KHULNA UNIVERSITY OF ENGINEERING & TECHNOLOGY

KHULNA-9203

LAB REPORT

Course No : CSE 3110
Course Title : Database Management System Laboratory
Project Name : Covid Bed Slot Booking System Database Project

SUBMITTED BY

Name : Md Mushfiqur Rahman
Department : Computer Science and Engineering
Roll: 2007027
Group : A1
Year : 3rd
Semester : 1st

NAME of THE PROJECT: COVID BED SLOT BOOKING SYSTEM

Objectives:

1) To understand how to translate real-world entities (hospitals, patients, beds) and their relationships into a formal ER model using Oracle tools.

2) To learn best practices for designing an Oracle database schema, including tables, columns, data types, constraints, and keys to ensure data integrity and efficiency.

3) To gain experience with PL/SQL, a procedural language extension for Oracle, to create stored procedures, functions, and triggers for managing bed availability, bookings, and cancellations.

4) To master the use of SQL statements (INSERT, UPDATE, DELETE) to manipulate data within the Oracle database, including adding new hospitals, beds, and booking information.

5) To develop proficiency in writing complex SQL queries (SELECT) to retrieve specific data from the database, such as searching for available beds based on location, type (ICU/general), or other criteria.

6) To learn how to implement security measures in the Oracle database, such as user accounts, roles, and permissions, to restrict access to sensitive patient data.

7) To understand and implement data integrity constraints (primary keys, foreign keys, check constraints) within the database schema to ensure data accuracy and consistency.

8) To learn how to manage database transactions using COMMIT and ROLLBACK statements to ensure data consistency in case of errors during booking or updates.

9) To gain experience with creating reports using Oracle tools (SQL queries, reports builder) to analyze bed occupancy, booking trends, and identify areas for improvement.

10) To explore considerations for deploying the developed system to a production environment and understand how to scale the Oracle database to accommodate future growth and increased usage.

Introduction:

In the wake of the global COVID-19 pandemic, the healthcare infrastructure faced unprecedented challenges, with hospitals struggling to manage patient inflows efficiently, particularly concerning critical resources like bed availability. In response to this pressing need, the development of a robust "Covid Bed Slot Booking System" database emerges as a crucial endeavor. This project aims to streamline and optimize the allocation of healthcare resources by implementing an advanced digital platform.

The Covid Bed Slot Booking System database project is designed to revolutionize the way hospitals manage bed allocation for COVID-19 patients. By leveraging modern database technologies, this system offers a comprehensive solution for hospitals to effectively monitor, manage, and allocate bed slots according to patient needs and resource availability. Through the integration of real-time data updates, intuitive user interfaces, and automated processes, this system promises to enhance operational efficiency and improve patient care outcomes amidst the ongoing healthcare crisis.

Description:

This database design for a Covid Bed Slot Booking System utilizes five interrelated tables within an Oracle database:

(1) Hospitals:

* Description: Stores information about registered hospitals in the system.
* Key Features:
    * Hospital_ID (NUMBER, Primary Key): Unique identifier for each hospital.
    * Name (VARCHAR2(255)): Name of the hospital.
    * Address (VARCHAR2(500)): Address of the hospital.
    * Phone_Number (VARCHAR2(20)): Hospital phone number for contact.

(2) Patients:

* Description: Stores information about patients seeking Covid-19 treatment.
* Key Features:
    * Patient_ID (NUMBER, Primary Key): Unique identifier for each patient.
    * Name (VARCHAR2(255)): Patient's full name.
    * Age (NUMBER): Age of the patient.
    * Phone_Number (VARCHAR2(20)): Patient's phone number for contact.
    * Address (VARCHAR2(500)): Patient's address.

(3) Staff:

* Description: Stores information about hospital staff involved in the booking process.

* Key Features:

    * Staff_ID (NUMBER, Primary Key): Unique identifier for each staff member.

    * Name (VARCHAR2(255)): Staff member's full name.

    * Role (VARCHAR2(100)): Staff member's role (e.g., Doctor, Nurse, Administrator).

    * Hospital_ID (NUMBER, Foreign Key): References the Hospital_ID in the Hospitals table, linking staff to their affiliated hospital.

(4) Beds:

* Description: Stores information about available beds within registered hospitals.

* Key Features:

    * Bed_ID (NUMBER, Primary Key): Unique identifier for each bed.

    * Hospital_ID (NUMBER, Foreign Key): References the Hospital_ID in the Hospitals table, linking beds to their respective hospitals.

    * Room_Number (VARCHAR2(20)): Room number where the bed is located.

    * Type (VARCHAR2(100)): Type of bed (e.g., General, ICU, Ventilator).

    * Status (VARCHAR2(50)): Current availability status of the bed (e.g., Available, Occupied, Reserved).

(5) Bookings:

* Description: Stores booking information for patients securing beds.

* Key Features:

    * Booking_ID (NUMBER, Primary Key): Unique identifier for each booking.

* Patient_ID (NUMBER, Foreign Key): References the Patient_ID in the Patients table, linking the booking to the patient.

