DATABASE PROJECT

RESTAURANT (FOOD CHAIN) RESERVATION SYSTEM

DILEVERABLE 3

PRESENTED TO:

DR. IRFANA BIBI

PRESENTED BY:

GROUP 11

NAME	ROLL#
M. DANIYAL QURESHI	BCSF24A051
M. FARRUKH HUMAYUN	BCSF24A042
M. ABDULLAH SANI	BCSF24A014
M. UZAIR KHAN	BCSF24A004

INTRODUCTION:

Restaurant (Food Chain) Reservation System allows one to make table reservations in any of the branches of a particular restaurant chain through a web-based service. The customers can easily register and browse the availability of tables accordingly to book their tables online. The managers of the restaurants can either approve or reject bookings, manage details of tables, and review customer feedback. The admin will have full control over the branches, users, and records of reservations.

The system will reduce manual booking mistakes, minimize double bookings, and provide customers with more convenience in making reservations from anywhere and at any time.

OBJECTIVES:

- To give clients a simple way to book tables online.
- To enable effective reservation management by restaurant managers.
- To assist administrators in keeping track of all branches records.
- To use an automated reservation process to avoid scheduling conflicts.
- To provide instant confirmations in order to save time and increase customer satisfaction.

FUNCTIONAL REQUIREMENTS

Functional requirements describe what the system will do, so the requirements for this project are listed below:

- FR-1: The system must allow managers to add, update, or delete restaurant branches.
- FR-2: Customers must be able to reserve tables at a specific branch, date, and time.
- FR-3: Customers must create an account to make reservations.
- FR-4: The system must generate a confirmation number for every reservation.
- FR-5: Customers must be able to modify or cancel their reservations.
- FR-6: The system must check table availability before confirming a reservation.
- FR-7: Admin must be able to manage (add/update/view) branches functionality.
- FR-8: Customers can leave a rating and feedback after dining.

NON-FUNCTIONAL REQUIREMENTS

Non-functional requirements describe how the system should perform and the quality standards it must maintain, these are:

- **Performance:** The system should respond to every user request within 3 seconds.
- **Availability:** The system should be accessible at any time.
- **Usability:** The interface should be simple, attractive, and easy to navigate.
- The system should be able to handle multiple users and branches at the same time.
- **Reliability:** The system must not lose data in case of unexpected failures.
- **Maintainability:** The system should be easy for developers to update and maintain.

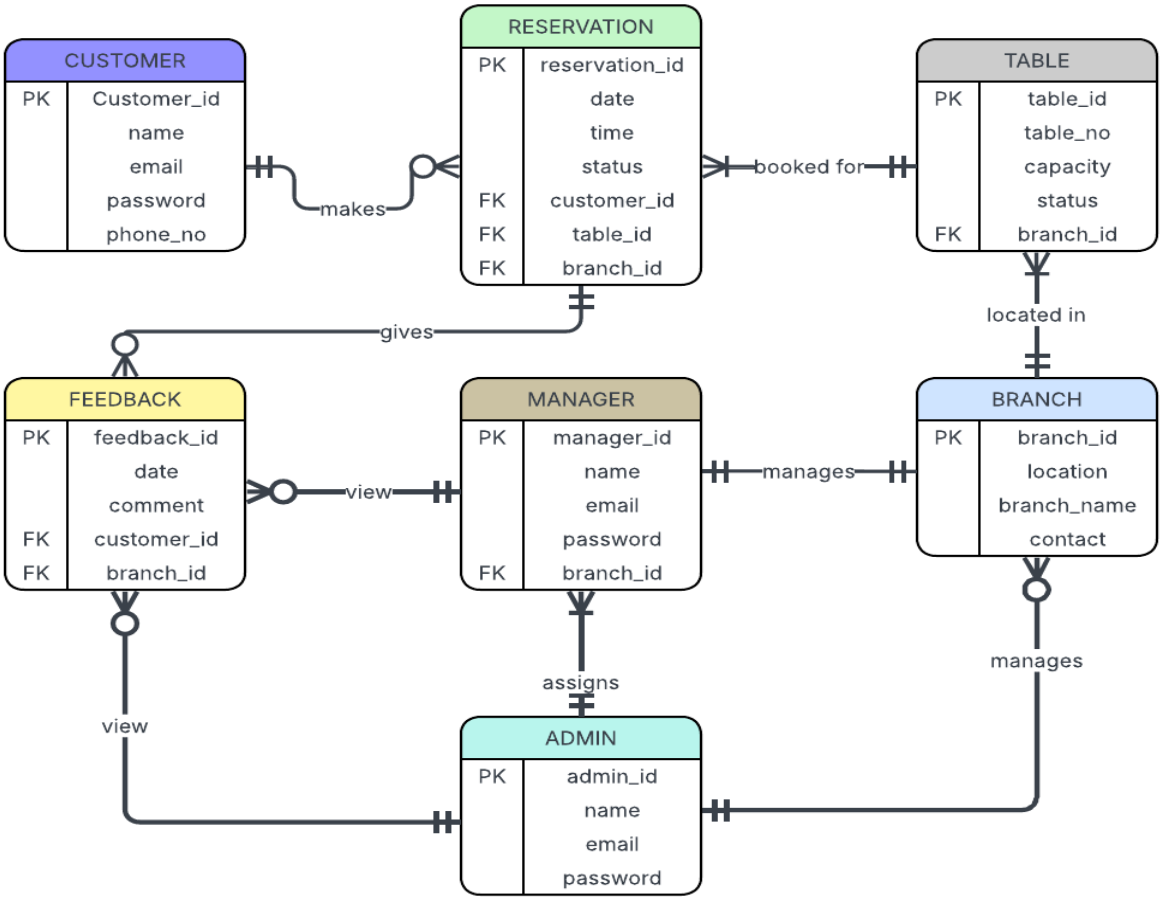
- **Portability:** The system should function properly on mobile devices, tablets, and desktops.

