASCADE
);
```

7. Warehouse

```
-- Create warehouse table
CREATE TABLE warehouse(
warehouseId varchar(25) PRIMARY KEY,
warehouseName varchar(30),
location varchar(100),
capacity bigint,
currentCapacity bigint
);
```

8. warehouse_storage

```
-- Create warehouse_storage table
CREATE TABLE warehouse_storage(
warehouseId varchar(25),
inventoryId varchar(25),
CONSTRAINT Fk_warehouseId FOREIGN KEY (warehouseId) REFERENCES warehouse(warehouseId) ON DELETE CASCADE,
CONSTRAINT Fk_inventoryId FOREIGN KEY (inventoryId) REFERENCES inventory(inventoryId) ON DELETE CASCADE
);
```

9. Customer

```
-- Create customer table
CREATE TABLE customer (
customerID VARCHAR(25) PRIMARY KEY,
firstName VARCHAR(50) NOT NULL,
lastName VARCHAR(50) NOT NULL,
email VARCHAR(100) UNIQUE NOT NULL,
phone BIGINT UNIQUE NOT NULL,
address VARCHAR(255),
city VARCHAR(50),
state VARCHAR(50),
zipCode VARCHAR(20)
);
```

10. Orders

```
-- Create orders table
CREATE TABLE orders(
orderId VARCHAR(25) PRIMARY KEY,
customerId VARCHAR(25),
orderDate datetime DEFAULT GETDATE(),
totalAmount BIGINT DEFAULT 0,
orderStatus VARCHAR(25),
CONSTRAINT Fk_customerId FOREIGN KEY (customerId) REFERENCES customer(customerId) ON DELETE CASCADE
);
```

11. order_details

```
-- Create order_details table
CREATE TABLE order_details(
orderId VARCHAR(25),
productId VARCHAR(25),
quantity INT,
CONSTRAINT Fk_orderId FOREIGN KEY (orderId) REFERENCES orders(orderId) ON DELETE CASCADE,
CONSTRAINT Fk_order_productId FOREIGN KEY (productId) REFERENCES products(productId) ON DELETE CASCADE
);
```

12. Carriers

```
-- Create carriers table
CREATE TABLE carriers (
carrierID VARCHAR(25) PRIMARY KEY,
carrierName VARCHAR(255) NOT NULL,
contactPerson VARCHAR(100),
contactEmail VARCHAR(100) UNIQUE,
contactPhone VARCHAR(20) UNIQUE
);
```

13. Transport

```
CREATE TABLE transport (
shipmentId VARCHAR(25) PRIMARY KEY,
carrierID VARCHAR(25),
shipmentStatus VARCHAR(15),
CONSTRAINT Fk_carriedId FOREIGN KEY (carrierID) REFERENCES carriers(carrierID) ON DELETE CASCADE
);
```

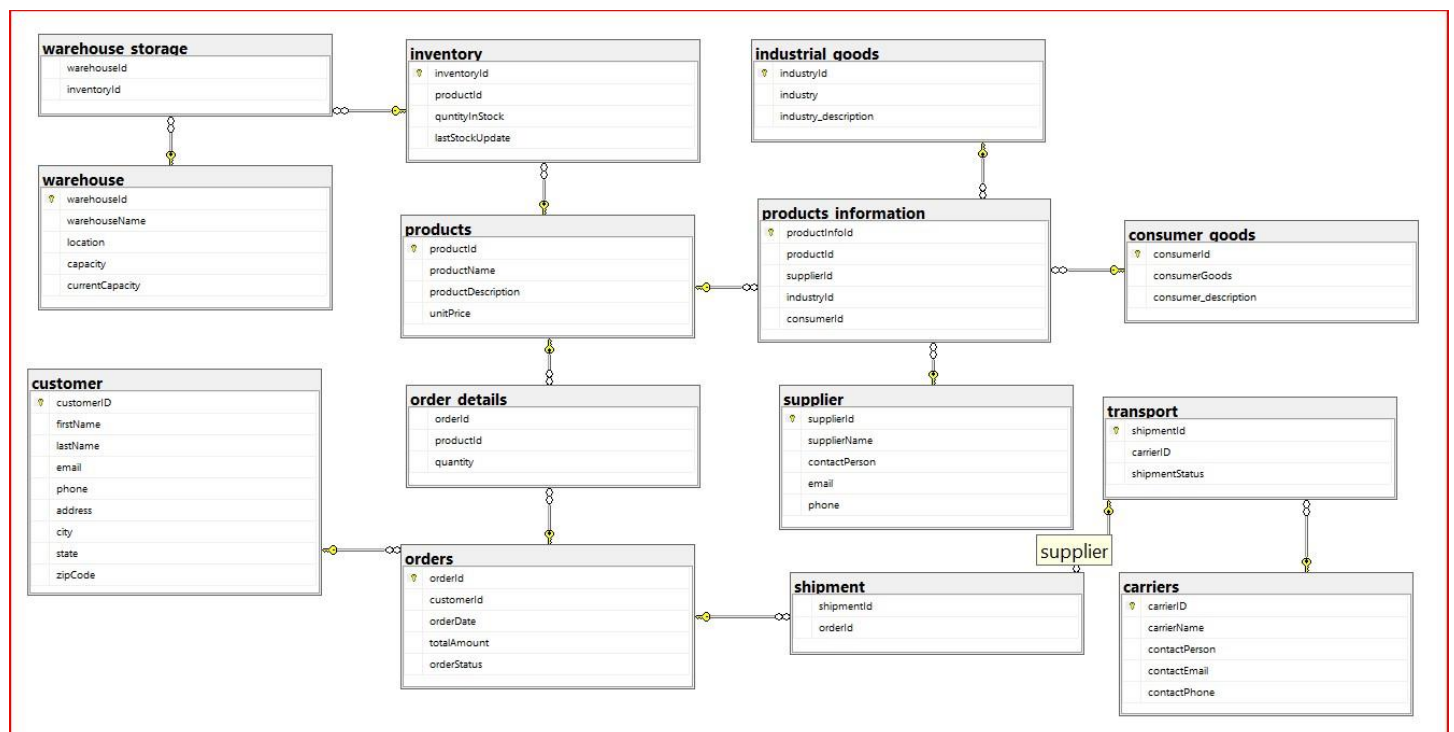
14. Shipment

