**Phase II Report – Restaurant Reservation System**

**1. Introduction and Goals**

The Restaurant Reservation System was developed to streamline the process of managing restaurant reservations, customer data, and table allocations. The motivation behind this project is to reduce manual scheduling errors, prevent double-booking, and improve restaurant efficiency through data-driven insights.

The main objectives of the system are:

* To provide a centralized database for restaurant, customer, and reservation management.
* To ensure referential integrity across all relational entities.
* To support analytical queries and visual reporting using SQL views.
* To enable future integration with web and mobile applications for real-time updates.

**2. Relational Schema and SQL Code**

This section defines the relational schema, constraints, and data relationships used in the database. The design ensures data normalization and consistency across multiple interconnected entities.

**Key Design Features:**

* **Foreign Keys** maintain relationships between entities such as Reservation, Customer, and Restaurant.
* **Primary Keys** uniquely identify records across all tables.
* **Constraints** (e.g., CHECK, UNIQUE, ENUM) guarantee data validity.

**SQL Schema:**

CREATE TABLE Restaurant (

restaurant\_id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100) NOT NULL,

location VARCHAR(100),

email VARCHAR(100),

opening\_time TIME,

closing\_time TIME

);

CREATE TABLE Table\_Info (

table\_id INT AUTO\_INCREMENT PRIMARY KEY,

restaurant\_id INT NOT NULL,

capacity INT NOT NULL CHECK (capacity > 0),

table\_number INT NOT NULL,

FOREIGN KEY (restaurant\_id) REFERENCES Restaurant(restaurant\_id)

ON DELETE CASCADE

ON UPDATE CASCADE

);

CREATE TABLE Customer (

customer\_id INT AUTO\_INCREMENT PRIMARY KEY,

full\_name VARCHAR(100) NOT NULL,

email VARCHAR(100) UNIQUE,

phone VARCHAR(20)

);

CREATE TABLE Cuisine (

cuisine\_id INT AUTO\_INCREMENT PRIMARY KEY,

cuisine\_name VARCHAR(50) UNIQUE NOT NULL

);

CREATE TABLE Restaurant\_Cuisine (

restaurant\_id INT NOT NULL,

cuisine\_id INT NOT NULL,

PRIMARY KEY (restaurant\_id, cuisine\_id),

FOREIGN KEY (restaurant\_id) REFERENCES Restaurant(restaurant\_id)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (cuisine\_id) REFERENCES Cuisine(cuisine\_id)

ON DELETE CASCADE

ON UPDATE CASCADE

);

CREATE TABLE Reservation (

reservation\_id INT AUTO\_INCREMENT PRIMARY KEY,

customer\_id INT NOT NULL,

restaurant\_id INT NOT NULL,

table\_id INT NOT NULL,

reservation\_date DATE NOT NULL,

start\_time TIME NOT NULL,

end\_time TIME NOT NULL,

num\_people INT NOT NULL CHECK (num\_people > 0),

status ENUM('Booked', 'Completed', 'Cancelled') DEFAULT 'Booked',

FOREIGN KEY (customer\_id) REFERENCES Customer(customer\_id)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (restaurant\_id) REFERENCES Restaurant(restaurant\_id)

