Em

Project DBD281

Group 281-V1-A

**Members:**

Petrus Human 577842  
Jonathan Reece Joubert 578085  
Heinrich Rudman 601001  
Sandhyaa Budhu 601240

Belgium Campus Itversity

DBD281-Galaletsang Itumelang Modimola



Table of Contents

[Introduction 2](#_Toc162432356)

[Background 2](#_Toc162432357)

[Entity Relationship Diagram 2](#_Toc162432358)

[3](#_Toc162432359)

[Normalization of my database 4](#_Toc162432360)

[Objects in my database 4](#_Toc162432361)

[Tables 4](#_Toc162432362)

[Triggers 4](#_Toc162432363)

[Other Objects 4](#_Toc162432364)

[Questions that can be answered with the database. 5](#_Toc162432365)

[Login 7](#_Toc162432366)

[Questions Answers 7](#_Toc162432367)

# Introduction and Background

Nestled within the vibrant cityscape of Pretoria, South Africa, Golden Gate Dental Clinic stands out as a shining example of top-tier oral healthcare. From the very beginning, the clinic has remained steadfast in its commitment to providing comprehensive dental services to the local community. This dedication stems from the leadership of Dr. John Doe.

Guided by Dr. Doe's expertise, a dedicated team of dental professionals including seasoned associate dentists, committed hygienists, attentive assistants, and proficient administrative staff work tirelessly to uphold the clinic's unwavering commitment to exceptional patient care.

Despite their dedication, Golden Gate Dental Clinic faces challenges in effectively managing billing, inventory, and regulatory compliance. Manual billing processes are time-consuming and prone to errors, while inventory management issues can disrupt operations. Additionally, maintaining compliance with patient privacy laws demands rigorous security measures and meticulous record-keeping practices.

To overcome these challenges and enhance efficiency, the clinic asked us to implement an relational database . This technological advancement will automate billing, monitor inventory levels, and ensure regulatory compliance, ultimately streamlining operations and improving patient care outcomes.

# ERD

**PK = Primary Key**

**FK = Foreign Key**

# Normalization of my database

Normalization is the process of organizing data in a database to reduce redundancy of data. There are several normal forms that can be achieved in a database, but Third Normal Form (3NF) is the most desired form. To achieve 3NF the database first needs to be in Second Normal Form (2NF) and then no transitive dependencies should exist for the database to be in 3NF. Before we can look at 2NF and 3NF we first needed to achieve First Normal Form (1NF).

First Normal Form (1NF):

This is when the data is organized in a table with each cell containing a single value, avoiding repeating groups of data. Columns have unique names, rows represent unique records, and the order of rows and columns is insignificant. After we achieve NF1 in the Dentist Database we can move forward to 2NF.

## Second Normal Form (2NF):

Requires that every non-prime attribute of a table is fully functionally dependent on the entire primary key. This means that each attribute should depend on the whole primary key, not just part of it. We can see this looking at the tables in the dentist database. That all tables have a primary key and there is no partial dependencies meaning we can move on to achieve 3NF.

## Third Normal Form (3NF):

Third Normal Form (3NF) eliminates transitive dependencies, which means that no nonprime attribute should be functionally dependent on another non-prime attribute. For example, let's consider the Employees table. The IsAvailable attribute is only dependent on the Employee\_ID, which is the primary key. There are no apparent transitive dependencies in this table. Meaning we achieve 3NF in our Dentist database because we meet the criteria for 2NF and there is no transitive dependencies

# Objects in my database

## Tables

* Employee\_Types
* Employees
* Patients
* Rooms
* Room\_Breaks
* Room\_Bookings
* Appontment\_Workers
* Treatments
* Patient\_Treatments
* Appointments
* Payment

## Stored procedures

--Stored procedures

GO

-- Parameterized query to find a patient by their ID, also prevents SQL injection:

CREATE PROCEDURE GetPatientByID @PatientID INT

AS

BEGIN

SELECT \*

FROM Patients

WHERE Patient\_ID = @PatientID;

END;

-- EXECUTE GetPatientByID @PatientID = 1;

GO

-- Stored procedure to update patient information

CREATE PROCEDURE UpdatePatientInfo

@PatientID INT,

@NewCity VARCHAR(50)

AS

BEGIN

UPDATE Patients

SET City = @NewCity

WHERE Patient\_ID = @PatientID;

END;

-- Execute UpdatePatientInfo stored procedure

EXECUTE UpdatePatientInfo @PatientID = 123, @NewCity = 'New York';

-- Replace 123 with the actual Patient\_ID and 'New York' with the new city value

GO

-- Stored procedure to calculate total revenue for a specific date range

CREATE PROCEDURE CalculateRevenue

@StartDate DATE,

@EndDate DATE

AS

BEGIN

SELECT SUM(p.Total) AS TotalRevenue

FROM Payment p

WHERE p.Payment\_Date BETWEEN @StartDate AND @EndDate;

END;

