Technical Report

GROUP 4

Omar Altalafeeh, Oghenemine Ejayeriese, Marcus Guillory, Bryce Holladay, Khanh Tran

The University of Texas at Dallas | JSOM UTD

BUAN 6320: Database Foundations for Business Analytics

Dr. Gasan Elkhodari

July 31, 2024

Introduction

This technical report details the design, implementation, and management of a hospital database schema. The schema includes tables for patients, rooms, appointments, treatments, and doctors. Each table is defined with specific fields and constraints to ensure data integrity and facilitate relationships between the tables. This report covers the SQL commands used to create and manage these tables and demonstrates querying the database using views.

Literature Review

Database design for healthcare systems requires careful consideration to ensure data integrity, privacy, and efficiency. Studies have shown that well-structured relational databases can significantly improve data retrieval times and accuracy in medical settings. Best practices is the use of foreign key constraints to maintain referential integrity, and designing for scalability to handle growing data volumes.

Assumptions and Special Considerations

The assumption is the hospital offers services for mild treatments or issues, and not severe cases for example in the ER when multiple doctors need to be present for a patient. We as well have constraints for the foreign keys we referenced in select tables. Some fields in the table can’t be a NULL value (for example if there is an entry in the appointment table there needs to be an appointment date).

Design Decisions

The design of this hospital database schema involved the following key decisions:

1. **Foreign Key Constraints**: These were implemented to maintain relationships between tables.
2. **Data Types and Constraints**: Appropriate data types and constraints (e.g., CHECK constraints on gender) were used to ensure data validity.
3. **Scalability**: The schema was designed to be scalable to handle increasing data volumes and complexity.
4. **Views**: Views were created to simplify complex queries and provide a user-friendly interface for data retrieval.

Project Scope

In-Scope Work

- Designing and creating tables for patients, rooms, appointments, treatments, and doctors.
- Implementing foreign key constraints to maintain relationships between tables.
- Creating views for simplified data retrieval.
- Testing the database schema for integrity and performance.
- Documenting the entire process and providing maintenance guidelines.

Out-of-Scope Work

- Developing a user interface for interacting with the database.

- Implementing advanced security measures beyond basic constraints and foreign keys.
- Integrating the database with external systems or software.
- Handling large-scale data migration from existing systems.

Database Goals

- **Data Integrity**: Ensure that all data entered into the database is accurate and consistent.
- **Efficiency**: Optimize the database for fast data retrieval and minimal redundancy.
- **Scalability**: Design the database to accommodate future growth in data volume and complexity.
- **Usability**: Create views and documentation to make the database user-friendly for administrators and other users.

Expectations and Deliverables

- **Entity Relationship Diagram (ERD)**: Visual representation of the database schema.
- **SQL Scripts**: Code to create tables, constraints, and views.
- **Technical Report**: Detailed documentation of the database design, implementation, and management.
- **Testing Results**: Evidence of successful database testing.
- **Maintenance Guidelines**: Instructions for maintaining and scaling the database.

Project Management Methodology

The project followed the Agile methodology, which allowed for iterative development and continuous feedback. Key stages included:

1. **Planning**: Defining the project scope, goals, and deliverables.
2. **Design**: Creating the ERD and detailed schema design.
3. **Implementation**: Writing and executing SQL scripts.
4. **Testing**: Verifying the functionality and performance of the database.
5. **Documentation**: Compiling all necessary documentation and guidelines.
6. **Review and Iteration**: Continuously reviewing progress and making necessary adjustments.

Requirements Definition Document

Business Rules

1. Only one ROOM must be assigned to one PATIENT.
2. Zero or one PATIENT may be assigned to one and only one ROOM.
3. A PATIENT has none or many APPOINTMENTS.
4. An APPOINTMENT has one and only one PATIENT.
5. An APPOINTMENT must have one TREATMENT.
6. A TREATMENT must be included by only one APPOINTMENT.
7. A DOCTOR can have zero, one, or many APPOINTMENTS.

