Report

1. Purpose of the Project

The core purpose of this project is to design and develop a comprehensive Supply Chain Management (SCM) database system that efficiently manages the entire flow of goods, information, and financial transactions among key stakeholders: suppliers, manufacturers, warehouses, logistics providers, and customers.

The system's primary objective is to enable real-time tracking of products from the initial source (supplier/manufacturer) to the final destination (customer). By providing an integrated database solution, the project aims to streamline critical business processes such as procurement, manufacturing, warehousing, shipments, and customer order management.

Ultimately, this system will empower businesses to make data-driven decisions, reduce operational costs, optimize inventory levels, mitigate delays, and significantly improve overall transparency and customer satisfaction within the supply chain.

2. Scope of the Project

The project encompasses the creation of a centralized relational database that manages ten core entities: Users, Supplier, Manufacturer, Product, Warehouse, Customer, Shipment, Payment, Orders, and Role_Permissions.

The scope includes:

- Implementing and supporting CRUD (Create, Read, Update, Delete) operations for all primary entities.
- Maintaining data integrity through established relationships and constraints (Primary Keys, Foreign Keys).
- The ability to generate real-time reports for key performance indicators (KPIs).

The system is designed to be scalable, allowing for the future expansion of suppliers, products, and customers without compromising system performance.

3. Detailed Description (Entity Model)

The SCM system is built around a set of interconnected tables, each representing a core entity in the supply chain.

Entity	Purpose	Key Attributes
Users	Manages access to the SCM system. Each user has a specific role that dictates their permissions.	user_id (PK), username, password_hash, role
Supplier	Represents vendors providing raw materials or components. Tracks contact information and performance ratings.	supplier_id (PK), supplier_name, contact_person, city, rating
Manufacturer	Represents entities that assemble or produce the final products. Stores details and production capabilities.	manufacturer_id (PK), manufacturer_name, production_capacity, license_number
Warehouse	Manages storage facilities. Tracks location, storage capacity, and current utilization to optimize inventory.	warehouse_id (PK), warehouse_name, city, capacity, current_utilization
Customer	Represents the end clients (individuals, retailers, distributors). Crucial for managing orders and sales data.	customer_id (PK), customer_name, customer_type, city
Product	The central entity, representing the goods being managed. Linked to its supplier and	product_id (PK), product_name, unit_price, quantity_available, supplier_id (FK),

	manufacturer.	manufacturer_id (FK)
Shipment	Tracks the logistics of moving products from a warehouse to a customer. Includes carrier, cost, and delivery status.	shipment_id (PK), carrier_name, status, warehouse_id (FK)
Payment	Stores information about financial transactions related to orders.	payment_id (PK), payment_mode, status
Orders	Primary transactional entity connecting a customer with products, shipment, and payment.	order_id (PK), total_amount, ordered_item, customer_id (FK), shipment_id (FK), payment_id (FK)
Role_Permissions	Crucial for implementing Role-Based Access Control (RBAC). Defines what actions each user role can perform.	role, entity_name, can_view, can_edit, can_delete

4. Functional Requirements

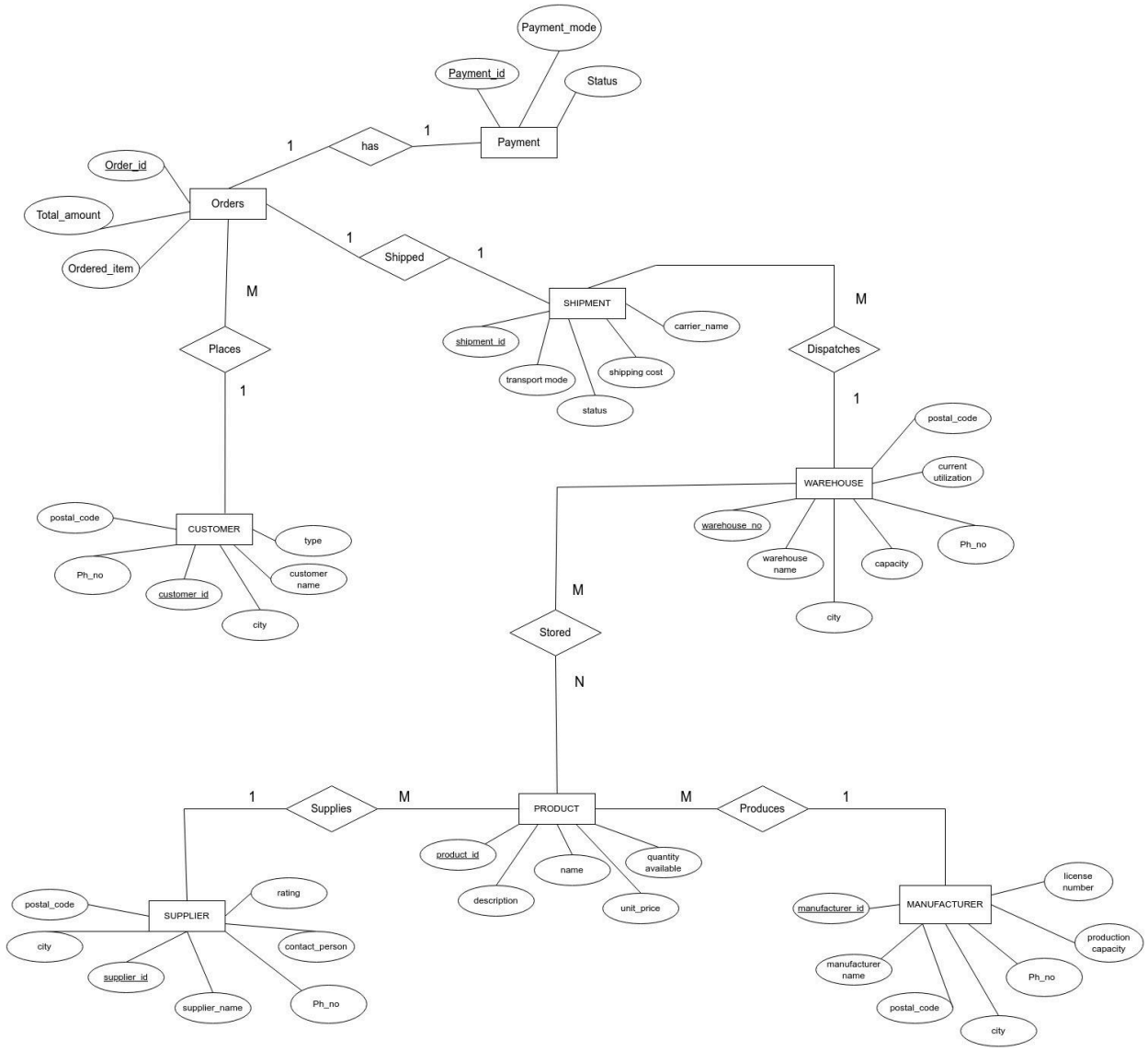
Functional Area	Requirement Description
User & Access Management	Admins must be able to create, update, and delete user accounts. Admins assign specific roles (e.g., 'Supplier', 'Warehouse'). System restricts actions based on Role_Permissions.
Supplier & Product Management	Ability to add new suppliers and products. quantity_available must update automatically.
Order & Fulfillment Processing	Customers/Sales staff place new orders linked to customers, shipments, and payments. Status updates allowed by authorized users.
Logistics & Warehouse Management	Track shipments from 'Pending' to 'Delivered'. Report on current_utilization vs. capacity of warehouses.
Reporting & Analytics	Generate reports on total orders, sales, stock levels, and supplier ratings.

5. List of Software Used

Category	Technology/Tool
Frontend	React.js
Backend	Node.js
Database	MySQL
Deployment & Hosting	Vercel
Version Control	Git

6. Data Model

6.1 Entity-Relationship Diagram (ERD)



6.2 Relational Schema

The schema defines the tables, primary keys (PK), and foreign keys (FK) for the SCM database.

- **CUSTOMER** (customer_id PK, customer_name, type, city, postal_code, ph_no)
- **ORDERS** (order_id PK, total_amount, ordered_item, customer_id FK, payment_id FK, shipment_id FK)
- **PAYMENT** (payment_id PK, payment_mode, status)
- **SHIPMENT** (shipment_id PK, carrier_name, transport_mode, shipping_cost, status, warehouse_no FK)
- **WAREHOUSE** (warehouse_no PK, warehouse_name, capacity, current_utilization, city, postal_code, ph_no)
- **PRODUCT** (product_id PK, name, description, quantity_available, unit_price, supplier_id FK, manufacturer_id FK)
- **SUPPLIER** (supplier_id PK, supplier_name, contact_person, rating, city, postal_code, ph_no)
- **MANUFACTURER** (manufacturer_id PK, manufacturer_name, license_number, production_capacity, city, postal_code, ph_no)

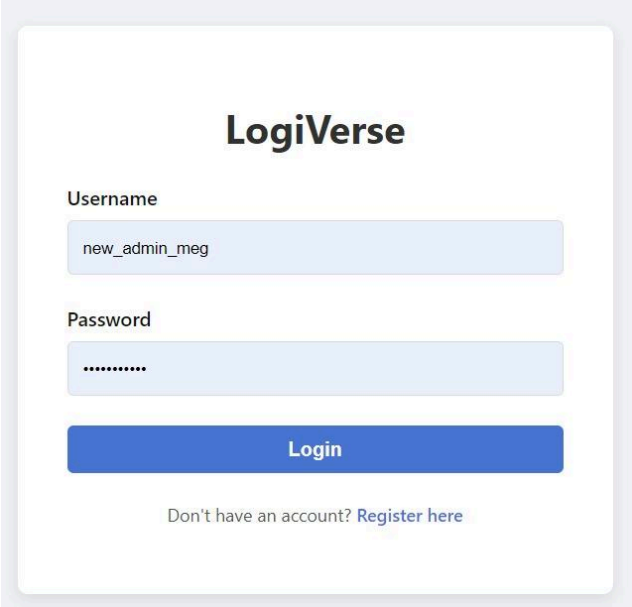
7. Implementation & Screenshot Gallery

This section demonstrates the practical implementation of the SCM system, highlighting the User Interface (UI) and the underlying SQL operations.

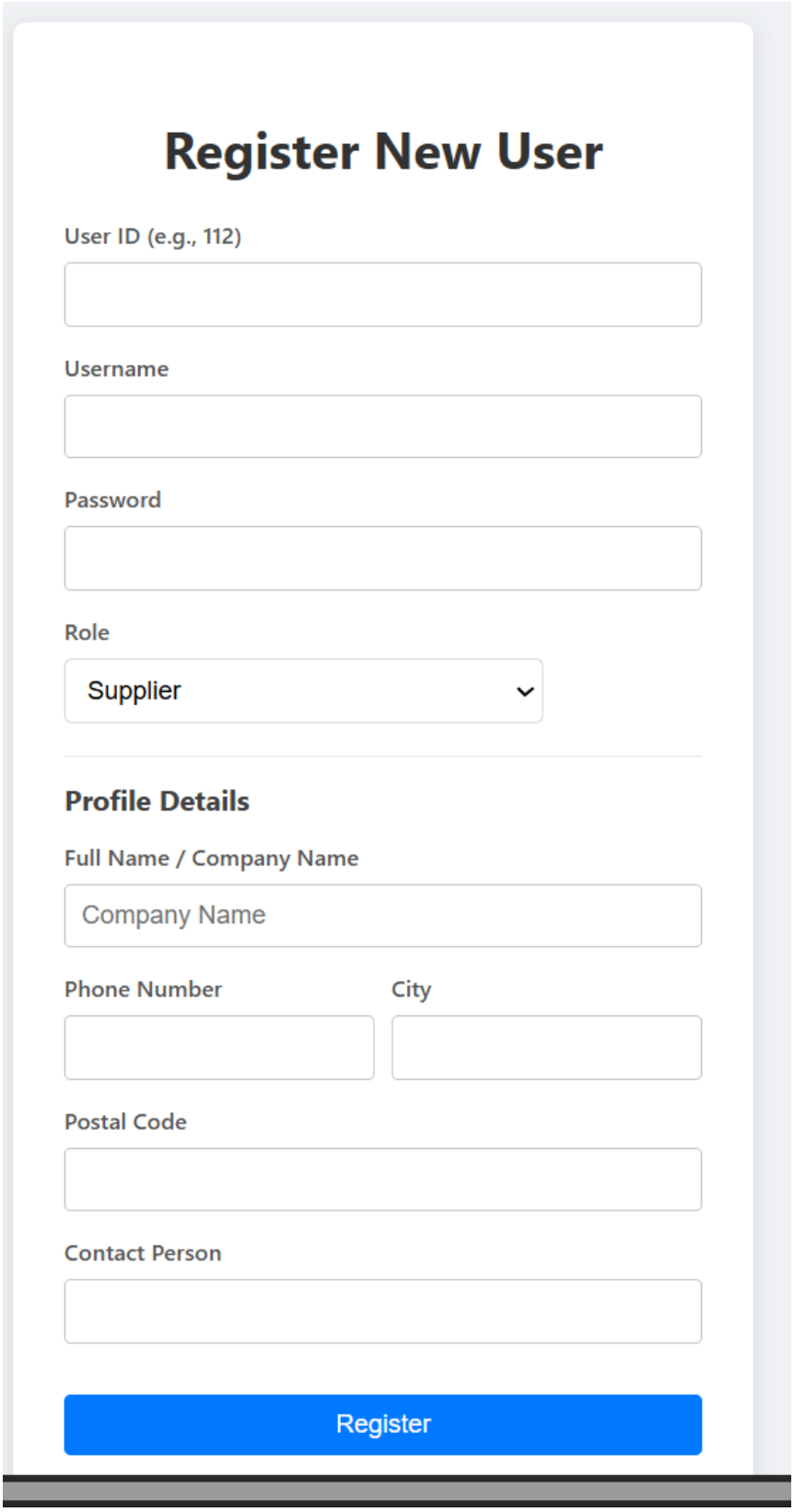
7.1 User Authentication & Access Control

The system secures access through a login interface and registration portal.