```
CREATE TABLE shipment (
shipmentId VARCHAR(25),
orderId VARCHAR(25),
CONSTRAINT Fk_shipmentId FOREIGN KEY (shipmentId) REFERENCES transport(shipmentId) ON DELETE CASCADE,
CONSTRAINT Fk_shipment_orderId FOREIGN KEY (orderId) REFERENCES orders(orderId) ON DELETE CASCADE
);
```

This includes defining each table's schema, such as columns, data types, and constraints, as well as setting up relationships between the tables to maintain data integrity and support the project's needs.

ER – Diagram:

The Entity-Relationship Diagram (ERD) for the project provides a visual representation of the database structure. It outlines the entities involved, their attributes, and the relationships between them. This diagram is essential for understanding the organization of the database and how different data elements interact with each other.



The various functionality involved in the project are:

1. Supplier:

→ Inserting records:


```
-- Inserting values into the supplier table
INSERT INTO supplier (supplierId, supplierName, contactPerson, email, phone) VALUES
('SUP1234567890', 'Acme Supplies', 'Alice Johnson', 'alice.johnson@acmesupplies.com', 9876543210),
('SUP1234567891', 'Global Corp', 'Bob Smith', 'bob.smith@globalcorp.com', 9876543211),
('SUP1234567892', 'Tech Gadgets Inc.', 'Charlie Brown', 'charlie.brown@techgadgets.com', 9876543212);
```

→ Retrieve all records from a table:

```
-- Select all details for all suppliers
SELECT * FROM supplier;
```

	supplierId	supplierName	contactPerson	email	phone
1	SUP1234567890	Acme Supplies	Alice Johnson	alice.johnson@acmesupplies.com	9876543210
2	SUP1234567891	Global Corp	Bob Smith	bob.smith@globalcorp.com	9876543211
3	SUP1234567892	Tech Gadgets Inc.	Charlie Brown	charlie.brown@techgadgets.com	9876543212

→ Filter records based on a condition:

```
-- Select details for a specific supplier
SELECT * FROM supplier WHERE supplierId = 'SUP1234567890';
```

	supplierId	supplierName	contactPerson	email	phone
1	SUP1234567890	Acme Supplies	Alice Johnson	alice.johnson@acmesupplies.com	9876543210

```
SELECT * FROM supplier WHERE supplierName LIKE '%a%';
```

	supplierId	supplierName	contactPerson	email	phone
1	SUP1234567890	Acme Supplies	Alice Johnson	alice.johnson@acmesupplies.com	9876543210
2	SUP1234567891	Global Corp	Bob Smith	bob.smith@globalcorp.com	9876543211
3	SUP1234567892	Tech Gadgets Inc.	Charlie Brown	charlie.brown@techgadgets.com	9876543212

→ Update Record

Query:

UPDATE supplier

SET

supplierName = 'Sanjeev pvt',

contactPerson = 'Jay',

email = 'sanjeev@gmail.com',

phone = 1234567890

WHERE supplierId = 'SUP1234567890';

Output:

<pre>-- Select all details for all suppliers SELECT * FROM supplier;</pre>					
4 %					
Results	Messages				
	supplierId	supplierName	contactPerson	email	phone
1	SUP1234567890	Sanjeev pvt	Jay	sanjeev@gmail.com	1234567890
2	SUP1234567891	Global Corp	Bob Smith	bob.smith@globalcorp.com	9876543211
3	SUP1234567892	Tech Gadgets Inc.	Charlie Brown	charlie.brown@techgadgets.com	9876543212

→ Delete record with condition:

-- Deleting a supplier using supplierId

DELETE FROM supplier

WHERE supplierId = 'SUP1234567890';

Output:

<pre>-- Deleting a supplier using supplierId DELETE FROM supplier WHERE supplierId = 'SUP1234567890';</pre>	
4 %	
Messages	
	(1 row affected)
	Completion time: 2024-07-31T14:57:14.6344420+05:30

→ Order by supplier name:

<pre>-- Select all suppliers sorted by supplierName SELECT * FROM supplier ORDER BY supplierName ASC;</pre>					
74 %					
Results	Messages				
	supplierId	supplierName	contactPerson	email	phone
1	SUP1234567891	Global Corp	Bob Smith	bob.smith@globalcorp.com	9876543211
2	SUP1234567892	Tech Gadgets Inc.	Charlie Brown	charlie.brown@techgadgets.com	9876543212

2. Product:

→ Inserting records: Inserting records in Product, Industrial, Consumer tables

```

-- Product Module --
-- Insert a new product into the products table
INSERT INTO products (productId, productName, productDescription, unitPrice)
VALUES ('PROD1234567890', 'Advanced Widget', 'High-tech widget for advanced applications', 99.99);
-- Insert data into consumer_goods table
INSERT INTO consumer_goods (consumerId, consumerGoods, consumer_description)
VALUES ('CONSUM1234567890', 'Consumer Product A', 'Description of Consumer Product A');
-- Insert data into industrial_goods table
INSERT INTO industrial_goods (industryId, industry, industry_description)
VALUES ('IND1234567890', 'High-Tech Electronics', 'High-tech electronics for industrial applications.');
```

4 %

Messages

(1 row affected)

(1 row affected)

(1 row affected)

Completion time: 2024-07-31T15:05:00.6229612+05:30

Inserting records from Product Information table:

```

-- Insert product information for an industrial good
INSERT INTO products_information (productInfoId, productId, supplierId, industryId, consumerId)
VALUES ('PROINFO1234567890', 'PROD1234567890', 'SUP1234567890', 'IND1234567890', NULL);
-- Insert product information for a consumer good
INSERT INTO products_information (productInfoId, productId, supplierId, industryId, consumerId)
VALUES ('PROINFO1234567891', 'PROD1234567890', 'SUP1234567890', NULL, 'CONSUM1234567890');
```

%

Messages

(1 row affected)

(1 row affected)

Completion time: 2024-07-31T15:07:38.2759158+05:30

→ Retrieve all records from table

```

-- select all values
select * from products_information;
select * from products;
select * from industrial_goods;
select * from consumer_goods;
```

74 %

Results Messages

	productInfoId	productId	supplierId	industryId	consumerId
1	PROINFO1234567890	PROD1234567890	SUP1234567890	IND1234567890	NULL
2	PROINFO1234567891	PROD1234567890	SUP1234567890	NULL	CONSUM1234567890

	productId	productName	productDescription	unitPrice
1	PROD1234567890	Advanced Widget	High-tech widget for advanced applications	99.99

	industryId	industry	industry_description
1	IND1234567890	High-Tech Electronics	High-tech electronics for industrial applications.

	consumerId	consumerGoods	consumer_description
1	CONSUM1234567890	Consumer Product A	Description of Consumer Product A

→ Delete specific record

```
-- Delete a specific record by productInfoId
DELETE FROM products_information
WHERE productInfoId = 'PROINFO1234567890';
```

74 %

Messages

(1 row affected)

Completion time: 2024-07-31T15:12:21.4423009+05:30

→ Update specific record

```
-- Update records based on supplierId and productId
UPDATE products_information
SET
    industryId = NULL,
    consumerId = 'CONSUM1234567890'
WHERE supplierId = 'SUP1234567890' AND productId = 'PROD1234567890';
```

74 %

Messages

(1 row affected)

Completion time: 2024-07-31T15:15:34.8724067+05:30

→ Retrieve all values using join

```
-- View detailed product information
SELECT
    pi.productInfoId,
    p.productId,
    p.productName,
    p.productDescription,
    p.unitPrice,
    s.supplierId,
    s.supplierName,
    i.industryId,
    i.industry AS industryName,
    i.industry_description,
    c.consumerId,
    c.consumerGoods AS consumerGoodsName,
    c.consumer_description
FROM products_information pi
JOIN products p ON pi.productId = p.productId
LEFT JOIN supplier s ON pi.supplierId = s.supplierId
LEFT JOIN industrial_goods i ON pi.industryId = i.industryId
LEFT JOIN consumer_goods c ON pi.consumerId = c.consumerId;
```

Output:

	productInfoId	productId	productName	productDescription	unitPrice	supplierId	supplierName	industryId	industryName	industry_desc
1	PROINFO1234567890	PROD1234567890	Advanced Widget	High-tech widget for advanced applications	99.99	SUP1234567890	Acme Supplies	NULL	NULL	NULL

→ Retrieve record specific condition


```
-- Search product information by productInfoId
SELECT
    pi.productInfoId,
    p.productId,
    p.productName,
    p.productDescription,
    p.unitPrice,
    s.supplierId,
    s.supplierName,
    i.industryId,
    i.industry AS industryName,
    i.industry_description,
    c.consumerId,
    c.consumerGoods AS consumerGoodsName,
    c.consumer_description
FROM products_information pi
JOIN products p ON pi.productId = p.productId
LEFT JOIN supplier s ON pi.supplierId = s.supplierId
LEFT JOIN industrial_goods i ON pi.industryId = i.industryId
LEFT JOIN consumer_goods c ON pi.consumerId = c.consumerId
WHERE pi.productInfoId = 'PROINFO1234567891';
```

productInfoId	productId	productName	productDescription	unitPrice	supplierId	supplierName	industryId	industryName	industry_desc
PROINFO1234567891	PROD1234567890	Advanced Widget	High-tech widget for advanced applications	99.99	SUP1234567890	Acme Supplies	NULL	NULL	NULL

Specific productInfoId using retrieve record

```
-- Search product information by productName
SELECT
    pi.productInfoId,
    p.productId,
    p.productName,
    p.productDescription,
    p.unitPrice,
    s.supplierId,
    s.supplierName,
    i.industryId,
    i.industry AS industryName,
    i.industry_description,
    c.consumerId,
    c.consumerGoods AS consumerGoodsName,
    c.consumer_description
FROM products_information pi
JOIN products p ON pi.productId = p.productId
LEFT JOIN supplier s ON pi.supplierId = s.supplierId
LEFT JOIN industrial_goods i ON pi.industryId = i.industryId
LEFT JOIN consumer_goods c ON pi.consumerId = c.consumerId
WHERE p.productName LIKE '%' + 'Advanced' + '%';
```

productInfoId	productId	productName	productDescription	unitPrice	supplierId	supplierName	industryId	industryName	industry_desc
PROINFO1234567891	PROD1234567890	Advanced Widget	High-tech widget for advanced applications	99.99	SUP1234567890	Acme Supplies	NULL	NULL	NULL

Specific producName using retrieve record

→ Counting product information table records

```
SELECT COUNT(*) AS TOTAL_PRODUCTS FROM products_information;
```

TOTAL_PRODUCTS
2

3. Inventory:

→ Inserting records

```
-- Insert inventory data for the products
INSERT INTO inventory (inventoryId, productId, quantityInStock, lastStockUpdate) VALUES
('INV1234567892', 'PROD1234567890', 56, GETDATE()),
('INV1234567890', 'PROD1234567892', 30, GETDATE()),
('INV1234567891', 'PROD1234567893', 20, GETDATE());
```

Messages

(3 rows affected)

Completion time: 2024-07-31T17:01:46.3614221+05:30

→ Retrieve the all records from table

