



CASE STUDY REPORT

ON

SHIP PORT MANAGEMENT SYSTEM

Student Name: Shivanshi

Branch: UIC/BCA

Semester: 4th

Submitted to: Mr. Arvinder Singh

Subject Code: 23CAP-252

UID: 23BCA10432

Section/Group: BCA-4/b

Subject Name: DBMS



ACKNOWLEDGEMENT

We would like to express our sincere gratitude to everyone who contributed to the development of the **Ship Port Management System** project. This project would not have been possible without the valuable support, guidance, and resources provided throughout the process.

First and foremost, we extend our deepest appreciation to our **Mr. Arvinder Singh**, whose expertise and encouragement played a crucial role in shaping our understanding of **database management, schema design, and SQL programming**. Their insightful feedback helped refine our approach and ensure accuracy in implementation.

We would also like to thank our **team members and colleagues** for their collaboration, dedication, and unwavering commitment. Their discussions, problem-solving skills, and contributions greatly enriched the project, making it more comprehensive and efficient.

Special thanks to **database experts and online communities** for sharing valuable knowledge and best practices in **SQL, normalization, and ER diagram design**. Resources such as **MySQL documentation, PostgreSQL forums, and dbdiagram.io** significantly aided in structuring and optimizing our system.

Additionally, we appreciate the **tools and platforms** used throughout this project, including **MySQL Workbench, SQL Server Management Studio, and dbdiagram.io**, which facilitated efficient database modeling and query execution.

Finally, we express gratitude to our **friends and family** for their continuous encouragement and support, providing motivation during every stage of development.

This project is a result of collective efforts, learning, and innovation, aimed at delivering a structured and efficient **Ship Port Management System** for port operations. 🚢⚓

Thank you to everyone who played a role in making this project a success!

INTRODUCTION

Introduction to Ship Port Management System 🚢⚓

Overview

The **Ship Port Management System** is a database-driven application designed to efficiently handle ship-related operations at ports. It ensures systematic tracking of ships, cargo movement, dock statuses, port schedules, employee records, and financial transactions. This system provides real-time data management and enhances port efficiency by organizing crucial information in a structured format.

Importance of DBMS in Port Management

Ports serve as critical hubs for international trade, handling thousands of cargo shipments and managing multiple ships simultaneously. Without a well-structured database, managing such vast amounts of data can become chaotic and inefficient. By implementing a **Database Management System (DBMS)**, ports can optimize operations, prevent logistical errors, ensure proper scheduling, and securely store essential records.

Key Objectives

1. **Ship Tracking:** Keep a record of ship arrivals, departures, and ownership details.
2. **Cargo Management:** Store cargo-related data, including type, weight, and movement history.
3. **Port and Dock Monitoring:** Maintain information on port locations, dock availability, and conditions.
4. **Employee Records:** Track port employees' positions and responsibilities.
5. **Financial Transactions:** Securely log transactions related to cargo shipments and port services.
6. **Operational Efficiency:** Ensure streamlined management of schedules, transactions, and port infrastructure.



System Functionality

The Ship Port Management System operates as a **relational database**, using **structured tables** interconnected via primary and foreign keys. The **eight tables** (Ship, Owner, Port, Dock, Cargo, Schedule, Employee, and Transaction) work together to store essential records while supporting **queries** for retrieval, updating, and analytical operations.

Technologies Used

- **Database Management System:** MySQL or PostgreSQL for relational data storage.
- **SQL Querying:** Structured Query Language (SQL) for performing operations like selection, insertion, updating, deletion, aggregation, and joins.
- **Entity-Relationship Diagram (ERD):** A visual representation of relationships among entities, helping understand dependencies and constraints.
- **Hardware Configuration:** Minimum system requirements include an Intel i5 processor, 8GB RAM, and SSD storage for optimal performance.

Expected Benefits

- ✓ **Improved Accuracy** – Eliminates manual errors in ship scheduling and cargo handling.
- ✓ **Enhanced Security** – Ensures safe storage and retrieval of sensitive port information.
- ✓ **Better Resource Allocation** – Optimizes dock usage and port operations.
- ✓ **Efficient Decision-Making** – Provides quick access to key insights for port managers.

