Hospital Management System

Database Project Report

Database Management Systems

Submitted by:	**Roll Numbers:**
Muhammad Faraz Ansari	29201
Ashhal Aamir	29114

IBA,Karachi

December 8, 2025

Contents

1 Introduction

1.1 Project Overview

This report documents the design and implementation of a Hospital Management System database using Oracle PL/SQL. The system manages hospital operations including patient registration, doctor appointments, billing, prescriptions, and department management. The database is designed following 3rd Normal Form (3NF) principles to ensure data integrity and minimize redundancy.

1.2 Objectives

- Design a normalized database schema for hospital management
- Implement business rules using constraints and triggers
- Create stored procedures and functions for common operations
- Demonstrate PL/SQL programming capabilities
- Ensure data integrity and security

1.3 Scope

The system supports three types of users: Administrators, Doctors, and Patients. It handles appointment scheduling, prescription management, billing, and department operations with comprehensive data validation and business logic enforcement.

2 Database Schema Design

2.1 Entity Relationship Diagram

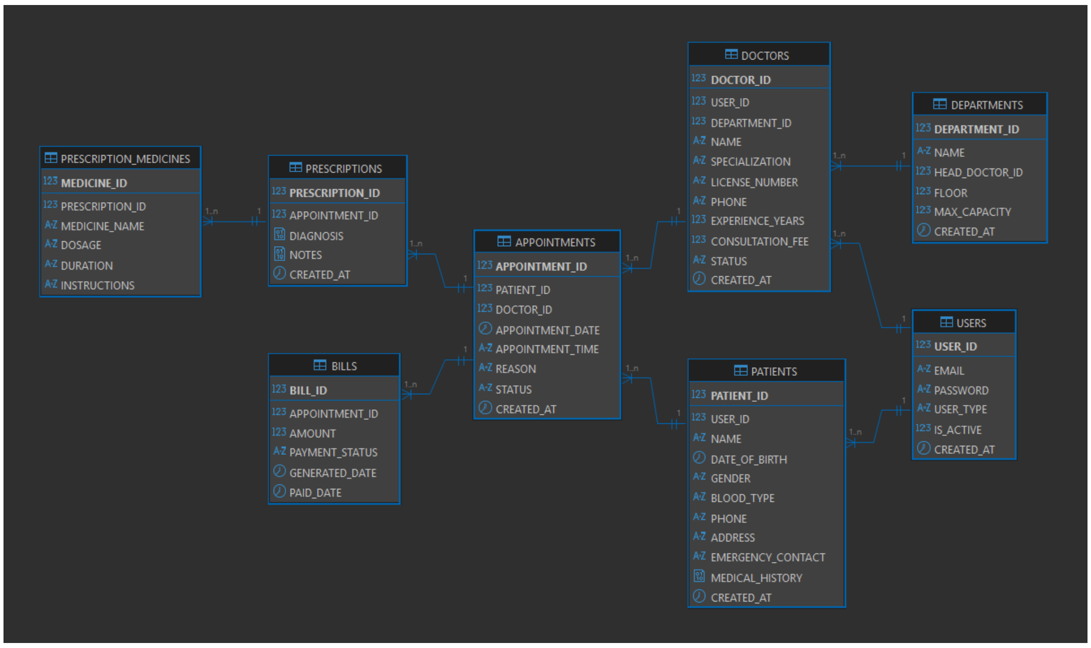


Figure 1: Entity Relationship Diagram of Hospital Management System

2.2 Normalization

The database is designed in 3rd Normal Form (3NF):

- **1NF**: All tables have atomic values, no repeating groups
- **2NF**: All non-key attributes fully dependent on primary keys
- **3NF**: No transitive dependencies between non-key attributes