* Bed_ID (NUMBER, Foreign Key): References the Bed_ID in the Beds table, linking the booking to the specific bed.

* Start_Date (DATE): Date on which the bed booking starts.

* End_Date (DATE): Date on which the bed booking ends.

Relationships:

* The database enforces referential integrity through foreign keys.

* Staff.Hospital_ID references Hospitals.Hospital_ID, ensuring staff are linked to existing hospitals.

* Beds.Hospital_ID references Hospitals.Hospital_ID, ensuring beds belong to registered hospitals.

* Bookings.Patient_ID references Patients.Patient_ID, linking bookings to patients.

* Bookings.Bed_ID references Beds.Bed_ID, linking bookings to specific beds.

This database design provides a solid foundation for a Covid Bed Slot Booking System, enabling functionalities like:

* Hospital registration and management.

* Patient registration and bed search based on location, type, and availability.

* Real-time bed availability display.

* Booking confirmation and management.

* Data storage for analysis and reporting on bed usage and booking trends.

By leveraging this database structure, the system can contribute to improved access to critical care for Covid-19 patients and optimize healthcare resource allocation.

Entity-Relationship Diagram:

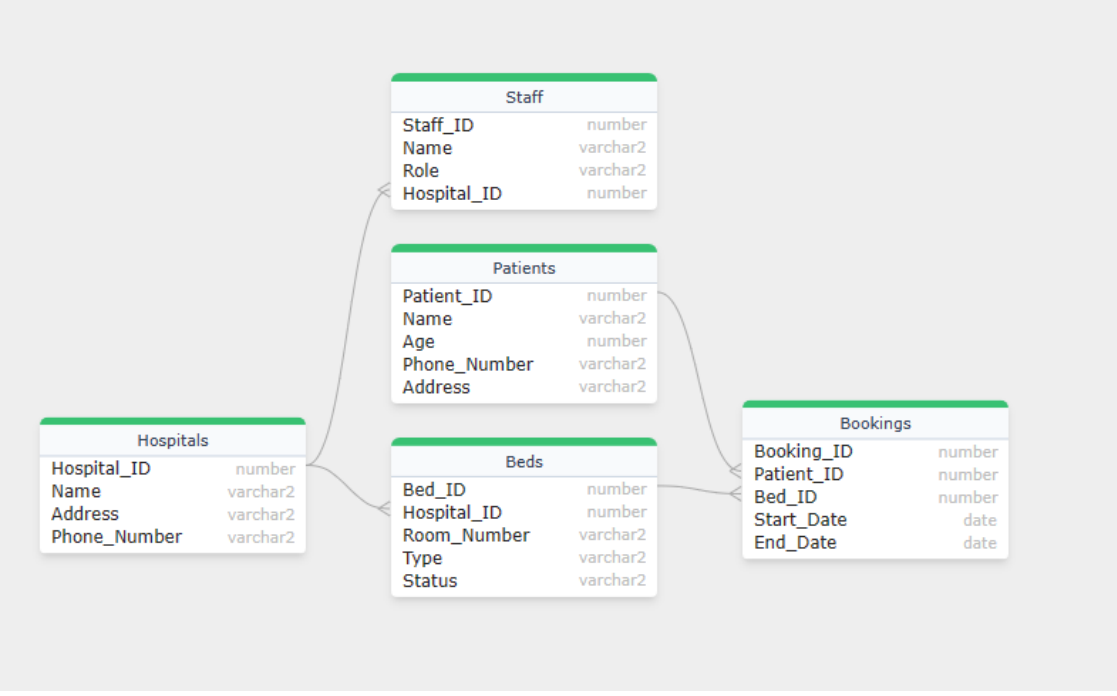

Figure : ER Diagram

SQL Operations:

(1) Creation of Data Tables:

```
CREATE TABLE Hospitals (
    Hospital_ID NUMBER PRIMARY KEY,
    Name VARCHAR2(255),
    Address VARCHAR2(500),
```

```
    Phone_Number VARCHAR2(20)
);
CREATE TABLE Patients (
    Patient_ID NUMBER PRIMARY KEY,
    Name VARCHAR2(255),
    Age NUMBER,
    Phone_Number VARCHAR2(20),
    Address VARCHAR2(500)
);

CREATE TABLE Staff (
    Staff_ID NUMBER PRIMARY KEY,
    Name VARCHAR2(255),
    Role VARCHAR2(100),
    Hospital_ID NUMBER,
    FOREIGN KEY (Hospital_ID) REFERENCES Hospitals(Hospital_ID)
);


CREATE TABLE Beds (
    Bed_ID NUMBER PRIMARY KEY,
    Hospital_ID NUMBER,
    Room_Number VARCHAR2(20),
    Type VARCHAR2(100),
    Status VARCHAR2(50),
    FOREIGN KEY (Hospital_ID) REFERENCES Hospitals(Hospital_ID)
```

```
);

CREATE TABLE Bookings (
    Booking_ID NUMBER PRIMARY KEY,
    Patient_ID NUMBER,
    Bed_ID NUMBER,
    Start_Date DATE,
    End_Date DATE,
    FOREIGN KEY (Patient_ID) REFERENCES Patients(Patient_ID),
    FOREIGN KEY (Bed_ID) REFERENCES Beds(Bed_ID)
);
```