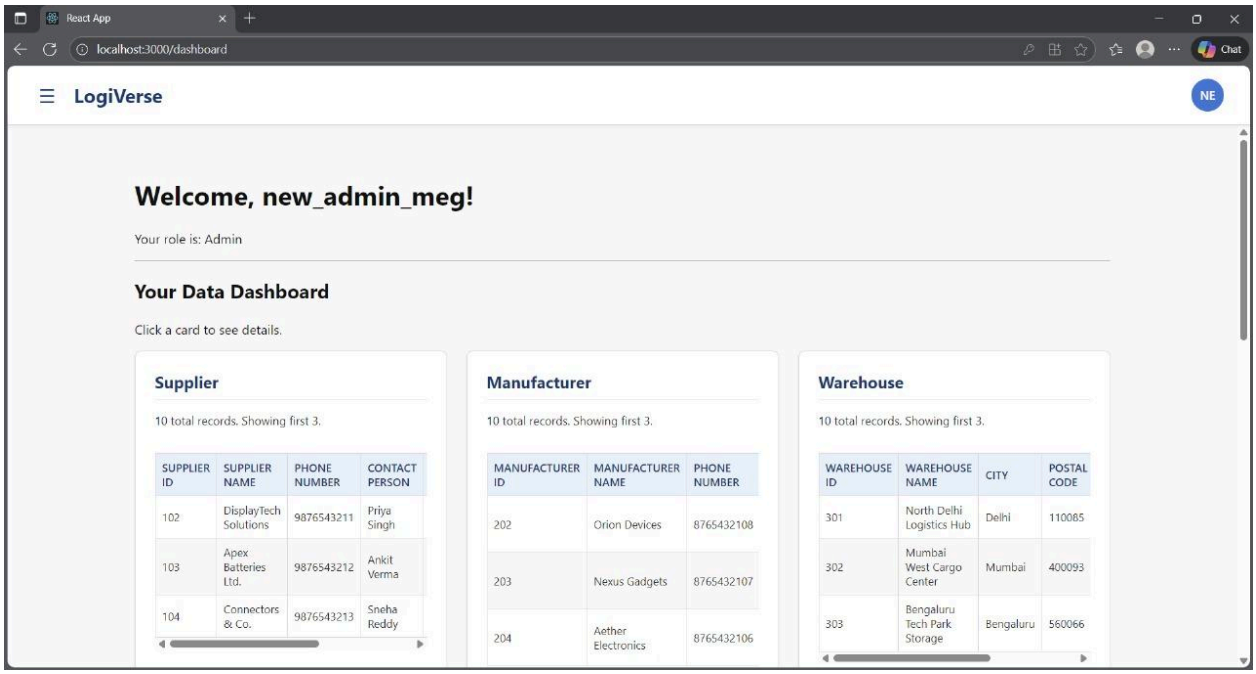
- Login Page:



- User Registration:



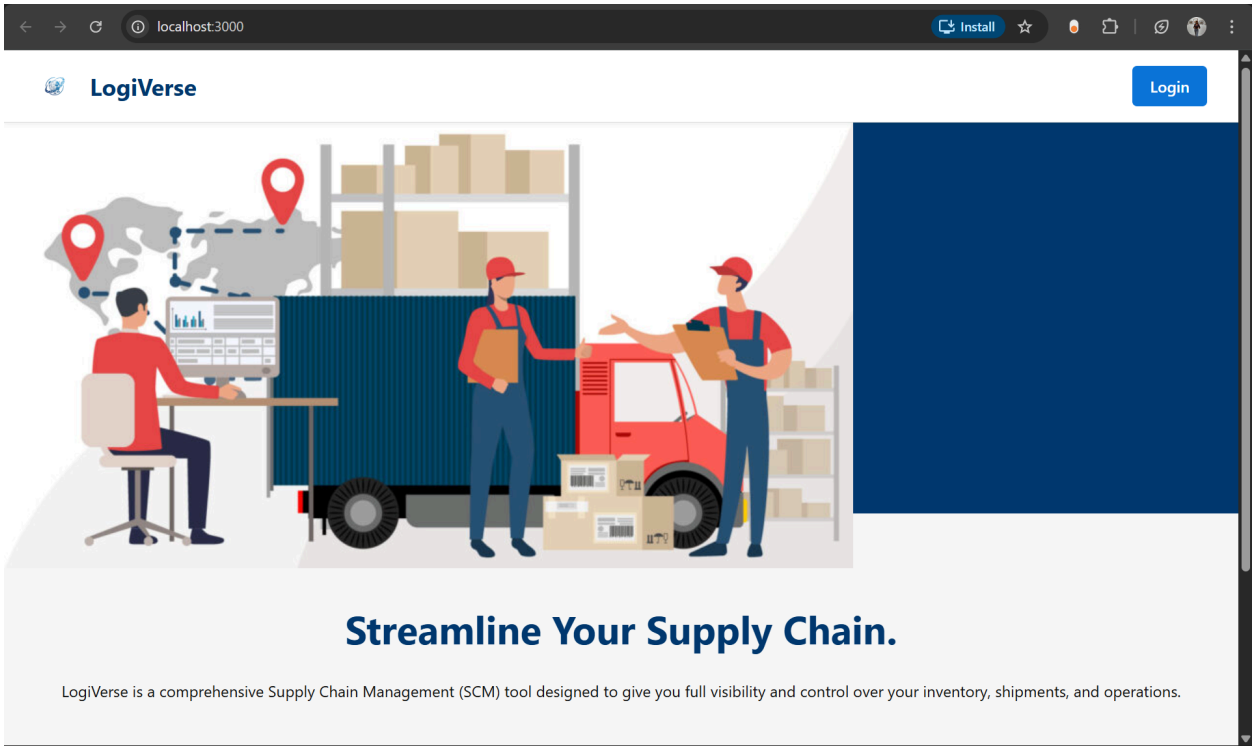
- Welcome Screen / Landing Page:



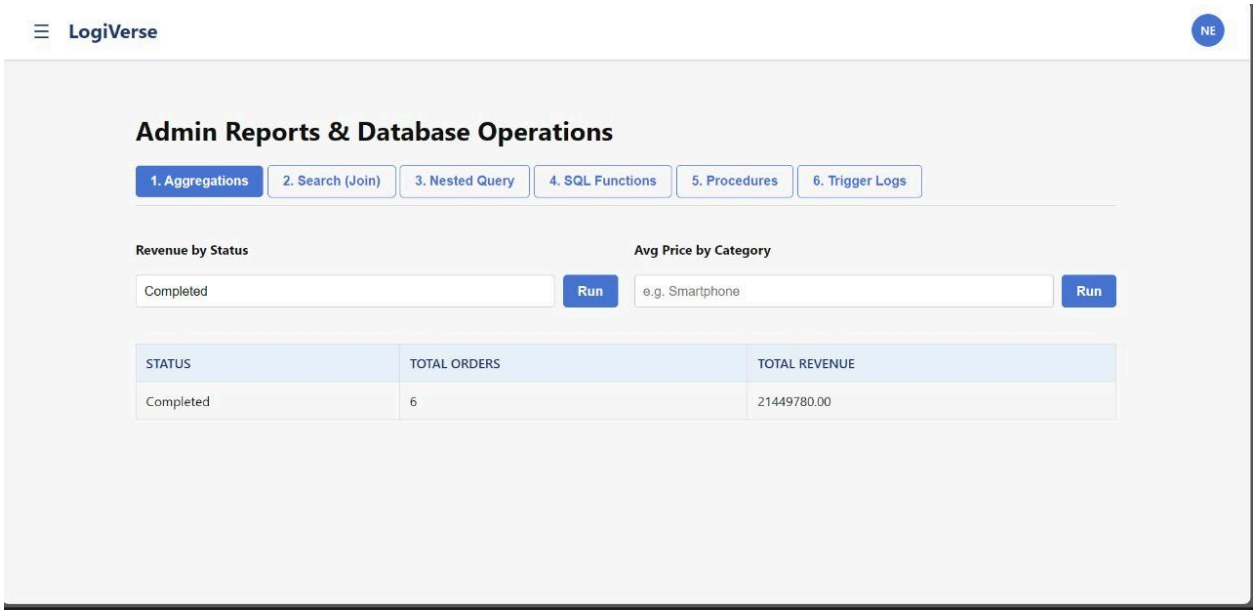
7.2 Dashboard & Navigation

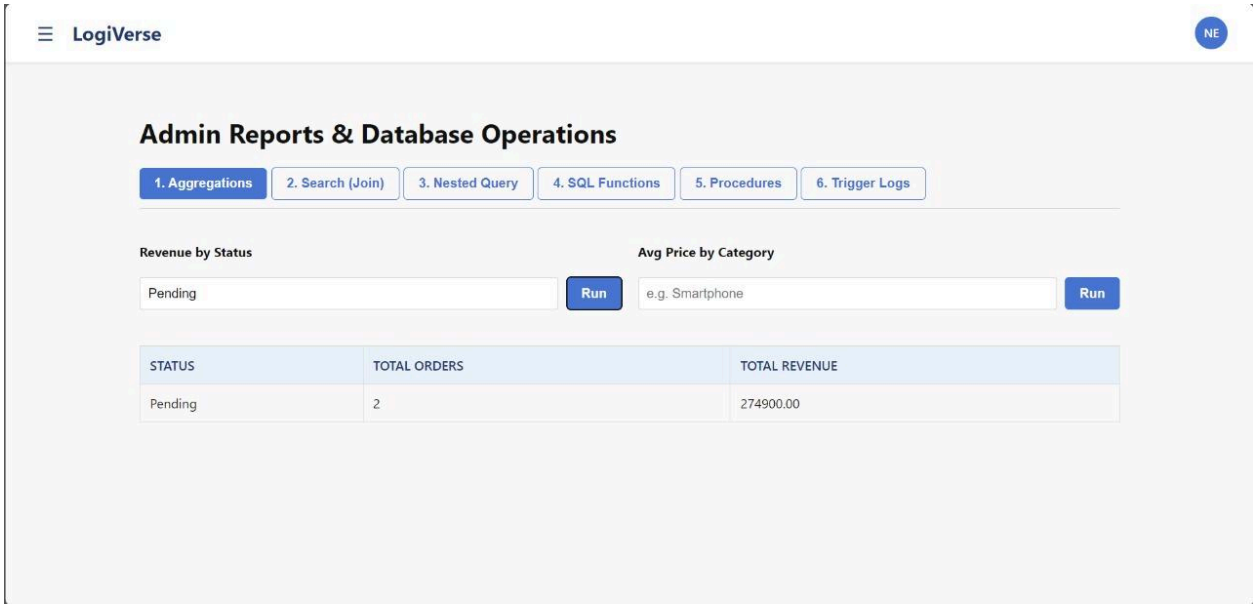
The central hub for navigating the application.

