PROJECT PART II

NICE Cruise Case Study

TEAM: EAM

MEMBERS:

Abhishek Koti (ak11546)

Ehtesham Siddiqui (es6663)

Meghna Sharma (ms16005)

Course Name:

Principles of Database Systems

Section B

CS GY-6083

Date of Submission: **December 8, 2024**

TABLE OF CONTENTS

SUMMARY

The NICE (Nature International Cruise Excellence) project involves developing a database-driven web application to streamline the operations of a cruise line company. This project is divided into two phases. In the first phase, we created a robust relational database schema to capture essential business information about cruises, including staterooms, restaurants, activities, ports, and customers. The second phase focuses on implementing a web-based frontend to interact with this database, providing functionality for user registration, data management, and enhanced business operations.

Business Case: NICE operates cruises from 35 ports in the USA, Canada, Mexico, and the Caribbean, offering a variety of staterooms, dining experiences, and entertainment options. Managing this complex array of services, customer data, and logistics is challenging without an integrated system. The proposed solution addresses these challenges by introducing a centralized database and web interface.

APPROACH:

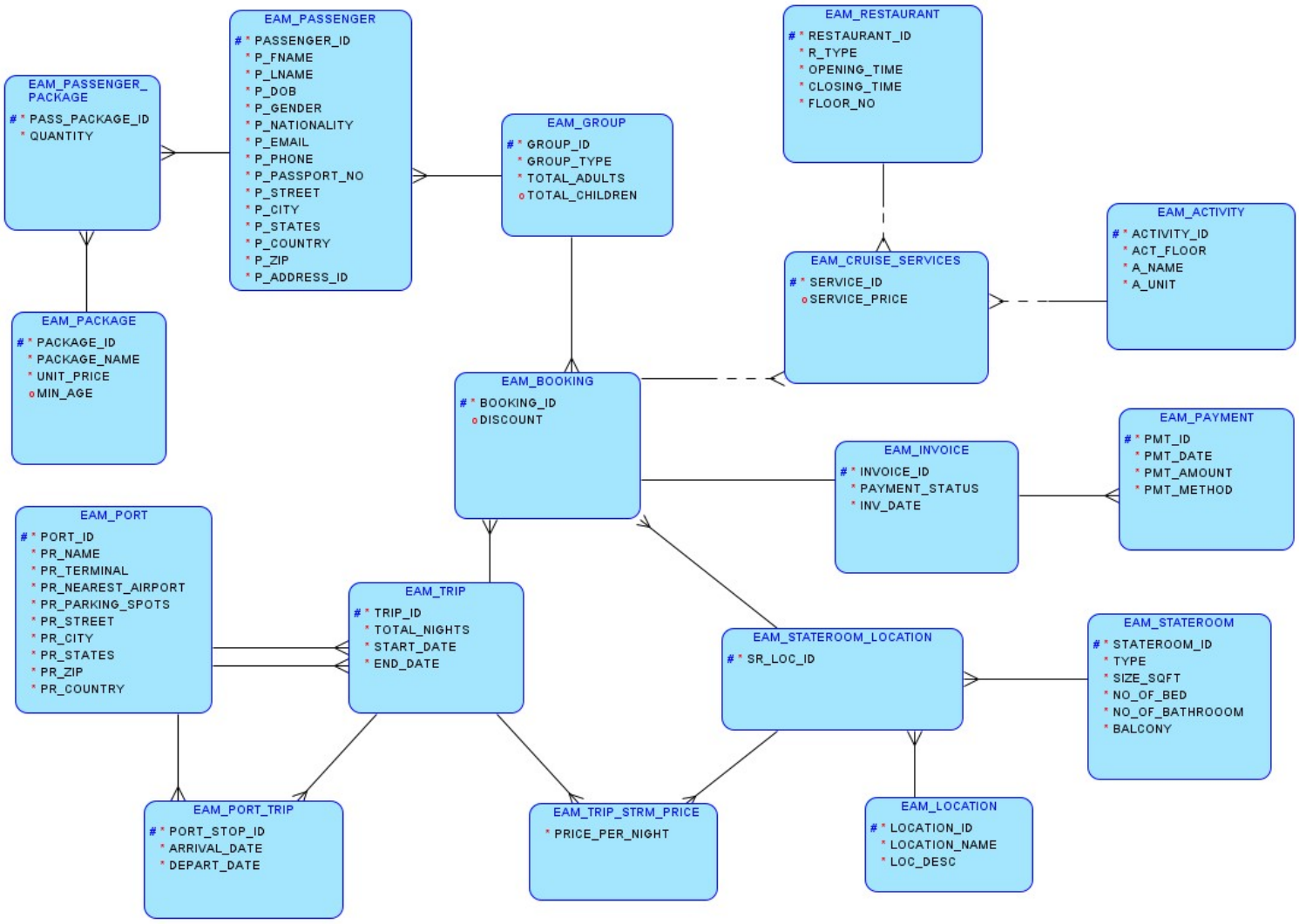
- Database Design: In Phase I, a detailed Entity-Relationship (ER) model was created to represent the system's data requirements. This included defining entities such as passengers, trips, staterooms, ports, and payment transactions, with relationships between them. Constraints and validation rules were implemented to ensure data consistency and integrity.
- Web-Based Interface: In Phase II, a **RESTful web application** was built using modern frameworks like **PHP** and **JavaScript**, to allow customers and employees to interact with the database. Key features include **user authentication**, **CRUD operations**, and **secure Session handling**. Security measures such as **input sanitization**, **SQL injection prevention**, **encrypted password storage**, **Cross-Site Scripting (XSS) Protection**, and other security features were implemented.
- Business Analytics: The system supports analytical queries, enabling insights into customer behavior, trip popularity, and revenue trends.

Business Impact: This solution enhances business performance in several ways:

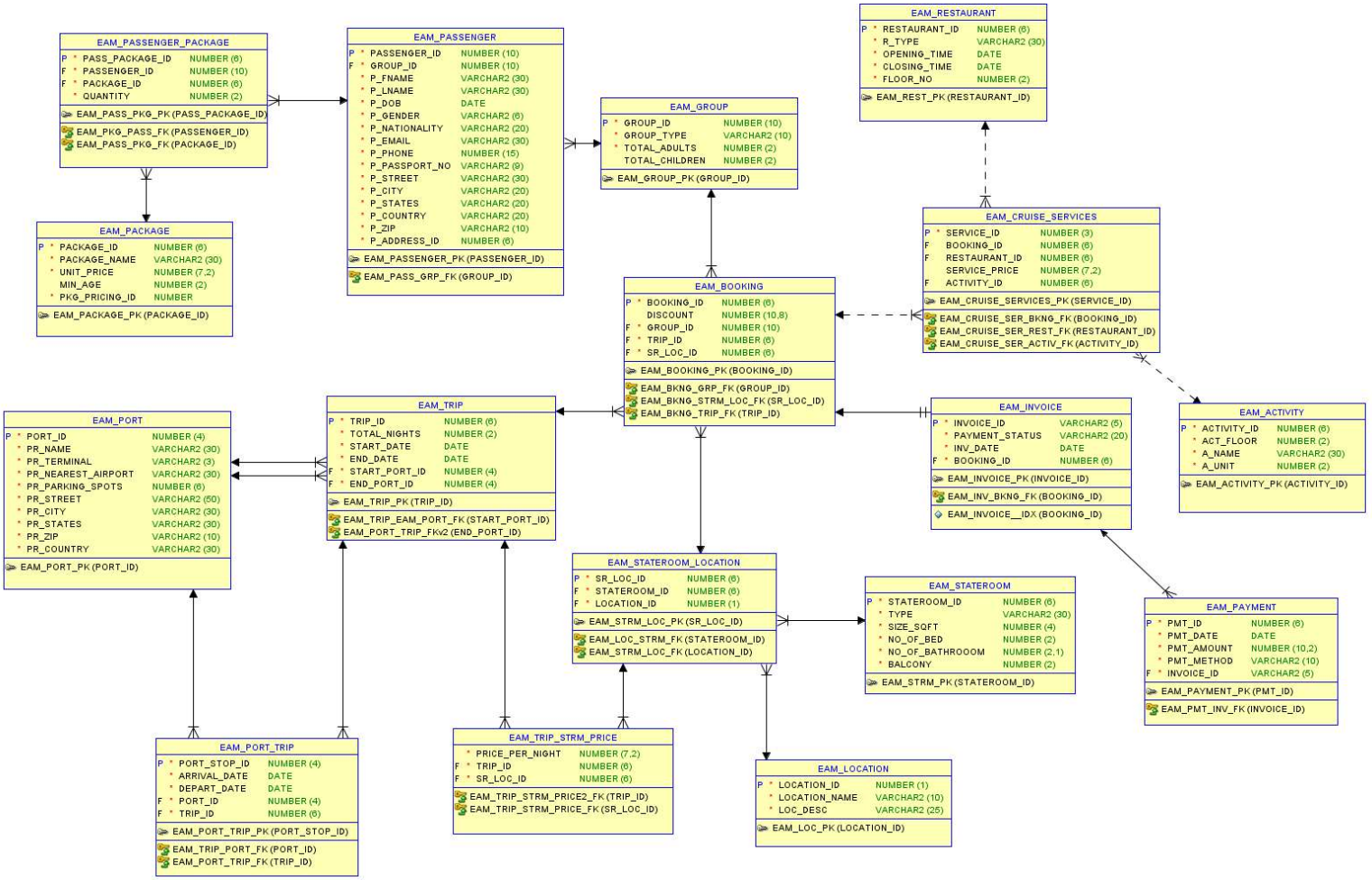
1. **Operational Efficiency:** Automating processes such as customer registration, booking management, and payment tracking reduces manual errors and administrative overhead.
2. **Improved Customer Experience:** A user-friendly interface allows customers to seamlessly book cruises, select packages, and view itineraries.
3. **Data-Driven Decision-Making:** The integration of analytics supports better decision-making by providing insights into occupancy rates, revenue streams, and customer preferences.

4. **Scalability and Security:** The robust database design and secure web application architecture ensure the system can handle increased demand as the business grows.

LOGICAL DESIGN



RELATIONAL DESIGN



ASSUMPTIONS FOR DESIGN

- Each passenger (EAM_PASSENGER) is linked to an EAM_GROUP through GROUP_ID. Individual Passengers are part of their own group "Solo". Each passenger is part of a single group and there can be many passengers in each group, forming a one-many relationship.
- A passenger links to the addresses through P_ADDRESS_ID as address is a multivalued attribute and has been resolved into a separate entity EAM_Pass_Address.
- Groups linked to bookings through a 1:M relationship. A group can have multiple bookings while each booking belongs to only one group.
- Each booking generates invoices having details about payment status and the invoice date.
- Invoice is linked to Payments. There can be multiple payments for a single invoice.
- Packages are associated with passengers through eam_passenger_package, since it is a many-to-many relationship. The passenger_package entity is an associative entity created to resolve the M:N relationship.
- For every package, a pricing scheme is given in EAM_PKG_PRICING which shows if the package is offered per night or for the entire trip. (This has been resolved as a separate entity as there are a few packages offered which are charged for the entire trip).
- Staterooms are linked with locations on the cruise, with EAM_STATEROOM_LOCATION giving stateroom locations and pricing variations. The cost of the stateroom varies as per the location.
- Since the stateroom price varies from trip to trip, we have included a Table Trip_strm_payment for stateroom prices per night which vary from trip to trip. This can decide the price for each trip.
- This table along with stateroom_location are taken along with stateroom as this makes it easier and more efficient and also expandable, to alter price per trip through booking instead of altering multiple attributes every time price changes with each trip.
- The trip price multiplier is stored in fractions and is multiplied to the base price as per the trip requirements. It is made optional as there may be no increment in a price for a particular trip.
- The EAM_PORT_TRIP table is an associative entity used to resolve the many-to-many relationship between ports and trips. As one port can have many trips and each trip will have many ports.
- The Total_Children attribute in Group entity indicates all the children under the age of 5. This is made optional as not all passengers may travel with children.
- The total amount to be paid, since it is a derived attribute, has not been mentioned in booking directly. It is calculated using Total Price = Total Packages Cost + (Stateroom Price per Night × Total Nights) × (total_adults). These values are taken from the respective foreign keys in the booking table.
- Age is a derived attribute in passengers and can be calculated using DOB.
- Separate Tax attribute has not been mentioned because we assume that the prices listed are all inclusive of taxes.
- The Final payable amount is all calculated in USD.

SOFTWARE USED:

IDE Code Editor: Visual Studio Code
Database Developer Tool: MySQL Workbench
Host Server Development Environment: XAMPP
Modelling: Oracle SQL Data Modeller
Host Server: Apache (PHP)

PROGRAMMING LANGUAGES USED:

- **HTML -** HTML structures the web pages, defining the content and layout of the application's interface. As a markup language, it forms the foundation for creating elements like forms, tables, headings, and navigation menus, providing a logical and semantic structure that's easy to maintain and extend. It bridges backend functionality with the visual presentation seen by users.
- **CSS -** CSS is responsible for styling and designing the application, making it visually appealing and user-friendly. It enhances the layout with responsive grids, flexible forms, and attractive color schemes. By ensuring responsiveness, CSS adapts the interface seamlessly to various devices and screen sizes, providing a consistent user experience across desktops, tablets, and mobile devices.
- **PHP -** The main server-side scripting language used in this project drives the backend logic, enabling dynamic content generation, user authentication, and database interaction. Its seamless integration with MySQL and ability to handle tasks like form submissions, session data, and data validation make it ideal for this purpose. Built-in functions like **password_hash()** further enhance the application's security, particularly for secure password storage.
- **MySQL -** This database management system is used to store and organize application data, including user accounts, bookings, trip details, and payment records. Its relational structure ensures efficient data storage and retrieval, with support for complex queries to handle various system functionalities. MySQL stored procedures are employed to encapsulate SQL logic within the database, enhancing security by mitigating SQL injection risks and improving consistency in database interactions.
- **JavaScript (for Statistics) -** JavaScript is used again for dynamically showing statistics about passengers in the application. The application utilizes JavaScript to manipulate the DOM and fetch and process data dynamically to display visual metrics through graphs, charts, or tables. These statistics are useful for admins to gain insights into aspects like passengers per trip, age group distribution, or other analytical details. Libraries and tools like Chart.js or D3.js can also be used to create graphics, improving the user experience by establishing a pleasing visual display of complex data.
- **Content Security Policy (CSP) -** CSP strengthens application security by preventing cross-site scripting (XSS) attacks. It restricts the sources from which scripts, styles, and

other content can be loaded, ensuring that only trusted resources are executed. This protects sensitive user data and prevents unauthorized script injections.