8. An APPOINTMENT can only be assigned to one DOCTOR at a time.

Entity and Attribute Description

Entities:

Entity Name: *PATIENT* Entity

Description: Information about a patient at the hospital

Main Attributes of *PATIENT*:

PatientID (PK): Unique identifier for each patient

FirstName: First name of the patient

LastName: Last name of the patient

BirthDate: Date of birth of the patient

Gender: Gender of the patient

ContactNumber: Best phone number (home, cell, etc) of the patient

Entity Name: *ROOM* Entity

Description: Area for which each patient is housed during treatment

Main Attributes of *ROOM*:

RoomID: Serves as a unique identifier for each room record in the database

RoomNumber: Represents the specific identifier assigned to each room within the facility to easily locate specific patients

RoomType: Describes the category or type of room based on intended use

AvailabilityStatus: Indicates current status of the room to aid in scheduling or resource allocation

Capacity: Specifies the maximum number of individuals that the room can accommodate

PatientID (FK): Indicates which patient is occupying this room

Entity Name: *APPOINTMENT* Entity

Description: The schedule when the patient is going to meet the doctor.

Main Attributes of *APPOINTMENT*:

AppointmentID (PK): A unique identifier for the Appointment entity.

PatientID (FK): References the Patient entity to indicate which patient the appointment is for.

DoctorID (FK): References the Doctor entity to indicate which doctor is conducting the appointment.

AppointmentDate: The date and time when the appointment is scheduled to take place.

ReasonForVisit: A brief description of the reason why the patient is visiting the doctor.

Duration: The length of the appointment, typically measured in minutes or hours.

Status: The current status of the appointment (e.g., scheduled, completed, canceled).

Entity Name: *TREATMENT* Entity

Description: What is the treatment the patient is on

Main Attributes of *TREATMENT*:

TreatmentID: A unique identifier for the treatment.

TreatmentName: What is the treatment name the patient is on.

Description: A brief description of what treatment the patient is on

Cost: How much the cost of the treatment will be

Meds: Describe the medication he is going to take.

AppointmentID(FK): References the Appointment entity to indicate which Appointment the patient is in.

Entity Name: *DOCTOR* Entity

Description: Profile of a Doctor at a hospital system

Main Attributes of *DOCTOR*:

DoctorID (PK): Unique identifier and primary key to be able to identify all doctors respectively

FirstName: This attribute lists the first name of the doctor

LastName: This attribute lists the last name of the doctor completing levels.

Specialty: This attribute identifies the practice the doctor is known for.

ContactNumber: This list the contact number of the doctor, if he/she needs to be reached.

Relationship and Cardinality Description

Relationship: assigned to between ROOM and PATIENT.

Cardinality: ROOM (1,1) to PATIENT (0,1).

Business rule: Only one ROOM must be assigned to one PATIENT; Zero or one PATIENT may be assigned to one and only one room.

Relationship: PATIENT has APPOINTMENT

Cardinality: PATIENT (1,1) to APPOINTMENT (0,M)

Business rule: A patient has none or many appointments; an appointment has one and only one patient

Relationship: includes between APPOINTMENT and TREATMENT.

Cardinality: APPOINTMENT (1,1) to TREATMENT (1,1).

Business rule: an APPOINTMENT must have one TREATMENT; a TREATMENT must be included by only one APPOINTMENT.

Relationship: A doctor at a hospital conducts appointments

Cardinality: DOCTOR (1,1) to APPOINTMENT (0,M)

Business Rule: A doctor can have 0, 1, or many appointments. An Appointment can only be assigned to one doctor at a time.