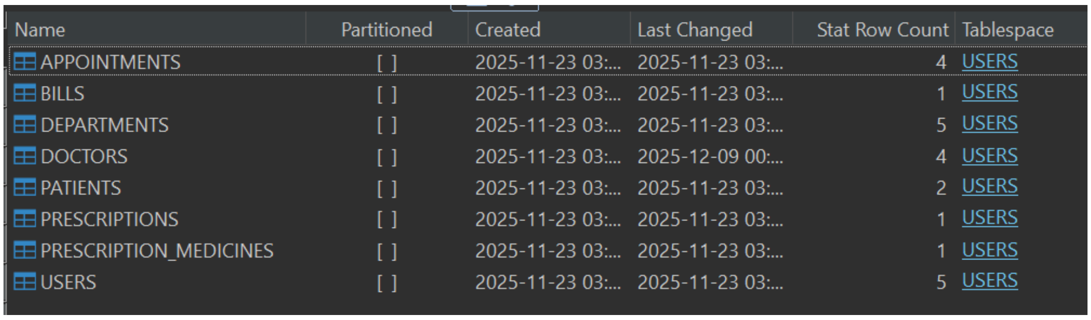
3 Table Definitions

3.1 CREATE TABLE Statements

```sql
-- ============================================================
-- Hospital Management System Database Schema
-- 3NF Compliant with proper relationships
-- ============================================================

-- 1. Users Table (Unified authentication)
CREATE TABLE users (
    user_id NUMBER PRIMARY KEY,
    email VARCHAR2(100) UNIQUE NOT NULL,
    password VARCHAR2(255) NOT NULL,
    user_type VARCHAR2(20) NOT NULL CHECK (user_type IN ('admin', 'doctor', 'patient')),
    is_active NUMBER(1) DEFAULT 1,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- 2. Departments Table
CREATE TABLE departments (
    department_id NUMBER PRIMARY KEY,
    name VARCHAR2(100) NOT NULL,
    head_doctor_id NUMBER,
    floor NUMBER,
    max_capacity NUMBER DEFAULT 10,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- 3. Doctors Table
CREATE TABLE doctors (
    doctor_id NUMBER PRIMARY KEY,
    user_id NUMBER NOT NULL,
    department_id NUMBER NOT NULL,
    name VARCHAR2(100) NOT NULL,
    specialization VARCHAR2(100) NOT NULL,
    license_number VARCHAR2(50) UNIQUE NOT NULL,
    phone VARCHAR2(20),
    experience_years NUMBER,
    consultation_fee NUMBER(10,2) NOT NULL,
    status VARCHAR2(20) DEFAULT 'Active' CHECK (status IN ('Active', 'On Leave')),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(user_id),
    FOREIGN KEY (department_id) REFERENCES departments(department_id)
);

-- 4. Patients Table
CREATE TABLE patients (
    patient_id NUMBER PRIMARY KEY,
    user_id NUMBER NOT NULL,
    name VARCHAR2(100) NOT NULL,
    date_of_birth DATE NOT NULL,
    gender VARCHAR2(10) CHECK (gender IN ('Male', 'Female', 'Other')),
    blood_type VARCHAR2(5),
    phone VARCHAR2(20),
    address VARCHAR2(255),
    emergency_contact VARCHAR2(100),
    medical_history CLOB,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(user_id)
);

-- 5. Appointments Table
CREATE TABLE appointments (
    appointment_id NUMBER PRIMARY KEY,
    patient_id NUMBER NOT NULL,
    doctor_id NUMBER NOT NULL,
    appointment_date DATE NOT NULL,
    appointment_time VARCHAR2(10) NOT NULL,
    reason VARCHAR2(500),
    status VARCHAR2(20) DEFAULT 'Scheduled' CHECK (status IN ('
   Scheduled', 'Completed', 'Cancelled', 'No Show')),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (patient_id) REFERENCES patients(patient_id),
    FOREIGN KEY (doctor_id) REFERENCES doctors(doctor_id)
);

-- 6. Bills Table
CREATE TABLE bills (
    bill_id NUMBER PRIMARY KEY,
    appointment_id NUMBER NOT NULL,
    amount NUMBER(10,2) NOT NULL,
    payment_status VARCHAR2(20) DEFAULT 'Unpaid' CHECK (payment_status
   IN ('Paid', 'Unpaid')),
    generated_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    paid_date TIMESTAMP,
    FOREIGN KEY (appointment_id) REFERENCES appointments(appointment_id
   )
);

-- 7. Prescriptions Table
CREATE TABLE prescriptions (
    prescription_id NUMBER PRIMARY KEY,
    appointment_id NUMBER NOT NULL,
    diagnosis CLOB,
    notes CLOB,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (appointment_id) REFERENCES appointments(appointment_id
   )
);

-- 8. Prescription\_Medicines Table
CREATE TABLE prescription_medicines (
    medicine_id NUMBER PRIMARY KEY,
    prescription_id NUMBER NOT NULL,
    medicine_name VARCHAR2(100) NOT NULL,
    dosage VARCHAR2(100) NOT NULL,
    duration VARCHAR2(100) NOT NULL,
    instructions VARCHAR2(500),
    FOREIGN KEY (prescription_id) REFERENCES prescriptions(
   prescription_id) ON DELETE CASCADE
);
```