- Main Home Dashboard:



- Admin Reports & Operations View:

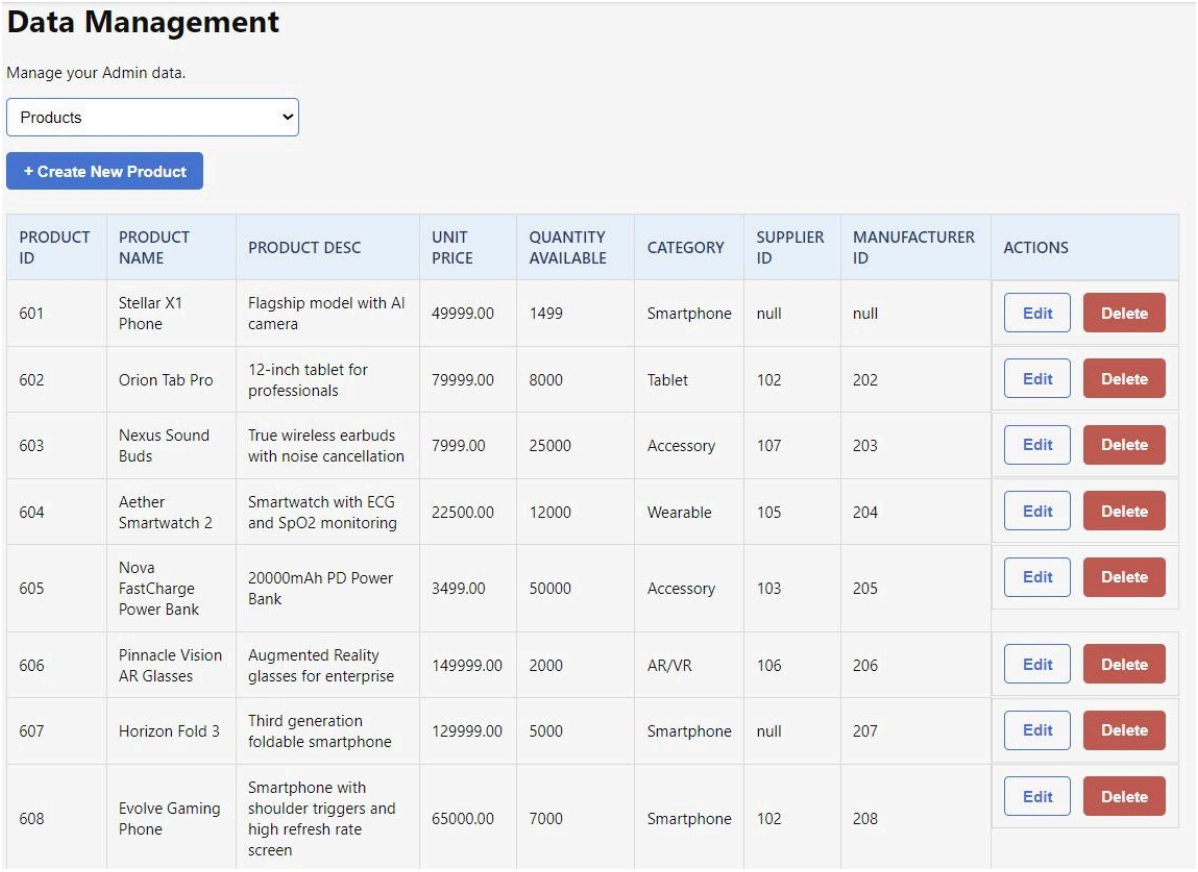



LogiVerse
NE
Admin Reports & Database Operations
1. Aggregations
2. Search (Join)
3. Nested Query
4. SQL Functions
5. Procedures
6. Trigger Logs
Revenue by Status
Pending
Run
Avg Price by Category
e.g. Smartphone
Run
STATUS
TOTAL ORDERS
TOTAL REVENUE
Pending
2
274900.00

7.3 Data Management Modules

The system provides interfaces to view and manage data across all entities.

- Product Management Interface:



Data Management

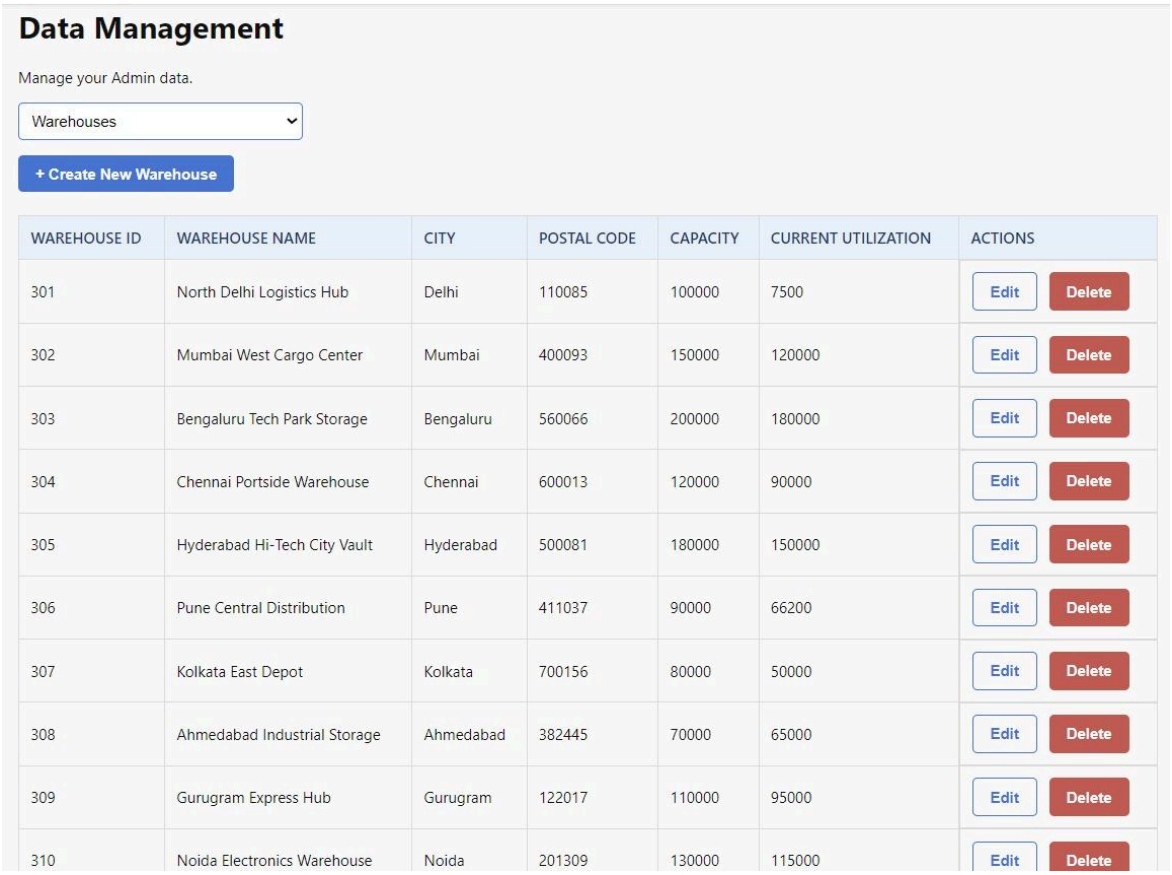
Manage your Admin data.

Products

+ Create New Product

PRODUCT ID	PRODUCT NAME	PRODUCT DESC	UNIT PRICE	QUANTITY AVAILABLE	CATEGORY	SUPPLIER ID	MANUFACTURER ID	ACTIONS
601	Stellar X1 Phone	Flagship model with AI camera	49999.00	1499	Smartphone	null	null	Edit Delete
602	Orion Tab Pro	12-inch tablet for professionals	79999.00	8000	Tablet	102	202	Edit Delete
603	Nexus Sound Buds	True wireless earbuds with noise cancellation	7999.00	25000	Accessory	107	203	Edit Delete
604	Aether Smartwatch 2	Smartwatch with ECG and SpO2 monitoring	22500.00	12000	Wearable	105	204	Edit Delete
605	Nova FastCharge Power Bank	20000mAh PD Power Bank	3499.00	50000	Accessory	103	205	Edit Delete
606	Pinnacle Vision AR Glasses	Augmented Reality glasses for enterprise	149999.00	2000	AR/VR	106	206	Edit Delete
607	Horizon Fold 3	Third generation foldable smartphone	129999.00	5000	Smartphone	null	207	Edit Delete
608	Evolve Gaming Phone	Smartphone with shoulder triggers and high refresh rate screen	65000.00	7000	Smartphone	102	208	Edit Delete

- Warehouse Management Interface:



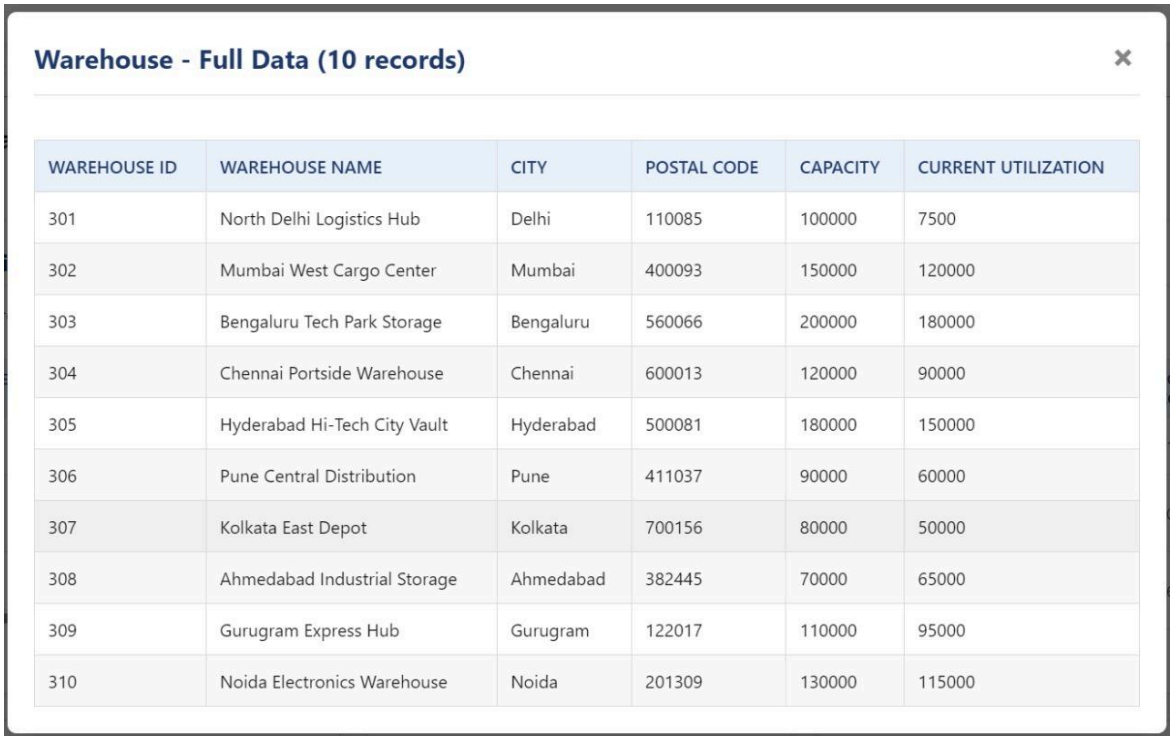
Data Management

Manage your Admin data.

Warehouses

+ Create New Warehouse

WAREHOUSE ID	WAREHOUSE NAME	CITY	POSTAL CODE	CAPACITY	CURRENT UTILIZATION	ACTIONS
301	North Delhi Logistics Hub	Delhi	110085	100000	7500	Edit Delete
302	Mumbai West Cargo Center	Mumbai	400093	150000	120000	Edit Delete
303	Bengaluru Tech Park Storage	Bengaluru	560066	200000	180000	Edit Delete
304	Chennai Portside Warehouse	Chennai	600013	120000	90000	Edit Delete
305	Hyderabad Hi-Tech City Vault	Hyderabad	500081	180000	150000	Edit Delete
306	Pune Central Distribution	Pune	411037	90000	66200	Edit Delete
307	Kolkata East Depot	Kolkata	700156	80000	50000	Edit Delete
308	Ahmedabad Industrial Storage	Ahmedabad	382445	70000	65000	Edit Delete
309	Gurugram Express Hub	Gurugram	122017	110000	95000	Edit Delete
310	Noida Electronics Warehouse	Noida	201309	130000	115000	Edit Delete



Warehouse - Full Data (10 records)

WAREHOUSE ID	WAREHOUSE NAME	CITY	POSTAL CODE	CAPACITY	CURRENT UTILIZATION
301	North Delhi Logistics Hub	Delhi	110085	100000	7500
302	Mumbai West Cargo Center	Mumbai	400093	150000	120000
303	Bengaluru Tech Park Storage	Bengaluru	560066	200000	180000
304	Chennai Portside Warehouse	Chennai	600013	120000	90000
305	Hyderabad Hi-Tech City Vault	Hyderabad	500081	180000	150000
306	Pune Central Distribution	Pune	411037	90000	60000
307	Kolkata East Depot	Kolkata	700156	80000	50000
308	Ahmedabad Industrial Storage	Ahmedabad	382445	70000	65000
309	Gurugram Express Hub	Gurugram	122017	110000	95000
310	Noida Electronics Warehouse	Noida	201309	130000	115000

- Manufacturer Data View:

Data Management

Manage your Admin data.