DDL Source Code

```
SET search_path to public


------- DROP statements to clean up objects from previous run

-- FK Constraints

ALTER TABLE room DROP CONSTRAINT fk_room_patientid;

ALTER TABLE appointment DROP CONSTRAINT fk_appointment_patientid;

ALTER TABLE appointment DROP CONSTRAINT fk_appointment_doctorid;

ALTER TABLE treatment DROP CONSTRAINT fk_treatment_appointmentid;


-- Trigger

DROP TRIGGER check_appointment_date ON appointment;


-- Function

DROP FUNCTION check_appointment_date_func();


-- Sequences

DROP SEQUENCE room_id_seq;

DROP SEQUENCE room_num_seq;

DROP SEQUENCE patient_id_seq;

DROP SEQUENCE doc_id_seq;

DROP SEQUENCE treatment_id_seq;

DROP SEQUENCE appointment_id_seq;


-- Views

DROP VIEW patient_appointments;


-- Tables

DROP TABLE patient;
```

```
DROP TABLE room;

DROP TABLE appointment;

DROP TABLE treatment;

DROP TABLE doctor;



-------------------------------------------------------------------------------------------
---



------- Create the patient table



CREATE TABLE patient (

    PatientID INT PRIMARY KEY,              -- Unique identifier for each patient

    FirstName VARCHAR(50) NOT NULL,         -- Patient's first name

    LastName VARCHAR(50) NOT NULL,          -- Patient's last name

    BirthDate DATE NOT NULL,                -- Patient's birth date

    Gender CHAR(1) CHECK (Gender IN ('M', 'F', 'O')), -- Patient's gender (M = Male, F =
Female, O = Other)

    ContactNumber VARCHAR(15)               -- Patient's contact number

);



------- Create the room table with foreign key constraint



CREATE TABLE room (

    RoomID INT PRIMARY KEY,                 -- Unique identifier for each room

    RoomNumber INT NOT NULL,                -- Unique Room number (e.g., '101', '202A')

    RoomType VARCHAR(50) NOT NULL,          -- Type of room (e.g., 'Private', 'Semi-Private',
'Shared')

    AvailabilityStatus BOOLEAN NOT NULL,    -- Availability status (TRUE = available, FALSE =
not available)

    Capacity INT NOT NULL,                  -- Capacity of the room (e.g., number of beds)

    PatientID INT,                          -- ID of the patient assigned to the room
```

```
       CONSTRAINT fk_room_patientid            -- Foreign key constraint
        FOREIGN KEY (PatientID)
        REFERENCES patient (PatientID) ON DELETE CASCADE
);


------- Create the doctor table


CREATE TABLE doctor (
    DoctorID INT PRIMARY KEY,               -- Unique identifier for each doctor
    FirstName VARCHAR(50) NOT NULL,         -- Doctor's first name
    LastName VARCHAR(50) NOT NULL,          -- Doctor's last name
    Specialty VARCHAR(100),                 -- Doctor's specialty (e.g., 'Cardiology',
'Orthopedics')
    ContactNumber VARCHAR(15)               -- Doctor's contact number
);


------- Create the appointment table with foreign key constraints


CREATE TABLE appointment (
    AppointmentID INT PRIMARY KEY,          -- Unique identifier for each appointment
    PatientID INT NOT NULL,                 -- ID of the patient for the appointment
    DoctorID INT NOT NULL,                  -- ID of the doctor for the appointment