Listing 1: Complete CREATE TABLE statements

3.2 Table Structure Screenshot



Name	Partitioned	Created	Last Changed	Stat Row Count	Tablespace
APPOINTMENTS	[ ]	2025-11-23 03:...	2025-11-23 03:...	4	USERS
BILLS	[ ]	2025-11-23 03:...	2025-11-23 03:...	1	USERS
DEPARTMENTS	[ ]	2025-11-23 03:...	2025-11-23 03:...	5	USERS
DOCTORS	[ ]	2025-11-23 03:...	2025-12-09 00:...	4	USERS
PATIENTS	[ ]	2025-11-23 03:...	2025-11-23 03:...	2	USERS
PRESCRIPTIONS	[ ]	2025-11-23 03:...	2025-11-23 03:...	1	USERS
PRESCRIPTION_MEDICINES	[ ]	2025-11-23 03:...	2025-11-23 03:...	1	USERS
USERS	[ ]	2025-11-23 03:...	2025-11-23 03:...	5	USERS

Figure 2: Table structure as shown in Oracle SQL Developer

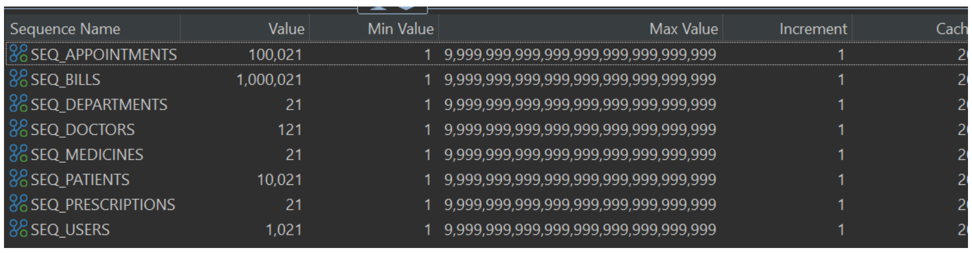
4 Sequences

4.1 Sequence Creation

```sql
--Sequences for auto-incrementing primary keys
CREATE SEQUENCE seq_users START WITH 1001 INCREMENT BY 1;
CREATE SEQUENCE seq_departments START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE seq_doctors START WITH 101 INCREMENT BY 1;
CREATE SEQUENCE seq_patients START WITH 10001 INCREMENT BY 1;
CREATE SEQUENCE seq_appointments START WITH 100001 INCREMENT BY 1;
CREATE SEQUENCE seq_bills START WITH 1000001 INCREMENT BY 1;
CREATE SEQUENCE seq_prescriptions START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE seq_medicines START WITH 1 INCREMENT BY 1;
```