- **XAMPP -** XAMPP serves as the local development environment, integrating Apache (web server), MySQL (database), PHP (server-side scripting), and other tools into one package. It facilitates local development and testing of the application before deployment, streamlining the development process. The inclusion of phpMyAdmin simplifies database management tasks.

DATABASE USED:

MySQL has been used as the database for this application because it is reliable and scalable, with great support for relational data management. MySQL is used to store and organize all key information regarding user accounts, bookings, details of trips, records of payments, etc., for smooth retrieval via optimized queries and indexing. It integrates perfectly with PHP, adding a lot to the power of back-end development. Such features as Stored Procedures enhance security and performance by providing an extra layer against SQL injection. MySQL enforces the integrity of data through relational constraints and allows scalability to meet the demands of more and more users. Security includes such measures as encryption of sensitive data and role-based access control, ensuring users' information is not breached, while regular backups ensure the safety of data and its recoverability in case of system failure. Thus, MySQL is ideal for the application's database needs.

DATABASE CODE - MySQL

DDL Code in MySQL:

(*Originally Generated using data modeler, minor additions made manually and converted from Oracle SQL to MySQL)

```
-- Generated by Oracle SQL Developer Data Modeler 23.1.0.087.0806


-- SQLINES FOR EVALUATION USE ONLY

CREATE TABLE eam_activity (
    activity_id INT NOT NULL COMMENT 'UNIQUE ID FOR EVERY ACTIVITY',
    act_floor   TINYINT NOT NULL COMMENT 'FLOOR NUMBER ON WHICH THE ACTIVITY TAKES PLACE',
    a_name      VARCHAR(30) NOT NULL COMMENT 'ACTIVITY NAME ',
    a_unit      TINYINT NOT NULL COMMENT 'NUMBER OF UNITS IN ACTIVITIES'
);


ALTER TABLE eam_activity ADD CONSTRAINT eam_activity_pk PRIMARY KEY ( activity_id );
```

```
CREATE TABLE eam_booking (
    booking_id INT NOT NULL COMMENT 'UNIQUE BOOKING ID FOR EVERY BOOKING',
    discount   DECIMAL(10, 8) COMMENT 'DISCOUNT GIVEN IF ANY',
    group_id   BIGINT NOT NULL,
    trip_id    INT NOT NULL,
    sr_loc_id  INT NOT NULL
);


ALTER TABLE eam_booking ADD CONSTRAINT eam_booking_pk PRIMARY KEY ( booking_id );

CREATE TABLE eam_cruise_services (
    service_id    SMALLINT NOT NULL COMMENT 'ID OF THE SERVICES TAKEN',
    booking_id    INT,
    restaurant_id INT,
    service_price DECIMAL(7, 2) COMMENT 'PRICE TO BE PAID IF ANY',
    activity_id   INT
);


ALTER TABLE eam_cruise_services ADD CONSTRAINT eam_cruise_services_pk PRIMARY KEY ( service_id
);

CREATE TABLE eam_group (
    group_id       BIGINT NOT NULL COMMENT 'UNIQUE GROUP ID FOR EVERY GROUP',
    group_type     VARCHAR(10) NOT NULL COMMENT 'TYPE OF GROUP ON A TRIP',
    total_adults   TINYINT NOT NULL COMMENT 'TOTAL ADULT PASSENGERS IN A GROUP',
    total_children TINYINT COMMENT 'TOTAL CHILDREN UNDER 5  IN A GROUP'
);


ALTER TABLE eam_group ADD CONSTRAINT eam_group_pk PRIMARY KEY ( group_id );

CREATE TABLE eam_invoice (
    invoice_id     VARCHAR(5) NOT NULL COMMENT 'UNIQUE INVOICE ID FOR EACH BOOKING',
    payment_status VARCHAR(20) NOT NULL COMMENT 'STATUS OF THE PAYMENT',
    inv_date       DATETIME NOT NULL COMMENT 'DATE OF  INVOICE GENERATED',
    booking_id     INT NOT NULL
);


CREATE UNIQUE INDEX eam_invoice__idx ON
    eam_invoice (
        booking_id
    ASC );

ALTER TABLE eam_invoice ADD CONSTRAINT eam_invoice_pk PRIMARY KEY ( invoice_id );

CREATE TABLE eam_location (
    location_id   TINYINT NOT NULL COMMENT 'UNIQUE LOCATION ID OF CRUISE',
    location_name VARCHAR(10) NOT NULL COMMENT 'LOCATION NAME OF CRUISE',
    loc_desc      VARCHAR(25) NOT NULL COMMENT 'DESCRIPTION OF THE LOCATION'
);
```

```
ALTER TABLE eam_location ADD CONSTRAINT eam_loc_pk PRIMARY KEY ( location_id );

CREATE TABLE eam_package (
    package_id     INT NOT NULL COMMENT 'UNIQUE ID FOR EVERY PACKAGE',
    package_name   VARCHAR(30) NOT NULL COMMENT 'TYPE OF PACKAGE ',
    unit_price     DECIMAL(7, 2) NOT NULL COMMENT 'PRICE OF THE PACKAGE',
    min_age        TINYINT COMMENT 'AGE LIMIT TO A PACKAGE IF APPLICABLE',
    pkg_pricing_id DECIMAL(18,6) NOT NULL
);


ALTER TABLE eam_package ADD CONSTRAINT eam_package_pk PRIMARY KEY ( package_id );

CREATE TABLE eam_passenger (
    passenger_id  BIGINT NOT NULL COMMENT 'UNIQUE ID FOR EVERY PASSENGER',
    group_id      BIGINT NOT NULL,
    p_fname       VARCHAR(30) NOT NULL COMMENT 'PASSENGER FIRST NAME',
    p_lname       VARCHAR(30) NOT NULL COMMENT 'PASSENGER LAST NAME',
    p_dob         DATETIME NOT NULL COMMENT 'PASSENGER DATE OF BIRTH',
    p_gender      VARCHAR(6) NOT NULL COMMENT 'PASSENGER GENDER (MALE/FEMALE)',
    p_nationality VARCHAR(20) NOT NULL COMMENT 'PASSENGER NATIONALITY',
    p_email       VARCHAR(30) NOT NULL COMMENT 'PASSENGER EMAIL ADDRESS',
    p_phone       BIGINT NOT NULL COMMENT 'PASSENGER PHONE NUMBER',
    p_passport_no VARCHAR(9) NOT NULL COMMENT 'PASSPORT NO OF PASSENGER',
    p_street      VARCHAR(30) NOT NULL COMMENT 'STREET ADDRESS OF PASSENGER',
    p_city        VARCHAR(20) NOT NULL COMMENT 'CITY OF PASSENGER',
    p_states      VARCHAR(20) NOT NULL COMMENT 'STATE OF PASSENGER',
    p_country     VARCHAR(20) NOT NULL COMMENT 'COUNTRY  OF PASSENGER',
    p_zip         VARCHAR(10) NOT NULL COMMENT 'ZIPCODE OF PASSENGER',
    p_address_id  INT NOT NULL
);


ALTER TABLE eam_passenger ADD CONSTRAINT eam_passenger_pk PRIMARY KEY ( passenger_id );

CREATE TABLE eam_passenger_package (
    pass_package_id INT NOT NULL COMMENT 'UNIQUE PASSENGER PACKAGE ID FOR EVERY PACKAGE ',
    passenger_id    BIGINT NOT NULL,
    package_id      INT NOT NULL,
    quantity        TINYINT NOT NULL COMMENT 'Number of units (nights for PER_NIGHT packages,
1 for ENTIRE_TRIP packages)'
);


ALTER TABLE eam_passenger_package ADD CONSTRAINT eam_pass_pkg_pk PRIMARY KEY ( pass_package_id
);

CREATE TABLE eam_payment (
    pmt_id     INT NOT NULL COMMENT 'UNIQUE PAYMENT ID FOR EVERY PAYMENT MADE',
    pmt_date   DATETIME NOT NULL COMMENT 'DATE OF PAYMENT',
    pmt_amount DECIMAL(10, 2) NOT NULL COMMENT 'AMOUNT PAID',
    pmt_method VARCHAR(10) NOT NULL COMMENT 'PAYMENT METHOD - CREDIT/DEBIT/CASH',
    invoice_id VARCHAR(5) NOT NULL
```

```
);


ALTER TABLE eam_payment ADD CONSTRAINT eam_payment_pk PRIMARY KEY ( pmt_id );

CREATE TABLE eam_port (
    port_id            SMALLINT NOT NULL COMMENT 'UNIQUE PORT ID FOR EVERY PORT',
    pr_name            VARCHAR(30) NOT NULL COMMENT 'NAME OF PORT',
    pr_terminal        VARCHAR(3) NOT NULL COMMENT 'TERMINAL  OF PORT',
    pr_nearest_airport VARCHAR(30) NOT NULL COMMENT 'NEAREST AIRPORT TO THE PORT',
    pr_parking_spots   INT NOT NULL COMMENT 'NUMBER OF PARKING SPOTS AT EACH PORT',
    pr_street          VARCHAR(50) NOT NULL COMMENT 'STREET ADDRESS  OF PORT',
    pr_city            VARCHAR(30) NOT NULL COMMENT 'CITY  OF PORT',
    pr_states          VARCHAR(30) NOT NULL COMMENT 'STATE OF PORT',
    pr_zip             VARCHAR(10) NOT NULL COMMENT 'ZIP CODE OF PORT',
    pr_country         VARCHAR(30) NOT NULL COMMENT 'COUNTRY OF PORT'
);


ALTER TABLE eam_port ADD CONSTRAINT eam_port_pk PRIMARY KEY ( port_id );

CREATE TABLE eam_port_trip (
    port_stop_id SMALLINT NOT NULL COMMENT 'UNIQUE PORT STOP ID FOR EVERY STOP',
    arrival_date DATETIME NOT NULL COMMENT 'ARRIVAL DATE ON THE PORT STOP',
    depart_date  DATETIME NOT NULL COMMENT 'DEPARTURE DATE FROM THE PORT STOP',
    port_id      SMALLINT NOT NULL,
    trip_id      INT NOT NULL
);



ALTER TABLE eam_port_trip ADD CONSTRAINT eam_port_trip_pk PRIMARY KEY ( port_stop_id );

CREATE TABLE eam_restaurant (
    restaurant_id INT NOT NULL COMMENT 'UNIQUE ID FOR EVERY RESTAURANT',
    r_type        VARCHAR(30) NOT NULL COMMENT 'TYPE OF RESTAURANT',
    opening_time  DATETIME NOT NULL COMMENT 'OPENING TIME OF RESTAURANT',
    closing_time  DATETIME NOT NULL COMMENT 'CLOSING TIME OF RESTAURANT',
    floor_no      TINYINT NOT NULL COMMENT 'FLOOR NO OF RESTAURANT'
);


ALTER TABLE eam_restaurant ADD CONSTRAINT eam_rest_pk PRIMARY KEY ( restaurant_id );

CREATE TABLE eam_stateroom (
    stateroom_id    INT NOT NULL COMMENT 'UNIQUE STATEROOM ID FOR EVERY STATEROOM',
    type            VARCHAR(30) NOT NULL COMMENT 'STATEROOM TYPE',
    size_sqft       SMALLINT NOT NULL COMMENT 'SIZE OF STATEROOM IN SQFT',
    no_of_bed       TINYINT NOT NULL COMMENT 'NUMBER OF BEDS IN A STATEROOM TYPE',
    no_of_bathrooom DECIMAL(2, 1) NOT NULL COMMENT 'NUMBER OF BATHROOM IN A STATEROOM TYPE',
    balcony         TINYINT NOT NULL COMMENT 'NUMBER OF BALCONIES IN A STATEROOM TYPE'
);
```

```
ALTER TABLE eam_stateroom ADD CONSTRAINT eam_strm_pk PRIMARY KEY ( stateroom_id );

CREATE TABLE eam_stateroom_location (
    sr_loc_id    INT NOT NULL COMMENT 'UNIQUE ID FOR EACH STATEROOM BOOKING (SURROGATE KEY)',
    stateroom_id INT NOT NULL,
    location_id  TINYINT NOT NULL
);


ALTER TABLE eam_stateroom_location ADD CONSTRAINT eam_strm_loc_pk PRIMARY KEY ( sr_loc_id );

CREATE TABLE eam_trip (
    trip_id       INT NOT NULL COMMENT 'UNIQUE TRIP ID ',
    total_nights  TINYINT NOT NULL COMMENT 'NUMBER OF NIGHTS SPENT IN A TRIP',
    start_date    DATETIME NOT NULL COMMENT 'START DATE OF TRIP',
    end_date      DATETIME NOT NULL COMMENT 'END DATE OF TRIP',
    start_port_id SMALLINT NOT NULL,
    end_port_id   SMALLINT NOT NULL
);



ALTER TABLE eam_trip ADD CONSTRAINT eam_trip_pk PRIMARY KEY ( trip_id );

CREATE TABLE eam_trip_strm_price (
    price_per_night DECIMAL(7, 2) NOT NULL COMMENT 'PRICE PER NIGHT FOR STATEROOM WHICH
CHANGES WITH TRIP',
    trip_id         INT NOT NULL,
    sr_loc_id       INT NOT NULL
);

CREATE TABLE users (
  user_id int(11) NOT NULL AUTO_INCREMENT,
  username varchar(50) NOT NULL,
  email varchar(100) NOT NULL,
  `password` varchar(255) NOT NULL,
  created_at datetime DEFAULT current_timestamp(),
  PRIMARY KEY (user_id),
  UNIQUE KEY email (email)
) ENGINE=InnoDB AUTO_INCREMENT=12 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;


ALTER TABLE eam_booking
    ADD CONSTRAINT eam_bkng_grp_fk FOREIGN KEY ( group_id )
        REFERENCES eam_group ( group_id );

ALTER TABLE eam_booking
    ADD CONSTRAINT eam_bkng_strm_loc_fk FOREIGN KEY ( sr_loc_id )
        REFERENCES eam_stateroom_location ( sr_loc_id );

ALTER TABLE eam_booking
    ADD CONSTRAINT eam_bkng_trip_fk FOREIGN KEY ( trip_id )
        REFERENCES eam_trip ( trip_id );
```