TECHNIQUE

Technique Used

The **Ship Port Management System** is built using a **relational database model**, ensuring structured data storage and efficient management of ship-related operations at ports. The main techniques involved in this project are:

1. Database Management System (DBMS)

- The project utilizes **Relational DBMS (RDBMS)** to store structured data using tables, relationships, and constraints.
- MySQL, PostgreSQL, or SQL Server can be used to implement this system.

2. Entity-Relationship (ER) Modeling

- The system is designed based on **Entity-Relationship Diagram (ERD)** to define relationships between various components like Ships, Owners, Ports, and Cargo.
- **Normalization** is applied to reduce redundancy and improve efficiency.

3. SQL Queries and Transactions

- SQL is used for table creation, data insertion, retrieval, updating, deletion, and performing complex queries.
- The project supports **aggregation (SUM, COUNT, AVG)** and **joins (INNER JOIN, LEFT JOIN, etc.)** for relational data retrieval.

4. Constraints and Data Integrity

- **Primary Keys:** Unique identifiers for each table entity (e.g., ShipID, PortID).
- **Foreign Keys:** Maintain referential integrity (e.g., ShipID in Cargo table references Ship table).
- **Constraints:** NOT NULL, UNIQUE, CHECK, and DEFAULT ensure data accuracy.

5. Data Security and Optimization

- Indexing is used for faster query execution.
- Role-based access control ensures secure handling of sensitive information.

SYSTEM CONFIGURATION

To efficiently run the Ship Port Management System, the following **software and hardware requirements** are recommended:

Software Requirements

- **Database Management System** – MySQL / PostgreSQL / SQL Server
- **Development Environment** – MySQL Workbench / pgAdmin / SQL Server Management Studio
- **Operating System** – Windows 10, Linux (Ubuntu), macOS
- **Programming Language (optional)** – Python, Java (for backend connectivity if needed)

Hardware Requirements

- **Processor:** Intel i5 or above (for smooth query execution)
- **RAM:** Minimum 8GB (Recommended: 16GB for high-performance data handling)
- **Storage:** SSD with at least 256GB for better speed
- **Network:** Stable internet connection for database access (if hosted remotely)

Performance Enhancements

- **Indexing** for faster query execution.
- **Security measures** like role-based access control and constraints (NOT NULL, UNIQUE).
- **Cloud hosting options** (AWS, Azure, Google Cloud) for scalability.

This setup ensures **efficient ship port management**, improving **speed, accuracy, and security** in maritime logistics.



INPUT

1. Ship Details

Ships arriving and departing from the port need to be registered in the database. Inputs include: Ship Name , Ship Capacity (Weight), Ship Owner

2. Owner Details

Each ship is owned by an individual or a company. Inputs for this include: Owner Name ,Contact Information , Address

3. Port Details

Different ports around the world handle ship operations. Inputs include: Port Name , Location (City, Country) ,Port Capacity

4. Dock Details

Ships require docking facilities at ports. Inputs include: Dock Number ,Port to which the dock belongs , Dock Status (Available, Occupied, Under Maintenance)

5. Cargo Details

Ships carry cargo that needs to be tracked. Inputs include: Cargo Type (Oil, Coal, Food, Electronics, etc.) ,Cargo Weight , Ship carrying the cargo

6. Schedule Information

A proper schedule ensures efficient ship operations at ports. Inputs include: Ship ID , Port ID , Arrival Date , Departure Date

7. Employee Details

Employees working at the port must be stored in the system. Inputs include: Employee Name , Position (Manager, Operator, Technician, etc.) , Port ID where they work

8. Transaction Records

Cargo transactions between ships and ports must be logged in the system. Inputs include: Cargo ID , Ship ID ,Transaction Date , Transaction Amount

➤ **Aim/Overview of the project:**

The Ship Port Management System is designed to efficiently manage ships, ports, cargo, transactions, and schedules at a port. It ensures smooth operations by tracking ship arrivals, dock statuses, cargo details, and financial transactions.