Listing 2: Sequence creation statements

4.2 Sequence Screenshot



Sequence Name	Value	Min Value	Max Value	Increment	Cach
SEQ_APPOINTMENTS	100,021	1	9,999,999,999,999,999,999,999,999,999	1	2
SEQ_BILLS	1,000,021	1	9,999,999,999,999,999,999,999,999,999	1	2
SEQ_DEPARTMENTS	21	1	9,999,999,999,999,999,999,999,999,999	1	2
SEQ_DOCTORS	121	1	9,999,999,999,999,999,999,999,999,999	1	2
SEQ_MEDICINES	21	1	9,999,999,999,999,999,999,999,999,999	1	2
SEQ_PATIENTS	10,021	1	9,999,999,999,999,999,999,999,999,999	1	2
SEQ_PRESCRIPTIONS	21	1	9,999,999,999,999,999,999,999,999,999	1	2
SEQ_USERS	1,021	1	9,999,999,999,999,999,999,999,999,999	1	2

Figure 3: Sequences in Oracle Database

5 Triggers

5.1 Business Rule Implementation via Triggers

5.1.1 1. Prevent Double Booking for Doctors

```sql
-- 1. Trigger: Prevent double booking for doctors
CREATE OR REPLACE TRIGGER trg_prevent_double_booking
BEFORE INSERT ON appointments
FOR EACH ROW
DECLARE
    v_count NUMBER;
BEGIN
    SELECT COUNT(*) INTO v_count
    FROM appointments
    WHERE doctor_id = :NEW.doctor_id
    AND appointment_date = :NEW.appointment_date
    AND appointment_time = :NEW.appointment_time
    AND status != 'Cancelled';

    IF v_count > 0 THEN
        RAISE_APPLICATION_ERROR(-20001,
            'Doctor already has an appointment at this time');
    END IF;
END;
/
```

Listing 3: Trigger to prevent doctor double booking

5.1.2 2. Auto-generate Bill on Appointment Completion

```sql
-- 2. Trigger: Auto-generate bill when appointment is completed
CREATE OR REPLACE TRIGGER trg_auto_generate_bill
AFTER UPDATE ON appointments
FOR EACH ROW
WHEN (OLD.status != 'Completed' AND NEW.status = 'Completed')
DECLARE
    v_consultation_fee NUMBER;
BEGIN
    SELECT consultation_fee INTO v_consultation_fee
    FROM doctors
    WHERE doctor_id = :NEW.doctor_id;

    INSERT INTO bills (bill_id, appointment_id, amount, payment_status)
    VALUES (seq_bills.NEXTVAL, :NEW.appointment_id,
            v_consultation_fee, 'Unpaid');

    DBMS_OUTPUT.PUT_LINE('Bill generated for appointment ' ||
                         :NEW.appointment_id);
END;
/
```

Listing 4: Trigger for automatic bill generation

5.1.3 3. Combined Business Rules for Appointment Booking

```
-- 3. Trigger: Validate appointment booking with multiple business
   rules
CREATE OR REPLACE TRIGGER validate_appointment_booking
BEFORE INSERT ON appointments
FOR EACH ROW
DECLARE
    scheduled_count NUMBER;
    max_appointments NUMBER := 2;
    max_booking_date DATE;
BEGIN
    -- Rule 1: Check booking date is within 7 days
    max_booking_date := TRUNC(SYSDATE) + 7;

    IF :NEW.appointment_date > max_booking_date THEN
        RAISE_APPLICATION_ERROR(-20002,
            'Appointments can only be booked within the next 7 days.');
    END IF;

    -- Rule 2: Check maximum scheduled appointments per patient
    SELECT COUNT(*)
    INTO scheduled_count
    FROM appointments
    WHERE patient_id = :NEW.patient_id
    AND status = 'Scheduled'
    AND appointment_id != NVL(:NEW.appointment_id, -1);

    IF scheduled_count >= max_appointments THEN
        RAISE_APPLICATION_ERROR(-20003,
            'Maximum of ' || max_appointments ||
            ' scheduled appointments allowed per patient.');
    END IF;
END;
/
```