ON DELETE CASCADE

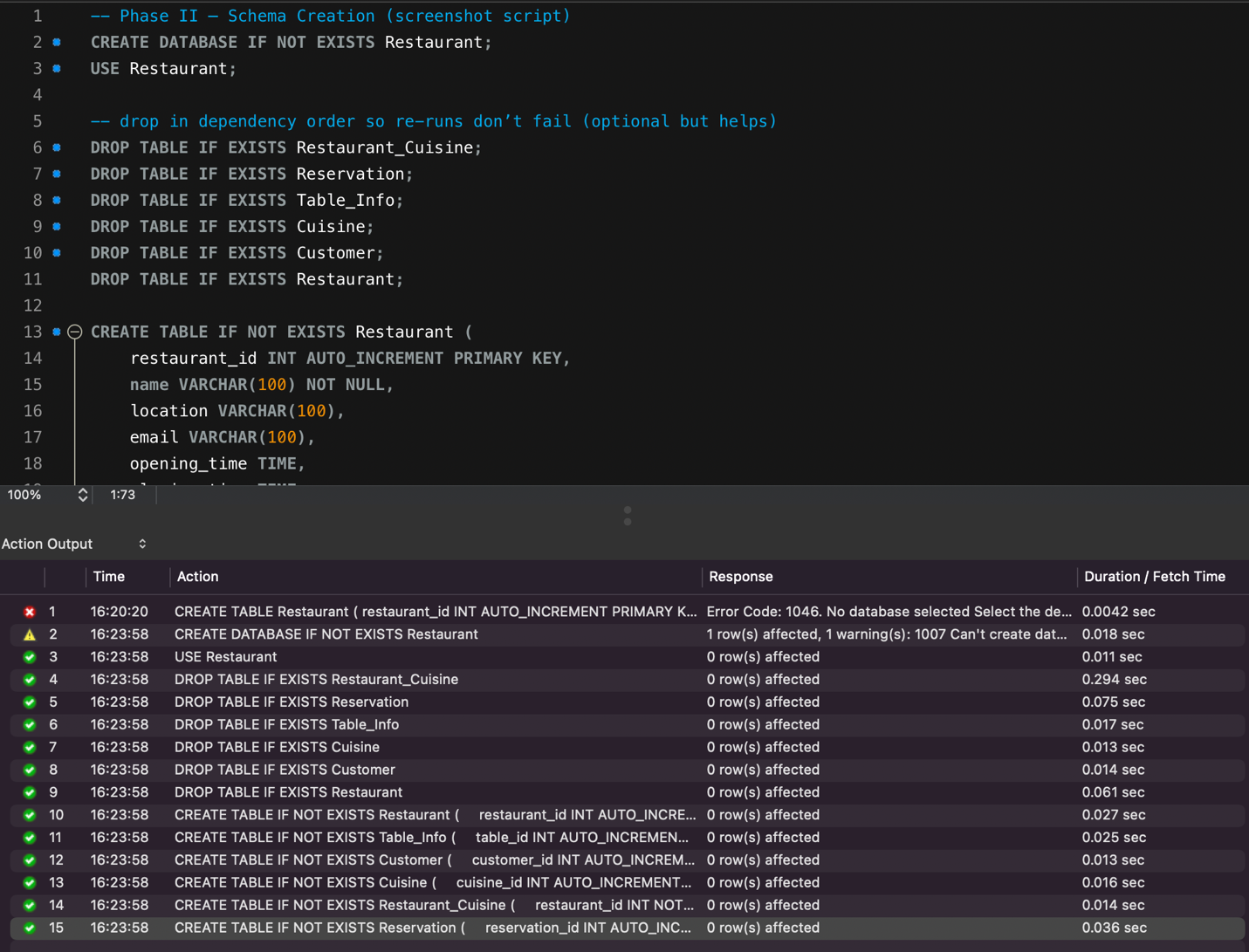
ON UPDATE CASCADE,

FOREIGN KEY (table\_id) REFERENCES Table\_Info(table\_id)

ON DELETE CASCADE

ON UPDATE CASCADE

);

**Screenshot Reference:**  


**3. ER Diagram**

The Entity-Relationship (ER) diagram provides a visual overview of all entities and their relationships. It ensures database normalization and depicts how tables interconnect.

**Entity Summaries:**

* **Restaurant:** Stores details about each restaurant, including operating hours.
* **Customer:** Contains personal and contact details for reservation tracking.
* **Reservation:** Manages booking details, connecting customers and restaurants.
* **Table\_Info:** Represents tables with capacity and numbering.
* **Cuisine & Restaurant\_Cuisine:** Define restaurant categories and their associations.

**ER Diagram:**  
A diagram of a restaurant

AI-generated content may be incorrect.

**4. Views and Data Retrieval**

This section presents ten SQL views created for the Restaurant Reservation System database. Each view is designed to simplify querying, improve maintainability, and provide quick access to key operational and analytical information. For each view, a description, SQL definition, and sample result are included.

**View 1: view\_full\_reservation\_info**

**Purpose:**  
Displays a complete overview of reservations, including restaurant, customer, and table details.

**SQL Definition:**

CREATE VIEW view\_full\_reservation\_info AS

SELECT rsv.reservation\_id, c.full\_name AS customer\_name, rest.name AS restaurant\_name,

rsv.reservation\_date, rsv.start\_time, rsv.end\_time, t.capacity, rsv.num\_people, rsv.status

FROM Reservation rsv

JOIN Customer c ON rsv.customer\_id = c.customer\_id

JOIN Restaurant rest ON rsv.restaurant\_id = rest.restaurant\_id

JOIN Table\_Info t ON rsv.table\_id = t.table\_id;

**Description:**  
Provides unified reservation details for reports and dashboards.

**Sample Output:**  
A screenshot of a computer

AI-generated content may be incorrect.

**View 2: view\_popular\_restaurants**

**Purpose:**  
Shows restaurants ranked by total number of reservations.

**SQL Definition:**

CREATE VIEW view\_popular\_restaurants AS

SELECT r.restaurant\_id, r.name AS restaurant\_name, COUNT(rs.reservation\_id) AS total\_reservations

FROM Reservation rs

JOIN Restaurant r ON rs.restaurant\_id = r.restaurant\_id

GROUP BY r.restaurant\_id, r.name

ORDER BY total\_reservations DESC;

**Sample Output:**  
A screenshot of a computer

AI-generated content may be incorrect.

**View 3: view\_available\_tables**

**Purpose:**  
Lists tables that are not reserved for a given date.

**SQL Definition:**

CREATE VIEW view\_available\_tables AS

SELECT t.table\_id, t.restaurant\_id, r.name AS restaurant\_name, t.table\_number, t.capacity

FROM Table\_Info t

JOIN Restaurant r ON r.restaurant\_id = t.restaurant\_id

WHERE t.table\_id NOT IN (

SELECT table\_id FROM Reservation WHERE reservation\_date = CURDATE()

);

**Sample Output:**  
A screenshot of a computer

AI-generated content may be incorrect.