➤ **Objective:**

Objectives of Ship Port Management System

- **Efficient Ship Management:**
 - Store and track ship details (capacity, schedules, ownership).
 - Maintain arrival and departure records for better coordination.
- **Cargo Handling and Tracking:**
 - Log cargo type, weight, and ship assignments.
 - Ensure proper tracking of cargo transactions between ships and ports.
- **Port and Dock Utilization Optimization:**
 - Manage dock status (Available, Occupied, Under Maintenance).
 - Prevent congestion by ensuring efficient use of port resources.
- **Employee Management at Ports:**
 - Maintain records of employees working at various ports.
 - Assign roles like managers, technicians, and supervisors.
- **Secure and Organized Financial Transactions:**
 - Keep records of transactions related to cargo shipments.
 - Ensure financial accountability with accurate logging of amounts and dates.
- **Automation of Operations to Improve Efficiency:**
 - Minimize manual errors in ship port operations.
 - Ensure quick retrieval and updating of data through SQL queries.
- **Implementation of Data Integrity and Security:**
 - Utilize primary and foreign keys to maintain referential integrity.
 - Apply constraints (NOT NULL, UNIQUE, CHECK) to ensure reliable data input.

By fulfilling these objectives, the **Ship Port Management System** ensures efficient port operations, accurate data handling, and seamless logistics management.

➤ ER Diagram & Schema

Database Schema

Here's a summary of the schema design:

1. Ship Table:

- **Columns:** ShipID (Primary Key), ShipName, Capacity, OwnerID (Foreign Key).

2. Owner Table:

- **Columns:** OwnerID (Primary Key), OwnerName, Contact, Address.

3. Port Table:

- **Columns:** PortID (Primary Key), PortName, Location, Capacity.

4. Dock Table:

- **Columns:** DockID (Primary Key), PortID (Foreign Key), DockNumber, Status.

5. Cargo Table:

- **Columns:** CargoID (Primary Key), CargoType, Weight, ShipID (Foreign Key).

6. Schedule Table:

- **Columns:** ScheduleID (Primary Key), ShipID (Foreign Key), PortID (Foreign Key), ArrivalDate, DepartureDate.

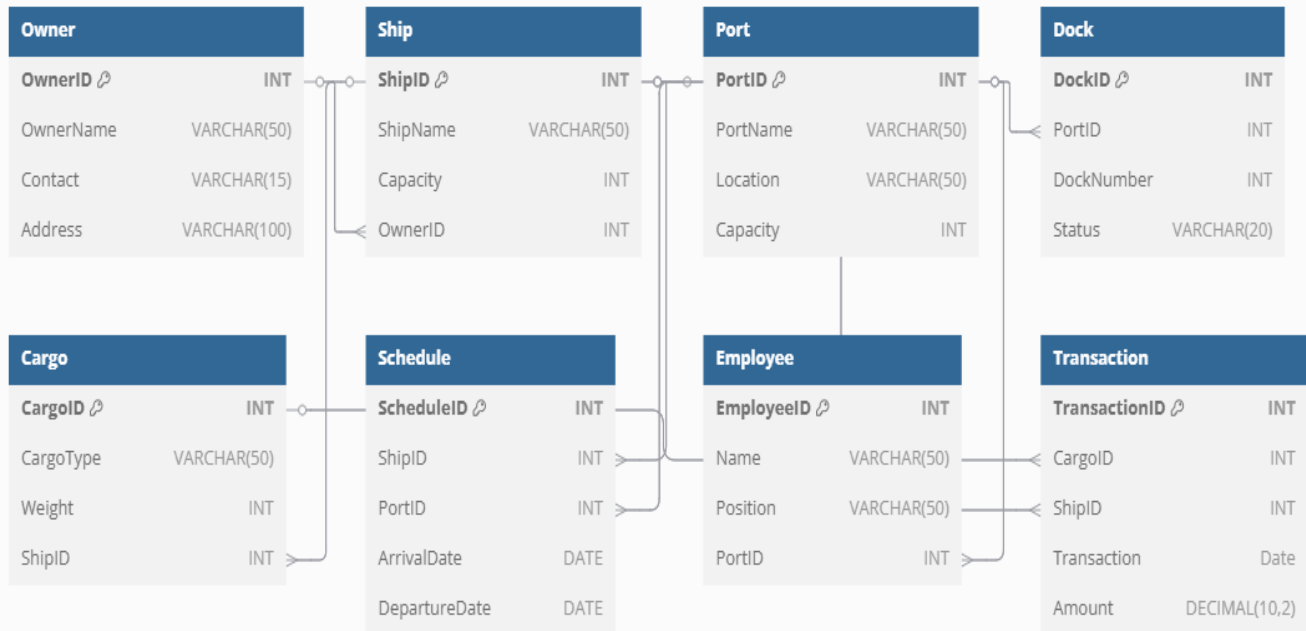
7. Employee Table:

- **Columns:** EmployeeID (Primary Key), Name, Position, PortID (Foreign Key).

8. Transaction Table:

- **Columns:** TransactionID (Primary Key), CargoID (Foreign Key), ShipID (Foreign Key), TransactionDate, Amount.

The schema adheres to **normalization principles**, ensuring **no data redundancy** while maintaining **data integrity** through **primary keys, foreign keys, and constraints**.



Relationships in tabular form

Entities and Their Attributes

- **Ship:** ShipID (PK), ShipName, Capacity, OwnerID (FK)
- **Owner:** OwnerID (PK), OwnerName, Contact, Address
- **Port:** PortID (PK), PortName, Location, Capacity
- **Dock:** DockID (PK), PortID (FK), DockNumber, Status
- **Cargo:** CargoID (PK), CargoType, Weight, ShipID (FK)
- **Schedule:** ScheduleID (PK), ShipID (FK), PortID (FK), ArrivalDate, DepartureDate
- **Employee:** EmployeeID (PK), Name, Position, PortID (FK)
- **Transaction:** TransactionID (PK), CargoID (FK), ShipID (FK), TransactionDate, Amount

Relationships

- **Ship ↔ Owner:** Many-to-One (Each ship belongs to one owner, but an owner can have multiple ships)
- **Ship ↔ Cargo:** One-to-Many (A ship can carry multiple cargo items, but each cargo belongs to only one ship)
- **Ship ↔ Schedule ↔ Port:** Many-to-Many (A ship can visit multiple ports, and ports receive multiple ships via schedules)
- **Port ↔ Dock:** One-to-Many (A port has multiple docks, but each dock belongs to only one port)
- **Port ↔ Employee:** One-to-Many (Each port has multiple employees managing operations)
- **Cargo ↔ Transaction ↔ Ship:** Many-to-Many (A ship can have multiple transactions for different cargo items)

This **relational structure** ensures **efficient data handling**, eliminating redundancy while maintaining **referential integrity** through **primary and foreign keys**.

Table Name	Primary Key (PK)	Foreign Key (FK)	Constraints
Ship	ShipID	OwnerID	NOT NULL, UNIQUE (ShipID)
Owner	OwnerID	None	NOT NULL, UNIQUE (OwnerID)
Port	PortID	None	NOT NULL, UNIQUE (PortID)
Dock	DockID	PortID	NOT NULL, UNIQUE (DockID), CHECK(Status IN ('Available', 'Occupied', 'Under Maintenance'))
Cargo	CargoID	ShipID	NOT NULL, UNIQUE (CargoID)
Schedule	ScheduleID	ShipID, PortID	NOT NULL, UNIQUE (ScheduleID)
Employee	EmployeeID	PortID	NOT NULL, UNIQUE (EmployeeID)
Transaction	TransactionID	CargoID, ShipID	NOT NULL, UNIQUE (TransactionID), Amount DECIMAL(10,2) CHECK(Amount >0)

Table Creation

CREATE TABLE Owner (

OwnerID INT PRIMARY KEY,

OwnerName VARCHAR(50),

Contact VARCHAR(15),

Address VARCHAR(100)

);