Listing 5: Trigger implementing multiple business rules

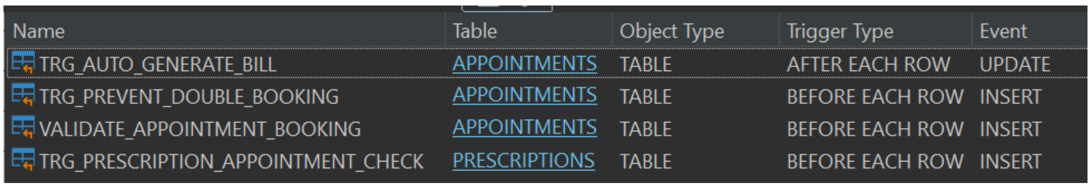
5.1.4 4. Prescription Validation Trigger

```
-- 4. Trigger: Prevent prescription for incomplete appointments
CREATE OR REPLACE TRIGGER trg_prescription_appointment_check
BEFORE INSERT ON prescriptions
FOR EACH ROW
DECLARE
    v_appointment_status VARCHAR2(20);
BEGIN
    SELECT status INTO v_appointment_status
    FROM appointments
    WHERE appointment_id = :NEW.appointment_id;

    IF v_appointment_status != 'Completed' THEN
        RAISE_APPLICATION_ERROR(-20003,
            'Prescriptions can only be created for completed
   appointments');
    END IF;
```

```
END;
/
```

Listing 6: Trigger for prescription validation

5.2 Triggers Screenshot



Name	Table	Object Type	Trigger Type	Event
TRG_AUTO_GENERATE_BILL	APPOINTMENTS	TABLE	AFTER EACH ROW	UPDATE
TRG_PREVENT_DOUBLE_BOOKING	APPOINTMENTS	TABLE	BEFORE EACH ROW	INSERT
VALIDATE_APPOINTMENT_BOOKING	APPOINTMENTS	TABLE	BEFORE EACH ROW	INSERT
TRG_PRESCRIPTION_APPOINTMENT_CHECK	PRESCRIPTIONS	TABLE	BEFORE EACH ROW	INSERT

Figure 4: Triggers in Oracle Database

6 Stored Procedures and Functions

6.1 Procedure: Complete Appointment

```sql
-- 1. Procedure: Complete appointment and create prescription
CREATE OR REPLACE PROCEDURE complete_appointment(
    p_appointment_id IN NUMBER,
    p_diagnosis IN CLOB,
    p_notes IN CLOB
)
AS
    v_appointment_count NUMBER;
BEGIN
    -- Check if appointment exists and is scheduled
    SELECT COUNT(*) INTO v_appointment_count
    FROM appointments
    WHERE appointment_id = p_appointment_id AND status = 'Scheduled';

    IF v_appointment_count = 0 THEN
        RAISE_APPLICATION_ERROR(-20004,
            'Appointment not found or not scheduled');
    END IF;

    -- Update appointment status
    UPDATE appointments
    SET status = 'Completed'
    WHERE appointment_id = p_appointment_id;

    -- Create prescription
    INSERT INTO prescriptions (prescription_id, appointment_id,
                               diagnosis, notes)
    VALUES (seq_prescriptions.NEXTVAL, p_appointment_id,
            p_diagnosis, p_notes);

    COMMIT;

    DBMS_OUTPUT.PUT_LINE('Appointment ' || p_appointment_id ||
                         ' completed and prescription created.');
END;
/
```

Listing 7: Procedure to complete appointment and create prescription

6.2 Procedure: Get Doctor Availability

```sql
-- 2. Procedure: Get doctor availability
CREATE OR REPLACE PROCEDURE get_doctor_availability(
    p_doctor_id IN NUMBER,
    p_date IN DATE,
    p_available_slots OUT SYS_REFCURSOR
)
AS
BEGIN
```

```
    OPEN p_available_slots FOR
    SELECT time_slot
    FROM (
        SELECT '09:00 AM' as time_slot FROM DUAL UNION
        SELECT '09:30 AM' FROM DUAL UNION
        SELECT '10:00 AM' FROM DUAL UNION
        SELECT '10:30 AM' FROM DUAL UNION
        SELECT '11:00 AM' FROM DUAL UNION
        SELECT '11:30 AM' FROM DUAL UNION
        SELECT '02:00 PM' FROM DUAL UNION
        SELECT '02:30 PM' FROM DUAL UNION
        SELECT '03:00 PM' FROM DUAL UNION
        SELECT '03:30 PM' FROM DUAL UNION
        SELECT '04:00 PM' FROM DUAL UNION
        SELECT '04:30 PM' FROM DUAL
    ) all_slots
    WHERE time_slot NOT IN (
        SELECT appointment_time
        FROM appointments
        WHERE doctor_id = p_doctor_id
        AND appointment_date = p_date
        AND status != 'Cancelled'
    );
END;
/
```