Manufacturers

+ Create New Manufacturer

MANUFACTURER ID	MANUFACTURER NAME	PHONE NUMBER	CITY	POSTAL CODE	PRODUCTION CAPACITY	LICENSE NUMBER	ACTIONS
202	Orion Devices	8765432108	Sriperumbudur	602105	75000	MFG-ORIN-6542	Edit Delete
203	Nexus Gadgets	8765432107	Gurugram	122001	60000	MFG-NEXS-3421	Edit Delete
204	Aether Electronics	8765432106	Pune	411014	45000	MFG-AETH-7893	Edit Delete
205	Nova Technologies	8765432105	Hyderabad	500081	80000	MFG-NOVA-4564	Edit Delete
206	Pinnacle Electronics	8765432104	Mumbai	400072	30000	MFG-PNCL-1235	Edit Delete
207	Horizon Mobile	8765432103	Bengaluru	560100	90000	MFG-HRZN-9086	Edit Delete
208	Evolve Tech	8765432102	Chennai	600096	55000	MFG-EVLV-5437	Edit Delete
209	Quantum Works	8765432101	Delhi	110044	25000	MFG-QWKS-2108	Edit Delete
210	Vertex Assembly	8765432100	Pune	411057	100000	MFG-VRTX-8769	Edit Delete

- Supplier Data View:

Supplier - Full Data (10 records)

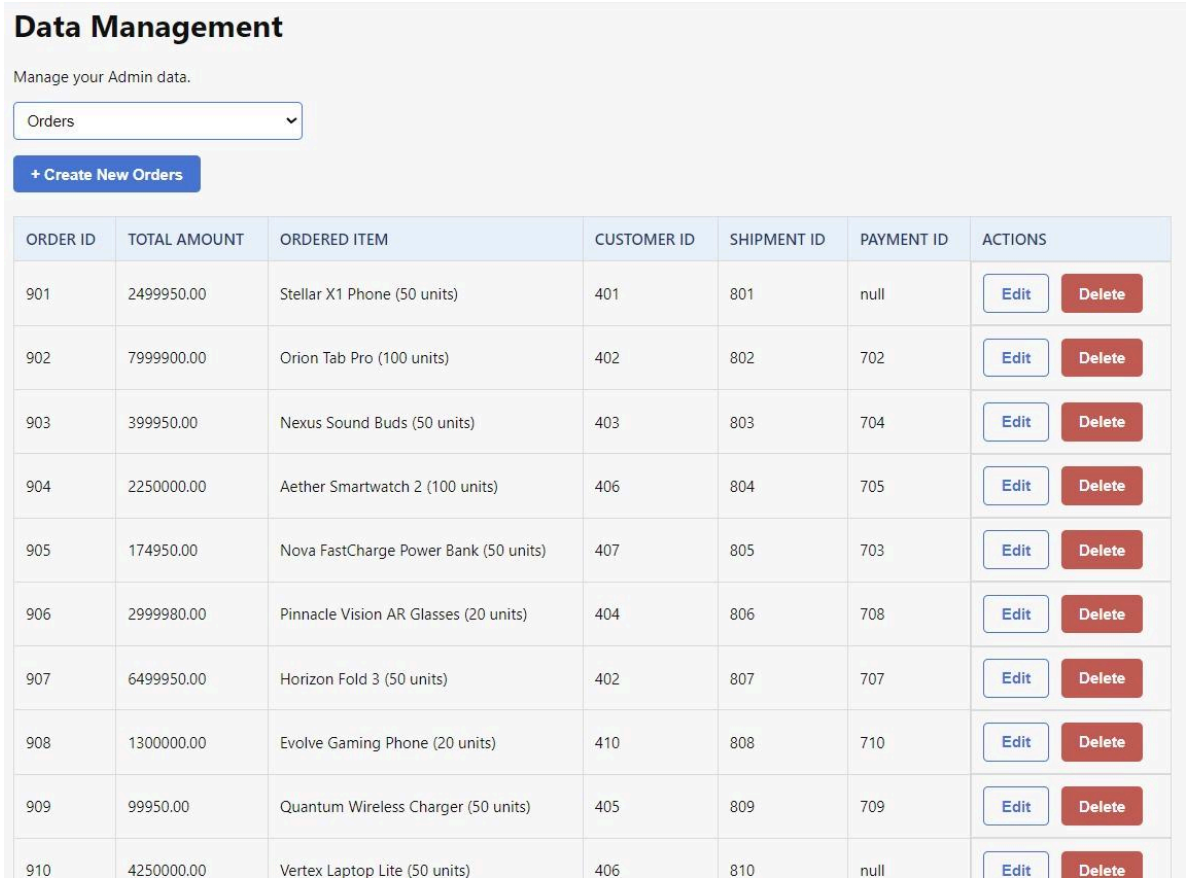
SUPPLIER ID	SUPPLIER NAME	PHONE NUMBER	CONTACT PERSON	CITY	POSTAL CODE	RATING
102	DisplayTech Solutions	9876543211	Priya Singh	Chennai	600001	4.80
103	Apex Batteries Ltd.	9876543212	Ankit Verma	Pune	411001	4.20
104	Connectors & Co.	9876543213	Sneha Reddy	Hyderabad	500001	4.00
105	Silicon Wafer Co.	9876543214	Vikram Rao	Bengaluru	560002	4.90
106	Optics Innovations	9876543215	Meera Iyer	Chennai	600002	4.60
107	SoundWave Audio	9876543216	Arjun Mehta	Mumbai	400001	4.30
108	CaseCrafters	9876543217	Diya Patel	Ahmedabad	380001	3.90
109	PowerCore Chargers	9876543218	Karan Gupta	Delhi	110001	4.70
110	Memory Lane Storage	9876543219	Nisha Desai	Pune	411002	4.52
112	Automatic Electronics Inc	999-888-777	Mr. Auto	Mumbai	400001	4.36

- Customer Data View:

Customer - Full Data (10 records)

CUSTOMER ID	CUSTOMER NAME	PHONE NUMBER	CUSTOMER TYPE	CITY	POSTAL CODE
401	Gadget Galaxy Retail	7654321098	Retail	Mumbai	400050
402	E-Mart Wholesale	7654321097	Wholesale	Delhi	110005
403	TechTree Online	7654321096	Online	Bengaluru	560034
404	Global Exports Inc.	7654321095	Export	Mumbai	400021
405	City Electronics Chain	7654321094	Retail	Pune	411004
406	Mega Distributors Ltd.	7654321093	Distributor	Hyderabad	500034
407	QuickMobile Store	7654321092	Online	Chennai	600017
408	Pioneer Wholesale	7654321091	Wholesale	Ahmedabad	380009
409	The Mobile Hub	7654321090	Retail	Kolkata	700016
410	NetCart Ecommerce	7654321089	Online	Bengaluru	560076

- Order Management:



Data Management

Manage your Admin data.

Orders

+ Create New Orders

ORDER ID	TOTAL AMOUNT	ORDERED ITEM	CUSTOMER ID	SHIPMENT ID	PAYMENT ID	ACTIONS
901	2499950.00	Stellar X1 Phone (50 units)	401	801	null	Edit Delete
902	7999900.00	Orion Tab Pro (100 units)	402	802	702	Edit Delete
903	399950.00	Nexus Sound Buds (50 units)	403	803	704	Edit Delete
904	2250000.00	Aether Smartwatch 2 (100 units)	406	804	705	Edit Delete
905	174950.00	Nova FastCharge Power Bank (50 units)	407	805	703	Edit Delete
906	2999980.00	Pinnacle Vision AR Glasses (20 units)	404	806	708	Edit Delete
907	6499950.00	Horizon Fold 3 (50 units)	402	807	707	Edit Delete
908	1300000.00	Evolve Gaming Phone (20 units)	410	808	710	Edit Delete
909	99950.00	Quantum Wireless Charger (50 units)	405	809	709	Edit Delete
910	4250000.00	Vertex Laptop Lite (50 units)	406	810	null	Edit Delete

- Payment Tracking:

Payment - Full Data (9 records)

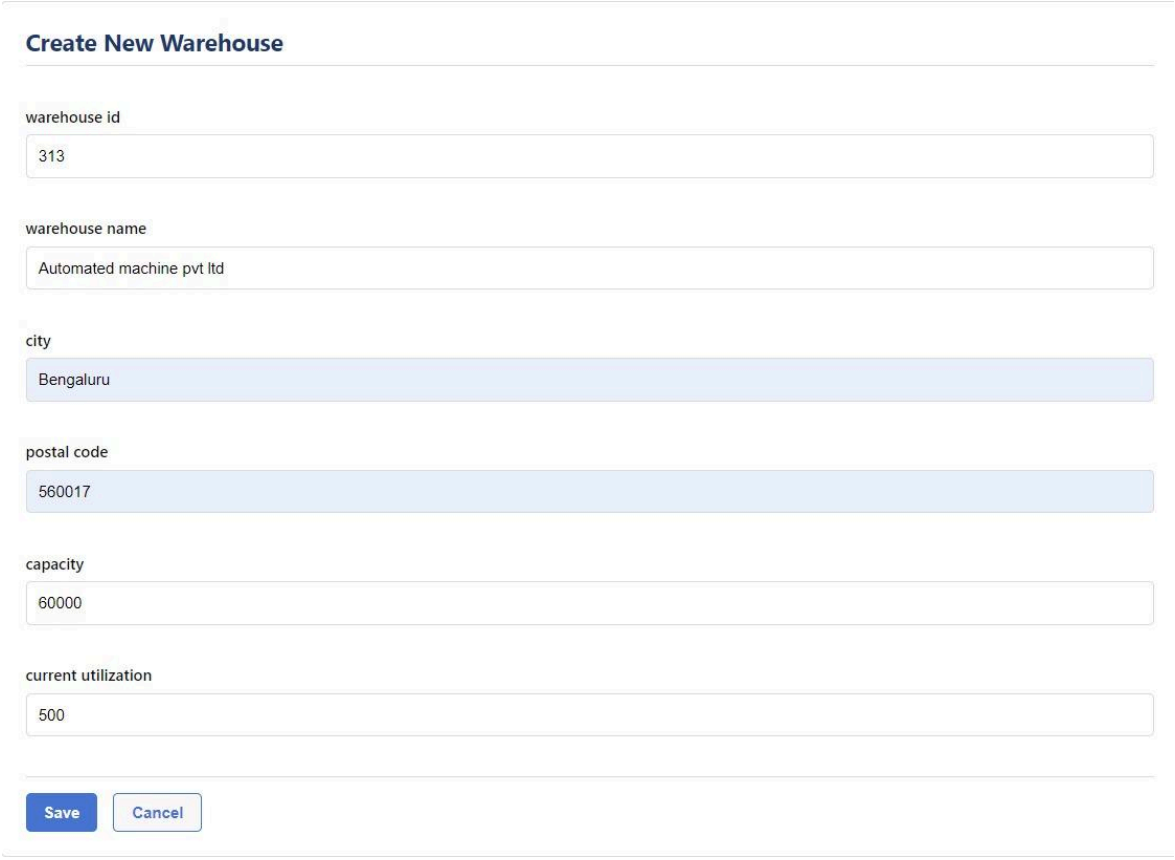
PAYMENT ID	PAYMENT MODE	STATUS
702	UPI	Completed
703	Bank Transfer	Pending
704	Credit Card	Completed
705	Net Banking	Completed
706	UPI	failed
707	Credit Card	Completed
708	Bank Transfer	Completed
709	UPI	Pending
710	Net Banking	Completed

7.4 Operational Workflows (CRUD Demonstration)

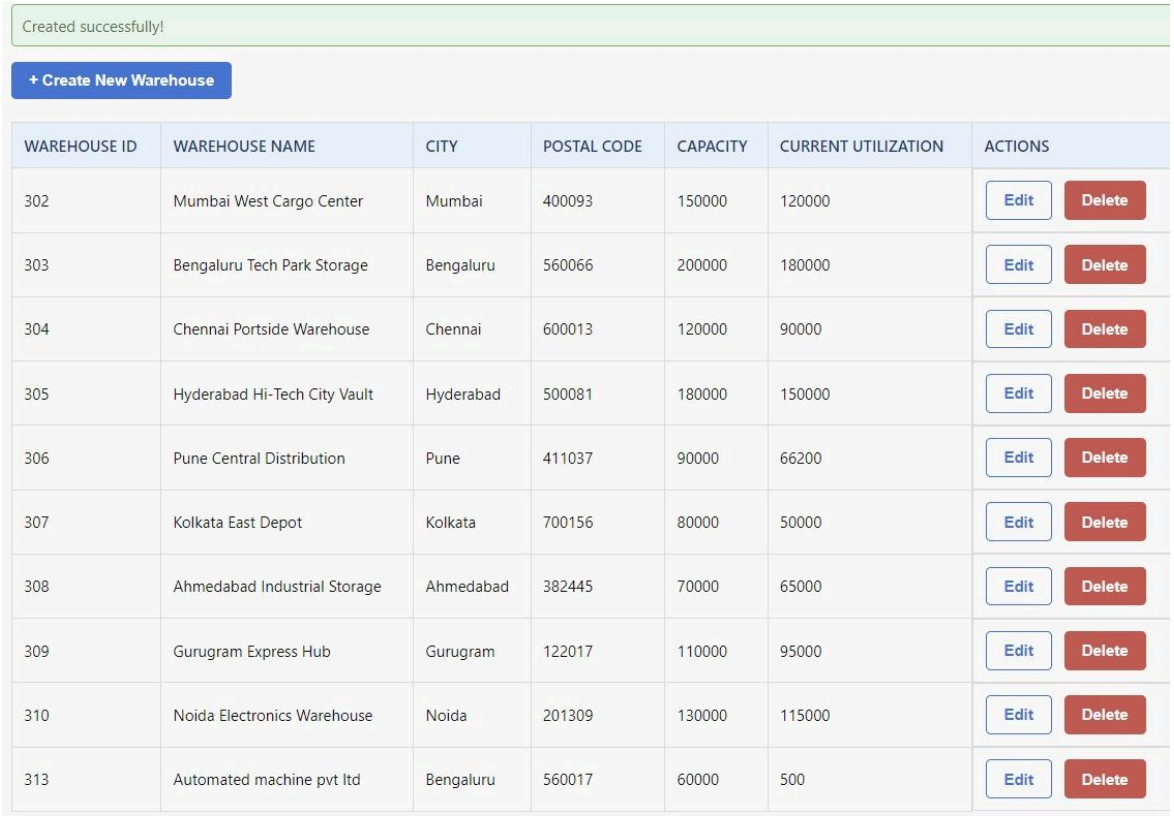
Demonstration of Create, Update, and Delete operations within the application.