BUSINESS RULES

Business rules for Restaurant (Food Chain) Reservation Systems are:

- There should be at least one table assigned to each restaurant branch.
- Every branch must have only one manager who will oversee the branch operations.
- One customer may have several reservations, but only one per table in a given time slot.
- Customers can make table reservations up to 7 days in advance, but they cannot for dates in the past.
- Reservations must be made at least 1 hour prior to the time of intended dining.
- Customers can cancel the reservation up to 2 hours in advance.
- The system should check table availability to avoid confirmation of any double booking.
- Only registered and currently logged-in customers may create, modify, or cancel reservations.
- Each branch manager can only modify reservations for the specific branch they manage.
- Only customers who have made a confirmed reservation are able to leave feedback.
- The admin has full authority to add, edit, or delete branches, managers, and bookings.
- If customers have not arrived within 30 minutes from the reserved time, the table will automatically be released for new reservations.

ER DIAGRAM



RELATIONAL SCHEMA

CUSTOMER :

Customer(<u>CustomerID</u>, Name, Email, Phone, Password)

BRANCH:

Branch(<u>BranchID</u>, BranchName, Location, Contact)

TABLE:

Table(<u>TableID</u>, TableNumber, Capacity, Status, BranchID*)

• BranchID* references Branch (BranchID)

RESERVATION:

Reservation(<u>ReservationID</u>, Date, Time, Status, CustomerID*, TableID*, BranchID*)

• CustomerID* references Customer (CustomerID)

• TableID* references Table (TableID)

• BranchID* references Branch (BranchID)

FEEDBACK:

Feedback(<u>FeedbackID</u>, Date, Comment, CustomerID*, BranchID*)

• CustomerID* references Customer (CustomerID)

• BranchID* references Branch (BranchID)

MANAGER:

Manager(<u>ManagerID</u>, Name, Email, Password, BranchID*)

• BranchID* references Branch (BranchID)

ADMIN:

Admin(<u>AdminID</u>, Name, Email, Password)

DDL & DML

- We designed and created all essential tables for the restaurant reservation system, including branches, customers, tables, reservations, feedback, managers, and admins.

- Each table was built with proper data types, primary keys, foreign keys, and constraints to maintain data integrity and enforce relationships between entities.
- The database structure was fully normalized, ensuring organized storage of information.
- We inserted sample data for all major entities to simulate real-world operations and test how the system responds under different scenarios.
- Various DML operations were performed to add, update, and retrieve data, confirming that the tables interacted correctly through their relationships.
- The test data helped verify that stored procedures, triggers, and constraints worked properly, such as handling reservation cancellations, table status updates, and rating validation.
- Overall, the combination of DDL and DML work ensured a reliable, accurate, and functional database environment ready for operational use.

NORMALIZATION

The database schema has been evaluated through all major levels of normalization, and it successfully meets the conditions up to Boyce–Codd Normal Form (BCNF).

1NF:
All tables use primary keys, every attribute contains a single value, and there are no repeating groups or mixed concepts in any column. This means the structure is properly organized at the most basic level.