Listing 8: Procedure to check doctor availability

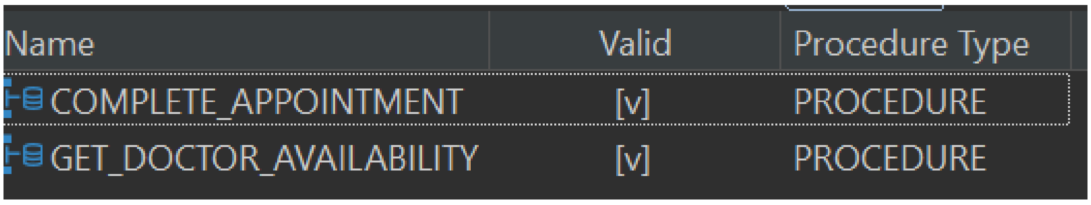
6.3 Function: Department Statistics

```
-- 3. Function: Calculate department statistics
CREATE OR REPLACE FUNCTION get_department_stats(
    p_department_id IN NUMBER
) RETURN SYS_REFCURSOR
AS
    v_stats SYS_REFCURSOR;
BEGIN
    OPEN v_stats FOR
    SELECT
        d.name as department_name,
        COUNT(DISTINCT doc.doctor_id) as doctor_count,
        COUNT(DISTINCT p.patient_id) as patient_count,
        COUNT(a.appointment_id) as appointment_count,
        SUM(CASE WHEN b.payment_status = 'Paid' THEN b.amount ELSE 0
   END)
            as total_revenue
    FROM departments d
    LEFT JOIN doctors doc ON d.department_id = doc.department_id
    LEFT JOIN appointments a ON doc.doctor_id = a.doctor_id
    LEFT JOIN patients p ON a.patient_id = p.patient_id
    LEFT JOIN bills b ON a.appointment_id = b.appointment_id
    WHERE d.department_id = p_department_id
    GROUP BY d.name;

    RETURN v_stats;
END;
```

```
/
```

Listing 9: Function to calculate department statistics

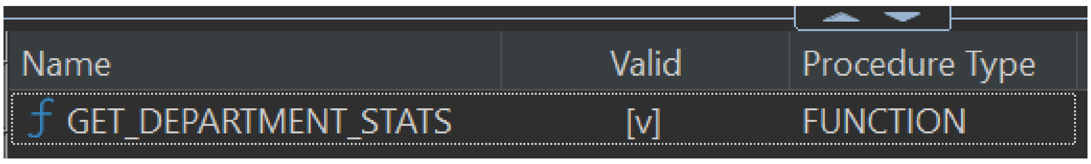
6.4 Stored Procedures Screenshot



Name	Valid	Procedure Type
COMPLETE_APPOINTMENT	[v]	PROCEDURE
GET_DOCTOR_AVAILABILITY	[v]	PROCEDURE