CREATE TABLE Ship (

ShipID INT PRIMARY KEY,

ShipName VARCHAR(50),

Capacity INT,

OwnerID INT,

FOREIGN KEY (OwnerID) REFERENCES Owner(OwnerID)

);

CREATE TABLE Port (

PortID INT PRIMARY KEY,

PortName VARCHAR(50),

Location VARCHAR(50),

Capacity INT

);

```
CREATE TABLE Dock (  
    DockID INT PRIMARY KEY,  
    PortID INT,  
    DockNumber INT,  
    Status VARCHAR(20),  
    FOREIGN KEY (PortID) REFERENCES Port(PortID)  
);  
  
CREATE TABLE Cargo (  
    CargoID INT PRIMARY KEY,  
    CargoType VARCHAR(50),  
    Weight INT,  
    ShipID INT,  
    FOREIGN KEY (ShipID) REFERENCES Ship(ShipID)  
);  
  
CREATE TABLE Schedule (  
    ScheduleID INT PRIMARY KEY,  
    ShipID INT,  
    PortID INT,  
    ArrivalDate DATE,  
    DepartureDate DATE,  
    FOREIGN KEY (ShipID) REFERENCES Ship(ShipID),  
    FOREIGN KEY (PortID) REFERENCES Port(PortID)
```

);

CREATE TABLE Employee (

EmployeeID INT PRIMARY KEY,

Name VARCHAR(50),

Position VARCHAR(50),

PortID INT,

FOREIGN KEY (PortID) REFERENCES Port(PortID)

);

CREATE TABLE Transaction (

TransactionID INT PRIMARY KEY,

CargoID INT,

ShipID INT,

TransactionDate DATE,

Amount DECIMAL(10,2),

FOREIGN KEY (CargoID) REFERENCES Cargo(CargoID),

FOREIGN KEY (ShipID) REFERENCES Ship(ShipID)

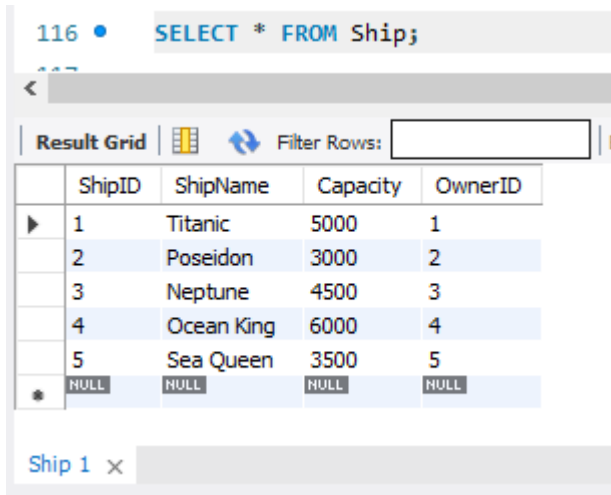
);

➤ SQL Queries & Output

1) Retrieve all ships from the database:

Query:

```
SELECT * FROM Ship;
```



116 • SELECT * FROM Ship;

Result Grid | Filter Rows:

	ShipID	ShipName	Capacity	OwnerID
▶	1	Titanic	5000	1
	2	Poseidon	3000	2
	3	Neptune	4500	3
	4	Ocean King	6000	4
	5	Sea Queen	3500	5
*	NULL	NULL	NULL	NULL

Ship 1 x

2) Insert a new ship into the database:

Query:

```
INSERT INTO Ship (ShipID, ShipName, Capacity, OwnerID)  
VALUES (6, 'Blue Wave', 5500, 3);
```

✓ 52 12:38:39 INSERT INTO Ship (ShipID, ShipName, Capacity, OwnerID) VALUES (6, 'Blue Wave', 5500, 3) 1 row(s) affected

3) Update the capacity of a specific ship:

Query:

UPDATE Ship SET Capacity = 6000 WHERE ShipID = 3;

53 12:39:24 UPDATE Ship SET Capacity = 6000 WHERE ShipID = 3 1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0

Result Grid Filter Rows: Edit:

	ShipID	ShipName	Capacity	OwnerID
▶	1	Titanic	5000	1
	2	Poseidon	3000	2
	3	Neptune	6000	3
	4	Ocean King	6000	4
	5	Sea Queen	3500	5
	6	Blue Wave	5500	3
*	NULL	NULL	NULL	NULL

Ship 2 x

5) Delete a ship record from the database.