Creating Data (Insert Operation)

- Form to Create New Warehouse:

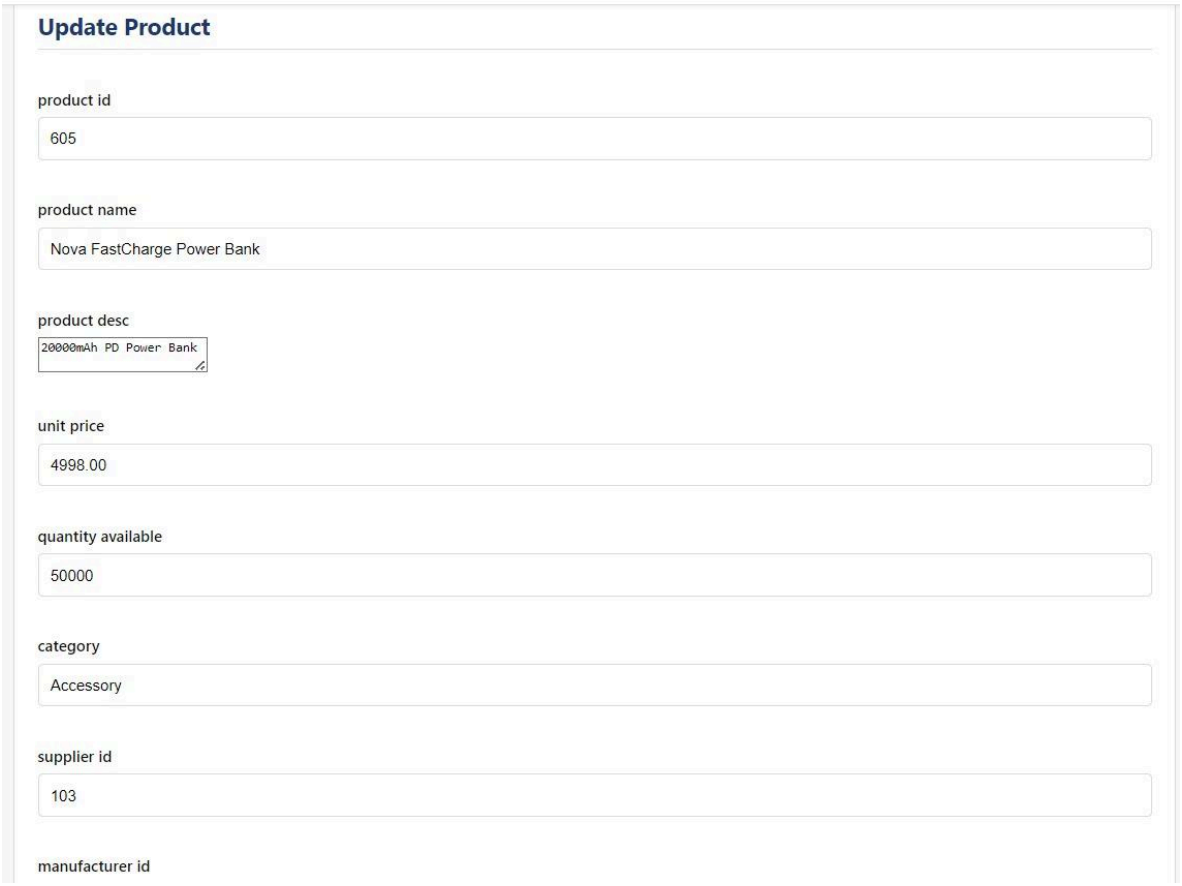


- Success Confirmation:

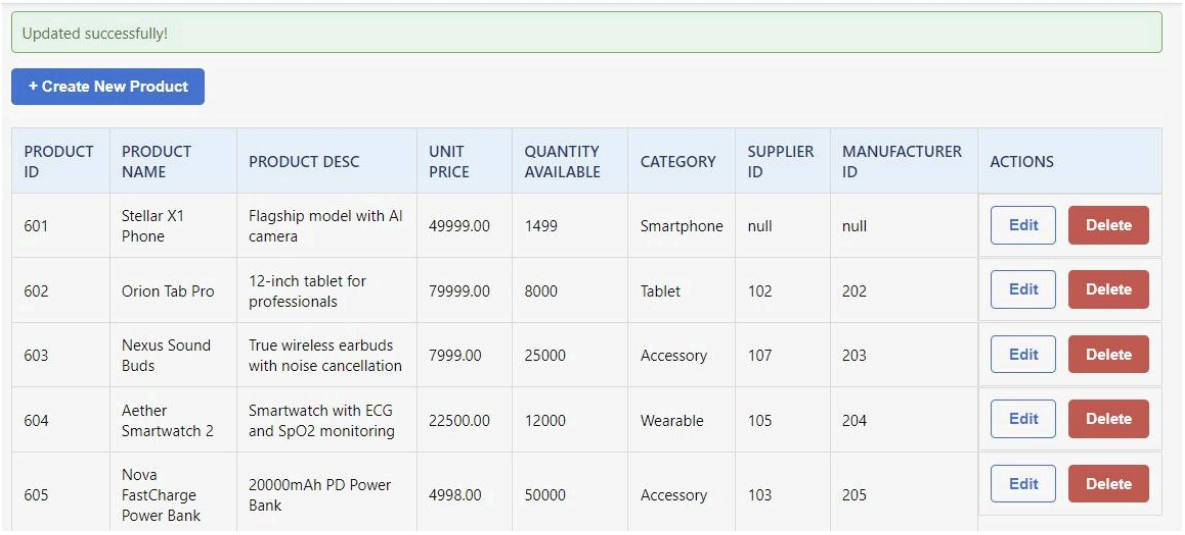


Updating Data (Update Operation)

- Form to Update Product Details:

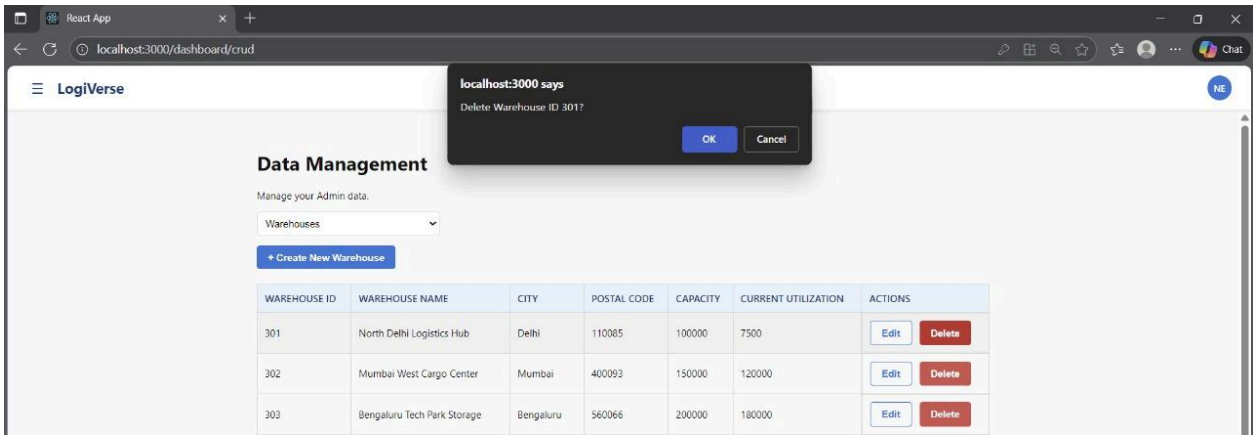


- Update Success Confirmation:

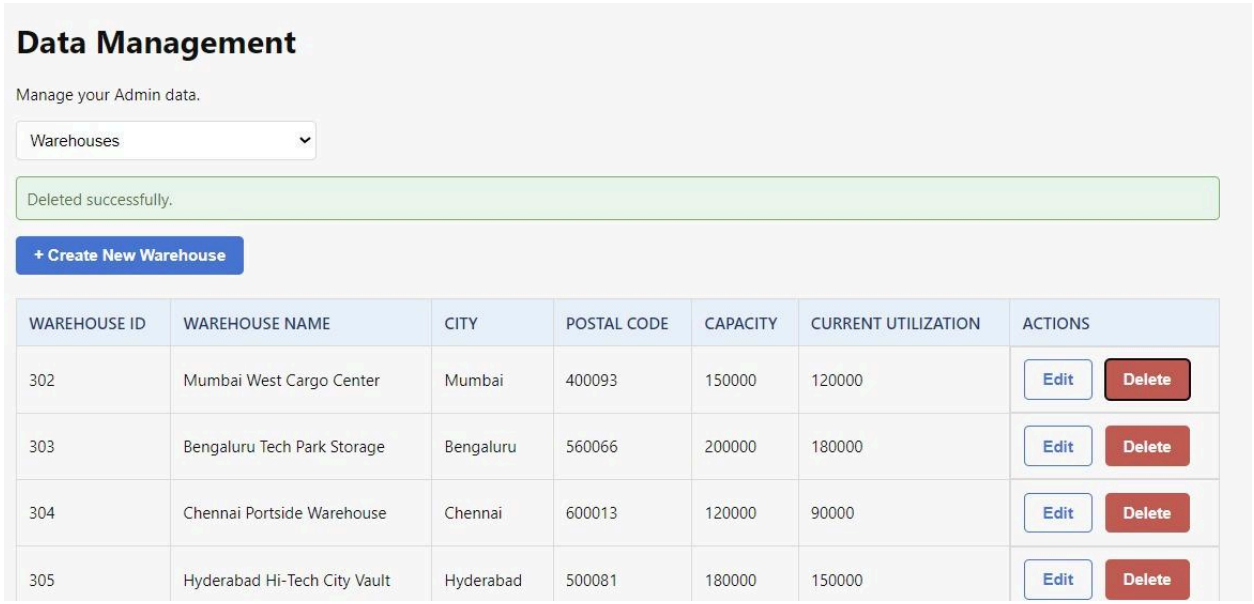


Deleting Data (Delete Operation)

- Deletion Process for Warehouse ID:



- Deletion Success Confirmation:

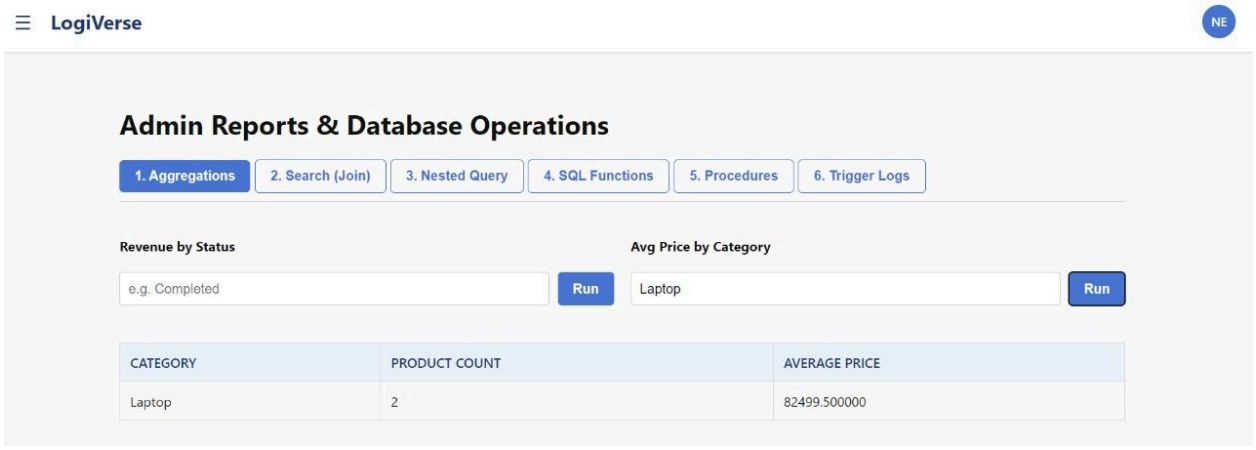


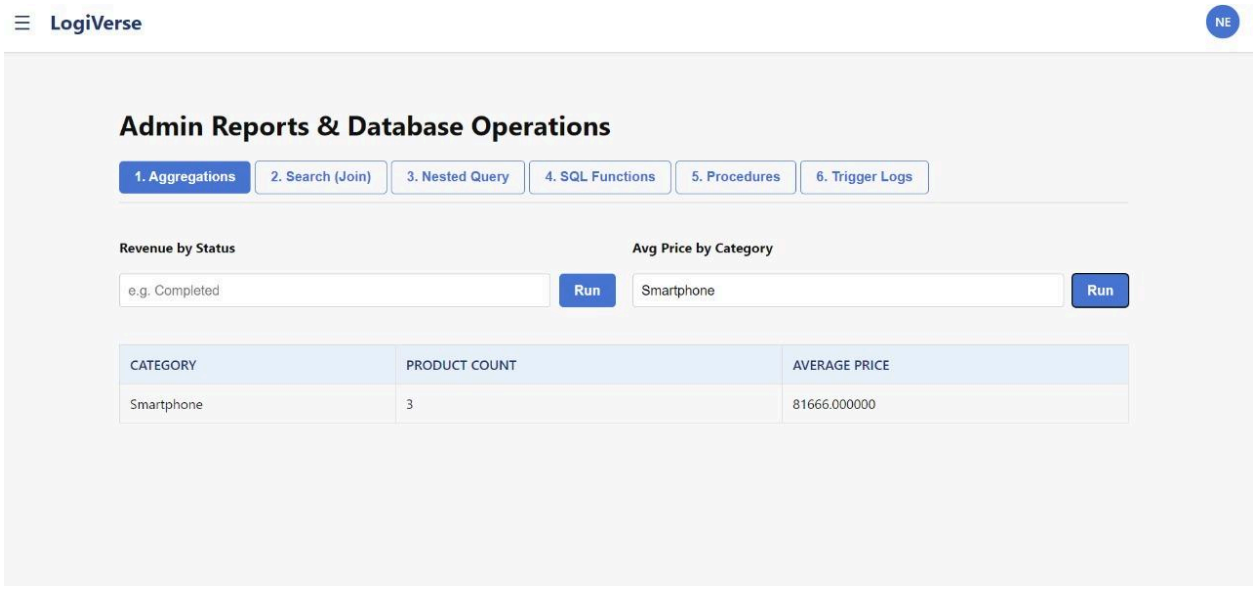
7.5 Advanced SQL Implementation

This section showcases the complex DBMS features implemented, including Aggregations, Joins, Nested Queries, Procedures, and Functions.