Figure 5: Stored Procedures in Oracle Database

6.5 Functions Screenshot



Name	Valid	Procedure Type
GET_DEPARTMENT_STATS	[v]	FUNCTION

Figure 6: Functions in Oracle Database

7 Sample Data

7.1 Sample Data Insertion

```sql
-- Insert Users
INSERT INTO users (user_id, email, password, user_type)
VALUES (seq_users.NEXTVAL, 'admin@hospital.com', 'admin123', 'admin');

INSERT INTO users (user_id, email, password, user_type)
VALUES (seq_users.NEXTVAL, 'sarah.johnson', 'doctor123', 'doctor');

INSERT INTO users (user_id, email, password, user_type)
VALUES (seq_users.NEXTVAL, 'michael.chen', 'doctor123', 'doctor');

INSERT INTO users (user_id, email, password, user_type)
VALUES (seq_users.NEXTVAL, 'john.doe', 'patient123', 'patient');

INSERT INTO users (user_id, email, password, user_type)
VALUES (seq_users.NEXTVAL, 'mary.johnson', 'patient123', 'patient');

-- Insert Departments
INSERT INTO departments (department_id, name, floor, max_capacity)
VALUES (seq_departments.NEXTVAL, 'Cardiology', 3, 8);

INSERT INTO departments (department_id, name, floor, max_capacity)
VALUES (seq_departments.NEXTVAL, 'Neurology', 4, 6);

INSERT INTO departments (department_id, name, floor, max_capacity)
VALUES (seq_departments.NEXTVAL, 'Pediatrics', 2, 10);

INSERT INTO departments (department_id, name, floor, max_capacity)
VALUES (seq_departments.NEXTVAL, 'Orthopedics', 5, 7);

INSERT INTO departments (department_id, name, floor, max_capacity)
VALUES (seq_departments.NEXTVAL, 'Emergency', 1, 12);

-- Insert Doctors
INSERT INTO doctors (doctor_id, user_id, department_id, name,
                     specialization, license_number, phone,
                     experience_years, consultation_fee)
VALUES (seq_doctors.NEXTVAL, 1002, 1, 'Dr. Sarah Johnson',
        'Cardiologist', 'CARD12345', '555-0101', 15, 200.00);

INSERT INTO doctors (doctor_id, user_id, department_id, name,
                     specialization, license_number, phone,
                     experience_years, consultation_fee)
VALUES (seq_doctors.NEXTVAL, 1003, 2, 'Dr. Michael Chen',
        'Neurologist', 'NEUR067890', '555-0102', 12, 180.00);

-- Insert Patients
INSERT INTO patients (patient_id, user_id, name, date_of_birth,
                      gender, blood_type, phone, address,
                      emergency_contact)
VALUES (seq_patients.NEXTVAL, 1004, 'John Doe', DATE '1990-05-15',
        'Male', 'O+', '555-0123', '123 Main St, City',
        'Jane Doe - 555-0124');
```

```sql

INSERT INTO patients (patient_id, user_id, name, date_of_birth,
                      gender, blood_type, phone, address,
                      emergency_contact)
VALUES (seq_patients.NEXTVAL, 1005, 'Mary Johnson', DATE '1985-08-22',
        'Female', 'A+', '555-0125', '456 Oak St, City',
        'Robert Johnson - 555-0126');

-- Add appointments
INSERT INTO appointments (appointment_id, patient_id, doctor_id,
                          appointment_date, appointment_time,
                          reason, status)
VALUES (seq_appointments.NEXTVAL, 10001, 101, DATE '2025-11-25',
        '10:00 AM', 'Heart palpitations', 'Scheduled');

INSERT INTO appointments (appointment_id, patient_id, doctor_id,
                          appointment_date, appointment_time,
                          reason, status)
VALUES (seq_appointments.NEXTVAL, 10002, 102, DATE '2025-11-26',
        '02:30 PM', 'Migraine consultation', 'Scheduled');

INSERT INTO appointments (appointment_id, patient_id, doctor_id,
                          appointment_date, appointment_time,
                          reason, status)
VALUES (seq_appointments.NEXTVAL, 10001, 103, DATE '2025-11-27',
        '11:00 AM', 'Child vaccination', 'Completed');

INSERT INTO appointments (appointment_id, patient_id, doctor_id,
                          appointment_date, appointment_time,
                          reason, status)
VALUES (seq_appointments.NEXTVAL, 10002, 104, DATE '2025-11-28',
        '03:00 PM', 'Knee pain evaluation', 'Scheduled');

COMMIT;
```