Query:

DELETE FROM Transaction WHERE ShipID = 5;

57 12:47:23 DELETE FROM Transaction WHERE ShipID = 5 1 row(s) affected

Result Grid Filter Rows: Edit: Export/Impo

	TransactionID	CargoID	ShipID	TransactionDate	Amount
▶	1	1	1	2025-04-15	50000.00
	2	2	2	2025-04-10	70000.00
	3	3	3	2025-04-12	90000.00
	4	4	4	2025-04-14	65000.00
*	NULL	NULL	NULL	NULL	NULL

Transaction 3 x

6) Find the Latest Scheduled Ship (ORDER BY + LIMIT):

Query:

```
SELECT * FROM Schedule
ORDER BY ArrivalDate DESC
LIMIT 1;
```

ScheduleID	ShipID	PortID	ArrivalDate	DepartureDate
5	5	5	2025-04-17	2025-04-23
* NULL	NULL	NULL	NULL	NULL

Schedule 4 x

7) Retrieve ship details along with their owner names using a join:

Query:

```
SELECT Ship.ShipName, Owner.OwnerName
FROM Ship
JOIN Owner ON Ship.OwnerID = Owner.OwnerID;
```

ShipName	OwnerName
Titanic	John Smith
Poseidon	Emma Davis
Neptune	Carlos Mendes
Blue Wave	Carlos Mendes
Ocean King	Ava Johnson
Sea Queen	Leo Brown

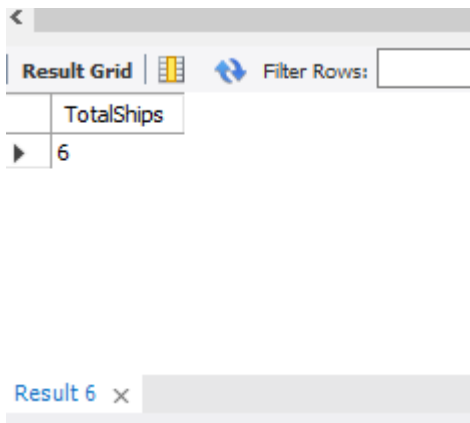
Result 5 x

Output

8) Count the total number of ships:

Query:

SELECT COUNT(*) AS TotalShips FROM Ship;



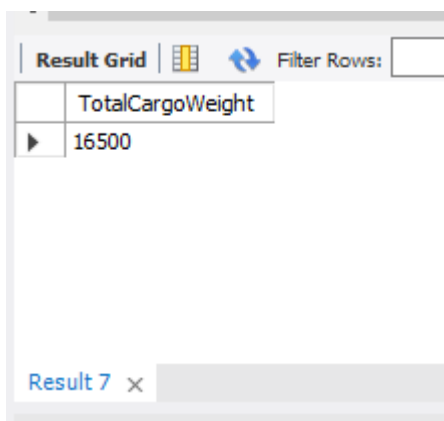
The screenshot shows a SQL query result grid. The header row is labeled 'TotalShips'. The data row shows the value '6'. The interface includes a 'Result Grid' tab, a 'Filter Rows' button, and a tab labeled 'Result 6'.

TotalShips
6

9) Calculate the total cargo weight handled by all ships:

Query:

SELECT SUM(Weight) AS TotalCargoWeight FROM Cargo;



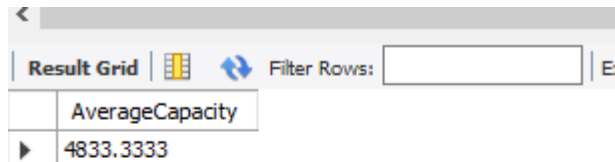
The screenshot shows a SQL query result grid. The header row is labeled 'TotalCargoWeight'. The data row shows the value '16500'. The interface includes a 'Result Grid' tab, a 'Filter Rows' button, and a tab labeled 'Result 7'.