A. SQL Aggregation

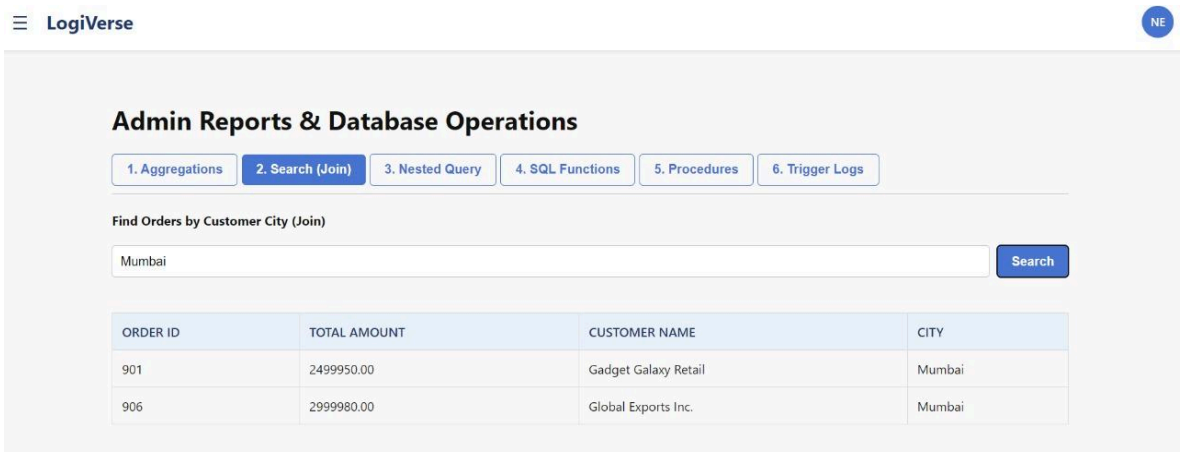
Using aggregate functions to summarize data (e.g., grouping products by category).

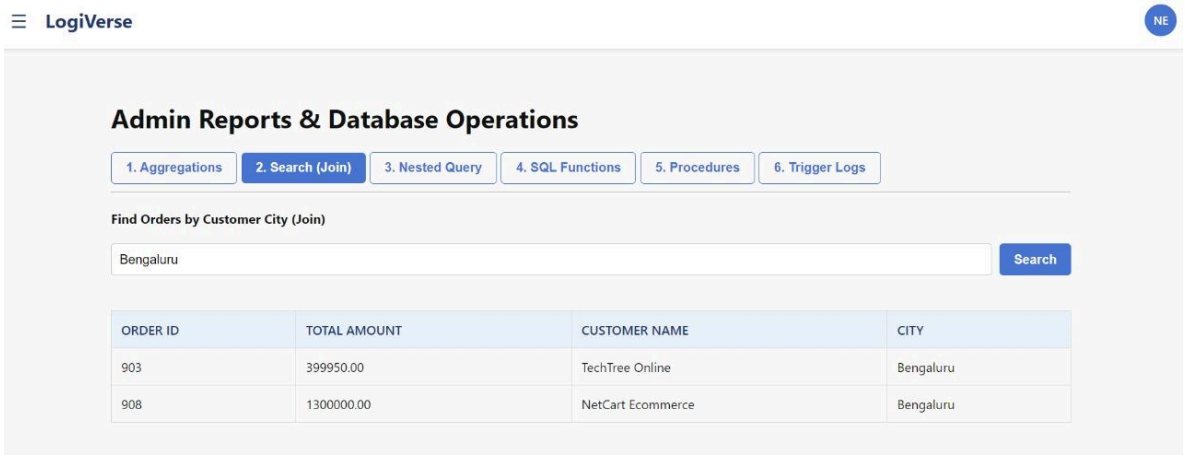
- 

- 

B. Joins & Search

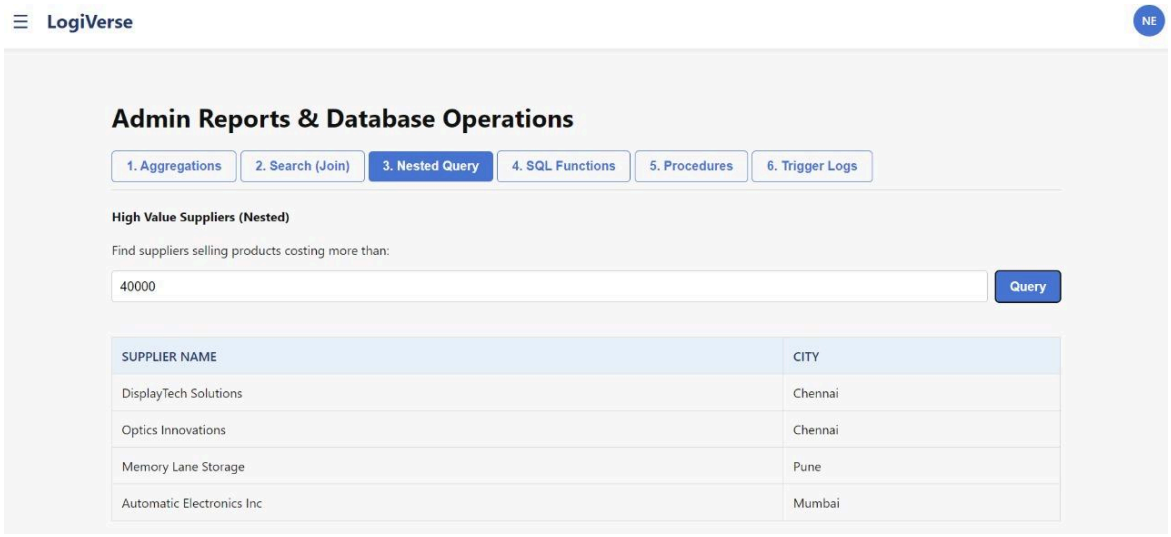
Demonstrating join operations to retrieve related data across tables (e.g., finding details based on city).

- 

- 

C. Nested Queries

Implementation of sub-queries to filter data based on dynamic criteria.

- 

CODE for reportRoutes.js

```
const express = require ('express');

const router = express.Router();

const pool = require('../config/db');

const { checkAuth, ensureIsAdmin } =
require('../middleware/authMiddleware');


// --- 1. AGGREGATION QUERIES (Updated) ---


// Query A: Total Sales Volume by Status (User inputs 'Completed',
'Pending', etc.)

router.post('/reports/agg-sales-by-status', checkAuth, async (req, res) =>
{

  try {

    const { status } = req.body;

    // Aggregation: SUM with WHERE clause

    const sql = `

      SELECT status, COUNT(order_id) as total_orders, SUM(total_amount) as
```

```
total_revenue
      FROM Orders
      JOIN Payment ON Orders.payment_id = Payment.payment_id
      WHERE Payment.status = ?
      GROUP BY status
    `;
    const [rows] = await pool.query(sql, [status]);
    res.status(200).json(rows);
  } catch (error) {
    res.status(500).json({ error: 'Database error: ' + error.message });
  }
});


// Query B: Average Product Price by Category (User inputs 'Smartphone', 'Laptop', etc.)
router.post('/reports/agg-avg-price-category', checkAuth, async (req, res) => {
  try {
    const { category } = req.body;
    // Aggregation: AVG with WHERE clause
    const sql = `
      SELECT category, COUNT(product_id) as product_count, AVG(unit_price) as average_price
      FROM Product
      WHERE category LIKE ?
      GROUP BY category
    `;
    const [rows] = await pool.query(sql, [`%${category}%`]);
```

```
    res.status(200).json(rows);
  } catch (error) {
    res.status(500).json({ error: 'Database error: ' + error.message });
  }
});

// 2. COMPLEX JOIN SEARCH (User inputs City -> Finds Orders)
// "Find all orders where the Customer is from [User Input City]"
router.post('/reports/complex-search', checkAuth, async (req, res) => {
  try {
    const { city } = req.body;

    // This is a JOIN query
    const sql = `
      SELECT o.order_id, o.total_amount, c.customer_name, c.city
      FROM Orders o
      JOIN Customer c ON o.customer_id = c.customer_id
      WHERE c.city LIKE ?
    `;

    const [rows] = await pool.query(sql, [`%${city}%`]);
    res.status(200).json(rows);
  } catch (error) {
    res.status(500).json({ error: 'Database error' });
  }
});

// 3. NESTED QUERY (User inputs Amount -> Finds Products cheaper than
```

```
that)
// "Find products that are cheaper than the average price of all products"
(Static Nested)
// OR "Find products cheaper than [User Input]"
router.post('/reports/nested', checkAuth, async (req, res) => {
  try {
    const { min_price } = req.body;

    // Example of Nested Query: Find suppliers who supply products costing
more than X
    const sql = `
      SELECT supplier_name, city
      FROM Supplier
      WHERE supplier_id IN (
        SELECT supplier_id FROM Product WHERE unit_price > ?
      )
    `;

    const [rows] = await pool.query(sql, [min_price]);
    res.status(200).json(rows);
  } catch (error) {
    res.status(500).json({ error: 'Database error' });
  }
});

// 4. LOGS VIEWER (For Testing Triggers)
router.get('/logs/triggers', checkAuth, async (req, res) => {
  try {
```

```
    // Assuming you created a table 'Product_Audit_Log' for your trigger
    // If not, creates one: CREATE TABLE Logs (log_msg VARCHAR(255), log_date DATETIME);
    // And trigger inserts into it.

    // For now, let's assume a table exists, or we return an empty array to prevent crash
    // Replace 'Product_Price_Log' with your actual log table name
    const [rows] = await pool.query('SELECT * FROM Product_Price_Log ORDER BY change_timestamp DESC LIMIT 10');
    res.status(200).json(rows);
  } catch (error) {
    console.log("Log table might not exist yet");
    res.status(200).json([]);
  }
});

// ====================================================================
//  NEW ADDITIONS: FUNCTIONS & PROCEDURES (Admin Only)
// ====================================================================

// --- SQL FUNCTIONS ---

router.post('/functions/utilization', checkAuth, ensureIsAdmin, async (req, res) => {
  try {
    const { warehouse_id } = req.body;
    const [rows] = await pool.query('SELECT fn_get_warehouse_utilization_percent(?) AS result', [warehouse_id]);
```

```
    res.status(200).json({ result: rows[0].result + '%' });
  } catch (error) { res.status(500).json({ error: error.message }); }
});

router.post('/functions/customer-order-count', checkAuth, ensureIsAdmin,
async (req, res) => {
  try {
    const { customer_id } = req.body;
    const [rows] = await pool.query('SELECT
fn_get_customer_total_orders(?) AS result', [customer_id]);
    res.status(200).json({ result: rows[0].result });
  } catch (error) { res.status(500).json({ error: error.message }); }
});

router.post('/functions/product-availability', checkAuth, ensureIsAdmin,
async (req, res) => {
  try {
    const { product_id } = req.body;
    const [rows] = await pool.query('SELECT fn_get_product_availability(?)
AS result', [product_id]);
    res.status(200).json({ result: rows[0].result });
  } catch (error) { res.status(500).json({ error: error.message }); }
});

router.post('/functions/tax', checkAuth, ensureIsAdmin, async (req, res)
=> {
  try {
    const { amount } = req.body;
    const [rows] = await pool.query('SELECT fn_calculate_order_tax(?) AS
```

```
result', [amount]);
    res.status(200).json({ result: rows[0].result });
  } catch (error) { res.status(500).json({ error: error.message }); }
});

router.post('/functions/manufacturer-count', checkAuth, ensureIsAdmin,
async (req, res) => {
  try {
    const { manufacturer_id } = req.body;
    const [rows] = await pool.query('SELECT
fn_get_manufacturer_product_count(?) AS result', [manufacturer_id]);
    res.status(200).json({ result: rows[0].result });
  } catch (error) { res.status(500).json({ error: error.message }); }
});

// --- STORED PROCEDURES ---

router.post('/procedures/customer-orders', checkAuth, ensureIsAdmin, async
(req, res) => {
  try {
    const { customer_id } = req.body;
    const [rows] = await pool.query('CALL sp_get_orders_by_customer(?)',
[customer_id]);
    res.status(200).json(rows[0]);
  } catch (error) { res.status(500).json({ error: error.message }); }
});

router.post('/procedures/low-stock', checkAuth, ensureIsAdmin, async (req,
res) => {
```

```
  try {
    const { threshold } = req.body;
    const [rows] = await pool.query('CALL sp_get_low_stock_products(?)', [threshold]);
    res.status(200).json(rows[0]);
  } catch (error) { res.status(500).json({ error: error.message }); }
});

router.post('/procedures/update-shipment', checkAuth, ensureIsAdmin, async (req, res) => {
  try {
    const { shipment_id, status } = req.body;
    await pool.query('CALL sp_update_shipment_status(?, ?)', [shipment_id, status]);
    res.status(200).json({ message: 'Shipment status updated successfully via Procedure.' });
  } catch (error) { res.status(500).json({ error: error.message }); }
});

router.post('/procedures/add-stock', checkAuth, ensureIsAdmin, async (req, res) => {
  try {
    const { warehouse_id, amount } = req.body;
    await pool.query('CALL sp_add_stock_to_warehouse(?, ?)', [warehouse_id, amount]);
    res.status(200).json({ message: 'Stock added to warehouse successfully.' });
  } catch (error) {
    // Return Trigger Error (400 Bad Request)
```

```
    if(error.sqlState === '45000') return res.status(400).json({ error: error.message });
    res.status(500).json({ error: error.message });
  }
});

// --- ADDITIONAL LOGS ---

router.get('/logs/price', checkAuth, ensureIsAdmin, async (req, res) => {
  try {
    const [rows] = await pool.query('SELECT * FROM Product_Price_Log ORDER BY change_timestamp DESC');
    res.status(200).json(rows);
  } catch (error) { res.status(200).json([]); }
});

router.get('/logs/shipment', checkAuth, ensureIsAdmin, async (req, res) => {
  try {
    const [rows] = await pool.query('SELECT * FROM Shipment_Status_Log ORDER BY change_timestamp DESC');
    res.status(200).json(rows);
  } catch (error) { res.status(200).json([]); }
});

module.exports = router;
```