(2) Data-Definition Language:

```
ALTER TABLE Patients ADD Email VARCHAR2(255);
ALTER TABLE Patients MODIFY Email VARCHAR2(100);
ALTER TABLE Patients RENAME COLUMN Email TO NewEmail;
ALTER TABLE Patients DROP COLUMN NewEmail;
```

(3) Insert Rows into Five Tables:

=>Insert into Hospitals Table:

```
INSERT INTO Hospitals (Hospital_ID, Name, Address, Phone_Number) VALUES (1, 'Hospital A', '123 Main St, CityA', '123-456-7890');
INSERT INTO Hospitals (Hospital_ID, Name, Address, Phone_Number) VALUES (2, 'Hospital B', '456 Oak St, CityB', '456-789-0123');
INSERT INTO Hospitals (Hospital_ID, Name, Address, Phone_Number) VALUES (3, 'Hospital C', '789 Maple St, CityC', '789-012-3456');
```

INSERT INTO Hospitals (Hospital_ID, Name, Address, Phone_Number) VALUES (4, 'Hospital D', '987 Elm St, CityD', '987-654-3210');

INSERT INTO Hospitals (Hospital_ID, Name, Address, Phone_Number) VALUES(5, 'Hospital E', '654 Pine St, CityE', '654-321-0987');

=>Insert into Patients Table:

INSERT INTO Patients (Patient_ID, Name, Age, Phone_Number, Address) VALUES (1, 'John Doe', 35, '111-222-3333', '123 Oak St, CityA');

INSERT INTO Patients (Patient_ID, Name, Age, Phone_Number, Address) VALUES (2, 'Jane Smith', 45, '222-333-4444', '456 Elm St, CityB');

INSERT INTO Patients (Patient_ID, Name, Age, Phone_Number, Address) VALUES (3, 'Michael Johnson', 50, '333-444-5555', '789 Maple St, CityC');

INSERT INTO Patients (Patient_ID, Name, Age, Phone_Number, Address) VALUES (4, 'Emily Davis', 28, '444-555-6666', '987 Pine St, CityD');

INSERT INTO Patients (Patient_ID, Name, Age, Phone_Number, Address) VALUES (5, 'David Wilson', 62, '555-666-7777', '654 Oak St, CityE');

=>Insert into Staff Table:

INSERT INTO Staff (Staff_ID, Name, Role, Hospital_ID) VALUES (1, 'Alice Johnson', 'Doctor', 1);

INSERT INTO Staff (Staff_ID, Name, Role, Hospital_ID) VALUES (2, 'Bob Smith', 'Nurse', 2);

INSERT INTO Staff (Staff_ID, Name, Role, Hospital_ID) VALUES (3, 'Charlie Brown', 'Surgeon', 3);

INSERT INTO Staff (Staff_ID, Name, Role, Hospital_ID) VALUES (4, 'Diana Lee', 'Nurse', 4);

INSERT INTO Staff (Staff_ID, Name, Role, Hospital_ID) VALUES (5, 'Eleanor Davis', 'Doctor', 5);

=>Insert into Beds Table:

```
INSERT INTO Beds (Bed_ID, Hospital_ID, Room_Number, Type, Status) VALUES (1, 1, '101', 'ICU', 'Occupied');

INSERT INTO Beds (Bed_ID, Hospital_ID, Room_Number, Type, Status) VALUES (2, 2, '201', 'General', 'Available');

INSERT INTO Beds (Bed_ID, Hospital_ID, Room_Number, Type, Status) VALUES (3, 3, '301', 'ICU', 'Occupied');

INSERT INTO Beds (Bed_ID, Hospital_ID, Room_Number, Type, Status) VALUES (4, 4, '401', 'General', 'Available');

INSERT INTO Beds (Bed_ID, Hospital_ID, Room_Number, Type, Status) VALUES (5, 5, '501', 'ICU', 'Occupied');
```

=>Insert into Bookings Table:

```
INSERT INTO Bookings (Booking_ID, Patient_ID, Bed_ID, Start_Date, End_Date)

VALUES (1, 1, 1, TO_DATE('2024-04-30', 'YYYY-MM-DD'), TO_DATE('2024-05-07', 'YYYY-MM-DD'));

INSERT INTO Bookings (Booking_ID, Patient_ID, Bed_ID, Start_Date, End_Date)

VALUES (2, 2, 2, TO_DATE('2024-04-30', 'YYYY-MM-DD'), TO_DATE('2024-05-05', 'YYYY-MM-DD'));

INSERT INTO Bookings (Booking_ID, Patient_ID, Bed_ID, Start_Date, End_Date)

VALUES (3, 3, 3, TO_DATE('2024-04-30', 'YYYY-MM-DD'), TO_DATE('2024-05-10', 'YYYY-MM-DD'));

INSERT INTO Bookings (Booking_ID, Patient_ID, Bed_ID, Start_Date, End_Date)

VALUES (4, 4, 4, TO_DATE('2024-04-30', 'YYYY-MM-DD'), TO_DATE('2024-05-03', 'YYYY-MM-DD'));
```