TotalCargoWeight
16500

10) Get the average ship capacity in the database:

Query:

SELECT AVG(Capacity) AS AverageCapacity FROM Ship;



AverageCapacity
4833.3333

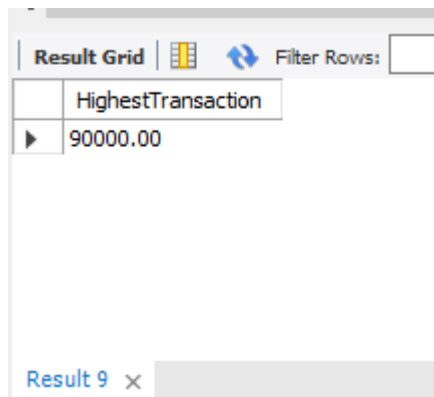


Output

11) Find the highest transaction amount :

Query:

SELECT MAX(Amount) AS HighestTransaction FROM Transaction;

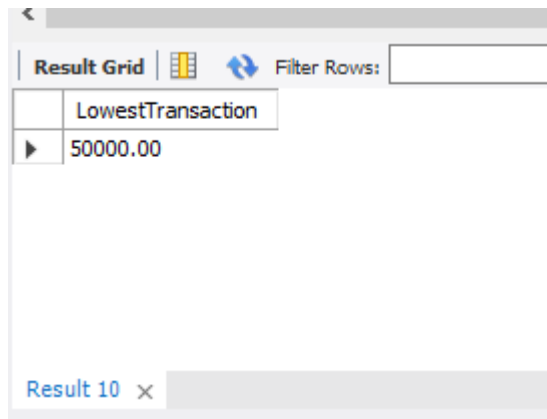


HighestTransaction
90000.00

12)Find the Lowest Transaction Amount:

Query:

SELECT MIN(Amount) AS LowestTransaction FROM Transaction;



Result Grid | Filter Rows:

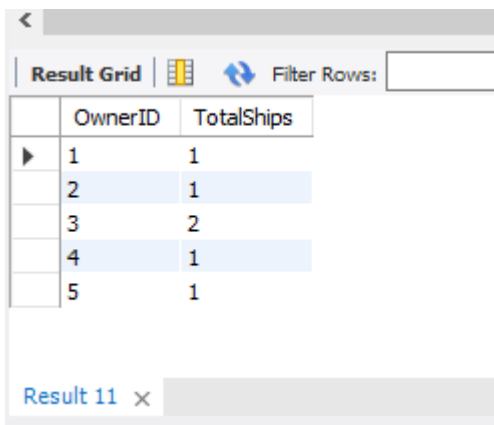
	LowestTransaction
▶	50000.00

Result 10 x

13) Count the number of ships each owner has:

Query:

```
SELECT OwnerID, COUNT(ShipID) AS TotalShips  
FROM Ship  
GROUP BY OwnerID;
```



Result Grid | Filter Rows:


	OwnerID	TotalShips
▶	1	1
	2	1
	3	2
	4	1
	5	1

Result 11 x

14) Retrieve ports with more than 2 docks:

Query:

```
SELECT PortID, COUNT(DockID) AS TotalDocks  
FROM Dock  
GROUP BY PortID  
HAVING COUNT(DockID) > 2;
```


Result Grid |  Filter Rows:




PortID	TotalDocks
--------	------------

Result 14 ×

15) Retrieve ship and port details for scheduled arrivals using joins:

Query:

```
SELECT Ship.ShipName, Port.PortName, Schedule.ArrivalDate,
Schedule.DepartureDate
FROM Schedule
JOIN Ship ON Schedule.ShipID = Ship.ShipID
JOIN Port ON Schedule.PortID = Port.PortID;
```

Result Grid |  Filter Rows: | Export:  | Wrap Cell Content: 

	ShipName	PortName	ArrivalDate	DepartureDate
▶	Titanic	New York Port	2025-04-15	2025-04-20
	Poseidon	London Port	2025-04-10	2025-04-18
	Neptune	Lisbon Port	2025-04-12	2025-04-19
	Ocean King	Sydney Port	2025-04-14	2025-04-21
	Sea Queen	Tokyo Port	2025-04-17	2025-04-23

Result 15 ×

SUMMARY

Summary of Ship Port Management System

The **Ship Port Management System** is a relational database project designed to streamline port operations by efficiently managing **ships, cargo, docks, schedules, employees, and financial transactions**. The project follows **structured schema design principles** with tables interconnected through **primary and foreign keys** to ensure data integrity and optimized functionality.