-- Execute CalculateRevenue stored procedure

DECLARE @StartDate DATE = '2024-01-01';

DECLARE @EndDate DATE = '2024-12-31';

EXECUTE CalculateRevenue @StartDate, @EndDate;

GO

-- Stored procedure to delete a patient and all related records

CREATE PROCEDURE DeletePatientCascade

@PatientID INT

AS

BEGIN

DELETE FROM Appointments WHERE Patient\_ID = @PatientID;

DELETE FROM Patients WHERE Patient\_ID = @PatientID;

END;

-- Execute DeletePatientCascade stored procedure

EXECUTE DeletePatientCascade @PatientID = 123;

-- Replace 123 with the actual Patient\_ID to be deleted

## Views

--Views

GO

--View for patient appointments and treatments with employee details:

CREATE VIEW PatientAppointmentDetails AS

SELECT a.Appointment\_ID, a.Appointment\_Date, a.Appointment\_time,

p.Patient\_FirstName, p.Patient\_LastName, p.City,

t.Treatment, py.Payment\_Method

FROM Appointments a

INNER JOIN Patients p ON a.Patient\_ID = p.Patient\_ID

INNER JOIN Patient\_Treatments pt ON a.PatientTreatment\_ID = pt.PatientTreatment\_ID

INNER JOIN Treatments t ON pt.Treatment\_ID = t.Treatment\_ID

INNER JOIN Payment py ON pt.PatientTreatment\_ID = py.PatientTreatment\_ID;

GO

-- View to look at employee performance like how many patients, revenue and avg duration per patient:

CREATE VIEW EmployeePerformance

AS

SELECT CONCAT(E.Employee\_FirstName,' ' ,E.Employee\_LastName) 'EmployeeFullName', COUNT(A.Patient\_ID) 'TotalAppointments',

SUM(A.Price) 'TotalRevenue', AVG(RB.Duration) 'AverageDuration'

FROM Appointments A

LEFT JOIN Appointment\_Workers AW ON A.Appointment\_WorkerID = AW.Appointment\_WorkerID

LEFT JOIN Employees E ON AW.Employee\_ID = E.Employee\_ID

LEFT JOIN Room\_Bookings RB ON A.RoomBooking\_ID = RB.RoomBooking\_ID

GROUP BY CONCAT(E.Employee\_FirstName,' ' ,E.Employee\_LastName);

GO

--View for appointments with patient details:

CREATE VIEW AppointmentDetails AS

SELECT A.\*, P.Patient\_FirstName, P.Patient\_LastName, P.City

FROM Appointments A

INNER JOIN Patients P ON A.Patient\_ID = P.Patient\_ID;

GO

--View for room utilization:

CREATE VIEW RoomUtilization AS

SELECT R.Room\_ID, R.RoomName, COUNT(RB.RoomBooking\_ID) AS Bookings

FROM Rooms R

LEFT JOIN Room\_Bookings RB ON R.Room\_ID = RB.Room\_ID

GROUP BY R.Room\_ID, R.RoomName;

GO

--View for patient treatments and payments:

CREATE VIEW PatientPayments AS

SELECT P.Patient\_ID, P.Patient\_FirstName, P.Patient\_LastName, T.Treatment, Py.Payment\_Method, Py.Total

FROM Patients P

INNER JOIN Patient\_Treatments PT ON P.Patient\_ID = PT.Patient\_ID

INNER JOIN Treatments T ON PT.Treatment\_ID = T.Treatment\_ID

INNER JOIN Payment Py ON PT.PatientTreatment\_ID = Py.PatientTreatment\_ID;

GO

--View for appointment statuses:

CREATE VIEW AppointmentStatus AS

SELECT Patient\_ID, Status, COUNT(\*) AS Status\_Count

FROM Appointments

GROUP BY Patient\_ID, Status;

GO

-- appointment details with revenue:

CREATE VIEW AppointmentRevenueDetails AS

SELECT A.Appointment\_ID, A.Appointment\_Date, A.Appointment\_time,

P.Patient\_FirstName, P.Patient\_LastName, P.City,

T.Treatment, Py.Payment\_Method, Py.Total AS Revenue

FROM Appointments A

INNER JOIN Patients P ON A.Patient\_ID = P.Patient\_ID

INNER JOIN Patient\_Treatments PT ON A.PatientTreatment\_ID = PT.PatientTreatment\_ID

INNER JOIN Treatments T ON PT.Treatment\_ID = T.Treatment\_ID

INNER JOIN Payment Py ON PT.PatientTreatment\_ID = Py.PatientTreatment\_ID;

Go

--Appointment Statistics by Employee

CREATE VIEW AppointmentStatisticsByEmployee AS

SELECT CONCAT(E.Employee\_FirstName, ' ', E.Employee\_LastName) AS EmployeeName,

COUNT(A.Appointment\_ID) AS TotalAppointments,

SUM(P.Total) AS TotalRevenue

FROM Employees E

INNER JOIN Appointment\_Workers AW ON E.Employee\_ID = AW.Employee\_ID

INNER JOIN Appointments A ON AW.Appointment\_WorkerID = A.Appointment\_WorkerID

INNER JOIN Patient\_Treatments PT ON A.PatientTreatment\_ID = PT.PatientTreatment\_ID