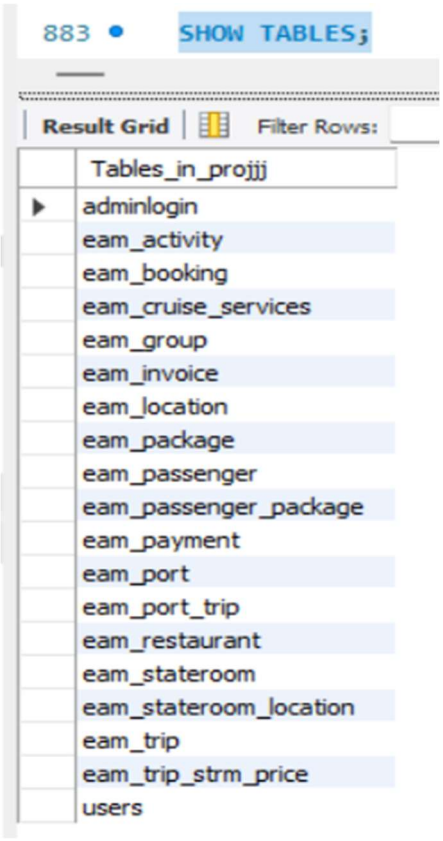
```
ALTER TABLE eam_cruise_services
    ADD CONSTRAINT eam_cruise_ser_activ_fk FOREIGN KEY ( activity_id )
        REFERENCES eam_activity ( activity_id );

ALTER TABLE eam_cruise_services
    ADD CONSTRAINT eam_cruise_ser_bkng_fk FOREIGN KEY ( booking_id )
        REFERENCES eam_booking ( booking_id );

ALTER TABLE eam_cruise_services
    ADD CONSTRAINT eam_cruise_ser_rest_fk FOREIGN KEY ( restaurant_id )
        REFERENCES eam_restaurant ( restaurant_id );

ALTER TABLE eam_invoice
    ADD CONSTRAINT eam_inv_bkng_fk FOREIGN KEY ( booking_id )
        REFERENCES eam_booking ( booking_id );

ALTER TABLE eam_stateroom_location
    ADD CONSTRAINT eam_loc_strm_fk FOREIGN KEY ( stateroom_id )
        REFERENCES eam_stateroom ( stateroom_id );

ALTER TABLE eam_passenger
    ADD CONSTRAINT eam_pass_grp_fk FOREIGN KEY ( group_id )
        REFERENCES eam_group ( group_id );

ALTER TABLE eam_passenger_package
    ADD CONSTRAINT eam_pass_pkg_fk FOREIGN KEY ( package_id )
        REFERENCES eam_package ( package_id );

ALTER TABLE eam_passenger_package
    ADD CONSTRAINT eam_pkg_pass_fk FOREIGN KEY ( passenger_id )
        REFERENCES eam_passenger ( passenger_id );

ALTER TABLE eam_payment
    ADD CONSTRAINT eam_pmt_inv_fk FOREIGN KEY ( invoice_id )
        REFERENCES eam_invoice ( invoice_id );

ALTER TABLE eam_port_trip
    ADD CONSTRAINT eam_port_trip_fk FOREIGN KEY ( trip_id )
        REFERENCES eam_trip ( trip_id );

ALTER TABLE eam_trip
    ADD CONSTRAINT eam_port_trip_fkv2 FOREIGN KEY ( end_port_id )
        REFERENCES eam_port ( port_id );

ALTER TABLE eam_stateroom_location
    ADD CONSTRAINT eam_strm_loc_fk FOREIGN KEY ( location_id )
        REFERENCES eam_location ( location_id );

ALTER TABLE eam_trip
    ADD CONSTRAINT eam_trip_eam_port_fk FOREIGN KEY ( start_port_id )
        REFERENCES eam_port ( port_id );

ALTER TABLE eam_port_trip
    ADD CONSTRAINT eam_trip_port_fk FOREIGN KEY ( port_id )
```

```
        REFERENCES eam_port ( port_id );

ALTER TABLE eam_trip_strm_price
    ADD CONSTRAINT eam_trip_strm_price_fk FOREIGN KEY ( sr_loc_id )
        REFERENCES eam_stateroom_location ( sr_loc_id );

ALTER TABLE eam_trip_strm_price
    ADD CONSTRAINT eam_trip_strm_price2_fk FOREIGN KEY ( trip_id )
        REFERENCES eam_trip ( trip_id );



-- SQLINES DEMO *** per Data Modeler Summary Report:
--
-- SQLINES DEMO ***                       17
-- SQLINES DEMO ***                        1
-- SQLINES DEMO ***                       35
-- SQLINES DEMO ***                        0
-- SQLINES DEMO ***                        0
-- SQLINES DEMO ***                        0
-- SQLINES DEMO *** DY                     0
-- SQLINES DEMO ***                        0
-- SQLINES DEMO ***                        0
-- SQLINES DEMO ***                        0
-- SQLINES DEMO ***                        0
-- SQLINES DEMO ***  TYPE                  0
-- SQLINES DEMO ***  TYPE                  0
-- SQLINES DEMO ***  TYPE BODY             0
-- SQLINES DEMO ***                        0
-- SQLINES DEMO ***                        0
-- SQLINES DEMO ***                        0
-- SQLINES DEMO ***                        0
-- SQLINES DEMO ***                        0
-- SQLINES DEMO ***                        0
-- SQLINES DEMO ***                        0
-- SQLINES DEMO *** EGMENT                 0
-- SQLINES DEMO ***                        0
-- SQLINES DEMO *** ED VIEW                0
-- SQLINES DEMO *** ED VIEW LOG            0
-- SQLINES DEMO ***                        0
-- SQLINES DEMO ***                        0
-- SQLINES DEMO ***                        0
--
-- SQLINES DEMO ***                        0
-- SQLINES DEMO ***                        0
--
-- SQLINES DEMO ***                        0
--
-- SQLINES DEMO ***                        0
-- SQLINES DEMO *** A                      0
-- SQLINES DEMO *** T                      0
--
-- SQLINES DEMO ***                        0
-- SQLINES DEMO ***                        0
```

LIST OF TABLES, AND THE TOTAL NUMBER OF RECORDS OF EACH TABLE

List of tables

1. EAM_PASSENGER
2. EAM_PASSENGER_PACKAGE
3. EAM _PACKAGE
4. EAM_PORT
5. EAM_TRIP
6. EAM_PORT_TRIP
7. EAM_GROUP
8. EAM_TRIP_STRM_PRICE
9. EAM_BOOKING
10. EAM_STATEROOM_LOCATION
11. EAM_LOCATION
12. EAM_STATEROOM
13. EAM_RESTAURANT
14. EAM_CRUISE_SERVICES
15. EAM_INVOICE
16. EAM_ACITIVITY
17. EAM_PAYMENT
18. USERS
19. ADMINLOGIN

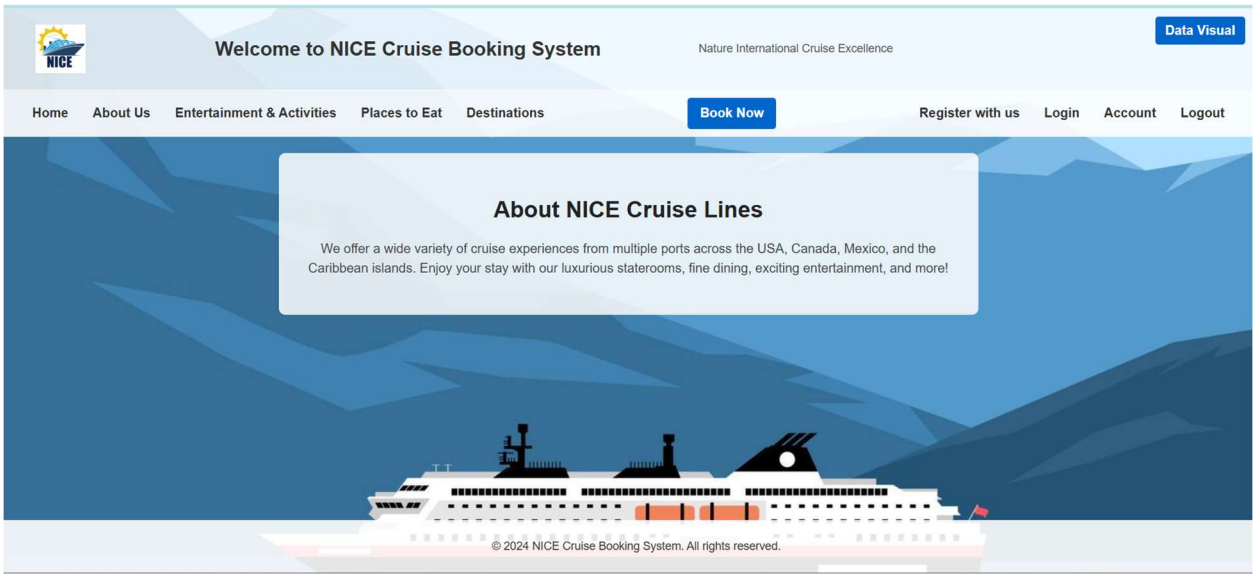


Total Number of Records in each Table:
(*as collected during initial testing, the number has varied upon further repeated entries)

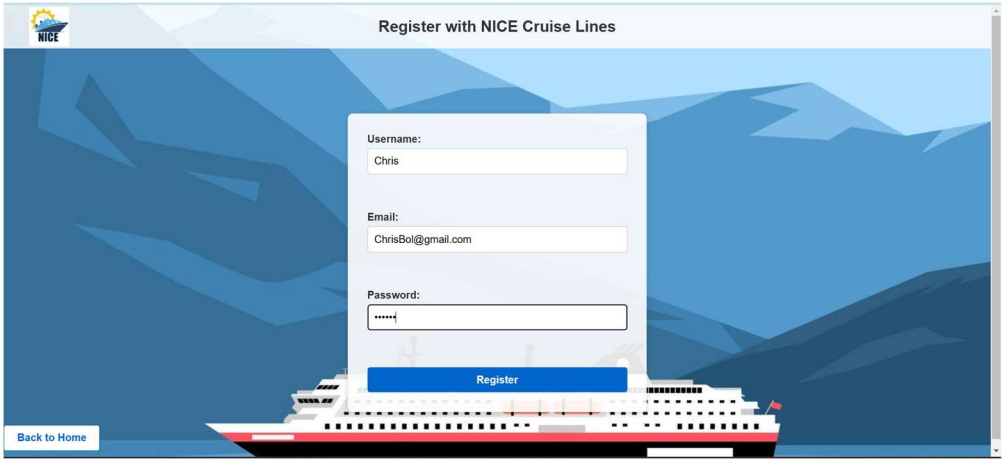
1. EAM_ACTIVITY 16
2. EAM_BOOKING 32
3. EAM_CRUISE_SERVICES 5
4. EAM_GROUP 29
5. EAM_INVOICE 10
6. EAM_LOCATION 4
7. EAM_PACKAGE 5
8. EAM_PASSENGER 44
9. EAM_PASSENGER_PACKAGE 74
10. EAM_PAYMENT 9
11. EAM_PORT 3
12. EAM_PORT_TRIP 3
13. EAM_RESTAURANT 9
14. EAM_STATEROOM 7
15. EAM_STATEROOM_LOCATION 7
16. EAM_TRIP 3
17. EAM_TRIP_STRM_PRICE 7
18. USERS 12
19. ADMINLOGIN 3

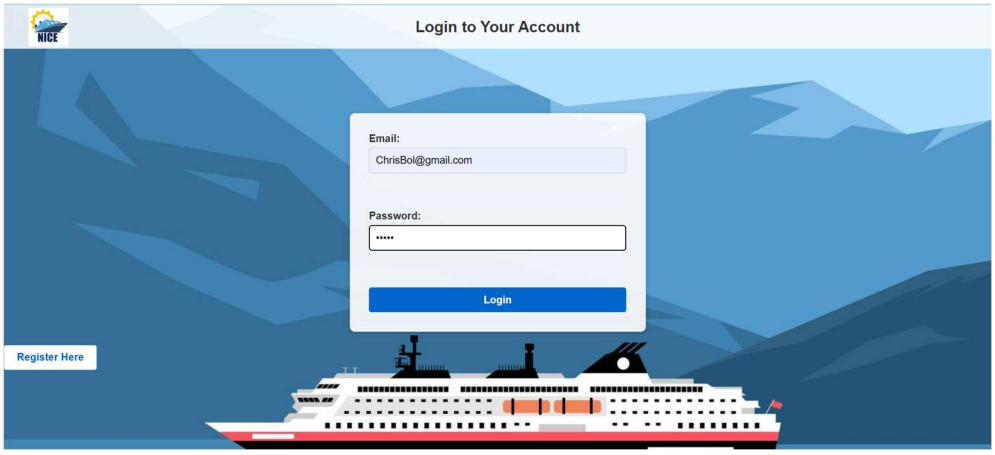
SCREENSHOTS OF SOME SESSIONS, PAGES, & MENUS OF WEB APPLICATION

Landing page (HOME Page):