Key Features:

- **Ships Management:** Registers ships, their capacity, and ownership details.
- **Cargo Tracking:** Logs cargo type, weight, and ship assignments.
- **Port & Dock Utilization:** Stores port details and tracks dock availability.
- **Scheduling System:** Records arrival and departure times of ships at various ports.
- **Employee Management:** Maintains staff working at each port.
- **Financial Transactions:** Logs cargo shipment transactions and amounts involved.
- **Aggregate Queries:** Supports analytics using SQL functions like COUNT, SUM, AVG.
- **Joins & Relationships:** Ensures efficient data retrieval with interlinked tables.

Technical Overview:

- **Database Schema:** Includes **eight tables** defining ship operations and relationships.
- **Normalization & Integrity:** Ensures **data consistency, reducing redundancy** through **primary and foreign keys**.
- **SQL Queries:** Supports **selection, insertion, deletion, updating, aggregation functions (COUNT, SUM, AVG), and joins** for efficient data retrieval.
- **System Configuration:** Utilizes **MySQL/PostgreSQL**, deployed on **Windows/Linux/macOS** with recommended hardware for **optimal performance**.

➤ Conclusion:

The **Ship Port Management System** helps port authorities manage ships, cargo, and transactions efficiently. By implementing a **structured relational database**, this project provides **seamless operations, accurate record-keeping, and improved decision-making** through SQL queries and aggregate functions.

Through **structured queries, normalization techniques, and optimized indexing**, the system provides **faster access to ship schedules, cargo movement, and financial transactions**, enabling **port authorities to make well-informed decisions**.

While the system **successfully improves operational efficiency**, it has **limitations**, such as the **lack of real-time GPS tracking and automation for notifications**. However, these features can be incorporated in future enhancements to create a **more advanced and scalable port management solution**.

Overall, this **Ship Port Management System** is a **powerful foundation for modernizing maritime logistics**, ensuring **accurate data handling, security, and resource optimization** for efficient port management.

This system successfully:

- ✓ **Enhances ship tracking and scheduling efficiency.**
- ✓ **Ensures accurate cargo management and transaction logging.**
- ✓ **Optimizes dock availability to prevent congestion at ports.**
- ✓ **Supports employee management at various port locations.**
- ✓ **Eliminates redundancy using normalization techniques.**

Overall, this **database-driven approach modernizes port operations**, making management faster, more reliable, and highly structured.



Observations

Efficiency Gains: The system improves **cargo tracking, dock management, and scheduling accuracy**, reducing logistical errors.

Data Integrity: Use of **primary and foreign keys** ensures relational integrity, preventing inconsistencies across tables.

Scalability: The database design allows for **future expansion**, accommodating new ports, employees, and ship transactions.

User-Friendly Queries: SQL queries enable **quick data retrieval**, ensuring fast decision-making in ship port management.

Security Measures: Data constraints (NOT NULL, UNIQUE, CHECK) prevent invalid entries and unauthorized modifications.

Limitations

Manual Data Entry Required: Initial setup requires entering ship, port, cargo, and employee data manually.

🔧 **Dependency on SQL:** Users must be familiar with **SQL query execution** for accessing data effectively.

🔧 **No Real-Time Tracking:** This system does not integrate **live GPS tracking** for ship movements.

🔧 **Limited Automation:** The database supports ship scheduling but does not automate **notifications or alerts**.

🔧 **Hardware Requirements:** Large-scale implementation may require **high-performance servers** for efficient data management.