CSCU9B3 Relational Database Assignment Spring 2023

**Report**

**Diagram

Description automatically generatedQ1)**

**Q2)**

The Boat table only stores information about the boats themselves, while the BookingParty table only stores information about the parties that make the bookings. This not only helps to reduce redundancy but also makes it easier to maintain data integrity since it is clear what data belongs where.

The use of foreign keys in the design also helps to maintain data integrity by enforcing referential integrity. Each foreign key in the design references a primary key in another table, which means that it must refer to a valid row in that table. This ensures that we cannot add a booking for a boat that does not exist or assign a skipper or crew member who does not have a valid person number.

In terms of normalisation, the design is in 3NF, which means that it eliminates all transitive dependencies and redundancy from the original BoatBooking table. This ensures that the data is stored efficiently and without any unnecessary duplication or inconsistencies. In addition, the design can be easily extended to support additional functionality, such as tracking maintenance or payments, without introducing any new data redundancy.

By storing data in separate tables based on their functional dependencies, we can reduce the likelihood of errors that can occur when data is duplicated across multiple tables. For example, if we had stored the skipper and crew member information directly in the BoatBooking table, we might accidentally enter different email addresses for the same person in different bookings.

Since the data is split into multiple tables, we can join the tables together based on their foreign keys to retrieve information from multiple tables at once. For example, we can easily retrieve the name and address of the skipper and crew members for a particular booking by joining the BookingParty table with the Skipper and Crew tables.

In conclusion, the design presented above ensures data integrity, reduces redundancy, and allows for efficient querying of the data, while also being highly normalised.

**Q3)**

CREATE TABLE WeekBooking (

WeekNo INT PRIMARY KEY,

BoatName VARCHAR(20),

FOREIGN KEY (BoatName) REFERENCES Boat(BoatName)

);

CREATE TABLE Boat (

BoatName VARCHAR(20) PRIMARY KEY,

BoatLength INT,

ExtraRequest VARCHAR(20)

);

CREATE TABLE Skipper (

SkipperPno INT PRIMARY KEY,

SkipperName VARCHAR(40),

SkipperAddress VARCHAR(40)

);

CREATE TABLE Crew (

CrewPno INT PRIMARY KEY,

CrewName VARCHAR(40),

CrewAddress VARCHAR(40)

);

CREATE TABLE BookingParty (

WeekNo INT,

SkipperPno INT,

CrewPno INT,

BoatName VARCHAR(20),

PRIMARY KEY (WeekNo, SkipperPno),

FOREIGN KEY (WeekNo) REFERENCES WeekBooking(WeekNo),

FOREIGN KEY (SkipperPno) REFERENCES Skipper(SkipperPno),

FOREIGN KEY (CrewPno) REFERENCES Crew(CrewPno),

FOREIGN KEY (BoatName) REFERENCES Boat(BoatName) );

**Q5)**

INSERT INTO WeekBooking (WeekNo, BoatName)

SELECT DISTINCT WeekNo, BoatName

FROM nohead

WHERE NOT EXISTS (

SELECT 1

FROM WeekBooking

WHERE WeekNo = nohead.WeekNo

);