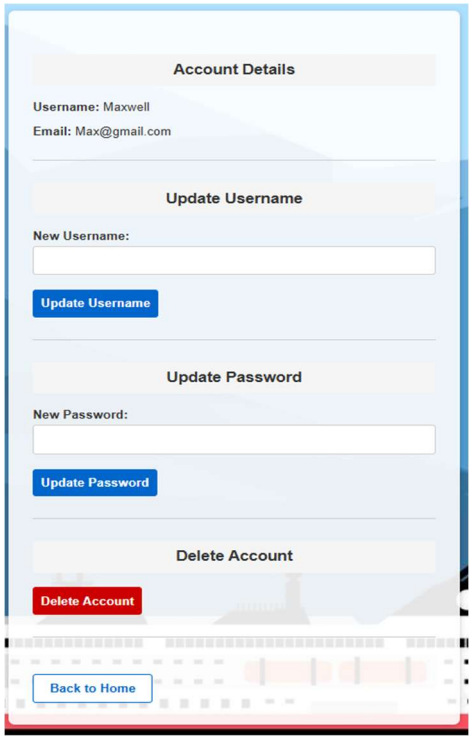


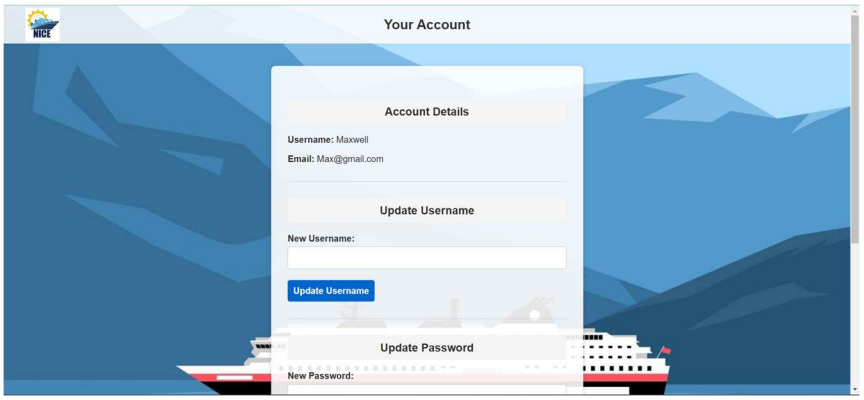
Register User and Login Pages:



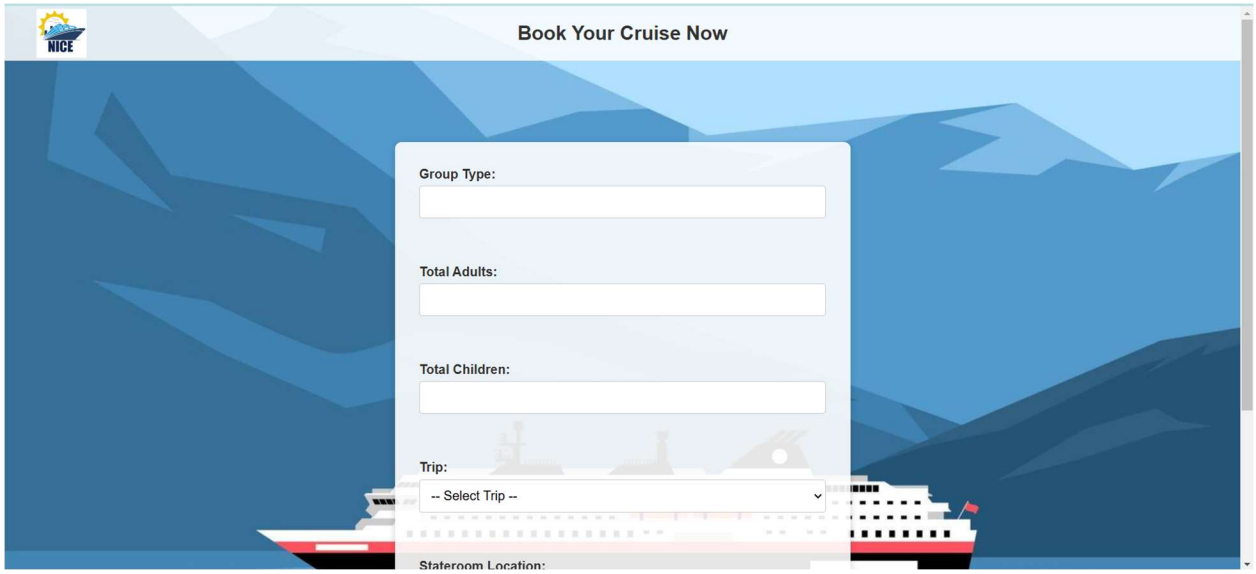


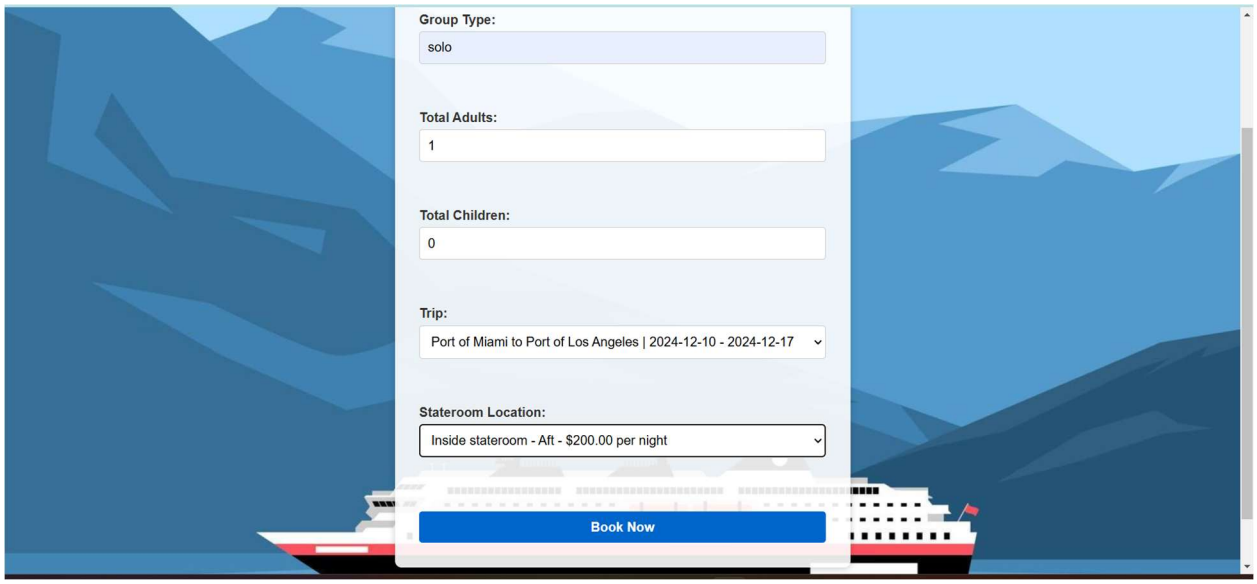
Account Page: (User can update username and password here or can delete their account)



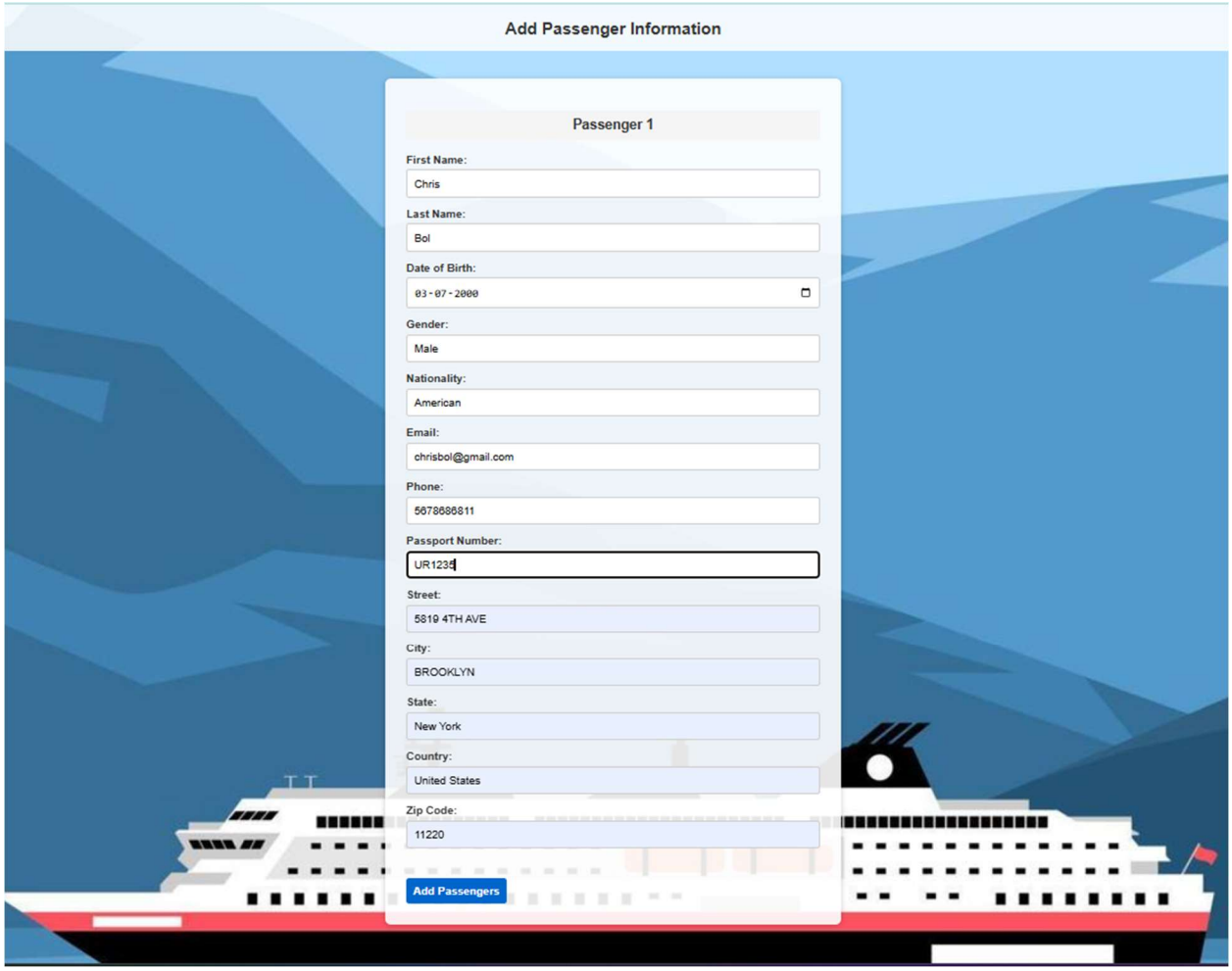


Cruise booking Page: User can select Trip, Stateroom (with price as per the trip and location)

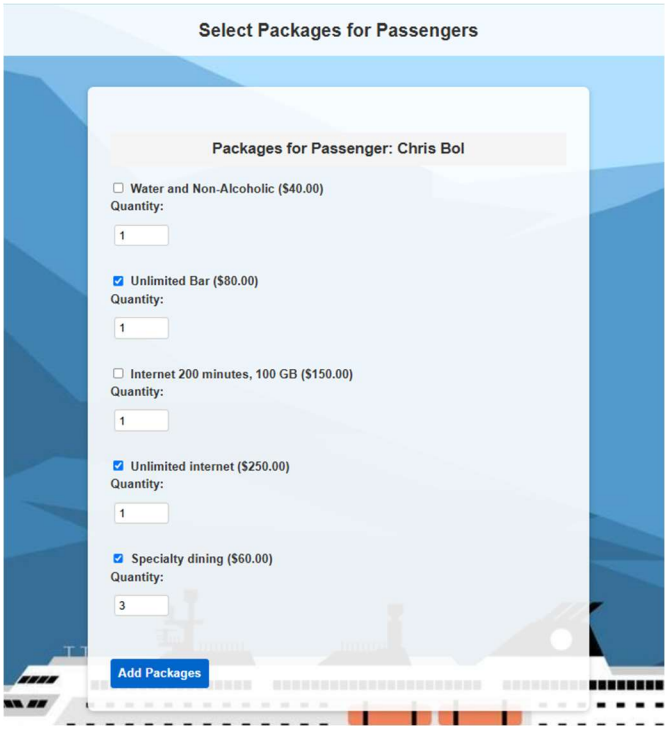




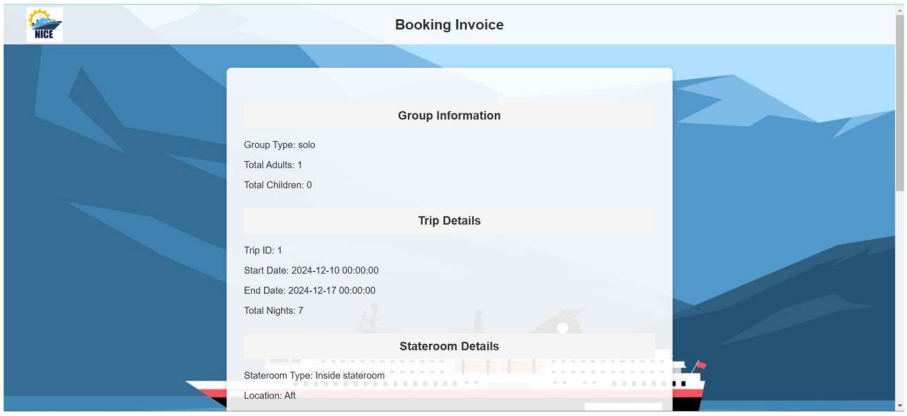
Passenger Information Page: Enter all the passenger details:

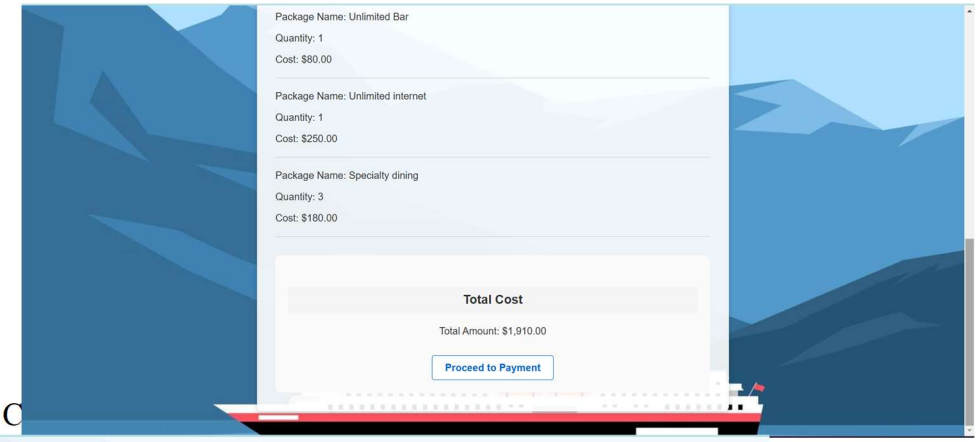


Package Selection Page. Each Passenger in the group can select their own packages:

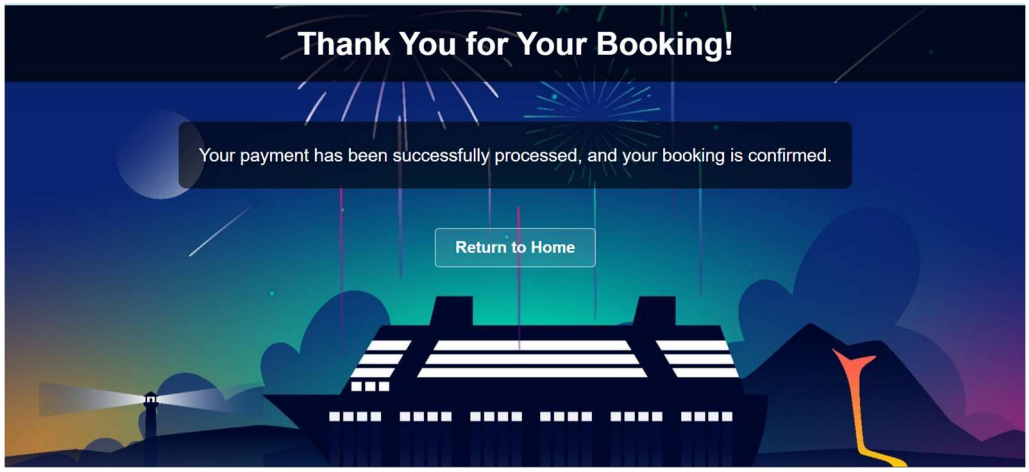


Final Booking Invoice Page with all the prices and final total amount:

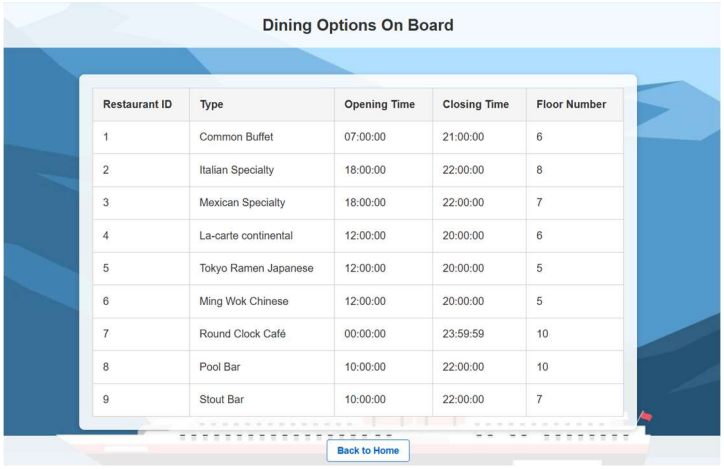




Payment Page and Payment C



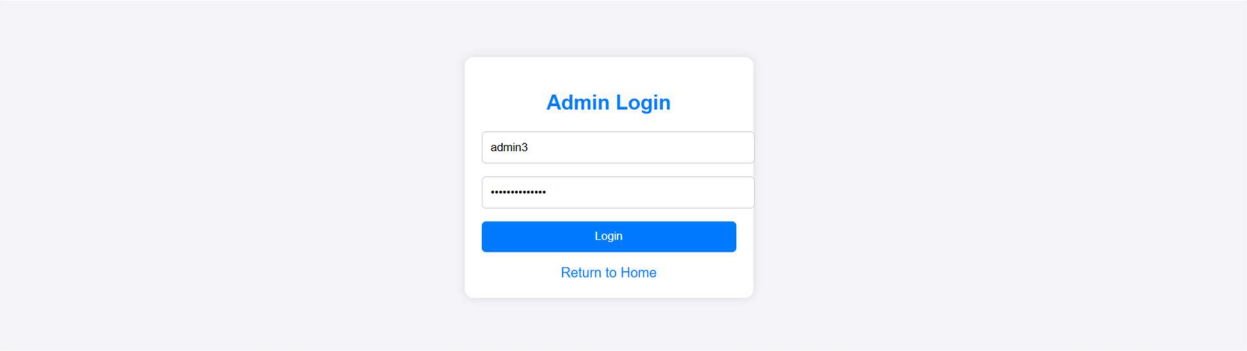
Other Pages like restaurants, activities of the cruise:



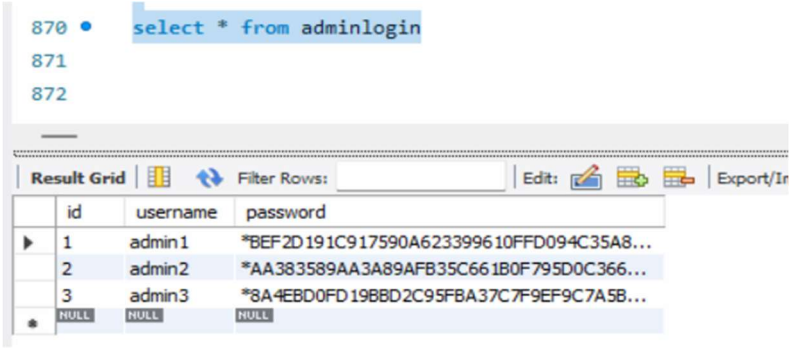
Port Destinat

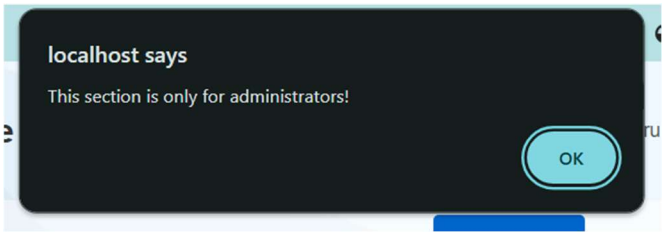
Port Name	Terminal	Nearest Airport	Parking Spots	Street	City	State	Zip	Country
Port of Miami	T1	Miami International Airport	500	1015 N America Way	Miami	FL	33132	USA
Port of Los Angeles	T2	Los Angeles International Airp	800	425 S Palos Verdes St	San Pedro	CA	90731	USA
Port of Vancouver	T3	Vancouver International Airpor	300	999 Canada Pl	Vancouver	BC	V6C 3E1	Canada

Admin Side Webpages:
Admin Login:



Admin Login enables admins to access restricted parts of the webpage like the data visualization page. This page is not accessible to regular users (customers)





DETAILS OF SECURITY FEATURES IMPLEMENTED

1) SQL Injection Prevention:

The stored procedure ensures that raw user input is never directly interpolated into SQL queries.

Prepared statements with bind_param() add another layer of protection for parameterized queries.

The code uses prepare and bind_param for SQL queries, which mitigate SQL injection attacks by separating query structure from user input.

```
912
913     DELIMITER //
914
915 •   CREATE PROCEDURE sp_loginUser(
916         IN input_email VARCHAR(255),
917         OUT output_user_id INT,
918         OUT output_password_hash VARCHAR(255)
919     )
920     BEGIN
921         SELECT user_id, password
922         INTO output_user_id, output_password_hash
923         FROM users
924         WHERE email = input_email;
925     END //
926
927     DELIMITER ;
928
```

```php
// Prepare variables for the stored procedure
$output_user_id = null;
$output_password_hash = null;

// Call the stored procedure
$stmt = $conn->prepare("CALL sp_loginUser(?, @output_user_id, @output_password_hash)");
if ($stmt) {
    $stmt->bind_param("s", $email);
    $stmt->execute();
    $stmt->close();
```

2) Cross-Site Scripting (XSS) Protection:

User inputs like email and error messages are sanitized using htmlspecialchars with ENT_QUOTES to ensure any special characters (<, >, " or ') are converted into their HTML-encoded forms, preventing script injection.
Error messages displayed to the user are also sanitized before being echoed to the browser.
Prevents attackers from injecting malicious scripts into the webpage, protecting users from XSS attacks.

```php
$error_message = "Error calling stored procedure: " . htmlspecialchars($conn->error, ENT_QUOTES, 'UTF-8');
```

```php
$email = htmlspecialchars(trim($_POST['email']), ENT_QUOTES, 'UTF-8');
```

3) Password Hashing and Verification:

Passwords stored in the database are hashed (during registration) using a secure hashing function like password_hash.
During login, the provided password is verified against the stored hash using password_verify.
This Hashing ensures passwords are not stored in plain text, making it computationally infeasible to reverse-engineer a password from the hash.
Even if the database is breached, the hashed passwords remain secure. Verification ensures only legitimate users are authenticated.

user_id	username	email	password	created_at
1	xys@gmail.com	ehtesham.siddiqui2001@gmail.com	$2y$10$hopCxhJzgR2v/f70nlPKRuHyeD3ARvU...	2024-12-04 04:01:11
2	xays@gmail.com	ehteshamsid9@gmail.com	$2y$10$GDQs7mKuSpwqO0AnxYLNi.tskEsTfE4...	2024-12-04 04:02:47
3	123456	123456@gmail.com	$2y$10$YU.3rhEk/KY2iTWE9Un7lurrFWzMlc7tG...	2024-12-04 04:05:47
5	wertyu	ehttsgdg@gmail.com	$2y$10$e4Q6gN6flzQ1BMCnEVpwYuvFBmj1KQ...	2024-12-05 00:54:15
6	abhishek	abcdefg@gmail.com	$2y$10$Sz2VlKuv8SUybVrTgPS90uxr3LzKMkBh...	2024-12-05 04:03:19
7	12345abcd	abcd@gmail.com	$2y$10$FvDQyKe0vp523eLfYkihmu9ivlx.JrcDq...	2024-12-07 02:23:54
8	Meghna	meghna@gmail.com	$2y$10$tJFBq2VzeGslxRru25DWtO0tGaDgBoyL...	2024-12-07 13:58:59
11	Robert01	robert@gmail.con	$2y$10$9DsoI6rbJlUopIAEuVR1teme.tcpJtkZm...	2024-12-07 14:46:20
12	Karthikeya	Karth@gmail.com	$2y$10$HOv1CnLGWgfjwVtzo3MJwe78yfa3hpi...	2024-12-07 17:22:57
13	Rahul	Rag@gmail.com	$2y$10$zzKmGtt9ZbG8wzNatoJhyehc.X69jaAm...	2024-12-08 02:59:01
14	Maxwell	Max@gmail.com	$2y$10$bQtBxgAlicHxWdsddtVHyuiEvix99zI2zq...	2024-12-08 04:52:16
15	Chris	ChrisBol@gmail.com	$2y$10$hQsxA.qGgVcRpbx3E7xFPuvq40uAt09...	2024-12-08 04:57:18
NULL	NULL	NULL	NULL	NULL

```php
$result = $conn->query("SELECT @output_user_id AS user_id, @output_password_hash AS password_hash");
if ($result->num_rows > 0) {
    $row = $result->fetch_assoc();
    $output_user_id = $row['user_id'];
    $output_password_hash = $row['password_hash'];

    // Verify the password using password_verify()
    if ($output_user_id && password_verify($password, $output_password_hash)) {
        // Password is correct, regenerate session ID
```

4) Transaction Concurrency:

The script explicitly starts a transaction with **$conn->begin_transaction()**. This ensures that all related operations (e.g., validating trip ID, validating stateroom location, inserting group and booking) are treated as a single atomic unit.
If any operation fails, the entire transaction is rolled back using **$conn->rollback(),** preventing partial updates that could lead to inconsistent states.
Queries for both trip and stateroom_location tables include the **FOR UPDATE** clause. This locks the rows being selected, preventing other transactions from modifying them until the current transaction is completed.
This ensures that no two bookings can reserve the same trip or stateroom simultaneously.
This approach ensures serialized access to critical resources, maintaining concurrency while avoiding conflicts.

```php
$conn->begin_transaction();

try {
    // Add FOR UPDATE to trip query to prevent deadlocks
    $trip_sql = "
        SELECT t.trip_id, t.start_date, t.end_date, sp.pr_name AS start_port, ep.pr_name AS end_port
        FROM eam_trip t
        JOIN eam_port sp ON t.start_port_id = sp.port_id
        JOIN eam_port ep ON t.end_port_id = ep.port_id
        WHERE t.trip_id = ?
        FOR UPDATE";
    $stmt_trip = $conn->prepare($trip_sql);
    $stmt_trip->bind_param("i", $trip_id);
    $stmt_trip->execute();
    $trip_result = $stmt_trip->get_result();
```

```php
    // Commit the transaction
    $conn->commit();
```

```php
} catch (Exception $e) {
    // Rollback the transaction in case of error
    $conn->rollback();
    echo "Transaction failed: " . $e->getMessage();
```

5) Deadlock Prevention:

The script accesses resources in a consistent order:
First, it locks the trip table row.
Then, it locks the stateroom_location table row.
By maintaining a consistent order of locking resources, it reduces the likelihood of deadlocks.
The FOR UPDATE clause ensures rows are locked only when necessary, minimizing unnecessary contention.

If a deadlock occurs (e.g., due to external factors), the catch block handles the exception and rolls back the transaction

```php
// FOR UPDATE to stateroom location query to prevent deadlocks
$stateroom_sql = "
    SELECT esl.sr_loc_id, es.type AS stateroom_type, el.location_name, etsp.price_per_night
    FROM eam_stateroom_location esl
    JOIN eam_stateroom es ON es.stateroom_id = esl.stateroom_id
    JOIN eam_location el ON el.location_id = esl.location_id
    JOIN eam_trip_strm_price etsp ON etsp.sr_loc_id = esl.sr_loc_id
    WHERE esl.sr_loc_id = ?
    FOR UPDATE";
$stmt_stateroom = $conn->prepare($stateroom_sql);
$stmt_stateroom->bind_param("i", $sr_loc_id);
$stmt_stateroom->execute();
$stateroom_result = $stmt_stateroom->get_result();
```

```php
try {
    // FOR UPDATE to trip query to prevent deadlocks
    $trip_sql = "
        SELECT t.trip_id, t.start_date, t.end_date, sp.pr_name AS start_port, ep.pr_name AS end_port
        FROM eam_trip t
        JOIN eam_port sp ON t.start_port_id = sp.port_id
        JOIN eam_port ep ON t.end_port_id = ep.port_id
        WHERE t.trip_id = ?
        FOR UPDATE";
    $stmt_trip = $conn->prepare($trip_sql);
    $stmt_trip->bind_param("i", $trip_id);
    $stmt_trip->execute();
    $trip_result = $stmt_trip->get_result();
```

```php
} catch (Exception $e) {
    // Rollback the transaction in case of error
    $conn->rollback();
    echo "Transaction failed: " . $e->getMessage();
} finally {
    // Close all prepared statements
    if (isset($stmt_trip)) $stmt_trip->close();
    if (isset($stmt_stateroom)) $stmt_stateroom->close();
    if (isset($stmt_group)) $stmt_group->close();
    if (isset($stmt_booking)) $stmt_booking->close();
}
```

6) **Minimal Attack Surface:**

The application logic is encapsulated in a stored procedure (sp_loginUser), reducing direct exposure of database queries in the PHP code. The use of prepared statements with bound

parameters minimizes the risk of database compromise. Reduces the attack surface by limiting direct interaction with the database and abstracting logic into the stored procedure.

```php
$stmt = $conn->prepare("CALL sp_loginUser(?, @output_user_id, @output_password_hash)");
if ($stmt) {
    $stmt->bind_param("s", $email);
    $stmt->execute();
    $stmt->close();
```

7) Session Security:

Session Initialization: **session_start()** starts a session to track user activity securely.