```
SELECT * FROM inventory;
```

74 %

Results Messages

	inventoryId	productId	quantityInStock	lastStockUpdate
1	INV1234567890	PROD1234567892	30	2024-07-31 04:31:46.547
2	INV1234567891	PROD1234567893	20	2024-07-31 04:31:46.547
3	INV1234567892	PROD1234567890	56	2024-07-31 04:31:46.547

→ Retrieving the records from specific condition

```
SELECT * FROM inventory WHERE inventoryId = 'INV1234567890';
```

74 %

Results Messages

	inventoryId	productId	quantityInStock	lastStockUpdate
1	INV1234567890	PROD1234567892	30	2024-07-31 04:31:46.547

-- Specific InventoryId

```
-- Search inventory records by productId
SELECT *
FROM inventory
WHERE productId = 'PROD1234567890';
```

74 %

Results Messages

	inventoryId	productId	quantityInStock	lastStockUpdate
1	INV1234567892	PROD1234567890	56	2024-07-31 04:31:46.547

-- Specific ProductId

→ Update records using specific id

```
-- Update inventory record
UPDATE inventory
SET quantityInStock = 39,
    lastStockUpdate = GETDATE()
WHERE inventoryId = 'INV1234567892';
```

74 %

Messages

(1 row affected)

Completion time: 2024-07-31T17:08:39.8505796+05:30

→ Delete record specific id

```
DELETE FROM inventory WHERE inventoryId = 'INV1234567892';
```

74 %

Messages

(1 row affected)

Completion time: 2024-07-31T17:10:31.9329513+05:30

→ Sum of all product quantity stock

```
SELECT SUM(quantityInStock) FROM inventory;
```

74 %

Results Messages

	(No column name)
1	50

4. Warehouse: warehouse table

→ Inserting records from table

```
-- Insert a new warehouse record
-- Insert values into the warehouse table
INSERT INTO warehouse (warehouseId, warehouseName, location, capacity, currentCapacity) VALUES
('WH001', 'Central Warehouse', '123 Main St, Coimbatore', 5000, 0),
('WH002', 'Northside Depot', '456 Elm St, Coimbatore', 3000, 0),
('WH003', 'Southside Storage', '789 Oak St, Coimbatore', 7000, 0),
('WH004', 'Eastside Facility', '321 Pine St, Coimbatore', 2000, 0),
('WH005', 'Westside Hub', '654 Maple St, Coimbatore', 4500, 0);
```

74 %

Messages

(5 rows affected)

Completion time: 2024-07-31T17:16:11.6117768+05:30

→ Retrieve all records from table

```
-- Search for a warehouse record
```

```
SELECT * FROM warehouse
```

	warehouseId	warehouseName	location	capacity	currentCapacity
1	WH001	Central Warehouse	123 Main St, Coimbatore	5000	0
2	WH002	Northside Depot	456 Elm St, Coimbatore	3000	0
3	WH003	Southside Storage	789 Oak St, Coimbatore	7000	0
4	WH004	Eastside Facility	321 Pine St, Coimbatore	2000	0
5	WH005	Westside Hub	654 Maple St, Coimbatore	4500	0

→ Retrieve records specific condition

```
-- Search for a warehouse record by warehouseId
```

```
SELECT * FROM warehouse WHERE warehouseId = 'WH004';
```

	warehouseId	warehouseName	location	capacity	currentCapacity
1	WH004	Eastside Facility	321 Pine St, Coimbatore	2000	0

→ Update records specific condition

```
-- Update a warehouse record
```

```
UPDATE warehouse
```

```
SET warehouseName = 'Eastside Facility',
```

```
    location = '321 Pine St, Coimbatore',
```

```
    currentCapacity = 5000
```

```
WHERE warehouseId = 'WH004';
```

(1 row affected)

Completion time: 2024-07-31T17:20:31.9181999+05:30

→ Delete record specific condition

```
-- Delete a warehouse record by warehouseId
```

```
DELETE FROM warehouse WHERE warehouseId = 'WH005';
```

(1 row affected)

Completion time: 2024-07-31T17:21:27.0904178+05:30

Warehouse_storage

→ Create triggers for when warehouse storage added inventory it's automatically update currentCapacity of warehouse storage


```

-- Trigger to update currentCapacity in warehouse table after inserting into warehouse_storage
GO
CREATE TRIGGER trg_UpdateCurrentCapacityWareHouseStorage
ON warehouse_storage
AFTER INSERT
AS
BEGIN
    -- Update currentCapacity for the warehouse
    UPDATE w
    SET w.currentCapacity = w.currentCapacity + i.quantityInStock
    FROM warehouse w
    JOIN inserted ins ON w.warehouseId = ins.warehouseId
    JOIN inventory i ON ins.inventoryId = i.inventoryId;
END;
GO

GO
CREATE TRIGGER trg_UpdateCurrentCapacity
ON inventory
AFTER UPDATE
AS
BEGIN
    -- Update currentCapacity in the warehouse based on changes in the inventory
    UPDATE w
    SET w.currentCapacity = w.currentCapacity + deltaQuantity
    FROM warehouse w
    INNER JOIN warehouse_storage ws ON w.warehouseId = ws.warehouseId
    INNER JOIN (
        SELECT i.inventoryId,
               i.quantityInStock - d.quantityInStock AS deltaQuantity
        FROM INSERTED i
        INNER JOIN DELETED d ON i.inventoryId = d.inventoryId
    ) AS changes ON changes.inventoryId = ws.inventoryId
    WHERE w.warehouseId = ws.warehouseId;
END;

```

→ Inserting records

```

-- Insert values into the warehouse_storage table
INSERT INTO warehouse_storage (warehouseId, inventoryId) VALUES
('WH001', 'INV1234567892'),
('WH001', 'INV1234567892'),
('WH002', 'INV1234567892'),
('WH003', 'INV1234567891'),
('WH004', 'INV1234567890');

select * from inventory;
select * from warehouse;

```

4 %

Messages

(4 rows affected)

(5 rows affected)

Completion time: 2024-07-31T19:13:26.4553117+05:30

→ Retrieve all records from table