CODE for scmRoutes.js

```
const express = require('express');

const router = express.Router();

const pool = require('../config/db');

const { checkAuth, checkPermission } =
require('../middleware/authMiddleware');


// --- HELPERS ---

const isOwner = async (table, idCol, idVal, foreignCol, foreignVal) => {

  const [rows] = await pool.query(`SELECT * FROM ${table} WHERE ${idCol} =
? AND ${foreignCol} = ?`, [idVal, foreignVal]);

  return rows.length > 0;

};


// 1. SUPPLIER ROUTES

router.post('/suppliers', checkAuth, checkPermission('Supplier',
'can_edit'), async (req, res) => {

  try {

    const { supplier_id, supplier_name, phone_number, contact_person,
city, postal_code, rating } = req.body;

    const sql = `INSERT INTO Supplier (supplier_id, supplier_name,
phone_number, contact_person, city, postal_code, rating) VALUES (?, ?, ?,
?, ?, ?, ?)`;

    await pool.query(sql, [supplier_id, supplier_name, phone_number,
contact_person, city, postal_code, rating]);

    res.status(201).json({ message: 'Supplier created successfully!' });

  } catch (error) {

    res.status(500).json({ error: 'Database error' });

  }
```

```
});


router.put('/suppliers/:id', checkAuth, checkPermission('Supplier',
'can_edit'), async (req, res) => {
  try {
    const { id } = req.params;
    const { role, entity_id } = req.user;


    // FIX: Convert both to String to ensure "112" matches 112
    if (role === 'Supplier' && String(id) !== String(entity_id)) {
      return res.status(403).json({ error: 'You can only update your own
profile.' });
    }


    const { supplier_name, phone_number, contact_person, city,
postal_code, rating } = req.body;
    const sql = `UPDATE Supplier SET supplier_name=?, phone_number=?,
contact_person=?, city=?, postal_code=?, rating=? WHERE supplier_id=?`;
    await pool.query(sql, [supplier_name, phone_number, contact_person,
city, postal_code, rating, id]);
    res.status(200).json({ message: 'Supplier updated successfully' });
  } catch (error) {
    console.error(error);
    res.status(500).json({ error: 'Database error' });
  }
});


// 2. MANUFACTURER ROUTES
router.post('/manufacturers', checkAuth, checkPermission('Manufacturer',
```

```
'can_edit'), async (req, res) => {
  try {
    const { manufacturer_id, manufacturer_name, phone_number, city,
postal_code, production_capacity, license_number } = req.body;
    const sql = `INSERT INTO Manufacturer (manufacturer_id,
manufacturer_name, phone_number, city, postal_code, production_capacity,
license_number) VALUES (?, ?, ?, ?, ?, ?, ?)`;
    await pool.query(sql, [manufacturer_id, manufacturer_name,
phone_number, city, postal_code, production_capacity, license_number]);
    res.status(201).json({ message: 'Manufacturer created successfully'
});
  } catch (error) {
    res.status(500).json({ error: 'Database error' });
  }
});


router.put('/manufacturers/:id', checkAuth,
checkPermission('Manufacturer', 'can_edit'), async (req, res) => {
  try {
    const { id } = req.params;
    const { role, entity_id } = req.user;


    // FIX: Added String() conversion
    if (role === 'Manufacturer' && String(id) !== String(entity_id)) {
      return res.status(403).json({ error: 'You can only update your own
profile.' });
    }


    const { manufacturer_name, phone_number, city, postal_code,
```

```
production_capacity, license_number } = req.body;
    const sql = `UPDATE Manufacturer SET manufacturer_name=?,
phone_number=?, city=?, postal_code=?, production_capacity=?,
license_number=? WHERE manufacturer_id=?`;
    await pool.query(sql, [manufacturer_name, phone_number, city,
postal_code, production_capacity, license_number, id]);
    res.status(200).json({ message: 'Manufacturer updated successfully'
});
  } catch (error) {
    res.status(500).json({ error: 'Database error' });
  }
});

// 3. WAREHOUSE ROUTES
router.post('/warehouses', checkAuth, checkPermission('Warehouse',
'can_edit'), async (req, res) => {
  try {
    const { warehouse_id, warehouse_name, city, postal_code, capacity,
current_utilization } = req.body;
    const sql = `INSERT INTO Warehouse (warehouse_id, warehouse_name,
city, postal_code, capacity, current_utilization) VALUES (?, ?, ?, ?, ?,
?)`;
    await pool.query(sql, [warehouse_id, warehouse_name, city,
postal_code, capacity, current_utilization]);
    res.status(201).json({ message: 'Warehouse created successfully' });
  } catch (error) {
    res.status(500).json({ error: 'Database error' });
  }
});
```

```
router.put('/warehouses/:id', checkAuth, checkPermission('Warehouse',
'can_edit'), async (req, res) => {
  try {
    const { id } = req.params;
    const { role, entity_id } = req.user;

    if (role === 'Warehouse' && String(id) !== String(entity_id)) {
        return res.status(403).json({ error: 'You can only update your own
warehouse.' });
    }

    const { warehouse_name, city, postal_code, capacity,
current_utilization } = req.body;

    const sql = `UPDATE Warehouse SET warehouse_name=?, city=?,
postal_code=?, capacity=?, current_utilization=? WHERE warehouse_id=?`;

    await pool.query(sql, [warehouse_name, city, postal_code, capacity,
current_utilization, id]);
    res.status(200).json({ message: 'Warehouse updated successfully' });

  } catch (error) {
    console.error("Update Error:", error);
    // --- CATCH TRIGGER ERROR ---
    if (error.sqlState === '45000') {
        // Send the specific trigger message (e.g., "Capacity Exceeded")
        return res.status(400).json({ error: error.message });
    }
    res.status(500).json({ error: error.message });
```

```
  }
});


// 4. CUSTOMER ROUTES
router.post('/customers', checkAuth, checkPermission('Customer',
'can_edit'), async (req, res) => {
  try {
    const { customer_id, customer_name, phone_number, customer_type, city,
postal_code } = req.body;
    const sql = `INSERT INTO Customer (customer_id, customer_name,
phone_number, customer_type, city, postal_code) VALUES (?, ?, ?, ?, ?,
?)`;
    await pool.query(sql, [customer_id, customer_name, phone_number,
customer_type, city, postal_code]);
    res.status(201).json({ message: 'Customer created successfully' });
  } catch (error) {
    res.status(500).json({ error: 'Database error' });
  }
});


router.put('/customers/:id', checkAuth, checkPermission('Customer',
'can_edit'), async (req, res) => {
  try {
    const { id } = req.params;
    const { role, entity_id } = req.user;


    // FIX: Added String() conversion
    if (role === 'Customer' && String(id) !== String(entity_id)) {
        return res.status(403).json({ error: 'You can only update your own
```

```
profile.' });
    }

    const { customer_name, phone_number, customer_type, city, postal_code
} = req.body;
    const sql = `UPDATE Customer SET customer_name=?, phone_number=?,
customer_type=?, city=?, postal_code=? WHERE customer_id=?`;
    await pool.query(sql, [customer_name, phone_number, customer_type,
city, postal_code, id]);
    res.status(200).json({ message: 'Customer updated successfully' });
  } catch (error) {
    res.status(500).json({ error: 'Database error' });
  }
});

// 5. PRODUCT ROUTES
router.post('/products', checkAuth, checkPermission('Product',
'can_edit'), async (req, res) => {
  try {
    let { product_id, product_name, product_desc, unit_price,
quantity_available, category, supplier_id, manufacturer_id } = req.body;
    const { role, entity_id } = req.user;

    if (role === 'Supplier') {
        supplier_id = entity_id;
    } else if (role === 'Manufacturer') {
        manufacturer_id = entity_id;
    }
```

```
    const sql = `INSERT INTO Product (product_id, product_name,
product_desc, unit_price, quantity_available, category, supplier_id,
manufacturer_id) VALUES (?, ?, ?, ?, ?, ?, ?, ?)`;
    await pool.query(sql, [product_id, product_name, product_desc,
unit_price, quantity_available, category, supplier_id, manufacturer_id]);
    res.status(201).json({ message: 'Product created successfully' });
  } catch (error) {
    console.error(error);
    res.status(500).json({ error: 'Database error: ' + error.message });
  }
});


router.put('/products/:id', checkAuth, checkPermission('Product',
'can_edit'), async (req, res) => {
  try {
    const { id } = req.params;
    const { role, entity_id } = req.user;


    if (role === 'Supplier') {
      const valid = await isOwner('Product', 'product_id', id,
'supplier_id', entity_id);
      if (!valid) return res.status(403).json({ error: 'Forbidden: You do
not own this product.' });
    }
    if (role === 'Manufacturer') {
      const valid = await isOwner('Product', 'product_id', id,
'manufacturer_id', entity_id);
      if (!valid) return res.status(403).json({ error: 'Forbidden: You do
not own this product.' });
```

```
    }

    const { product_name, product_desc, unit_price, quantity_available,
category, supplier_id, manufacturer_id } = req.body;
    const sql = `UPDATE Product SET product_name=?, product_desc=?,
unit_price=?, quantity_available=?, category=?, supplier_id=?,
manufacturer_id=? WHERE product_id=?`;
    await pool.query(sql, [product_name, product_desc, unit_price,
quantity_available, category, supplier_id, manufacturer_id, id]);
    res.status(200).json({ message: 'Product updated successfully' });
  } catch (error) {
    res.status(500).json({ error: 'Database error' });
  }
});

// 6. SHIPMENT ROUTES
router.post('/shipments', checkAuth, checkPermission('Shipment',
'can_edit'), async (req, res) => {
  try {
    let { shipment_id, carrier_name, transport_mode, shipping_cost,
status, warehouse_id } = req.body;
    const { role, entity_id } = req.user;

    if (role === 'Warehouse') {
        warehouse_id = entity_id;
    }

    const sql = `INSERT INTO Shipment (shipment_id, carrier_name,
transport_mode, shipping_cost, status, warehouse_id) VALUES (?, ?, ?, ?,
```

```
?, ?)`;
    await pool.query(sql, [shipment_id, carrier_name, transport_mode,
shipping_cost, status, warehouse_id]);
    res.status(201).json({ message: 'Shipment created successfully' });
  } catch (error) {
    res.status(500).json({ error: 'Database error' });
  }
});


router.put('/shipments/:id', checkAuth, checkPermission('Shipment',
'can_edit'), async (req, res) => {
  try {
    const { id } = req.params;
    const { role, entity_id } = req.user;


    if (role === 'Warehouse') {
        const valid = await isOwner('Shipment', 'shipment_id', id,
'warehouse_id', entity_id);
        if (!valid) return res.status(403).json({ error: 'Forbidden: This
shipment is not from your warehouse.' });
    }


    const { carrier_name, transport_mode, shipping_cost, warehouse_id } =
req.body;
    const sql = `UPDATE Shipment SET carrier_name=?, transport_mode=?,
shipping_cost=?, warehouse_id=? WHERE shipment_id=?`;
    await pool.query(sql, [carrier_name, transport_mode, shipping_cost,
warehouse_id, id]);
    res.status(200).json({ message: 'Shipment details updated
```

```
successfully' });
  } catch (error) {
    res.status(500).json({ error: 'Database error' });
  }
});

// SPECIAL: Update Shipment Status
router.put('/shipments/:id/status', checkAuth, checkPermission('Shipment',
'can_edit'), async (req, res) => {
  try {
    const { id } = req.params;
    const { role, entity_id } = req.user;
    const { status } = req.body;

    if (role === 'Warehouse') {
        const valid = await isOwner('Shipment', 'shipment_id', id,
'warehouse_id', entity_id);
        if (!valid) return res.status(403).json({ error: 'Forbidden: Not
your shipment.' });
    }

    await pool.query('CALL sp_update_shipment_status(?, ?)', [id,
status]);
    res.status(200).json({ message: 'Shipment status updated!' });
  } catch (error) {
    res.status(500).json({ error: 'Database error' });
  }
});
```

```
// 7. PAYMENT & 8. ORDERS ROUTES
router.post('/payments', checkAuth, checkPermission('Orders', 'can_edit'),
async (req, res) => {
    try {
        const { payment_id, payment_mode, status } = req.body;
        const sql = `INSERT INTO Payment (payment_id, payment_mode,
status) VALUES (?, ?, ?)`;
        await pool.query(sql, [payment_id, payment_mode, status]);
        res.status(201).json({ message: 'Payment created successfully' });
      } catch (error) {
        res.status(500).json({ error: 'Database error' });
      }
});

router.put('/payments/:id', checkAuth, checkPermission('Orders',
'can_edit'), async (req, res) => {
    try {
        const { id } = req.params;
        const { payment_mode, status } = req.body;
        const sql = `UPDATE Payment SET payment_mode=?, status=? WHERE
payment_id=?`;
        await pool.query(sql, [payment_mode, status, id]);
        res.status(200).json({ message: 'Payment updated successfully' });
      } catch (error) {
        res.status(500).json({ error: 'Database error' });
      }
});
```