Session Regeneration: After login, **session_regenerate_id(true)** generates a new session ID, invalidating the old one to prevent session fixation attacks.

Sensitive user data, like user_id, is stored securely in the **$_SESSION superglobal**.

Protects against session hijacking and fixation attacks, ensuring user sessions are securely maintained. User authentication is managed through the **$_SESSION['user_id']** variable, ensuring secure session tracking.

```php
// Password is correct, regenerate session ID
session_regenerate_id(true);
$_SESSION['user_id'] = $output_user_id; // Store user ID in the session

// Redirect to the homepage or booking page
header("Location: index.php");
exit();
```

8) Error Handling:

Error messages are sanitized before being displayed to the user to prevent information leakage. Internal errors, such as database connection or procedure call failures, are captured and logged (though explicit logging isn't shown, it's implied). Messages like Invalid email or password are generic, avoiding detailed information that could aid attackers. Prevents attackers from gaining insight into system behavior while keeping users informed in a secure manner. Ensures all dynamically generated content is HTML-escaped before rendering.

```php
        } else {
            $error_message = "Invalid email or password.";
        }
    } else {
        $error_message = "Invalid email or password.";
    }
} else {
    $error_message = "Error calling stored procedure: " . htmlspecialchars($conn->error, ENT_QUOTES, 'UTF-8');
}
```

SUMMARY OF SECURITY FEATURES IMPLEMENTED:

PROTECTION	IMPLEMENTATION
SQL Injection Prevention	Stored Procedures, Prepared Statements, prepare and bind_param for SQL queries
Cross-Site Scripting (XSS)	htmlspecialchars sanitization for inputs and outputs.
Password Hashing	Use of password_hash (assumed in registration) and password_verify for secure authentication.
Transaction Concurrency	Use transactions with $conn->begin_transaction() , $conn->rollback(), use FOR UPDATE
Deadlock Prevention	Explicit Locking using FOR UPDATE, catch block handles the exception and rolls back the transaction to remove deadlock.
Minimal Attack Surface	Use of stored procedures and prepared statements with bound parameters to limit direct database interaction
Session Security	Session management with session_start and session_regenerate_id(true)
Error Handling	Sanitization of error messages and generic feedback to users to avoid exposing system details. Ensures all dynamically generated content is HTML-escaped before rendering.

LESSON LEARNED AND REFLECTIONS ABOUT PROJECT WORK AND CONSTRAINTS FACED:

Working together on this NICE Cruise- Case study project was a very interesting experience for our team. It helped us to learn many important lessons in database design, web application development, and also in team collaboration. We were able to learn the importance of proper planning and development. The database schema we designed in the initial phase was the foundation for the web-based interface in the second phase, this taught us how critical a strong and well-thought-out base is to a project's success.

One of the biggest challenges we had faced as a team was learning PHP. This was hard because none of us had any experience with it and we had to dedicate more time to understanding its core concepts before we could move forward with connecting PHP to MySQL and integrating it with frontend. This learning curve was very time taking but it was a significant milestone in our project because it helped us gain the skills that were needed to build the core functionality of our project.

Another challenge we had faced was implementing important security features like protecting against SQL injection and encrypting sensitive data. Initially we were struggling with understanding and applying these concepts. However, with the support and guidance of our professor we were able to implement these features. This had showed importance of taking assistance and receiving help using the available resources when faced with challenges.

The logical and relational modeling part went smoothly due to clear guidelines and well-coordinated teamwork, integrating the backend database with the frontend interface required more effort. Ensuring seamless user interactions involved trial and error and collaborative problem-solving, which ultimately strengthened our technical and teamwork skills.

Time management was another place where we faced issues. Balancing project deliverables with our academic commitments required careful planning and prioritization. Having remote coordination was tricky but we overcame this by maintaining open and proactive communication within the group.

Despite these challenges, the project was an extremely rewarding experience for all of us. It reinforced us with the value of adaptability, collaboration, and continuous learning. Together, we not only built a functional and secure application but also grew as a team, overcoming obstacles and learning from each other throughout this process. This project has left us better prepared for future projects and made us more confident in tackling new challenges together as a team.

BUSINESS ANALYSIS WITH 6 SQLS USING PROJECT DATA

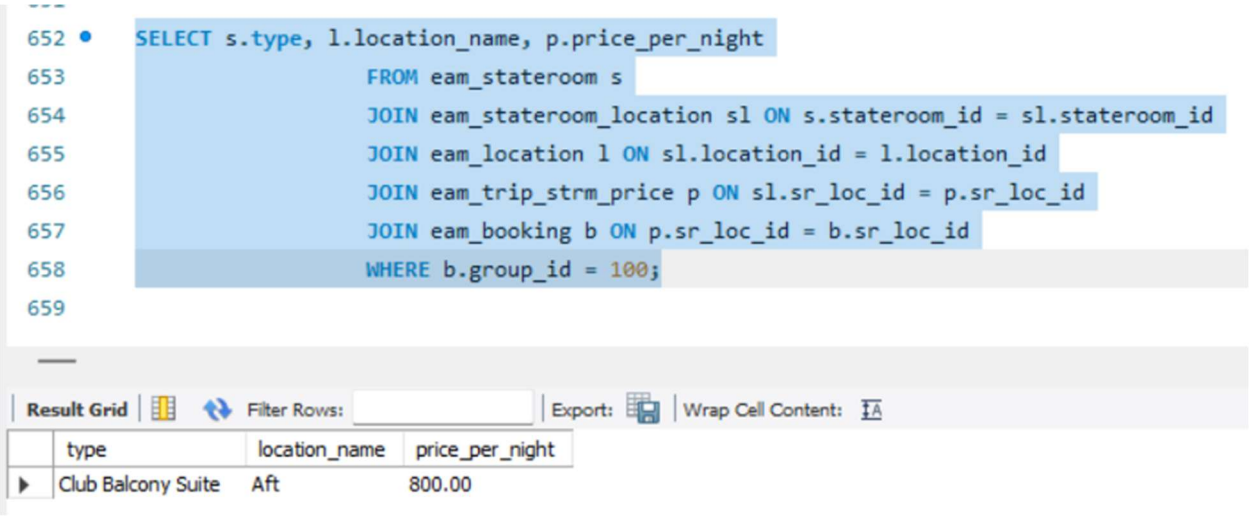
Write SQL queries using each of the following:

Q1) Table joins with at least 3 tables in join.

1A1) Select Statement:

```
SELECT s.type, l.location_name, p.price_per_night
        FROM eam_stateroom s
        JOIN eam_stateroom_location sl ON s.stateroom_id = sl.stateroom_id
        JOIN eam_location l ON sl.location_id = l.location_id
        JOIN eam_trip_strm_price p ON sl.sr_loc_id = p.sr_loc_id
        JOIN eam_booking b ON p.sr_loc_id = b.sr_loc_id
        WHERE b.group_id = 100;
```

1A2)Result of the query



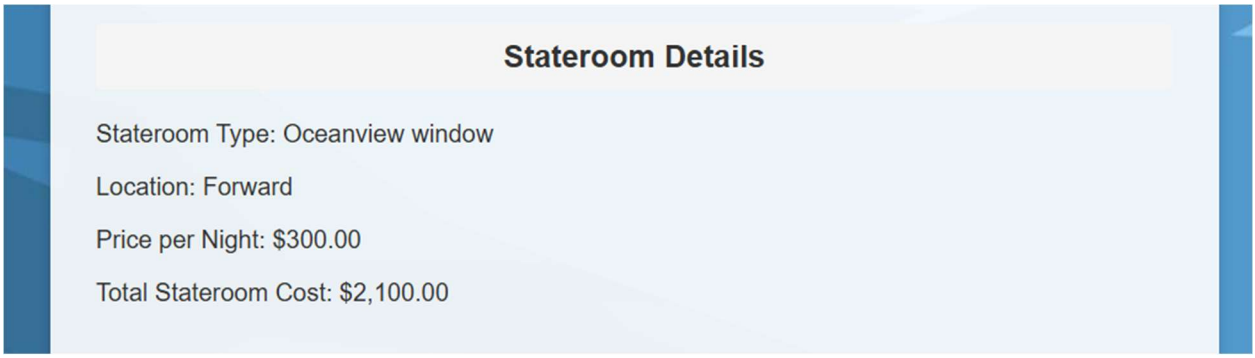
1A3) The business question that query is answering to:

In the Final Invoice: Getting the booked Stateroom Details and price per_night in the Invoice based on the trip and location for a particular booking and a specific group. (as Stateroom price vary based on the location and each trip may have different prices). This value is used to calculate the multivalued attribute Total Stateroom Cost for the trip.

PHP Implementation of above:

```php
$sql_stateroom = "SELECT s.type, l.location_name, p.price_per_night
                  FROM eam_stateroom s
                  JOIN eam_stateroom_location sl ON s.stateroom_id = sl.stateroom_id
                  JOIN eam_location l ON sl.location_id = l.location_id
                  JOIN eam_trip_strm_price p ON sl.sr_loc_id = p.sr_loc_id
                  JOIN eam_booking b ON p.sr_loc_id = b.sr_loc_id
                  WHERE b.group_id = ?";

$stmt_stateroom = $conn->prepare($sql_stateroom);
$stmt_stateroom->bind_param("i", $group_id);
```

Website Displaying the above Select Statement in the Invoice:



Q2) Multi-row subquery

2A1) Select Statement:

```
SELECT pp.quantity, p.package_name, p.unit_price
            FROM eam_passenger_package pp
            JOIN eam_package p ON p.package_id = pp.package_id
            WHERE pp.passenger_id IN (
                SELECT pa.passenger_id
                  FROM eam_passenger pa
                     WHERE pa.group_id = 121)
```

2A2) Result of the Query:

```
663 •  SELECT pp.quantity, p.package_name, p.unit_price
664                       FROM eam_passenger_package pp
665                       JOIN eam_package p ON p.package_id = pp.package_id
666                       WHERE pp.passenger_id IN (
667                             SELECT pa.passenger_id
668                               FROM eam_passenger pa
669                                  WHERE pa.group_id = 121)
```

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

quantity	package_name	unit_price
4	Specialty dining	60.00

2A3) The business question that query is answering to:

This multi-row subquery is used to get the packages details in the Invoice of a particular booking based on the particular group which is ordering the package. It fetches the quantity of the package and the name and the unit price. This is used to calculate the total packages cost of all the packages that have been booked.

PHP Implementation of Above:

```php
// Fetch package details
$sql_packages = "SELECT pp.quantity, p.package_name, p.unit_price
                 FROM eam_passenger_package pp
                 JOIN eam_package p ON p.package_id = pp.package_id
                 WHERE pp.passenger_id IN (
                       SELECT pa.passenger_id
                         FROM eam_passenger pa
                            WHERE pa.group_id = ?)";
$stmt_packages = $conn->prepare($sql_packages);
$stmt_packages->bind_param("i", $group_id);
$stmt_packages->execute();
$packages = $stmt_packages->get_result();
```

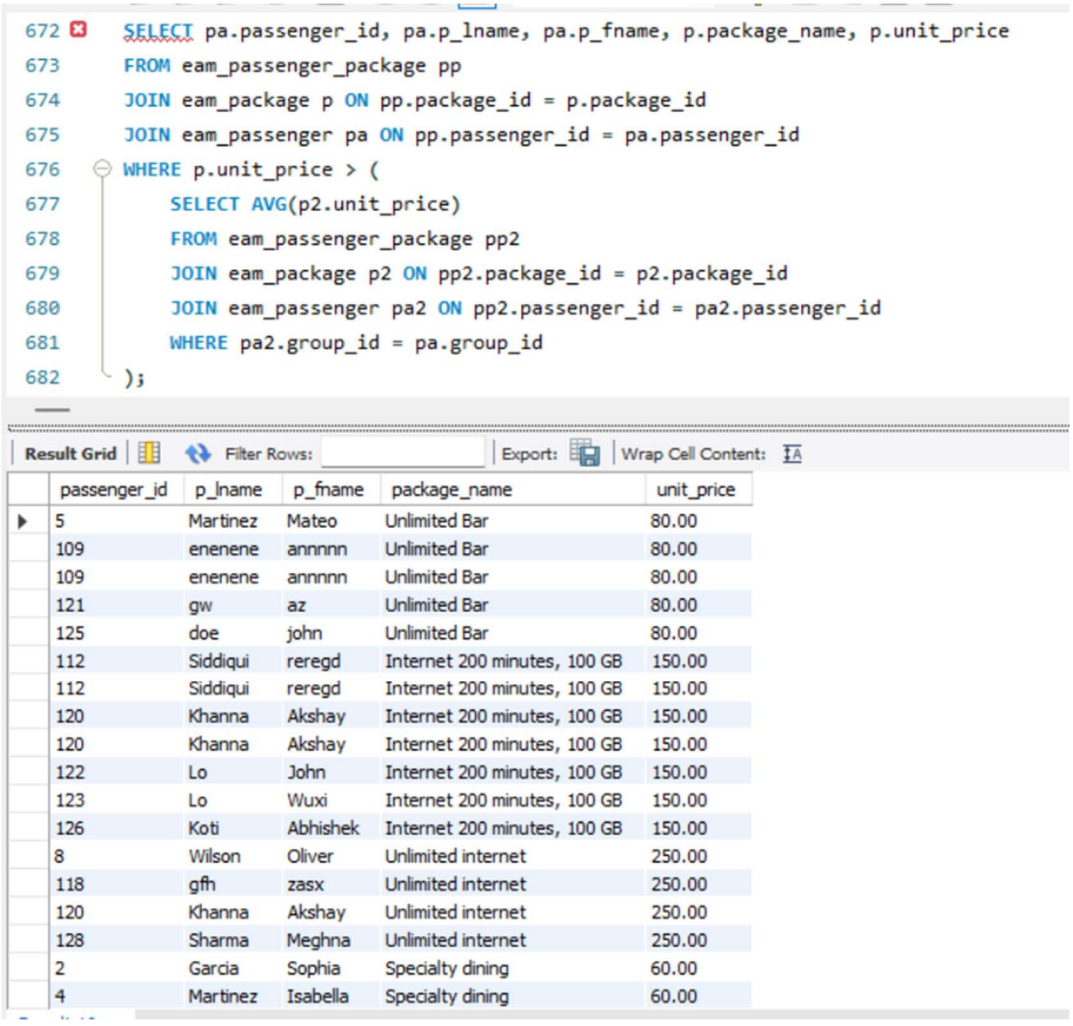
Website Displaying the above Select Statement in the Invoice:

Q3) Correlated subquery.