```

-- Search for records in the warehouse_storage table
SELECT * FROM warehouse_storage

```

74 %

Results Messages

	warehouseId	inventoryId
1	WH001	INV1234567892
2	WH001	INV1234567892
3	WH002	INV1234567892
4	WH003	INV1234567891
5	WH004	INV1234567890

→ Retrieve records specific condition

```
-- Search for records in the warehouse_storage table by warehouseId and inventoryId
SELECT * FROM warehouse_storage WHERE warehouseId = 'WH001' AND inventoryId = 'INV1234567892';
```

74 %

Results Messages

	warehouseId	inventoryId
1	WH001	INV1234567892
2	WH001	INV1234567892

→ Retrieve all records from inventory and warehouse

```
select * from inventory;
select * from warehouse;
```

74 %

Results Messages

	inventoryId	productId	quantityInStock	lastStockUpdate
1	INV1234567890	PROD1234567892	30	2024-07-31 04:31:46.547
2	INV1234567891	PROD1234567893	20	2024-07-31 04:31:46.547
3	INV1234567892	PROD1234567890	56	2024-07-31 04:55:26.333

	warehouseId	warehouseName	location	capacity	currentCapacity
1	WH001	Central Warehouse	123 Main St, Coimbatore	5000	56
2	WH002	Northside Depot	456 Elm St, Coimbatore	3000	56
3	WH003	Southside Storage	789 Oak St, Coimbatore	7000	20
4	WH004	Eastside Facility	321 Pine St, Coimbatore	2000	30

→ Delete specific record from table

```
-- Delete a record from the warehouse_storage table
DELETE FROM warehouse_storage
WHERE warehouseId = 'WH004'
AND inventoryId = 'INV1234567890';
```

4 %

Messages

(1 row affected)

Completion time: 2024-07-31T19:23:47.7671556+05:30

5. Order:

Order & Customer table

→ Insert records from customer

```
-- Insert a new customer into the customer table
-- Sample data for the Customer table
INSERT INTO customer VALUES
('CUST1234567890', 'Alice', 'Johnson', 'alice.johnson@example.com', 9876543210, '321 Elm St', 'Somewhere', 'TX', '78901'),
('CUST1234567891', 'John', 'Doe', 'john.doe@example.com', 9874586325, '123 Main St', 'Anytown', 'CA', '12345'),
('CUST1234567892', 'Jane', 'Smith', 'jane.smith@example.com', 9874586328, '456 Oak Ave', 'Somewhere', 'NY', '67890'),
('CUST1234567893', 'Bob', 'Johnson', 'bob.johnson@example.com', 9874586327, '789 Pine Rd', 'Nowhere', 'TX', '54321');
```

4 %

Messages

(4 rows affected)

Completion time: 2024-07-31T19:33:43.4275808+05:30

→ Retrieve all records from customer table

`SELECT * FROM customer;`

74 %

Results Messages

	customerID	firstName	lastName	email	phone	address	city	state	zipCode
1	CUST1234567890	Alice	Johnson	alice.johnson@example.com	9876543210	321 Elm St	Somewhere	TX	78901
2	CUST1234567891	John	Doe	john.doe@example.com	9874586325	123 Main St	Anytown	CA	12345
3	CUST1234567892	Jane	Smith	jane.smith@example.com	9874586328	456 Oak Ave	Somewhere	NY	67890
4	CUST1234567893	Bob	Johnson	bob.johnson@example.com	9874586327	789 Pine Rd	Nowhere	TX	54321

→ Insert order using stored procedure

```

CREATE PROCEDURE AddOrder
    @orderId VARCHAR(25),
    @customerId VARCHAR(25),
    @totalAmount BIGINT = 0,
    @orderStatus VARCHAR(25)
AS
BEGIN
    -- Check if the customer exists before adding the order
    IF NOT EXISTS (SELECT 1 FROM customer WHERE customerID = @customerId)
    BEGIN
        PRINT 'Customer does not exist.';
        RETURN;
    END
    -- Check if the order already exists
    IF EXISTS (SELECT 1 FROM orders WHERE orderId = @orderId)
    BEGIN
        PRINT 'Order already exists.';
        RETURN;
    END
    -- Insert the new order
    INSERT INTO orders (orderId, customerId, orderDate, totalAmount, orderStatus)
    VALUES (@orderId, @customerId, GETDATE(), @totalAmount, @orderStatus);

    PRINT 'Order added successfully.';
END;
GO

```

Output:

```

-- Call AddOrder Procedure
EXEC AddOrder
    @orderId = 'ORD1234567890',
    @customerId = 'CUST1234567890',
    @totalAmount = 1500,
    @orderStatus = 'Processing';

```

67 %

Messages

(1 row affected)
Order added successfully.

Completion time: 2024-07-31T19:50:11.9880214+05:30

→ Update order using the stored procedure

```

-- Create or replace the stored procedure to update order details
GO
CREATE PROCEDURE UpdateOrder
    @orderId VARCHAR(25),
    @customerId VARCHAR(25) = NULL,
    @orderDate DATETIME = NULL,
    @totalAmount BIGINT = 0,
    @orderStatus VARCHAR(25) = NULL
AS
BEGIN
    -- Check if the order exists
    IF NOT EXISTS (SELECT 1 FROM orders WHERE orderId = @orderId)
    BEGIN
        PRINT 'Order does not exist.';
        RETURN;
    END

    -- Update order details
    UPDATE orders
    SET customerId = ISNULL(@customerId, customerId),
        orderDate = ISNULL(@orderDate, orderDate),
        totalAmount = @totalAmount,
        orderStatus = ISNULL(@orderStatus, orderStatus)
    WHERE orderId = @orderId;

    PRINT 'Order updated successfully.';
END;
GO

```

Output:

```

-- Call UpdateOrder Procedure
EXEC UpdateOrder
    @orderId = 'ORD1234567890',
    @customerId = 'CUST1234567891',
    @orderDate = '2024-08-01',
    @totalAmount = 2000,
    @orderStatus = 'Shipped';

```

67 %

Messages

(1 row affected)
Order updated successfully.

Completion time: 2024-07-31T19:51:41.7727221+05:30

→ Retrieve records specific condition

```

-- Create or replace the stored procedure to search for an order by ID
GO
CREATE PROCEDURE SearchOrderByID
    @orderId VARCHAR(25)
AS
BEGIN
    -- Retrieve order details by ID
    SELECT *
    FROM orders
    WHERE orderId = @orderId;
END;
GO

```

Output:

```

-- Call SearchOrderByID Procedure
EXEC SearchOrderByID
    @orderId = 'ORD1234567890';

```

67 %

Results Messages

	orderId	customerId	orderDate	totalAmount	orderStatus
1	ORD1234567890	CUST1234567891	2024-08-01 00:00:00.000	2000	Shipped

Order Details table:

→ Inserting records order details with order, Product available or not

```
/* Order Details Table Operations */
-- Insert order details after checking if the order and product exist
IF EXISTS (SELECT 1 FROM orders WHERE orderId = 'ORD1234567890') AND EXISTS (SELECT 1 FROM products WHERE productId = 'PROD1234567890')
BEGIN
    INSERT INTO order_details (orderId, productId, quantity)
    VALUES ('ORD1234567890', 'PROD1234567890', 10);
    PRINT 'Order details added successfully.';
END
ELSE
BEGIN
    PRINT 'Order or Product does not exist.';
END
```

67 %

Messages

(1 row affected)
Order details added successfully.

Completion time: 2024-07-31T20:23:21.5230922+05:30

→ Update records order details with order, Product available or not

```
-- Update order details after checking if the record exists
IF EXISTS (SELECT 1 FROM order_details WHERE orderId = 'ORD1234567890' AND productId = 'PROD1234567890')
BEGIN
    UPDATE order_details
    SET quantity = 20
    WHERE orderId = 'ORD1234567890' AND productId = 'PROD1234567890';
    PRINT 'Order details updated successfully.';
END
ELSE
BEGIN
    PRINT 'Order details record does not exist.';
END
```

7 %

Messages

(1 row affected)
Order details updated successfully.

Completion time: 2024-07-31T20:24:26.2660786+05:30

→ Retrieve all records from order details table

```
-- Search for order details
SELECT * FROM order_details
```

57 %

Results Messages

	orderId	productId	quantity
1	ORD1234567890	PROD1234567890	20

→ Retrieve specific record from table

```
-- Search for order details by orderId and productId
SELECT * FROM order_details
WHERE orderId = 'ORD1234567890' AND productId = 'PROD1234567890';
```

67 %

Results Messages

	orderId	productId	quantity
1	ORD1234567890	PROD1234567890	20

6. Transportation:

→ Inserting records from carriers details

```
-- Sample data for the carriers table
INSERT INTO carriers VALUES
('CAIR09876567890', 'FedEx', 'John Smith', 'john.smith@fedex.com', '+91 9856475632'),
('CAIR09876567891', 'UPS', 'Jane Doe', 'jane.doe@ups.com', '+91 9856475631'),
('CAIR09876567892', 'DHL', 'Bob Johnson', 'bob.johnson@dhl.com', '+91 9856475633');
```

57 %

Messages

(3 rows affected)

Completion time: 2024-07-31T20:31:38.9864361+05:30

→ Retrieve all carrier records from table

```
SELECT * FROM carriers;
```

67 %

Results Messages

	carrierID	carrierName	contactPerson	contactEmail	contactPhone
1	CAIR09876567890	FedEx	John Smith	john.smith@fedex.com	+91 9856475632
2	CAIR09876567891	UPS	Jane Doe	jane.doe@ups.com	+91 9856475631
3	CAIR09876567892	DHL	Bob Johnson	bob.johnson@dhl.com	+91 9856475633

→ Insert record from transport table

```
-- Insert a new record into the transport table
INSERT INTO transport (shipmentId, carrierID, shipmentStatus)
VALUES ('SH1234567890', 'CAIR09876567890', 'In Transit');
```

74 %

Messages

(1 row affected)

Completion time: 2024-07-31T20:36:59.8906518+05:30

→ Insert record from shipment table

```
-- Insert a new record into the shipment table
INSERT INTO shipment (shipmentId, orderId)
VALUES ('SH1234567890', 'ORD1234567890');
```

74 %

Messages

(1 row affected)

Completion time: 2024-07-31T20:39:30.5781346+05:30

→ Retrieve records with condition or without condition using stored procedure

```

GO
CREATE PROCEDURE GetTransportationDetails
    @shipmentId VARCHAR(25) = NULL,
    @orderId VARCHAR(25) = NULL
AS
BEGIN
    -- Retrieve transportation details along with related carrier and shipment information
    SELECT t.shipmentId,
           t.carrierID,
           t.shipmentStatus,
           c.carrierName,
           c.contactPerson,
           c.contactEmail,
           c.contactPhone,
           s.orderId,
           o.orderDate,
           o.totalAmount,
           o.orderStatus
    FROM transport t
    LEFT JOIN carriers c ON t.carrierID = c.carrierID
    LEFT JOIN shipment s ON t.shipmentId = s.shipmentId
    LEFT JOIN orders o ON s.orderId = o.orderId
    WHERE (@shipmentId IS NULL OR t.shipmentId = @shipmentId)
           AND (@orderId IS NULL OR s.orderId = @orderId);
END;

-- Get transportation details for a specific shipmentId

```

74 %

Messages

Commands completed successfully.

Completion time: 2024-07-31T20:41:16.7837377+05:30

Output:

```

-- Get transportation details for a specific shipmentId
EXEC GetTransportationDetails @shipmentId = 'SH1234567890';

```

74 %

Results Messages

	shipmentId	carrierID	shipmentStatus	carrierName	contactPerson	contactEmail	contactPhone	orderId	orderDate	totalAmount	orderStatus
1	SH1234567890	CAIR09876567890	In Transit	FedEx	John Smith	john.smith@fedex.com	+91 9856475632	ORD1234567890	2024-08-01 00:00:00.000	2000	Shipped
2	SH1234567890	CAIR09876567890	In Transit	FedEx	John Smith	john.smith@fedex.com	+91 9856475632	ORD1234567890	2024-08-01 00:00:00.000	2000	Shipped

→ Retrieve all records from order, order details, carrier, transport tables using join

```

-- Retrieve all transport details along with related carrier and shipment information
SELECT t.shipmentId,
       t.carrierID,
       t.shipmentStatus,
       c.carrierName,
       c.contactPerson,
       c.contactEmail,
       c.contactPhone,
       s.orderId,
       s.shipmentId AS shipmentInShipmentTable,
       o.orderDate,
       o.totalAmount,
       o.orderStatus
FROM transport t
JOIN carriers c ON t.carrierID = c.carrierID
JOIN shipment s ON t.shipmentId = s.shipmentId
JOIN orders o ON s.orderId = o.orderId;

-- Stored procedure, retrieve all details

```

4 %

Results Messages

	shipmentId	carrierID	shipmentStatus	carrierName	contactPerson	contactEmail	contactPhone	orderId	shipmentInShipmentTable	orderDate
1	SH1234567890	CAIR09876567890	In Transit	FedEx	John Smith	john.smith@fedex.com	+91 9856475632	ORD1234567890	SH1234567890	2024-08-01 00:00:00.000
2	SH1234567890	CAIR09876567890	In Transit	FedEx	John Smith	john.smith@fedex.com	+91 9856475632	ORD1234567890	SH1234567890	2024-08-01 00:00:00.000

→ Manage transportation using the stored procedure

```

GO
CREATE PROCEDURE ManageTransportationDetails
    @shipmentId VARCHAR(25) = NULL,
    @orderId VARCHAR(25) = NULL,
    @newShipmentStatus VARCHAR(15) = NULL,
    @newCarrierID VARCHAR(25) = NULL
AS
BEGIN
    -- Retrieve transportation details if parameters are provided
    IF @shipmentId IS NOT NULL OR @orderId IS NOT NULL
    BEGIN
        SELECT t.shipmentId,
            t.carrierID,
            t.shipmentStatus,
            c.carrierName,
            c.contactPerson,
            c.contactEmail,
            c.contactPhone,
            s.orderId,
            o.orderDate,
            o.totalAmount,
            o.orderStatus
        FROM transport t
        LEFT JOIN carriers c ON t.carrierID = c.carrierID
        LEFT JOIN shipment s ON t.shipmentId = s.shipmentId
        LEFT JOIN orders o ON s.orderId = o.orderId
        WHERE (@shipmentId IS NULL OR t.shipmentId = @shipmentId)
            AND (@orderId IS NULL OR s.orderId = @orderId);
    END
END

```

Output: Retrieve specific record

```

-- Retrieve details for a specific shipmentId
EXEC ManageTransportationDetails @shipmentId = 'SH1234567890';

```

61 %

Results Messages

	shipmentId	carrierID	shipmentStatus	carrierName	contactPerson	contactEmail	contactPhone	orderId	orderDate	totalAmount	orderStatus
1	SH1234567890	CAIR09876567890	In Transit	FedEx	John Smith	john.smith@fedex.com	+91 9856475632	ORD1234567890	2024-08-01 00:00:00.000	2000	Shipped
2	SH1234567890	CAIR09876567890	In Transit	FedEx	John Smith	john.smith@fedex.com	+91 9856475632	ORD1234567890	2024-08-01 00:00:00.000	2000	Shipped

→ Update and retrieve records using Stored procedure with case

Query:

GO

CREATE PROCEDURE ManageTransportationDetailsCase

@shipmentId VARCHAR(25) = NULL,

@orderId VARCHAR(25) = NULL,

@newShipmentStatus VARCHAR(15) = NULL,

@newCarrierID VARCHAR(25) = NULL

AS

BEGIN

-- Retrieve transportation details if parameters are provided

IF @shipmentId IS NOT NULL OR @orderId IS NOT NULL

BEGIN

SELECT t.shipmentId,

t.carrierID,

t.shipmentStatus,

c.carrierName,

c.contactPerson,

c.contactEmail,

c.contactPhone,

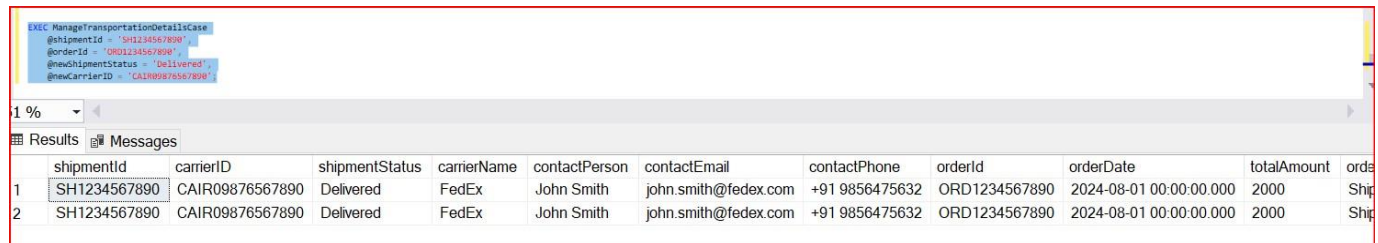
s.orderId,


```

        o.orderDate,
        o.totalAmount,
        o.orderStatus
FROM transport t
LEFT JOIN carriers c ON t.carrierID = c.carrierID
LEFT JOIN shipment s ON t.shipmentId = s.shipmentId
LEFT JOIN orders o ON s.orderId = o.orderId
WHERE (@shipmentId IS NULL OR t.shipmentId = @shipmentId)
      AND (@orderId IS NULL OR s.orderId = @orderId);
END