    AppointmentDate DATE NOT NULL,          -- Date and time of the appointment
    ReasonForVisit VARCHAR(200),            -- Reason for the visit (e.g., symptoms,
check-up)
    Duration INT NOT NULL,                  -- Duration of the appointment (e.g., '1 hour',
'30 minutes')
    Status VARCHAR(50) NOT NULL,            -- Status of the appointment (e.g., 'Scheduled',
'Completed', 'Cancelled')
       CONSTRAINT fk_appointment_patientid      -- Foreign key constraint
        FOREIGN KEY (PatientID)
```

```
        REFERENCES patient (PatientID) ON DELETE CASCADE,
       CONSTRAINT fk_appointment_doctorid       -- Foreign key constraint
        FOREIGN KEY (DoctorID)
        REFERENCES doctor (DoctorID) ON DELETE CASCADE
);

------- Create the treatment table with a foreign key constraint

CREATE TABLE treatment (
    TreatmentID INT PRIMARY KEY,             -- Unique identifier for each treatment
    TreatmentName VARCHAR(100) NOT NULL,     -- Name of the treatment
    Description VARCHAR(200),                -- Detailed description of the treatment
    Cost NUMERIC(6, 2) NOT NULL,             -- Cost of the treatment (e.g., 100.00)
    Meds VARCHAR(100),                       -- Medications or drugs used in the treatment
    AppointmentID INT,                       -- ID of the associated appointment
       CONSTRAINT fk_treatment_appointmentid    -- Foreign key constraint
        FOREIGN KEY (AppointmentID)
        REFERENCES appointment (AppointmentID) ON DELETE CASCADE
);

-------------------------------------------------------------------------------------------
---

-- Create the trigger function for AppointmentDate " can not be in the past "
CREATE OR REPLACE FUNCTION check_appointment_date_func()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.AppointmentDate < CURRENT_DATE THEN
        RAISE EXCEPTION 'AppointmentDate cannot be in the past';
    END IF;
```

```
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;


-- Create the trigger for AppointmentDate
CREATE TRIGGER check_appointment_date
BEFORE INSERT OR UPDATE ON appointment
FOR EACH ROW
EXECUTE FUNCTION check_appointment_date_func();




-------------------------------------------------------------------------------------------
---


-- SEQUENCE for column 'RoomID'
CREATE SEQUENCE room_id_seq
START WITH 1
INCREMENT BY 1
MINVALUE 1
MAXVALUE 999999                      -- MAX VAL is set to 999999
CYCLE
CACHE 20;


-- SEQUENCE for column 'RoomNumber'
CREATE SEQUENCE room_num_seq
START WITH 101
INCREMENT BY 1
MINVALUE 101
MAXVALUE 999999
CYCLE
```

```
CACHE 20;


-- SEQUENCE for column 'PatientID'

CREATE SEQUENCE patient_id_seq

START WITH 1

INCREMENT BY 1

MINVALUE 1

MAXVALUE 999999

CYCLE

CACHE 20;


-- SEQUENCE for column 'DoctorID'

CREATE SEQUENCE doc_id_seq

START WITH 1

INCREMENT BY 1

MINVALUE 1

MAXVALUE 999999

CYCLE

CACHE 20;


-- SEQUENCE for column 'TreatmentID'

CREATE SEQUENCE treatment_id_seq

START WITH 1

INCREMENT BY 1

MINVALUE 1

MAXVALUE 999999

CYCLE

CACHE 20;
```

```
-- SEQUENCE for column 'AppointmentID'

CREATE SEQUENCE appointment_id_seq

START WITH 1

INCREMENT BY 1

MINVALUE 1

MAXVALUE 999999

CYCLE

CACHE 20;
```