INSERT INTO Bookings (Booking_ID, Patient_ID, Bed_ID, Start_Date, End_Date)

VALUES (5, 5, 5, TO_DATE('2024-04-30', 'YYYY-MM-DD'), TO_DATE('2024-05-09', 'YYYY-MM-DD'));

(4) Displaying Data From Tables:

=> From Hospitals Table: SELECT * FROM Hospitals;

=> From Patients Table: SELECT * FROM Patients;

=> From Staff Table: SELECT * FROM Staff;

=> From Beds Table: SELECT * FROM Beds;

=> From Bookings Table: SELECT * FROM Bookings;

(5) Displaying Data Using Condition:

=> From Hospitals Table: SELECT * FROM Hospitals WHERE Hospital_ID = 1;

=> From Patients Table: SELECT * FROM Patients WHERE Age >= 40;

=> From Staff Table: SELECT * FROM Staff WHERE Role = 'Doctor';

=> From Beds Table: SELECT * FROM Beds WHERE Type = 'ICU' AND Status = 'Occupied';

=> From Bookings Table: SELECT * FROM Bookings WHERE Bed_ID > 2;

(6) Updating A Row:

=> For Hospitals Table:

UPDATE Hospitals

SET Name = 'Updated Hospital Name', Address = 'Updated Address', Phone_Number = '555-555-5555'

WHERE Hospital_ID = 5;

=>For Patients Table:

```
UPDATE Patients
SET Name = 'Updated Patient Name', Age = 30, Phone_Number = '555-555-5555', Address = 'Updated Address'
WHERE Patient_ID = 5;
```

=>For Staff Table:

```
UPDATE Staff
SET Name = 'Updated Staff Name', Role = 'Updated Role', Hospital_ID = 1
WHERE Staff_ID = 5;
```

=>For Beds Table:

```
UPDATE Beds
SET Hospital_ID = 2, Room_Number = '502', Type = 'General', Status = 'Available'
WHERE Bed_ID = 5;
```

=>For Bookings Table:

```
UPDATE Bookings
SET Patient_ID = 4, Bed_ID = 4,
    Start_Date = TO_DATE('2024-05-01', 'YYYY-MM-DD'),
    End_Date = TO_DATE('2024-05-08', 'YYYY-MM-DD')
WHERE Booking_ID = 5;
```

(7) Deleting A Row:

=> From Hospitals Table: `DELETE FROM Hospitals WHERE Hospital_ID = 5;`

=> From Patients Table: `DELETE FROM Patients WHERE Patient_ID = 5;`

=> From Staff Table: `DELETE FROM Staff WHERE Staff_ID = 5;`

=> From Beds Table: DELETE FROM Beds WHERE Bed_ID = 5;

=> From Bookings Table: DELETE FROM Bookings WHERE Booking_ID = 5;

(8) Union Operation:

```
SELECT Hospital_ID, Name, Address, Phone_Number FROM Hospitals
UNION
SELECT Hospital_ID, Name, NULL AS Address, NULL AS Phone_Number FROM Staff;
```

(9) Intersect Operation:

```
SELECT Hospital_ID, Name, Address, Phone_Number FROM Hospitals
 INTERSECT
 SELECT Hospital_ID, Name, NULL AS Address, NULL AS Phone_Number FROM Staff;
```

(10) Minus Operation:

```
SELECT P.Patient_ID, P.Name
FROM Patients P
MINUS
SELECT P.Patient_ID, P.Name
FROM Bookings B
JOIN Patients P ON B.Patient_ID = P.Patient_ID;
```

(11) Use of Aggregate Function:

=> min aggregate function : SELECT MIN(Bed_ID) AS Min_Bed_ID FROM Beds;

=> max aggregate function : SELECT MAX(Bed_ID) AS Max_Bed_ID FROM Beds;

=> count aggregate function : SELECT COUNT(*) AS Total_Beds FROM Beds;

=> sum aggregate function : SELECT SUM(Bed_ID) AS Total_Bed_IDs FROM Beds;

=> avg aggregate function : SELECT AVG(Bed_ID) AS Avg_Bed_ID FROM Beds;

(12) Use of With Clause:

```
WITH BookingDetails AS (
   SELECT Booking_ID, Patient_ID, Bed_ID, Start_Date, End_Date, End_Date - Start_Date AS Duration
   FROM Bookings
)
SELECT * FROM BookingDetails;
```

(13) Use of Having Clause:

```
SELECT Role, COUNT(*) AS Role_Count
FROM Staff
GROUP BY Role
HAVING COUNT(*) > 1;
```

(14) Use of Group By Clause:

```
SELECT Hospital_ID, COUNT(*) AS Total_Beds
FROM Beds
GROUP BY Hospital_ID;
```

(15) Use of Nested Subquery:

```
SELECT *
FROM Beds
```

```
WHERE Bed_ID IN (
    SELECT DISTINCT Bed_ID
    FROM Bookings
    WHERE Patient_ID IN (
        SELECT Patient_ID
        FROM Patients
        WHERE Age > 40
    )
);
```