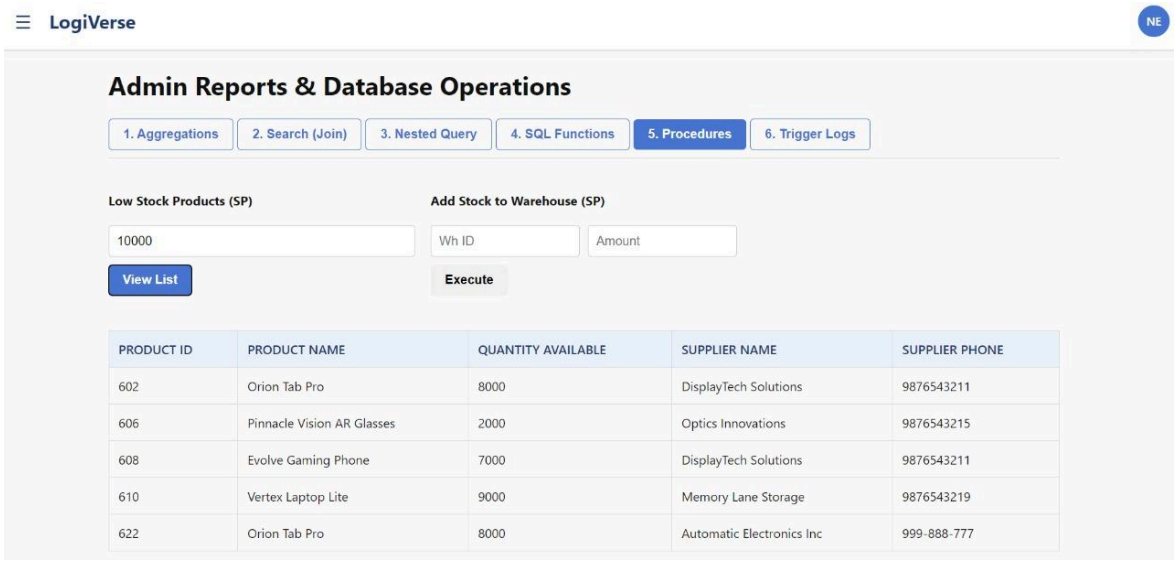
```
router.post('/orders', checkAuth, checkPermission('Orders', 'can_edit'),
async (req, res) => {
    try {
        let { order_id, total_amount, ordered_item, customer_id,
shipment_id, payment_id } = req.body;
        if (req.user.role === 'Customer') customer_id =
req.user.entity_id;

        const sql = `INSERT INTO Orders (order_id, total_amount,
ordered_item, customer_id, shipment_id, payment_id) VALUES (?, ?, ?, ?, ?,
?)`;
        await pool.query(sql, [order_id, total_amount, ordered_item,
customer_id, shipment_id, payment_id]);
        res.status(201).json({ message: 'Order created successfully' });
      } catch (error) {
        res.status(500).json({ error: 'Database error' });
      }
});

router.put('/orders/:id', checkAuth, checkPermission('Orders',
'can_edit'), async (req, res) => {
    try {
        const { id } = req.params;
        const { total_amount, ordered_item, customer_id, shipment_id,
payment_id } = req.body;

        if (req.user.role === 'Customer') {
            const valid = await isOwner('Orders', 'order_id', id,
'customer_id', req.user.entity_id);
```

```
            if (!valid) return res.status(403).json({error: 'Not your
order'});
        }

        const sql = `UPDATE Orders SET total_amount=?, ordered_item=?,
customer_id=?, shipment_id=?, payment_id=? WHERE order_id=?`;
        await pool.query(sql, [total_amount, ordered_item, customer_id,
shipment_id, payment_id, id]);
        res.status(200).json({ message: 'Order updated successfully' });
      } catch (error) {
        res.status(500).json({ error: 'Database error' });
      }
});

module.exports = router;
```

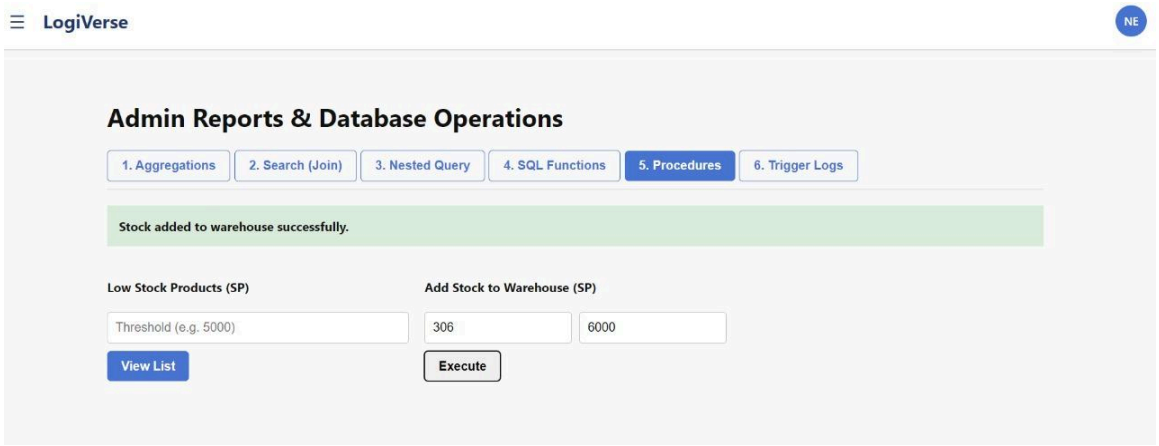
D. Stored Procedures

Database procedures used to automate tasks like adding stock.

- Procedure Execution:



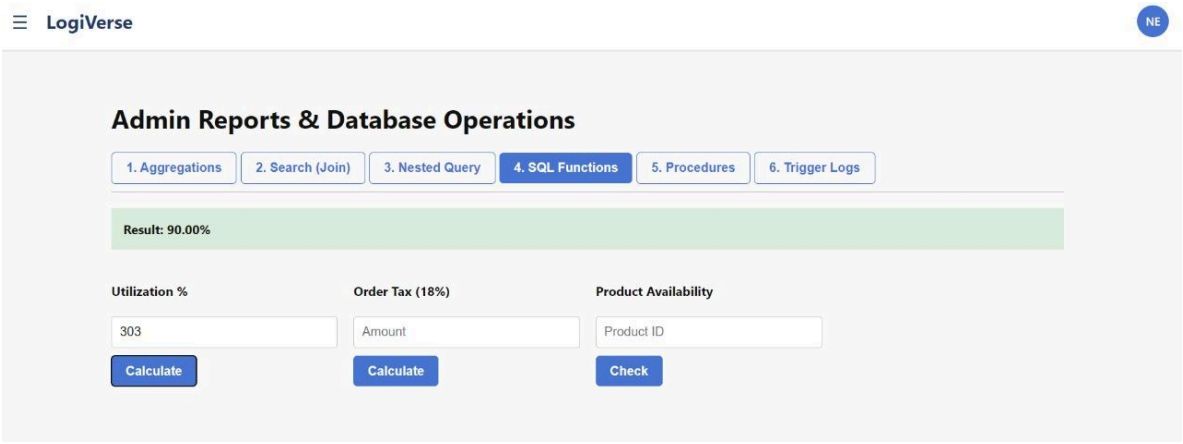
- Result of Stock Addition:



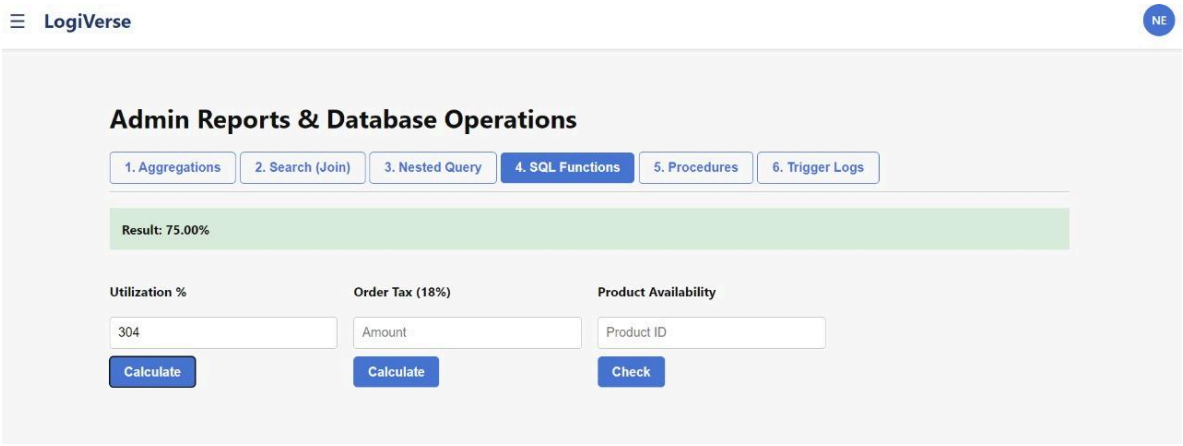
E. SQL Functions

Custom functions created to calculate warehouse utilization and determine stock status.

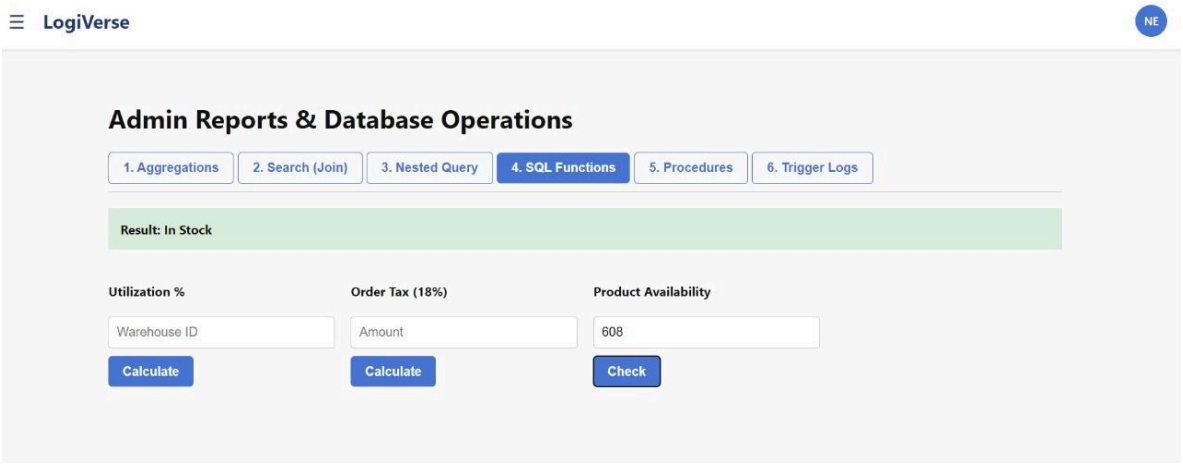
- Function Definition/Call:



- Utilization Calculation Result (75%):



- Stock Status Result:



F. System Reporting

General admin reports generated from the database.

-