3A1) Select Statement:

```
SELECT pa.passenger_id, pa.p_lname, pa.p_fname, p.package_name, p.unit_price
FROM eam_passenger_package pp
JOIN eam_package p ON pp.package_id = p.package_id
JOIN eam_passenger pa ON pp.passenger_id = pa.passenger_id
WHERE p.unit_price > (
   SELECT AVG(p2.unit_price)
   FROM eam_passenger_package pp2
   JOIN eam_package p2 ON pp2.package_id = p2.package_id
   JOIN eam_passenger pa2 ON pp2.passenger_id = pa2.passenger_id
   WHERE pa2.group_id = pa.group_id
);
```

3A2) Result of the Query:



```
672  SELECT pa.passenger_id, pa.p_lname, pa.p_fname, p.package_name, p.unit_price
673  FROM eam_passenger_package pp
674  JOIN eam_package p ON pp.package_id = p.package_id
675  JOIN eam_passenger pa ON pp.passenger_id = pa.passenger_id
676  WHERE p.unit_price > (
677      SELECT AVG(p2.unit_price)
678      FROM eam_passenger_package pp2
679      JOIN eam_package p2 ON pp2.package_id = p2.package_id
680      JOIN eam_passenger pa2 ON pp2.passenger_id = pa2.passenger_id
681      WHERE pa2.group_id = pa.group_id
682  );
```

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

passenger_id	p_lname	p_fname	package_name	unit_price
5	Martinez	Mateo	Unlimited Bar	80.00
109	enenene	annnnn	Unlimited Bar	80.00
109	enenene	annnnn	Unlimited Bar	80.00
121	gw	az	Unlimited Bar	80.00
125	doe	john	Unlimited Bar	80.00
112	Siddiqui	reregd	Internet 200 minutes, 100 GB	150.00
112	Siddiqui	reregd	Internet 200 minutes, 100 GB	150.00
120	Khanna	Akshay	Internet 200 minutes, 100 GB	150.00
120	Khanna	Akshay	Internet 200 minutes, 100 GB	150.00
122	Lo	John	Internet 200 minutes, 100 GB	150.00
123	Lo	Wuxi	Internet 200 minutes, 100 GB	150.00
126	Koti	Abhishek	Internet 200 minutes, 100 GB	150.00
8	Wilson	Oliver	Unlimited internet	250.00
118	gfh	zasx	Unlimited internet	250.00
120	Khanna	Akshay	Unlimited internet	250.00
128	Sharma	Meghna	Unlimited internet	250.00
2	Garcia	Sophia	Specialty dining	60.00
4	Martinez	Isabella	Specialty dining	60.00

3A3) The business question that query is answering to:

Business Question can be: Find all passengers who have purchased a package more expensive than the average package unit price purchased by passengers in their own group.

This is a common question that the company may want to look at to analyze what is the trend in package purchasing, and how much each package is being purchased.

The outer query selects passenger details along with the packages they purchased.
The inner (correlated) subquery computes the average unit price of packages purchased by all passengers in that same passenger's group.
The correlation occurs through pa.group_id in both the outer and inner queries.

Q4) SET operator query.

4A1) Select Statement:
USING SET OPERATOR <u>EXCEPT</u>:

```
SELECT pa.passenger_id
FROM eam_passenger pa
JOIN eam_passenger_package pp ON pa.passenger_id = pp.passenger_id

EXCEPT

SELECT pa2.passenger_id
FROM eam_passenger pa2
JOIN eam_passenger_package pp2 ON pa2.passenger_id = pp2.passenger_id
JOIN eam_package p ON pp2.package_id = p.package_id
WHERE p.package_name = 'Speciality dining';
```

4A2) Result of the Query:

```
715  SELECT pa.passenger_id
716  FROM eam_passenger pa
717  JOIN eam_passenger_package pp ON pa.passenger_id = pp.passenger_id
718
719  EXCEPT
720
721  SELECT pa2.passenger_id
722  FROM eam_passenger pa2
723  JOIN eam_passenger_package pp2 ON pa2.passenger_id = pp2.passenger_id
724  JOIN eam_package p ON pp2.package_id = p.package_id
725  WHERE p.package_name = 'Speciality dining';
```

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

passenger_id
2
3
4
5
6
7
8
9
10
11

4A3) The business question that query is answering to:

Business Question can be: Which passengers have purchased at least one package, but have never purchased the 'Speciality dining Package'

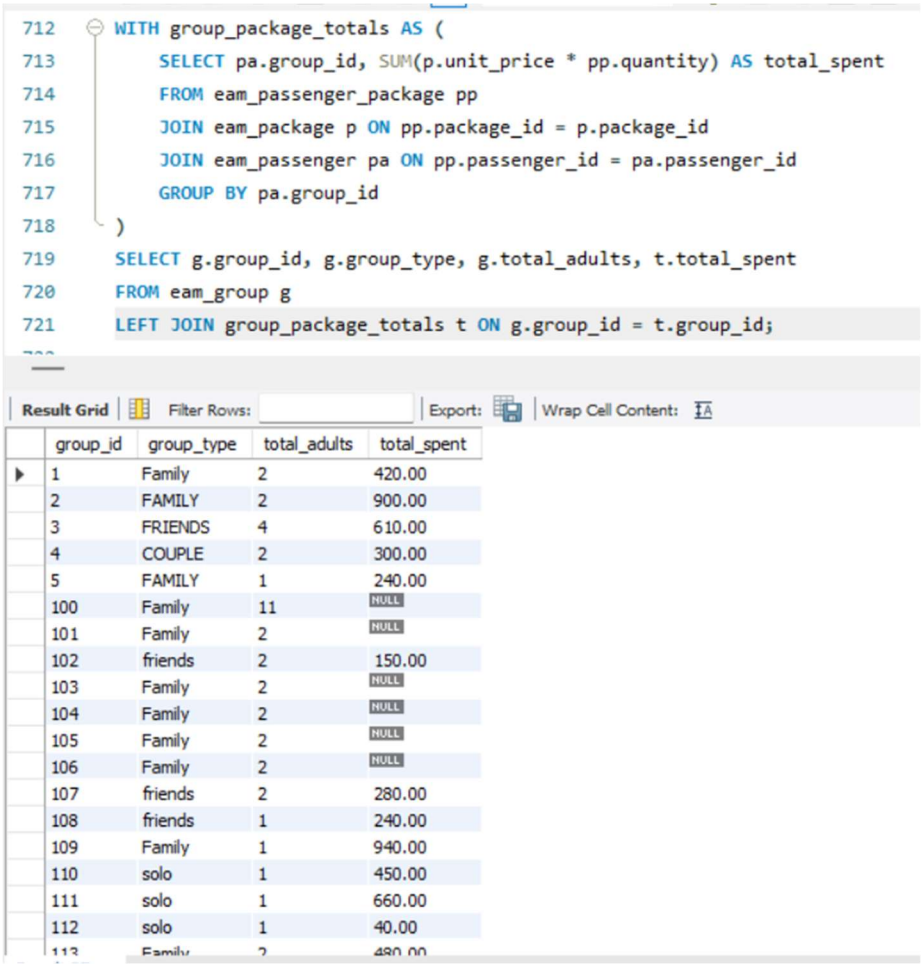
The first query retrieves all passengers who have purchased any package at all.
The second query retrieves all passengers who specifically purchased the 'Basic Meal Package'.
By using EXCEPT, we get all package-purchasing passengers except those who bought that particular package.

Q5) Query with in-line view or WITH clause

5A1) Select Statement:

```
WITH group_package_totals AS (
    SELECT pa.group_id, SUM(p.unit_price * pp.quantity) AS total_spent
    FROM eam_passenger_package pp
    JOIN eam_package p ON pp.package_id = p.package_id
    JOIN eam_passenger pa ON pp.passenger_id = pa.passenger_id
    GROUP BY pa.group_id
)
SELECT g.group_id, g.group_type, g.total_adults, t.total_spent
FROM eam_group g
LEFT JOIN group_package_totals t ON g.group_id = t.group_id;
```

5A2) Result of the Query:



```
712  WITH group_package_totals AS (
713      SELECT pa.group_id, SUM(p.unit_price * pp.quantity) AS total_spent
714      FROM eam_passenger_package pp
715      JOIN eam_package p ON pp.package_id = p.package_id
716      JOIN eam_passenger pa ON pp.passenger_id = pa.passenger_id
717      GROUP BY pa.group_id
718  )
719  SELECT g.group_id, g.group_type, g.total_adults, t.total_spent
720  FROM eam_group g
721  LEFT JOIN group_package_totals t ON g.group_id = t.group_id;
```

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

group_id	group_type	total_adults	total_spent
1	Family	2	420.00
2	FAMILY	2	900.00
3	FRIENDS	4	610.00
4	COUPLE	2	300.00
5	FAMILY	1	240.00
100	Family	11	NULL
101	Family	2	NULL
102	friends	2	150.00
103	Family	2	NULL
104	Family	2	NULL
105	Family	2	NULL
106	Family	2	NULL
107	friends	2	280.00
108	friends	1	240.00
109	Family	1	940.00
110	solo	1	450.00
111	solo	1	660.00
112	solo	1	40.00
113	Family	2	480.00

5A3) The business question that query is answering to:

The Business Question would be: What is the total amount spent on packages by each group? We first calculate the total spending by each group on packages. Then we join it to the EAM_GROUP table to show group details alongside their total package spending.
This shows each group's total package expenditure.

Q6) TOP-N/BOTTOM-N query

6A1) Select Statement:

We implement this using Analytic Functions - RANK( )

```
SELECT passenger_id, p_lname, p_fname, total_spent
FROM (
   SELECT
      pa.passenger_id, pa.p_lname, pa.p_fname,
      SUM(p.unit_price * pp.quantity) AS total_spent,
      RANK() OVER (ORDER BY SUM(p.unit_price * pp.quantity) DESC) AS spend_rank
   FROM eam_passenger_package pp
   JOIN eam_package p ON pp.package_id = p.package_id
   JOIN eam_passenger pa ON pp.passenger_id = pa.passenger_id
   GROUP BY pa.passenger_id, pa.p_lname, pa.p_fname
) s
WHERE spend_rank <= 3;
```

6A2) Result of the Query:

```sql
SELECT passenger_id, p_lname, p_fname, total_spent
FROM (
    SELECT
        pa.passenger_id,
        pa.p_lname,
        pa.p_fname,
        SUM(p.unit_price * pp.quantity) AS total_spent,
        RANK() OVER (ORDER BY SUM(p.unit_price * pp.quantity) DESC) AS spend_rank
    FROM eam_passenger_package pp
    JOIN eam_package p ON pp.package_id = p.package_id
    JOIN eam_passenger pa ON pp.passenger_id = pa.passenger_id
    GROUP BY pa.passenger_id, pa.p_lname, pa.p_fname
) s
WHERE spend_rank <= 3;
```

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

passenger_id	p_lname	p_fname	total_spent
112	Siddiqui	reregd	940.00
120	Khanna	Akshay	1270.00
129	Rathod	Adarshini	750.00

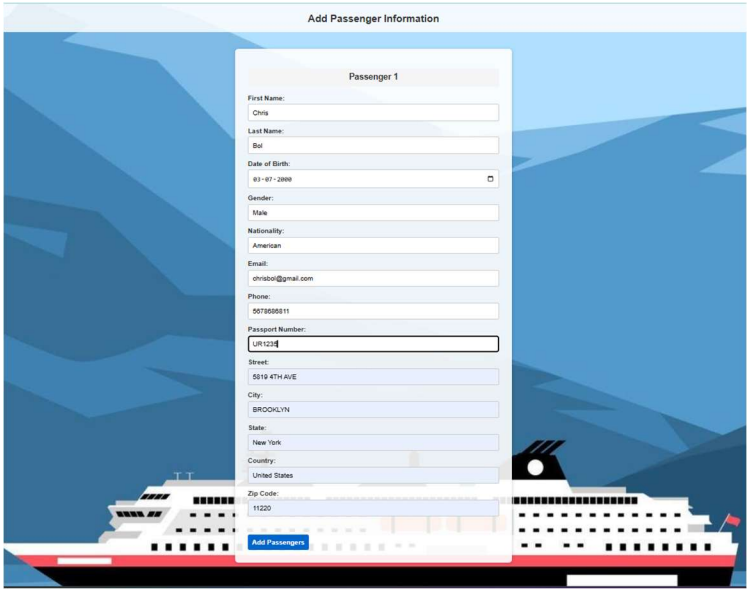
6A3) The business question that query is answering to:

The Business Question would be: TOP 3 passengers who have spent the most on packages?