(16) Use of AND Operator:

```
SELECT *
FROM Patients
WHERE Age >= 30 AND Age <= 50;
```

(17) Use of OR Operator:

```
SELECT *
FROM Patients
WHERE Age < 30 OR Age >= 40;
```

(18) Use of NOT Operator:

```
SELECT *
FROM Patients
WHERE NOT (Age < 30 OR Age >= 40);
```

(19) Use of ‘Exists’ Keyword:

```
SELECT *
FROM Patients p
WHERE EXISTS (
    SELECT 1
    FROM Patients
    WHERE Age >= 30
);
```

(20) Use of ‘Some’ Keyword:

```
SELECT *
FROM Patients
WHERE Age > SOME (
    SELECT Age
    FROM Patients
    WHERE Age >= 30
);
```

(21) Use of ‘In’ Keyword:

```
SELECT Name
FROM Staff
WHERE Hospital_ID IN
    (SELECT Hospital_ID
     FROM Hospitals);
```

(22) Use of ‘Unique’ Keyword:

```
SELECT UNIQUE Bed_ID
FROM Bookings;
```

(23) Use of ‘LIKE’ Keyword:

```
SELECT *
FROM Patients
WHERE Name LIKE 'J%';
```

(24) Perform Natural Join Operation:

```
SELECT *
FROM Beds
NATURAL JOIN Hospitals;
```

(25) Perform Left Outer Join Operation:

```
SELECT *
FROM Beds
LEFT OUTER JOIN Hospitals ON Beds.Hospital_ID = Hospitals.Hospital_ID;
```

(26) Perform Right Outer Join Operation:

```
SELECT *
FROM Beds
RIGHT OUTER JOIN Hospitals ON Beds.Hospital_ID = Hospitals.Hospital_ID;
```

(27) Perform Full Outer Join Operation:

```
SELECT *
FROM Beds
FULL OUTER JOIN Hospitals ON Beds.Hospital_ID = Hospitals.Hospital_ID;
```

(28) Perform Inner Join Operation:

```
SELECT *
FROM Beds
INNER JOIN Hospitals ON Beds.Hospital_ID = Hospitals.Hospital_ID;
```

(29) Creation of View:

```
CREATE VIEW AdultPatients AS
SELECT *
FROM Patients
WHERE Age > 18;

SELECT * FROM AdultPatients;
```

(30) Creation of View Using Other View:

```
CREATE VIEW SeniorPatients AS
SELECT *
FROM AdultPatients
WHERE Age >= 40;

SELECT * FROM SeniorPatients;
```

(31) PL/SQL Variable Declaration & Print Value:

```
SET SERVEROUTPUT ON;
DECLARE
    v_message VARCHAR2(100);
    v_patient_name VARCHAR2(100);
BEGIN
    -- Assigning value to the variable
    v_message := 'Patient Name: ';

    -- Retrieving patient name from the table
    SELECT Name INTO v_patient_name
    FROM Patients
    WHERE Patient_ID = 1; -- Specify the patient ID you want to retrieve

    -- Printing the message along with the patient name
    DBMS_OUTPUT.PUT_LINE(v_message || v_patient_name);
END;
/
```

(32) Insert & Set Default Value to PL/SQL Variable:

```
SET SERVEROUTPUT ON;
DECLARE
    v_patient_id Patients.Patient_ID%TYPE := 100; -- Choose a unique Patient_ID
    v_patient_name Patients.Name%TYPE := 'John Doe';
    v_patient_age Patients.Age%TYPE := 30;
    v_patient_phone Patients.Phone_Number%TYPE := 'N/A';
```

```
    v_patient_address Patients.Address%TYPE := 'N/A';
BEGIN
    INSERT INTO Patients (Patient_ID, Name, Age, Phone_Number, Address)
    VALUES (v_patient_id, v_patient_name, v_patient_age, v_patient_phone,
v_patient_address);


    DBMS_OUTPUT.PUT_LINE('Inserted Patient ID: ' || v_patient_id);
    DBMS_OUTPUT.PUT_LINE('Inserted Patient Name: ' || v_patient_name);
    DBMS_OUTPUT.PUT_LINE('Inserted Patient Age: ' || v_patient_age);
    DBMS_OUTPUT.PUT_LINE('Inserted Patient Phone Number: ' || v_patient_phone);
    DBMS_OUTPUT.PUT_LINE('Inserted Patient Address: ' || v_patient_address);
END;
/
```