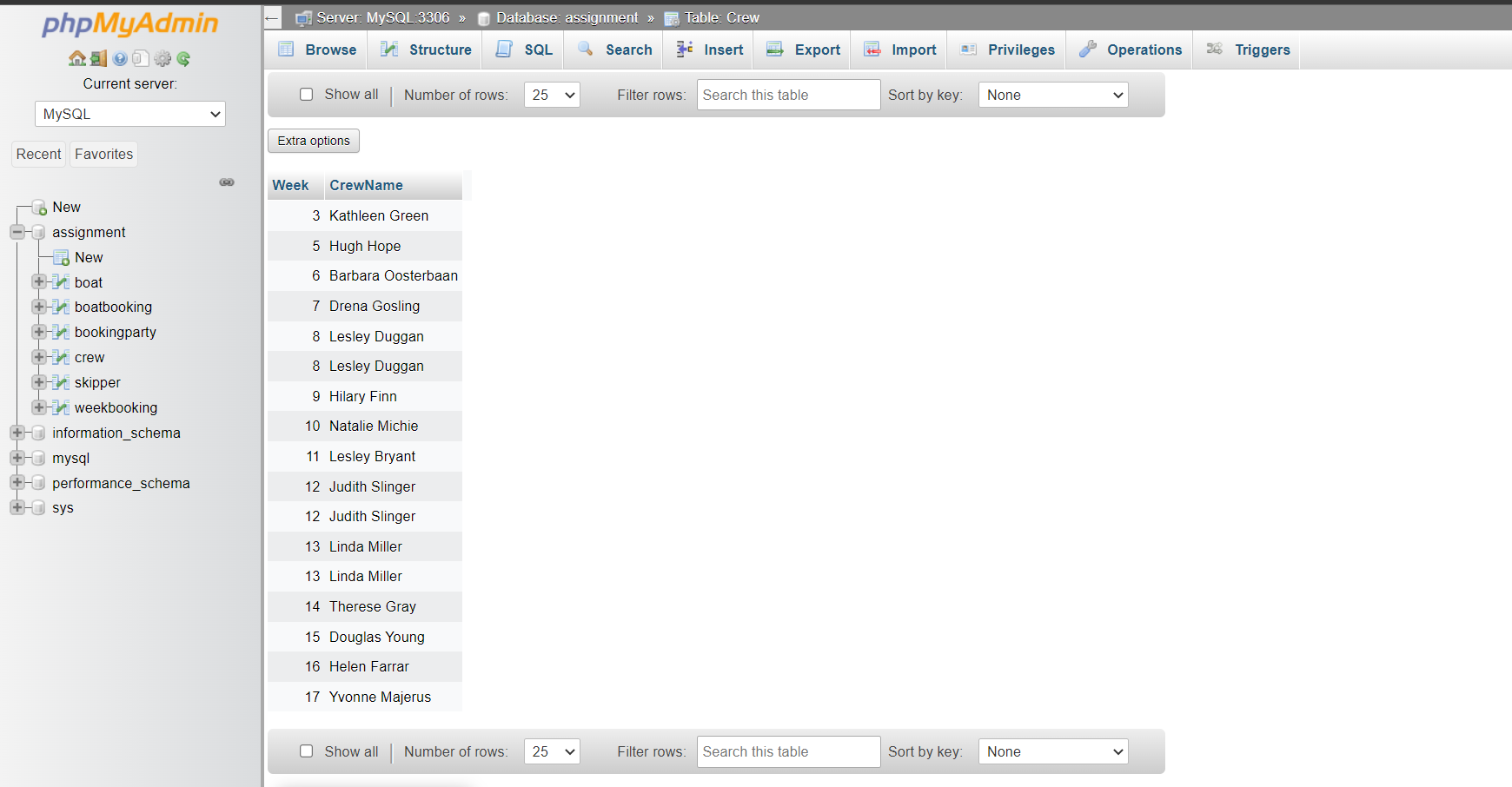
**Q6a)**

SELECT Week, Crew.CrewName

FROM BoatBooking

INNER JOIN Crew ON BoatBooking.CrewPno = Crew.CrewPno

WHERE BoatName = 'Janet';



**Q6b)**

SELECT WeekNo, COUNT(DISTINCT WeekNo) AS NumBookings

FROM bookingparty

GROUP BY WeekNo;

Graphical user interface, application, Word

Description automatically generated

**Q6c)**

SELECT BoatName, SUM(IF(ExtraRequest = '(no extras)', 0, 1)) AS NumExtras

FROM BoatBooking

GROUP BY BoatName

ORDER BY NumExtras DESC;

Graphical user interface, application, Word

Description automatically generated

**6d i)**

SELECT

CASE WHEN boat.BoatLength <= 20 THEN 'Shorter boats' ELSE 'Longer boats' END AS BoatType,

COUNT(\*) AS NumBookings

FROM

BoatBooking

JOIN Boat ON Boat.BoatName = BoatBooking.BoatName

GROUP BY

BoatType;

Graphical user interface, text, application, email

Description automatically generated

**6d ii)**

SELECT

ROUND(AVG(ExtraRequest), 2) AS AvgExtras,

COUNT(\*) AS NumBookings,

boatbooking.CrewPno + boatbooking.SkipperPno AS NumPeople

FROM

BoatBooking

JOIN BookingParty ON BoatBooking.CrewPno = BookingParty.CrewPno AND BoatBooking.SkipperPno = BookingParty.SkipperPno

WHERE

ExtraRequest != '(no extras)'

GROUP BY

NumPeople

ORDER BY

NumPeople ASC;

Graphical user interface, table

Description automatically generated

**Q7)**

<html>

<head>

<title>Assignment Template</title>

</head>

<body>

<form action="assignment.php" method="post">

Search player name: <input type="text" name="name">

<input type="submit">

</form>

<?php

$servername = "localhost";

$username = "root";

$password = "";

$database = "assignment";

// Create connection

$conn = mysqli\_connect($servername, $username, $password, $database);

// Check connection

if (!$conn) {

die("Connection failed: " . mysqli\_connect\_error());

}

// Check if form has been submitted

if ($\_SERVER['REQUEST\_METHOD'] == 'POST') {

$search\_term = $\_POST['name'];

$search\_query = "SELECT \* FROM BoatBooking WHERE skippername LIKE '%$search\_term%' OR crewname LIKE '%$search\_term%' ORDER BY skippername";

$result = mysqli\_query($conn, $search\_query);

// Display search results

if ($result && mysqli\_num\_rows($result) > 0) {

echo "<h2>Search Results:</h2>";

echo "<table><tr><th>Skipper Name</th><th>Skipper Address</th><th>Boat Name</th><th>Week</th></tr>";

while($row = mysqli\_fetch\_assoc($result)) {

if ($row['skippername'] == '') {

echo "<tr><td>".$row['crewname']."</td><td>".$row['crewaddress']."</td><td>Not a skipper</td><td></td></tr>";

} else {

echo "<tr><td>".$row['skippername']."</td><td>".$row['skipperaddress']."</td><td>".$row['boatname']."</td><td>".$row['week']."</td></tr>";

}

}

echo "</table>";

} else {

echo "<p>No results found.</p>";

}

mysqli\_close($conn);

}

?>

</body>

</html>