-- Update transportation details if new values are provided
IF @newShipmentStatus IS NOT NULL OR @newCarrierID IS NOT NULL
BEGIN
    UPDATE t
    SET t.shipmentStatus =
        CASE
            WHEN @newShipmentStatus IS NOT NULL THEN @newShipmentStatus
            ELSE t.shipmentStatus
        END,
        t.carrierID =
        CASE
            WHEN @newCarrierID IS NOT NULL THEN @newCarrierID
            ELSE t.carrierID
        END
    FROM transport t
    INNER JOIN shipment s ON t.shipmentId = s.shipmentId
    WHERE (@shipmentId IS NOT NULL AND t.shipmentId = @shipmentId)
          AND (@orderId IS NOT NULL AND s.orderId = @orderId);
END
END;
```

Output:



The screenshot shows a SQL Server Enterprise Manager interface. At the top, a query window displays the execution of a stored procedure: `EXEC ManageTransportationDetailsCase @shipmentId = 'SH1234567890', @orderId = 'ORD1234567890', @newShipmentStatus = 'Delivered', @newCarrierID = 'CAIR09876567890'.` Below the query window, the 'Results' tab is active, showing a table with 12 columns: `shipmentId`, `carrierID`, `shipmentStatus`, `carrierName`, `contactPerson`, `contactEmail`, `contactPhone`, `orderId`, `orderDate`, `totalAmount`, and `orderId`. The table contains two rows of data, both representing the same shipment and order.

	shipmentId	carrierID	shipmentStatus	carrierName	contactPerson	contactEmail	contactPhone	orderId	orderDate	totalAmount	orderId
1	SH1234567890	CAIR09876567890	Delivered	FedEx	John Smith	john.smith@fedex.com	+91 9856475632	ORD1234567890	2024-08-01 00:00:00.000	2000	Shp
2	SH1234567890	CAIR09876567890	Delivered	FedEx	John Smith	john.smith@fedex.com	+91 9856475632	ORD1234567890	2024-08-01 00:00:00.000	2000	Shp

Use Case Specifications:

This will outline various functionalities for the system involving suppliers, products, inventory, warehouses, customers, orders, and carriers.

1. Use Case: Add New Supplier

Description:

Allows the user to add a new supplier to the system.

Actors:

Admin/Inventory Manager

Preconditions:

- User is logged in.
- All required fields are provided.

Postconditions:

- A new supplier record is created in the supplier table.

Basic Flow:

User navigates to the "Add Supplier" section.

User enters supplier details (ID, name, contact person, email, and phone).

System validates the input.

System inserts the new supplier into the supplier table.

System confirms successful addition.

Alternative Flow:

- **Invalid Email/Phone:** If the email or phone already exists, the system prompts the user to enter a unique value.

2. Use Case: Add New Product

Description:

Allows the user to add a new product to the inventory.

Actors:

Admin/Inventory Manager

Preconditions:

- User is logged in.
- All required fields are provided.

Postconditions:

- A new product record is created in the products table.

Basic Flow:

User navigates to the "Add Product" section.

User enters product details (ID, name, description, unit price).

System validates the input.

System inserts the new product into the products table.

System confirms successful addition.

Alternative Flow:

- **Duplicate Product Name:** If a product with the same name already exists, the system prompts the user to enter a unique name.

3. Use Case: Update Inventory**Description:**

Updates the quantity in stock for a product.

Actors:

Inventory Manager

Preconditions:

- User is logged in.
- The product and inventory record exist.

Postconditions:

- The inventory table is updated with new stock quantities.

Basic Flow:

User navigates to the "Update Inventory" section.

User selects the product and enters the new quantity in stock.

System validates the input.

System updates the inventory table with the new quantity.

System confirms successful update.

Alternative Flow:

- **Invalid Quantity:** If the quantity is negative, the system prompts the user to enter a valid number.

4. Use Case: Place an Order

Description:

Allows a customer to place an order for products.

Actors:

Customer

Preconditions:

- Customer is logged in.
- Products are available in the inventory.

Postconditions:

- An order record is created in the orders table.
- Order details are recorded in the order_details table.

Basic Flow:

Customer navigates to the "Place Order" section.

Customer selects products and specifies quantities.

System validates the availability of products.

System creates an order record in the orders table.

System creates entries in the order_details table for each product.

System confirms successful order placement.

Alternative Flow:

- **Out of Stock:** If any product is out of stock, the system prompts the customer to adjust quantities.

5. Use Case: Assign Warehouse to Inventory

Description:

Assigns a specific warehouse to a product inventory for storage purposes.

Actors:

Warehouse Manager

Preconditions:

- User is logged in.
- The warehouse and inventory records exist.

Postconditions:

- An entry is created in the warehouse_storage table.

Basic Flow:

User navigates to the "Assign Warehouse" section.

User selects a warehouse and inventory item.

System validates the input.

System creates an entry in the warehouse_storage table.

System confirms successful assignment.

Alternative Flow:

- **Warehouse Capacity Exceeded:** If assigning the inventory would exceed warehouse capacity, the system prompts the user to select a different warehouse.

6. Use Case: Manage Shipments**Description:**

Manages the shipment of orders through carriers.

Actors:

Shipping Coordinator

Preconditions:

- User is logged in.
- Orders and carriers are available.

Postconditions:

- Shipment records are created and linked to orders and carriers.

Basic Flow:

User navigates to the "Manage Shipments" section.

User selects an order and a carrier.

System validates the input.

System creates a shipment record in the transport table.

System links the shipment to the order in the shipment table.

System confirms successful shipment creation.

Alternative Flow:

- **Carrier Availability:** If the selected carrier is unavailable, the system prompts the user to choose a different carrier.