(33) Use of Row Type:

```
SET SERVEROUTPUT ON;
DECLARE
    -- Declare a row type based on the structure of the Patients table
    TYPE patient_row_type IS RECORD (
        Patient_ID Patients.Patient_ID%TYPE,
        Name Patients.Name%TYPE,
        Age Patients.Age%TYPE,
        Phone_Number Patients.Phone_Number%TYPE,
        Address Patients.Address%TYPE
    );
```

```
    -- Declare a variable of the row type
    v_patient patient_row_type;
BEGIN
    -- Assign values to the row type variable from the Patients table
    SELECT *
    INTO v_patient
    FROM Patients
    WHERE ROWNUM = 1; -- Fetching the first row for demonstration

    -- Printing the values of the row type variable
    DBMS_OUTPUT.PUT_LINE('Patient ID: ' || v_patient.Patient_ID);
    DBMS_OUTPUT.PUT_LINE('Name: ' || v_patient.Name);
    DBMS_OUTPUT.PUT_LINE('Age: ' || v_patient.Age);
    DBMS_OUTPUT.PUT_LINE('Phone Number: ' || v_patient.Phone_Number);
    DBMS_OUTPUT.PUT_LINE('Address: ' || v_patient.Address);
END;
/
```

(34) Use of Cursor & Row Count:

```
SET SERVEROUTPUT ON;
DECLARE
    -- Declare a cursor to fetch data from the Patients table
    CURSOR patient_cursor IS
        SELECT *
        FROM Patients;
```

```
    -- Declare a variable to store the count of rows fetched
    v_row_count NUMBER := 0;

    -- Declare variables to hold column values
    v_patient_id Patients.Patient_ID%TYPE;
    v_patient_name Patients.Name%TYPE;
    v_patient_age Patients.Age%TYPE;
    v_patient_phone Patients.Phone_Number%TYPE;
    v_patient_address Patients.Address%TYPE;
BEGIN
    -- Open the cursor
    OPEN patient_cursor;

    -- Loop through the cursor and fetch each row
    LOOP
        -- Fetch row into variables
        FETCH patient_cursor INTO v_patient_id, v_patient_name, v_patient_age, v_patient_phone, v_patient_address;

        -- Exit the loop if no more rows to fetch
        EXIT WHEN patient_cursor%NOTFOUND;

        -- Increment row count
        v_row_count := v_row_count + 1;

        -- Printing the values of the fetched row
```

```
        DBMS_OUTPUT.PUT_LINE('Patient ID: ' || v_patient_id);
        DBMS_OUTPUT.PUT_LINE('Name: ' || v_patient_name);
        DBMS_OUTPUT.PUT_LINE('Age: ' || v_patient_age);
        DBMS_OUTPUT.PUT_LINE('Phone Number: ' || v_patient_phone);
        DBMS_OUTPUT.PUT_LINE('Address: ' || v_patient_address);
        DBMS_OUTPUT.PUT_LINE('--------------------------------');
    END LOOP;

    -- Close the cursor
    CLOSE patient_cursor;

    -- Printing the total number of rows fetched
    DBMS_OUTPUT.PUT_LINE('Total number of rows fetched: ' || v_row_count);
END;
/
```

(35) For Loop & While Loop With Extend() Function:

```
DECLARE
      -- Define a collection type for storing patient data
       TYPE patient_array IS TABLE OF Patients%ROWTYPE;

        -- Declare an array variable
        v_patients patient_array := patient_array();

        -- Counter variable for loops
        v_counter NUMBER := 0;
```

```
    -- Variable to hold the total count of patients
    v_total_patients NUMBER;

    -- Cursor variable
    CURSOR patients_cursor IS
        SELECT * FROM Patients;
BEGIN
    -- Fetch total count of patients
    SELECT COUNT(*) INTO v_total_patients FROM Patients;

    -- Open cursor
    OPEN patients_cursor;

    -- Fetch data from Patients table into the array using a cursor and loop
    LOOP
        -- Extend the array size
        v_patients.EXTEND;

        -- Fetch data from cursor into record
        FETCH patients_cursor INTO v_patients(v_counter+1);

        -- Exit loop if no more rows
        EXIT WHEN patients_cursor%NOTFOUND;

        -- Increment counter
```

```
        v_counter := v_counter + 1;
    END LOOP;

    -- Close cursor
    CLOSE patients_cursor;

    -- Print the fetched data using a loop
    DBMS_OUTPUT.PUT_LINE('Fetched data using cursor and loop:');
    FOR i IN 1..v_patients.COUNT LOOP
        DBMS_OUTPUT.PUT_LINE('Patient ID: ' || v_patients(i).Patient_ID);
        -- Print other patient details similarly
    END LOOP;
END;
/
```