DML and Query Source Code

```
SET search_path to public


-- Insert data into the Patient table
INSERT INTO patient (PatientID, Firstname, Lastname, Birthdate, Gender, ContactNumber) VALUES
(NEXTVAL('patient_id_seq'), 'John', 'Doe', '1980-01-15', 'M', '(123)-456-7890'),
(NEXTVAL('patient_id_seq'), 'Jane', 'Smith', '1990-02-20', 'F', '(234)-567-8901'),
(NEXTVAL('patient_id_seq'), 'Alice', 'Johnson', '1975-03-25', 'F', '(345)-678-9012'),
(NEXTVAL('patient_id_seq'), 'Bob', 'Brown', '1985-04-30', 'M', '(456)-789-0123'),
(NEXTVAL('patient_id_seq'), 'Charlie', 'Davis', '1995-05-10', 'M', '(567)-890-1234');


-- Insert data into the Room table using existing PatientIDs
INSERT INTO room (RoomID, RoomNumber, RoomType, AvailabilityStatus, Capacity, PatientID)
VALUES
(NEXTVAL('room_id_seq'), NEXTVAL('room_num_seq'), 'Single', TRUE, 1, 1),  -- Assuming
PatientID 1 exists
(NEXTVAL('room_id_seq'), NEXTVAL('room_num_seq'), 'Double', FALSE, 2, 2), -- Assuming
PatientID 2 exists
(NEXTVAL('room_id_seq'), NEXTVAL('room_num_seq'), 'Single', TRUE, 1, 3),  -- Assuming
PatientID 3 exists
(NEXTVAL('room_id_seq'), NEXTVAL('room_num_seq'), 'Suite', TRUE, 3, 4),   -- Assuming
PatientID 4 exists
```

```
(NEXTVAL('room_id_seq'), NEXTVAL('room_num_seq'), 'Single', FALSE, 1, 5); -- Assuming
PatientID 5 exists

-- Insert data into the Doctor table
INSERT INTO doctor (DoctorID, Firstname, Lastname, Specialty, ContactNumber) VALUES
(NEXTVAL('doc_id_seq'), 'Dr. Emily', 'White', 'Cardiology', '(123)-456-7890'),
(NEXTVAL('doc_id_seq'), 'Dr. James', 'Green', 'Neurology', '(234)-567-8901'),
(NEXTVAL('doc_id_seq'), 'Dr. Linda', 'Black', 'Orthopedics', '(345)-678-9012'),
(NEXTVAL('doc_id_seq'), 'Dr. Michael', 'Blue', 'Pediatrics', '(456)-789-0123'),
(NEXTVAL('doc_id_seq'), 'Dr. Sarah', 'Red', 'Dermatology', '(567)-890-1234');

-- Insert data into the Appointment table using existing Patient & Doctor IDs
INSERT INTO appointment (AppointmentID, AppointmentDate, ReasonForVisit, Duration, Status,
PatientID, DoctorID) VALUES
(NEXTVAL('appointment_id_seq'), '2024-08-25', 'Regular Checkup', 30, 'Scheduled', 1, 1), --
Assuming PatientID 1 and DoctorID 1 exists
(NEXTVAL('appointment_id_seq'), '2024-08-26', 'Follow-up', 45, 'Scheduled', 2, 2),       --
Assuming PatientID 2 and DoctorID 2 exists
(NEXTVAL('appointment_id_seq'), '2024-08-27', 'Consultation', 20, 'Scheduled', 3, 3),    --
Assuming PatientID 3 exists and DoctorID 3 exists
(NEXTVAL('appointment_id_seq'), '2024-08-28', 'Emergency', 60, 'Completed', 4, 4),       --
Assuming PatientID 4 exists and DoctorID 4 exists
(NEXTVAL('appointment_id_seq'), '2024-09-29', 'Routine Check', 30, 'Scheduled', 5, 5);   --
Assuming PatientID 5 exists and DoctorID 5 exists

-- Insert data into the Treatment table
INSERT INTO treatment (TreatmentID, TreatmentName, Description, Cost, Meds, AppointmentID)
VALUES
(NEXTVAL('treatment_id_seq'), 'Physical Therapy, Surgery', 'Therapeutic exercises', 3100.00,
'Painkillers', 1),
(NEXTVAL('treatment_id_seq'), 'Chemotherapy', 'Cancer treatment', 2000.00, 'Chemotherapy
drugs', 2),
(NEXTVAL('treatment_id_seq'), 'Radiology', 'X-ray imaging', 150.00, 'None', 3),
(NEXTVAL('treatment_id_seq'), 'Surgery', 'Appendectomy', 5000.00, 'Antibiotics', 4),
(NEXTVAL('treatment_id_seq'), 'Vaccination', 'Flu vaccine', 50.00, 'None', 5);
```

```
-------------------------------------------------------------------------------------------
---



-- Query 1:
SELECT * FROM Patient;

-- Query 2:
SELECT PatientID, FirstName, LastName, BirthDate, Gender FROM Patient;

-- Query 3:
-- Create the VIEW
CREATE VIEW patient_appointments AS
SELECT
    p.PatientID,
    p.FirstName AS PatientFirstName,
    p.LastName AS PatientLastName,
    a.AppointmentID,
    a.AppointmentDate,
    a.ReasonForVisit,
    a.Status AS AppointmentStatus,
    d.FirstName AS DoctorFirstName,