2NF:
Since the tables that use composite keys do not have partial dependencies, and all other tables use simple primary keys, every non-key attribute depends on the whole primary key. Therefore, no partial dependencies exist, and the schema satisfies 2NF.

3NF:
There are no transitive dependencies, non-key attributes do not rely on other non-key attributes. Each attribute depends directly on its table's primary key, confirming the schema meets 3NF requirements.

BCNF:
The schema further satisfies the stricter conditions of BCNF. In every table, the primary key is the sole determinant, meaning all functional dependencies have a candidate key on the left side. There are no overlapping candidate keys, no attributes determining part of a key, and no situations where a non-key attribute determines another attribute. Because each determinant is a super key, the schema fully meets the requirements of BCNF.

Overall:

The database schema is well-structured, free from unnecessary redundancy, and successfully normalized up to 3NF.

1-Table Customer

Customer-id(PK)	Name	Email	Password	Phone_no

Functional Dependencies:

Name,Email,Password and phone all depend upon Customer_id(PK).

2-Table Reservation

Reservation_id(pk)	Date	Time	Status	Customer_id(FK)	Table_id(FK)	Branch_id(FK)

Functional Dependencies:

Date,Time,Status,Customer_id(FK),Table_id(FK) and Branch_id(PK) all depend upon Reservation_id.

3-Table Table:

Table_id(PK)	Table_No	Capacity	Status	Branch_id(FK)

Functional Dependencies:

Table_No,Capacity,Status,Branch_id(FK) all depend upon Table_id.

4-Table Branch:

Branch_id(PK)	Location	Branch_name	Contact

Functional Dependencies:

Location,Branch_name,Contact all depend upon Branch_id.

5-Table Manager:

Manager_id(PK)	Name	Email	Password	Branch_id(FK)

Functional Dependencies:

Name,Email,Password,Branch_id(FK) all depend upon Manager_id.

6-Table Feedback:

Feedback_id(PK)	Date	Comment	Customer_id(FK)	Branch_id(FK)

Functional Dependencies:

Date,Comment,Customer_id(FK) ,Branch_id(FK) all depend upon Feedback_id.

7-Table Admin:

Admin_id(PK)	Name	Email	Password

Functional Dependencies:

Name ,Email and password all depend upon Admin_id.

STORED LOGIC & TEST SCRIPT

Stored Procedure: (sp_cancel_reservation)

What the Procedure Does?

This procedure cancels a reservation by finding the table linked to it, changing the reservation status to "cancelled," and updating the table status to "available." It commits the updates and handles errors such as non-existent reservation IDs.

Related Test Script:

Test 1: Successful Cancellation

- Insert a test reservation with ID 100.
- Confirm the reservation was inserted.
- Call sp_cancel_reservation(100).
- Check that the reservation status is now "cancelled."
- Check that the table is now marked "available."

Test 2: Error Case

- Attempt to cancel reservation ID 9999 (which does not exist).
- Confirm that the procedure outputs an appropriate error message.

Stored Procedure: (sp_branch_stats)

What the Procedure Does?

- This procedure displays statistics for a branch.
- It counts how many reservations the branch has and calculates its average customer rating.
- If no feedback exists, the average rating is shown as zero.

Related Test Script:

Test 3: Branch Statistics

- Call sp_branch_stats(1) to display reservation count and average rating for Branch 1.

➢ **Verify the output by running manual SELECT queries:**

• Count of reservations for Branch 1

• Average of ratings for Branch 1

Trigger: trg_check_rating (Validates Feedback Ratings)

What the Trigger Does?

- This trigger activates before a feedback record is inserted.
- It checks that the rating is between 1.0 and 5.0.
- If the rating falls outside this range, the trigger blocks the insert.

Related Test Script:

Test 4: Valid Rating

- Insert a feedback record with rating 4.5.
- Confirm the feedback was inserted successfully.

Test 5: Invalid Rating

- Attempt to insert feedback with rating 6.0.
- The trigger blocks the insert and raises an error.
- Confirm that no feedback with this invalid rating exists in the table.

Trigger: trg_occupy_table (Automatically Marks Table as Occupied)

What the Trigger Does?

- This trigger runs after a reservation is inserted.
- It automatically changes the status of the reserved table to "occupied."

Related Test Script:

Test 6: Auto Table Occupancy

- Check the status of table 2 before the reservation.
- Insert a reservation for table 2.
- Confirm that table 2 is now marked “occupied.”

Cleanup Script:

What It Does?

- After all tests are completed, the script removes test records to return the database to its clean state.