**View 4: view\_customer\_history**

**Purpose:**  
Displays a list of all past reservations made by each customer.

**SQL Definition:**

CREATE VIEW view\_customer\_history AS

SELECT c.full\_name, r.name AS restaurant\_name, rs.reservation\_date, rs.num\_people, rs.status

FROM Reservation rs

JOIN Customer c ON rs.customer\_id = c.customer\_id

JOIN Restaurant r ON rs.restaurant\_id = r.restaurant\_id

WHERE rs.reservation\_date < CURDATE();

**Sample Output:**  
A screenshot of a computer

AI-generated content may be incorrect.

**View 5: view\_cuisine\_summary**

**Purpose:**  
Summarizes how many restaurants offer each cuisine type.

**SQL Definition:**

CREATE VIEW view\_cuisine\_summary AS

SELECT c.cuisine\_name, COUNT(rc.restaurant\_id) AS total\_restaurants

FROM Cuisine c

JOIN Restaurant\_Cuisine rc ON c.cuisine\_id = rc.cuisine\_id

GROUP BY c.cuisine\_name;

**Sample Output:**  
A screenshot of a computer

AI-generated content may be incorrect.

**View 6: view\_reservations\_by\_day**

**Purpose:**  
Aggregates reservation counts by day of week.

**SQL Definition:**

CREATE VIEW view\_reservations\_by\_day AS

SELECT DAYNAME(reservation\_date) AS day\_of\_week, COUNT(\*) AS total\_reservations

FROM Reservation

GROUP BY DAYNAME(reservation\_date)

ORDER BY total\_reservations DESC;

**Sample Output:**  
A screenshot of a computer

AI-generated content may be incorrect.

**View 7: view\_active\_bookings**

**Purpose:**  
Displays all upcoming reservations.

**SQL Definition:**

CREATE VIEW view\_active\_bookings AS

SELECT rsv.reservation\_id, c.full\_name, rest.name AS restaurant\_name,

rsv.reservation\_date, rsv.start\_time, rsv.status

FROM Reservation rsv

JOIN Customer c ON rsv.customer\_id = c.customer\_id

JOIN Restaurant rest ON rsv.restaurant\_id = rest.restaurant\_id

WHERE rsv.reservation\_date >= CURDATE() AND rsv.status = 'Booked';

**Sample Output:**  
A screenshot of a computer

AI-generated content may be incorrect.

**View 8: view\_table\_utilization**

**Purpose:**  
Shows number of reservations per table, useful for load analysis.

**SQL Definition:**

CREATE VIEW view\_table\_utilization AS

SELECT t.table\_id, r.name AS restaurant\_name, COUNT(rs.reservation\_id) AS total\_reservations

FROM Table\_Info t

LEFT JOIN Reservation rs ON rs.table\_id = t.table\_id

JOIN Restaurant r ON r.restaurant\_id = t.restaurant\_id

GROUP BY t.table\_id, r.name;

**Sample Output:**  
A screenshot of a computer

AI-generated content may be incorrect.

**View 9: view\_top\_customers**

**Purpose:**  
Identifies customers with the highest number of reservations.

**SQL Definition:**

CREATE VIEW view\_top\_customers AS

SELECT c.full\_name, COUNT(rs.reservation\_id) AS total\_reservations

FROM Reservation rs

JOIN Customer c ON rs.customer\_id = c.customer\_id

GROUP BY c.full\_name

ORDER BY total\_reservations DESC;

**Sample Output:**  
A screenshot of a computer

AI-generated content may be incorrect.

**View 10: view\_cancelled\_reservations**

**Purpose:**  
Lists all reservations that were cancelled.

**SQL Definition:**

CREATE VIEW view\_cancelled\_reservations AS

SELECT rs.reservation\_id, c.full\_name, r.name AS restaurant\_name, rs.reservation\_date, rs.status

FROM Reservation rs

JOIN Customer c ON rs.customer\_id = c.customer\_id

JOIN Restaurant r ON rs.restaurant\_id = r.restaurant\_id

WHERE rs.status = 'Cancelled';

**Sample Output:**  
A screenshot of a computer

AI-generated content may be incorrect.

**5. Contribution Matrix**

| **Team Member** | **Section(s) Contributed** | **Description of Work** |
| --- | --- | --- |
| Mohammad Taqi | Document Lead, Views | Authored report, integrated SQL, coordinated sections |
| Ayaan Ahmed | Screenshots, Testing | Captured database outputs and visual evidence |
| Mohamed Ali | Schema & Integrity | Developed and tested constraints and relationships |
| Ali Hakkani | Report Formatting, ERD | Created ER diagram and formatted report structure |

**6. Conclusion**

The Phase II implementation successfully integrates schema design, referential integrity, and analytical data views for the Restaurant Reservation System. Each module of the project aligns with database management principles, enabling efficient querying, strong data consistency, and readiness for future application development.

**Next Steps:**

* Implement stored procedures for automated booking conflict checks.
* Connect to a web front-end for user-based reservation management.
* Extend analytics to track daily performance trends and cancellation ratios.

**End of Report**