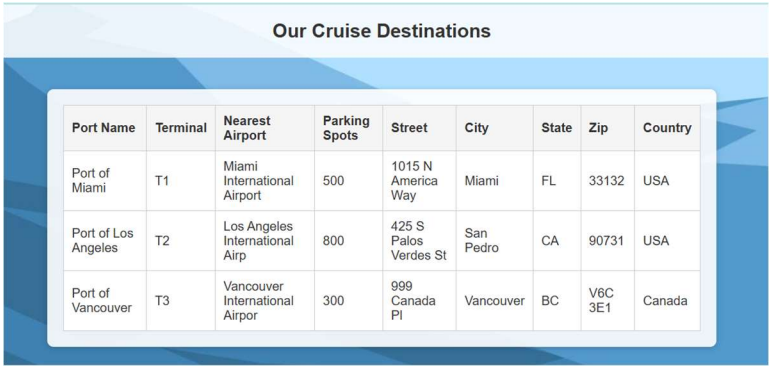
The inner query calculates the total amount each passenger spent on packages (SUM(p.unit_price * pp.quantity)), grouping by passenger details.
The RANK() analytic function assigns a rank to each passenger based on the total_spent in descending order (highest spenders get rank 1).
The outer query filters to include only the top 3 ranked passengers.
This query will return the passenger IDs, names, and their total spent amounts for the three top spenders.

CRUD OPERATIONS

Create: Inserting new data into the database
This is done in multiple places. One example is inserting data like New Passenger Details when booking a trip.



Read: Retrieves data from the database
This is also done in multiple places. One example is Fetching the ports that the cruise ship has for destinations.



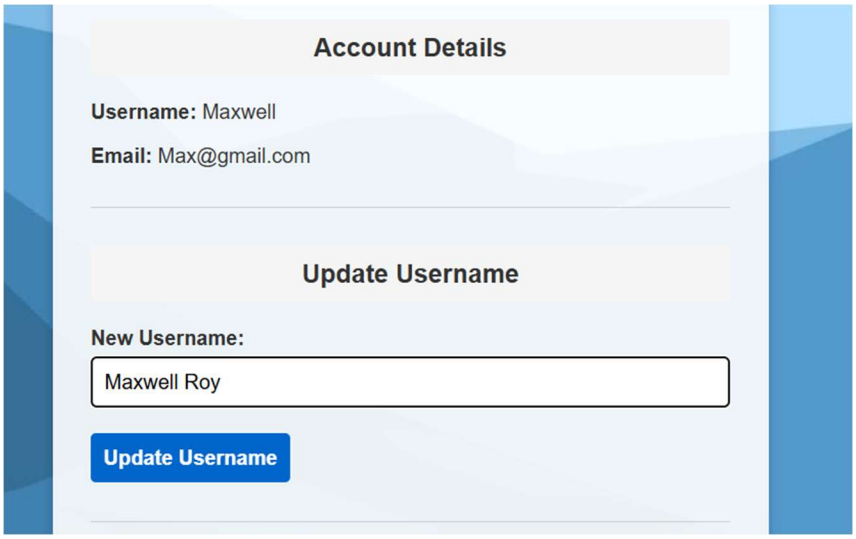
Port Name	Terminal	Nearest Airport	Parking Spots	Street	City	State	Zip	Country
Port of Miami	T1	Miami International Airport	500	1015 N America Way	Miami	FL	33132	USA
Port of Los Angeles	T2	Los Angeles International Airp	800	425 S Palos Verdes St	San Pedro	CA	90731	USA
Port of Vancouver	T3	Vancouver International Airpor	300	999 Canada Pl	Vancouver	BC	V6C 3E1	Canada

```
870 •  select * from eam_port
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

port_id	pr_name	pr_terminal	pr_nearest_airport	pr_parking_spots	pr_street	pr_city	pr_states	pr_zip	pr_country
1	Port of Miami	T1	Miami International Airport	500	1015 N America Way	Miami	FL	33132	USA
2	Port of Los Angeles	T2	Los Angeles International Airp	800	425 S Palos Verdes St	San Pedro	CA	90731	USA
3	Port of Vancouver	T3	Vancouver International Airpor	300	999 Canada Pl	Vancouver	BC	V6C 3E1	Canada
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Update: Modifies existing data in the database
This modification of data can be seen in the account page which will modify username and password of the user.

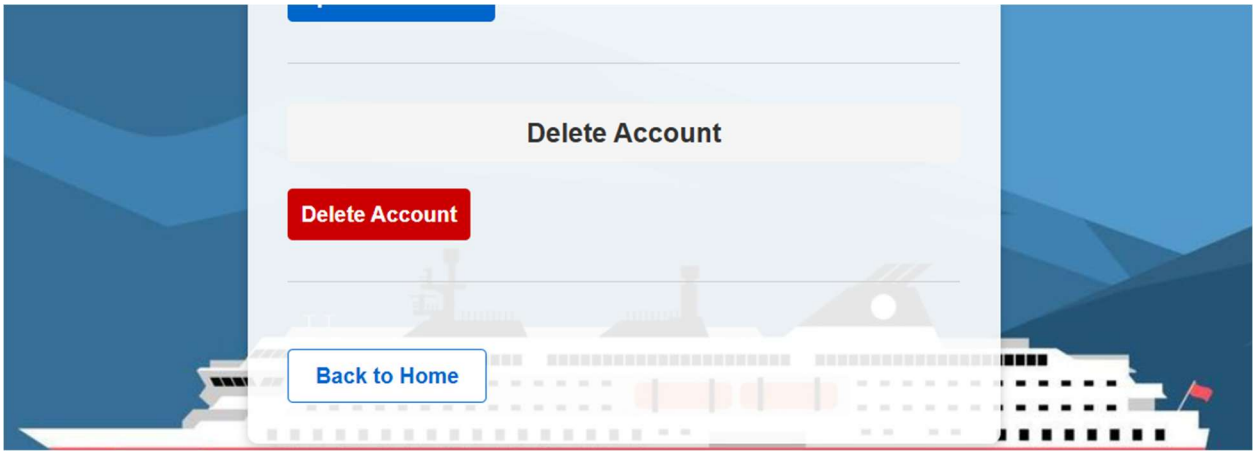



```
870 •  select * from users
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

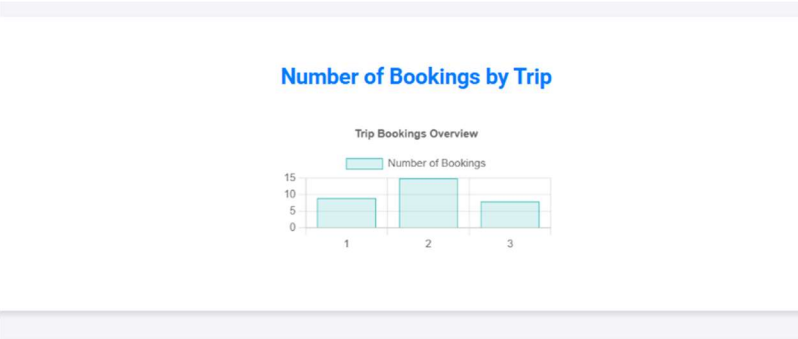
user_id	username	email	password	created_at
11	Robert01	robert@gmail.con	$2y$10$9DsoI6rbJlUopIAEuVR1teme.tcpJtkZm...	2024-12-07 14:46:20
12	Karthikeya	Karth@gmail.com	$2y$10$HOv1CnLGWgfjwVtzo3MJwe78yfa3hpi...	2024-12-07 17:22:57
13	Rahul	Rag@gmail.com	$2y$10$zzKmGtt9ZbG8wzNatoJhyehc.X69jaAm...	2024-12-08 02:59:01
14	Maxwell Roy	Max@gmail.com	$2y$10$bQtBxgAlicHxWdsddtVHyuiEvix99zI2zq...	2024-12-08 04:52:16
15	Chris	ChrisBol@gmail.com	$2y$10$hQsxA.qGgVcRpbx3E7xFPuvq40uAt09...	2024-12-08 04:57:18
NULL	NULL	NULL	NULL	NULL

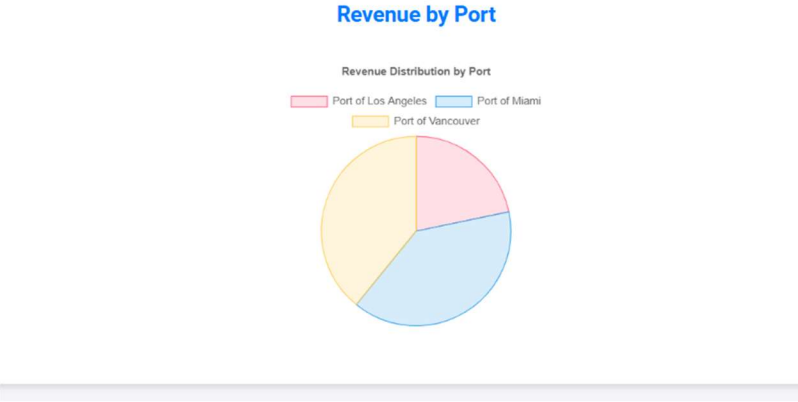
Delete: Removes data from the database
This can also be seen in accounts where we can delete a user account using the delete account button.

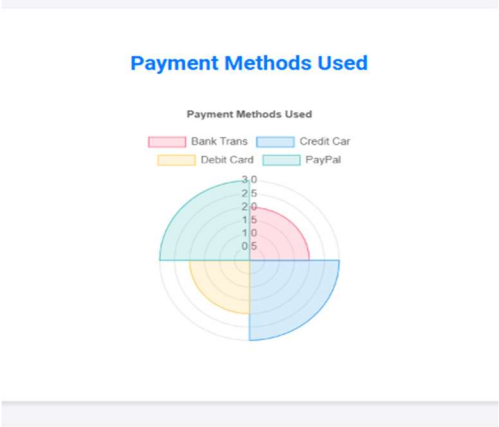


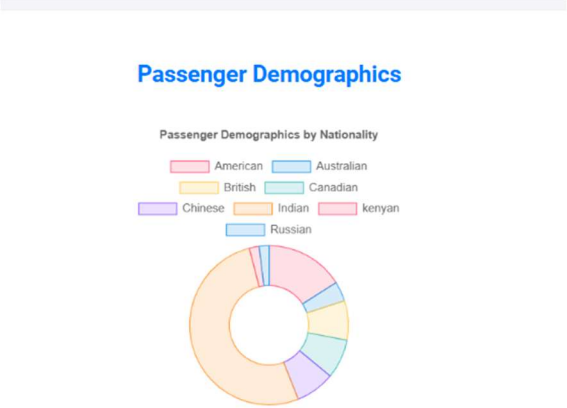
EXTRA FEATURES

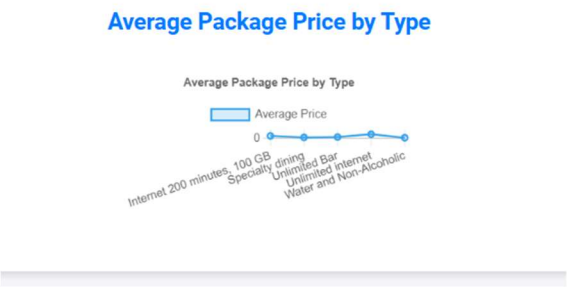
1) Data Visualization (data analysis by Graphs/Charts)

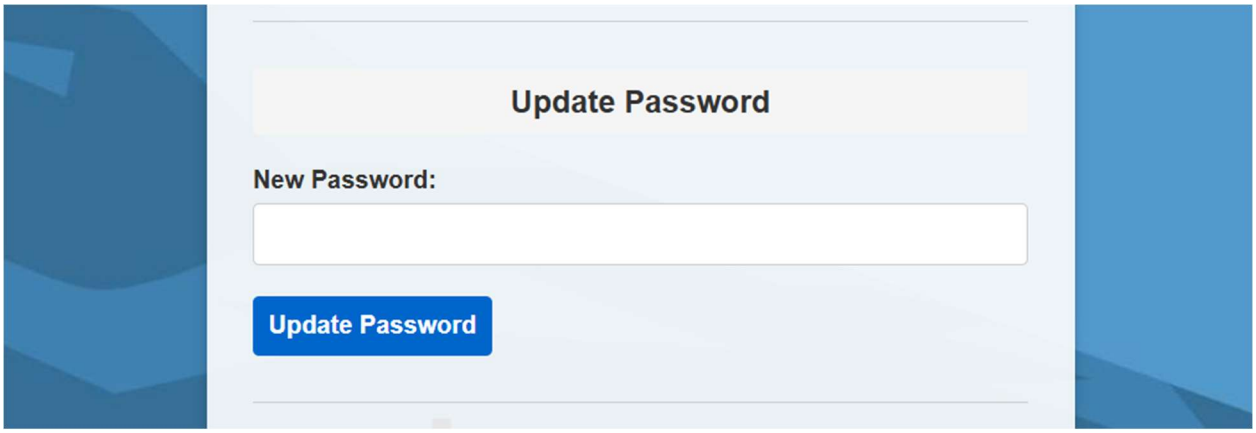













```
870 •  select * from users
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

user_id	username	email	password	created_at
11	Robert01	robert@gmail.con	$2y$10$9DsoI6rbJlUopIAEuVR1teme.tcpJtkZm...	2024-12-07 14:46:20
12	Karthikeya	Karth@gmail.com	$2y$10$HOv1CnLGWgfjwVtzo3MJwe78yfa3hpi...	2024-12-07 17:22:57
13	Rahul	Rag@gmail.com	$2y$10$zzKmGtt9ZbG8wzNatoJhyehc.X69jaAm...	2024-12-08 02:59:01
14	Maxwell Roy	Max@gmail.com	$2y$10$bQtBxgAlicHxWdsddtVHyuiEvix99zI2zq...	2024-12-08 04:52:16
15	Chris	ChrisBol@gmail.com	$2y$10$hQsxA.qGgVcRpbx3E7xFPuvq40uAt09...	2024-12-08 04:57:18
NULL	NULL	NULL	NULL	NULL

3) Stored Procedures

```
912
913  DELIMITER //
914
915 •  CREATE PROCEDURE sp_loginUser(
916      IN input_email VARCHAR(255),
917      OUT output_user_id INT,
918      OUT output_password_hash VARCHAR(255)
919  )
920  BEGIN
921      SELECT user_id, password
922      INTO output_user_id, output_password_hash
923      FROM users
924      WHERE email = input_email;
925  END //
926
927  DELIMITER ;
928
```