(36) For Loop & While Loop Without Extend() Function:

```
DECLARE
    -- Define a collection type for storing patient data
    TYPE patient_array IS TABLE OF Patients%ROWTYPE INDEX BY PLS_INTEGER;

    -- Declare an array variable
    v_patients patient_array;

    -- Counter variable for loops
    v_counter NUMBER := 0;
```

```
    -- Cursor variable
    patient_cursor SYS_REFCURSOR;
BEGIN
    -- Fetch data from Patients table into the array using a FOR loop without EXTEND function
    FOR patient_rec IN (SELECT * FROM Patients) LOOP
        v_counter := v_counter + 1;
        v_patients(v_counter) := patient_rec; -- Assign fetched row to array element
    END LOOP;

    -- Print the fetched data using a FOR loop
    DBMS_OUTPUT.PUT_LINE('Fetched data using FOR loop without EXTEND function:');
    FOR i IN 1..v_patients.COUNT LOOP
        DBMS_OUTPUT.PUT_LINE('Patient ID: ' || v_patients(i).Patient_ID);
        -- Print other patient details similarly
    END LOOP;

    -- Reset counter for WHILE loop
    v_counter := 0;

    -- Fetch data from Patients table into the array using a WHILE loop without EXTEND function
    OPEN patient_cursor FOR SELECT * FROM Patients;
    LOOP
        FETCH patient_cursor INTO v_patients(v_counter+1);
        EXIT WHEN patient_cursor%NOTFOUND;
```

```
        v_counter := v_counter + 1;
    END LOOP;
    CLOSE patient_cursor;


    -- Print the fetched data using a WHILE loop
    DBMS_OUTPUT.PUT_LINE('Fetched data using WHILE loop without EXTEND function:');
    v_counter := 1;
    WHILE v_counter <= v_patients.COUNT LOOP
        DBMS_OUTPUT.PUT_LINE('Patient ID: ' || v_patients(v_counter).Patient_ID);
        -- Print other patient details similarly
        v_counter := v_counter + 1;
    END LOOP;
END;
/
```

(37) Use of IF/ ELSEIF/ ELSE:

```
SET SERVEROUTPUT ON;
DECLARE
    v_patient_age Patients.Age%TYPE;
BEGIN
    -- Assume we are fetching the age of a patient from the Patients table
    SELECT Age INTO v_patient_age FROM Patients WHERE Patient_ID = 1; -- Fetching age of patient with ID = 1


    -- Check the age and classify the patient into different groups
```

```
    IF v_patient_age < 18 THEN
        DBMS_OUTPUT.PUT_LINE('Patient is a child.');
    ELSIF v_patient_age >= 18 AND v_patient_age < 60 THEN
        DBMS_OUTPUT.PUT_LINE('Patient is an adult.');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Patient is a senior citizen.');
    END IF;
END;
/
```

(38) Creation of Procedure:

```
CREATE OR REPLACE PROCEDURE GetPatientDetails(
    p_patient_id IN Patients.Patient_ID%TYPE
)
IS
    v_patient_name Patients.Name%TYPE;
    v_patient_age Patients.Age%TYPE;
    v_patient_phone Patients.Phone_Number%TYPE;
    v_patient_address Patients.Address%TYPE;
BEGIN
    -- Fetch patient details based on the provided patient ID
    SELECT Name, Age, Phone_Number, Address
    INTO v_patient_name, v_patient_age, v_patient_phone, v_patient_address
    FROM Patients
    WHERE Patient_ID = p_patient_id;
```

```
    -- Print the fetched patient details
    DBMS_OUTPUT.PUT_LINE('Patient ID: ' || p_patient_id);
    DBMS_OUTPUT.PUT_LINE('Name: ' || v_patient_name);
    DBMS_OUTPUT.PUT_LINE('Age: ' || v_patient_age);
    DBMS_OUTPUT.PUT_LINE('Phone Number: ' || v_patient_phone);
    DBMS_OUTPUT.PUT_LINE('Address: ' || v_patient_address);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Patient with ID ' || p_patient_id || ' not found.');
END;
/



SET SERVEROUTPUT ON;
BEGIN
    GetPatientDetails(1); -- Pass the patient ID as an argument
END;
/
```

(39) Creation of Function:

```
CREATE OR REPLACE FUNCTION GetPatientAge(
    p_patient_id IN Patients.Patient_ID%TYPE
) RETURN NUMBER
IS
    v_patient_age Patients.Age%TYPE;
BEGIN
```

```
    -- Fetch the age of the patient based on the provided patient ID
    SELECT Age INTO v_patient_age
    FROM Patients
    WHERE Patient_ID = p_patient_id;

    -- Return the age of the patient
    RETURN v_patient_age;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Patient with ID ' || p_patient_id || ' not found.');
        RETURN NULL;
END;
/


SET SERVEROUTPUT ON;
DECLARE
    v_age NUMBER;
BEGIN
    v_age := GetPatientAge(1); -- Pass the patient ID as an argument
    IF v_age IS NOT NULL THEN
        DBMS_OUTPUT.PUT_LINE('Patient Age: ' || v_age);
    END IF;
END;
/
```

(40) Creation of Trigger:

```
CREATE OR REPLACE TRIGGER Beds_Before_Insert
BEFORE INSERT ON Beds
FOR EACH ROW
BEGIN
    -- Update the Status column to "Available" before insertion
    :NEW.Status := 'Available';
END;
/

-- Step 1: Insert a new row into the Beds table
INSERT INTO Beds (Bed_ID, Hospital_ID, Room_Number, Type)
VALUES (5, 1, '501', 'ICU');

-- Step 2: Verify the trigger execution
-- Check the status of the inserted row
SELECT * FROM Beds WHERE Bed_ID = 5;

-- Output: The Status column of the inserted row should be "Available" due to the trigger execution
```

Real World Applications of This Project:

The Covid Bed Slot Booking System offers significant benefits in a real-world scenario by addressing critical challenges during a pandemic:

Improved Access to Critical Care:

- Streamlined Booking Process: Patients or their representatives can search for available beds based on specific criteria (location, type) and make confirmed bookings, eliminating the need for physically visiting hospitals and navigating complex procedures.
- Reduced Wait Times: Real-time availability information allows patients to identify available beds promptly, minimizing wait times and ensuring faster access to essential medical care.
- Centralized Platform: The system acts as a central hub for bed booking, reducing confusion and frustration for patients seeking Covid-19 treatment.