    d.LastName AS DoctorLastName,
    d.Specialty AS DoctorSpecialty
FROM
    patient p
JOIN
    appointment a ON p.PatientID = a.PatientID
JOIN
    doctor d ON a.DoctorID = d.DoctorID;

-- RUN the VIEW
SELECT * FROM patient_appointments;

-- Query 4:
```

```
SELECT *
FROM Appointment a
JOIN Patient p ON a.PatientID = p.PatientID;

-- Query 5:
SELECT * FROM Doctor
ORDER BY doctorid ASC;

-- Query 6:
SELECT p.FirstName, p.LastName, a.AppointmentDate, d.FirstName AS
DoctorFirstName, d.LastName AS DoctorLastName
FROM Appointment a
JOIN Patient p ON a.PatientID = p.PatientID
JOIN Doctor d ON a.DoctorID = d.DoctorID
LIMIT 3;

-- Query 7:
SELECT DISTINCT p.FirstName, p.LastName, d.FirstName AS
DoctorFirstName, d.LastName AS DoctorLastName, r.RoomNumber
FROM Appointment a
JOIN Patient p ON a.PatientID = p.PatientID
JOIN Doctor d ON a.DoctorID = d.DoctorID
JOIN Room r ON a.PatientID = r.PatientID;

-- Query 8:
SELECT
       d.DoctorID,
       d.FirstName,
       d.LastName,
ROUND(AVG(t.Cost), 2) AS AverageCost
FROM
doctor d
JOIN
appointment a ON d.DoctorID = a.DoctorID
JOIN
treatment t ON a.AppointmentID = t.AppointmentID
```

```
GROUP BY
       d.DoctorID, d.FirstName, d.LastName
HAVING
AVG(t.Cost) > 200;

-- Query 9:
SELECT * FROM Appointment
WHERE DoctorID IN (SELECT DoctorID FROM Doctor WHERE Specialty = 'Cardiology');

-- Query 10:
SELECT FirstName, LENGTH(FirstName) AS FirstNameLength
FROM Patient;

-- Query 11:
SELECT * FROM Patient WHERE PatientID = 1;      -- View Table pre-DELETE statement

BEGIN;
DELETE FROM Patient WHERE PatientID = 1;        -- DELETE statement

SELECT * FROM Patient WHERE PatientID = 1;      -- View Table after DELETE statement

ROLLBACK;                                       -- ROLLBACK DELETE statement

-- Query 12:
SELECT * FROM Patient WHERE PatientID = 1;      -- View Table pre-UPDATE statement

BEGIN;
UPDATE Patient SET FirstName = 'Jacob' WHERE PatientID = 1;  -- UPDATE statement

SELECT * FROM Patient WHERE PatientID = 1;      -- View Table after UPDATE statement

ROLLBACK;                                       -- ROLLBACK UPDATE statement

-- Advanced Query 1: Number of Patients per Room Type and Average Cost of Treatments
SELECT
r.RoomType,
```

```
        COUNT(DISTINCT p.PatientID) AS NumberOfPatients,
        ROUND(AVG(t.Cost), 2) AS AverageTreatmentCost
        FROM
        Room r
        JOIN
        Patient p ON r.PatientID = p.PatientID
        JOIN
        Appointment a ON p.PatientID = a.PatientID
        JOIN
        Treatment t ON a.AppointmentID = t.AppointmentID
GROUP BY
r.RoomType
ORDER BY
NumberOfPatients DESC;

-- Advanced Query 2: Retrieve Detailed Patient Visit Information
SELECT
    p.PatientID,
    p.FirstName || ' ' || p.LastName AS PatientName,
    d.FirstName || ' ' || d.LastName AS DoctorName,
    r.RoomNumber,
    r.RoomType,
    a.AppointmentDate,
    a.ReasonForVisit,
    STRING_AGG(t.TreatmentName, ', ') AS Treatments,
    SUM(t.Cost) AS TotalTreatmentCost
FROM
    patient p
JOIN
    appointment a ON p.PatientID = a.PatientID
JOIN
    doctor d ON a.DoctorID = d.DoctorID
LEFT JOIN
    room r ON p.PatientID = r.PatientID
LEFT JOIN
    treatment t ON a.AppointmentID = t.AppointmentID
```

```
GROUP BY
    p.PatientID, p.FirstName, p.LastName,
    d.FirstName, d.LastName,
    r.RoomNumber, r.RoomType,
    a.AppointmentDate, a.ReasonForVisit
ORDER BY
    p.LastName, p.FirstName, a.AppointmentDate;
```

Conclusion

This report presents a comprehensive approach to designing, implementing, and managing a hospital database schema. The schema includes well-defined tables with appropriate constraints and relationships. The use of views simplifies complex queries and enhances data accessibility. This robust schema ensures efficient data management and supports various hospital operations.