Related Test Script:

- Delete all feedback and reservations with IDs 100 or higher.
- Confirm that all test data has been removed.

VIEWS & INDEXES

View: (vw_customer_res_summary)

What the View Shows?

This view provides a summary of each customer's reservation activity. It displays the customer name along with counts of total reservations and the number of confirmed, cancelled, and pending reservations. The view joins the CUSTOMER and RESERVATION tables. Customers without reservations are still included through a left join.

Purpose:

- Give an overview of each customer's reservation history.
- Support reporting on customer behavior and booking trends.

Related Test Script:

- Check whether the view exists in the database by querying USER_VIEWS.

- Select all rows from the view to confirm that the summary values are calculated correctly.

View: (vw_branch_table_summary)

What the View Shows?

This view presents a summary of table availability for each branch. It includes the total number of tables as well as counts of tables marked as available or occupied. It uses a left join between BRANCH and RES_TABLE to ensure all branches are included even if they have no tables assigned.

Purpose:

- Provide a quick look at table availability across branches.
- Assist in capacity management and branch-level planning.

Related Test Script:

- Verify the view exists by checking USER_VIEWS.
- Retrieve data from the view to confirm that table counts and statuses are calculated correctly.

Index: (idx_fk_customer_res)

What the Index Does?

This index is created on the customer_id column of the RESERVATION table. Since customer_id is a foreign key, this index improves the performance of queries that join or filter reservations by customer.

Purpose:

- Speed up queries that depend on the customer-reservation relationship.
- Improve performance for reservation lookups, summaries, and reporting.

Index: (idx_fk_branch_table)

What the Index Does?

This index is applied to the branch_id column in the RES_TABLE table. This helps optimize queries that retrieve tables based on their branch.

Purpose:

- Improve performance for branch-related table queries.
- Support faster execution of branch summaries and lookups.

Index: (idx_status_reservation)

What the Index Does?

This index is created on the status column in the RESERVATION table. Since reservation status is frequently used for filtering (such as pending, confirmed, or cancelled), this index improves the performance of status-based queries.

Purpose:

- Speed up reports and operations that categorize reservations by status.
- Support efficient execution of views and stored procedures that rely on status filters.

TECHNICAL CHALLENGES

- **WHILE MAKING ERD:**

  We faced challenges in accurately representing business rules and relationships. Determining one-to-many vs. many-to-many links, handling composite keys, and avoiding data redundancy required careful planning. Translating real-world rules, like table reservations and branch management, into clear database constraints was tricky. Balancing simplicity with completeness was also challenging, but iterative design ensured a clean, normalized, and practical ERD.

- **ORACLE ISSUES:**
  Oracle APEX supports only pure SQL and PL/SQL, and it does not run SQL*Plus commands. For example, commands like PROMPT are not supported, so we replaced them with DBMS_OUTPUTPUT_LINE. Therefore, we adjusted our project to work fully within the SQL/PLSQL environment that APEX provides!

CONCLUSION

This project successfully delivered a complete and efficient restaurant reservation management system supported by a well-structured and fully normalized database. Through carefully designed tables, stored logic, triggers, views, and performance-enhancing indexes, the system ensures accurate reservation handling, smooth feedback processing, and reliable branch-level management. The implemented DDL, DML, and automated test scripts helped validate every component, ensuring the system runs with consistency, integrity, and real-time accuracy. Overall, the database foundation is stable, scalable, and ready for practical use in a real restaurant chain environment.

CONTRIBUTIONS

M. DANIYAL QURESHI	• BUSINESS RULES. • ERD. • DDL & NORMALIZATION JUSTIFICATION. • TECHNICAL REPORT.
M. FARRUKH HUMAYUN	• FUNCTIONAL REQ. • NON-FUNCTIONAL REQ. • DML. • COMPLEX QURIES.
M. ABDULLAH SANI	• FUNCTIONAL DEPENDENCIES. • RELATIONAL SCHEMA. • DML. • STORED LOGIC & TEST SCRIPT.
M. UZAIR KHAN	• NORMALIZATION. • INDEXES & VIEWS. • COMPLEX QURIES. • TECHNICAL REPORT.

Appendices:

The complete, well-commented SQL scripts for this project are submitted as separate files in accordance with the submission guidelines. They are not included in the body of this report to adhere to the specified page limit.

- **Appendix 1:** (DDL & DML) Script.
- **Appendix 2:** Indexes and Views Script.
- **Appendix 3:** Complex Queries.
- **Appendix 4:** Stored Logic & Test Script.