Listing 10: Sample data insertion statements

8 Indexes

8.1 Automatically Generated Indexes

Oracle Database automatically creates B-tree indexes for all PRIMARY KEY and UNIQUE constraints to ensure data integrity and optimize query performance. The following 10 indexes were automatically generated:

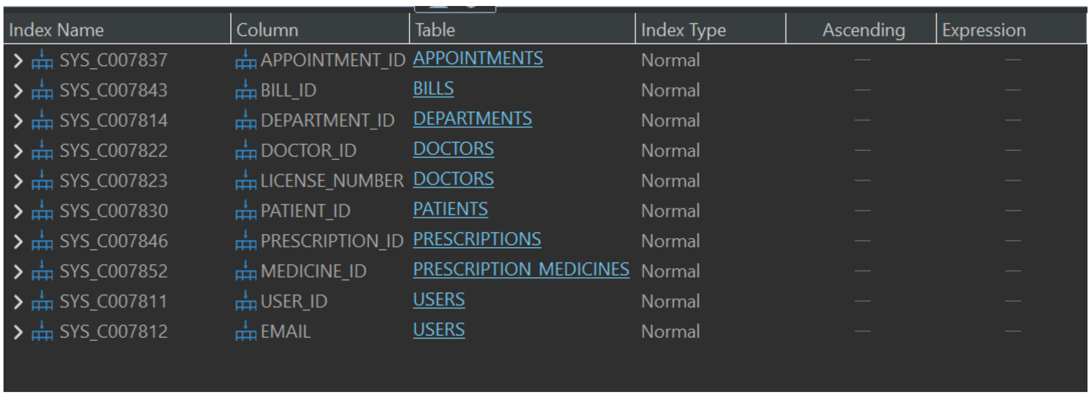
Index Name	Table	Purpose
appointment_id	appointments	Primary Key
bill_id	bills	Primary Key
department_id	departments	Primary Key
doctor_id	doctors	Primary Key
license_number	doctors	Unique Constraint
patient_id	patients	Primary Key
prescription_id	prescriptions	Primary Key
medicine_id	prescription_medicines	Primary Key
user_id	users	Primary Key
email	users	Unique Constraint

Table 1: Automatically Generated Indexes

8.2 Benefits of Automatic Indexing

- **Performance**: Accelerates data retrieval for primary key lookups
- **Data Integrity**: Enforces uniqueness constraints efficiently
- **Referential Integrity**: Improves foreign key constraint validation
- **Query Optimization**: Optimizer uses indexes for JOIN operations

8.3 Indexes Screenshot



Index Name	Column	Table	Index Type	Ascending	Expression
SYS_C007837	APPOINTMENT_ID	APPOINTMENTS	Normal	—	—
SYS_C007843	BILL_ID	BILLS	Normal	—	—
SYS_C007814	DEPARTMENT_ID	DEPARTMENTS	Normal	—	—
SYS_C007822	DOCTOR_ID	DOCTORS	Normal	—	—
SYS_C007823	LICENSE_NUMBER	DOCTORS	Normal	—	—
SYS_C007830	PATIENT_ID	PATIENTS	Normal	—	—
SYS_C007846	PRESCRIPTION_ID	PRESCRIPTIONS	Normal	—	—
SYS_C007852	MEDICINE_ID	PRESCRIPTION_MEDICINES	Normal	—	—
SYS_C007811	USER_ID	USERS	Normal	—	—
SYS_C007812	EMAIL	USERS	Normal	—	—

Figure 7: Automatically Generated Indexes in Oracle Database

9 Conclusion

9.1 Achievements

- Successfully designed and implemented a 3NF compliant database schema
- Implemented comprehensive business rules using triggers
- Created reusable stored procedures and functions
- Ensured data integrity through constraints and validation
- Demonstrated advanced PL/SQL programming techniques

9.2 Key Features

- Unified user authentication system
- Appointment scheduling with double-booking prevention
- Automatic bill generation
- Prescription management with validation
- Department statistics and reporting
- Comprehensive data validation