Optimized Healthcare Resource Allocation:

- Efficient Bed Management: Hospitals can update bed availability in real-time, enabling healthcare authorities to effectively manage resources and direct patients to facilities with available beds.
- Data-Driven Decision Making: The system facilitates data collection on bed occupancy and booking trends. This data can be analyzed to identify areas with high demand and optimize resource allocation across hospitals.
- Reduced Pressure on Healthcare Systems: By streamlining bed booking and ensuring efficient utilization, the system can alleviate pressure on overwhelmed healthcare institutions during peak pandemic times.

Enhanced Transparency and Communication:

- Accurate Information: Real-time bed availability data promotes transparency for both patients and healthcare providers. Patients can make informed decisions based on accurate information, and hospitals can effectively manage patient expectations.
- Improved Communication: The system can be integrated with communication channels to notify patients of available beds or booking confirmations, fostering better communication and reducing uncertainty.
- Public Awareness: Public health authorities can leverage the system's data to provide the public with insights into bed availability in different regions, promoting awareness and preparedness.

Beyond the Pandemic:

While designed specifically for the Covid-19 crisis, this system's core functionalities can be adapted for broader applications in the healthcare sector. It can serve as a foundation for managing bed bookings for various critical illnesses or even regular hospital admissions, promoting efficiency and improved patient care throughout the healthcare system.

Learning Outcomes of This Project:

This project provided a valuable opportunity to gain practical experience in several key areas of database design and development:

(1) Database Modeling: The project solidified understanding of database models, specifically the Entity-Relationship (ER) model, allowing for the effective translation of real-world entities (hospitals, beds, patients) and their relationships into a structured database schema.

(2) SQL Querying: Through extensive practice, the project honed proficiency in writing essential SQL queries (SELECT, INSERT, UPDATE, DELETE) for data manipulation within the Oracle database. This included retrieving specific bed information, adding new hospital entries, and managing bookings.

(3) Data Management: The project fostered an understanding of data management principles, including the importance of data integrity constraints (primary keys, foreign keys) and effective data manipulation techniques to ensure data accuracy and consistency within the database.

(4) Real-World Application: By focusing on a critical healthcare issue (Covid-19 bed allocation), the project provided a practical context for applying database skills. It demonstrated how efficient database management can optimize resource allocation and improve access to essential medical care.

(5) Problem-Solving: The project encouraged problem-solving skills through the design and implementation of queries to address specific needs, such as searching for available beds based on location and type.

Discussion:

The completion of this project was followed ten highly engaging and interactive class sessions. Our instructors had dedicated themselves to thoroughly explaining database models, entity-relationship diagrams, and fundamental query operations. Notably, from the outset, SQL operations were demonstrated practically through SQL*PLUS, solidifying our understanding of database management. Following extensive practice, this project was constructed. The project primarily showcased various query operations that could be utilized to create an efficient database program for effectively allocating beds in hospitals, whether for Covid patients or others. Consequently, this system had gained the potential to significantly reduce the likelihood of bed mismanagement issues faced by hospital authorities. Overall, this project proved to be a success.

Conclusion:

In conclusion, the "Covid Bed Slot Booking System" database project represents a critical tool in managing healthcare resources during the COVID-19 pandemic. By efficiently organizing bed availability and reservations, it ensures timely access to medical care for patients in need. The implementation of such a system not only optimizes resource allocation but also enhances transparency and accountability in healthcare facilities. Through robust database management, it enables healthcare administrators to track bed utilization, allocate resources effectively, and make data-driven decisions to address emerging needs. Furthermore, it fosters collaboration among healthcare providers, facilitating the sharing of vital information for better patient care. As the pandemic evolves, the importance of agile and adaptable systems like this becomes increasingly evident. Moving forward, continued refinement and integration of technological solutions will be essential in bolstering healthcare resilience and responsiveness in the face of future challenges.

References:

Book References:

Silberschatz, A., Korth, H. F., & Sudarshan, S. (2023). Database System Concepts (7th ed.).

URL References:

https://www.javatpoint.com/dbms-sql-operator/

https://www.w3schools.com/sql/sql_operators.asp/

https://www.programiz.com/sql/operators/

https://github.com/arkprocoder/DBMS-COVID-BED-SLOT-BOOKING-MINI-PROJECT/

https://www.scribd.com/document/656265504/Covid-bed-slot-management-system/

https://www.cronj.com/blog/hospital-bed-management-system/

https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/About-SQL-Operators.html#GUID-6A0C265F-3A7E-4E1C-8F79-8C6BCA26CFBA/

---------- o ----------