INNER JOIN Payment P ON PT.PatientTreatment\_ID = P.PatientTreatment\_ID

GROUP BY E.Employee\_ID, E.Employee\_FirstName, E.Employee\_LastName;

## Triggers

--Triggers

GO

-- Trigger to log patient deletions

CREATE TRIGGER LogPatientDeletions

ON Patients

INSTEAD OF DELETE

AS

BEGIN

INSERT INTO DeletedPatientsLog (PatientID, DeletionDate)

SELECT Patient\_ID, GETDATE()

FROM deleted;

DELETE FROM Patients WHERE Patient\_ID IN (SELECT Patient\_ID FROM deleted);

END;

GO

-- Trigger to enforce unique emails for patients

CREATE TRIGGER EnforceUniqueEmail

ON Patients

AFTER INSERT, UPDATE

AS

BEGIN

IF EXISTS (SELECT 1 FROM inserted GROUP BY Email HAVING COUNT(\*) > 1)

BEGIN

RAISERROR ('Email must be unique.', 16, 1);

ROLLBACK TRANSACTION;

END;

END;

## Other Objects

--other objects (indexes,views, and constraints)

--index

CREATE INDEX IX\_Appointment\_Date ON Appointments (Appointment\_Date);

CREATE INDEX IX\_Patient\_LastName ON Patients (Patient\_LastName);

--views

CREATE VIEW AppointmentDetails AS

SELECT A.Appointment\_ID, A.Appointment\_Date, A.Appointment\_time,

P.Patient\_FirstName, P.Patient\_LastName, P.City,

T.Treatment, Py.Payment\_Method

FROM Appointments A

INNER JOIN Patients P ON A.Patient\_ID = P.Patient\_ID

INNER JOIN Patient\_Treatments PT ON A.PatientTreatment\_ID = PT.PatientTreatment\_ID

INNER JOIN Treatments T ON PT.Treatment\_ID = T.Treatment\_ID

INNER JOIN Payment Py ON PT.PatientTreatment\_ID = Py.PatientTreatment\_ID;

/\*if the object exists, rename and drop:

CREATE VIEW NewAppointmentDetails AS

SELECT A.Appointment\_ID, A.Appointment\_Date, A.Appointment\_time,

P.Patient\_FirstName, P.Patient\_LastName, P.City,

T.Treatment, Py.Payment\_Method

FROM Appointments A

INNER JOIN Patients P ON A.Patient\_ID = P.Patient\_ID

INNER JOIN Patient\_Treatments PT ON A.PatientTreatment\_ID = PT.PatientTreatment\_ID

INNER JOIN Treatments T ON PT.Treatment\_ID = T.Treatment\_ID

INNER JOIN Payment Py ON PT.PatientTreatment\_ID = Py.PatientTreatment\_ID;

DROP VIEW AppointmentDetails;

and thern continute to run the code

\*/

CREATE VIEW RoomUtilization AS

SELECT R.Room\_ID, R.RoomName, COUNT(RB.RoomBooking\_ID) AS Bookings

FROM Rooms R

LEFT JOIN Room\_Bookings RB ON R.Room\_ID = RB.Room\_ID

GROUP BY R.Room\_ID, R.RoomName;

/\*if the object exists, rename and drop:

CREATE VIEW NewRoomUtilization AS

SELECT R.Room\_ID, R.RoomName, COUNT(RB.RoomBooking\_ID) AS Bookings

FROM Rooms R

LEFT JOIN Room\_Bookings RB ON R.Room\_ID = RB.Room\_ID

GROUP BY R.Room\_ID, R.RoomName;

DROP VIEW RoomUtilization;

and thern continute to run the code

\*/

--constraints

ALTER TABLE Patients

ADD Age INT;

ALTER TABLE Patients

ADD CONSTRAINT CHK\_Age CHECK (Age >= 0 AND Age < 120);

ALTER TABLE Appointments

ADD CONSTRAINT FK\_Appointments\_Patients FOREIGN KEY (Patient\_ID) REFERENCES Patients(Patient\_ID);

# Login

You may Login to the database with the following created Login:

-- logins

CREATE LOGIN SampleLogin WITH PASSWORD = 'DentistStrongPassword';

CREATE USER SampleUser FOR LOGIN SampleLogin;

CREATE ROLE DataEntryRole;

CREATE ROLE DoctorRole;

CREATE ROLE AdministratorRole;

--assigning the permissions to the roles

GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Appointments TO DataEntryRole;

GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Patients TO DataEntryRole;

GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Appointments TO DoctorRole;

GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Patient\_Treatments TO DoctorRole;

GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Appointments TO AdministratorRole;

GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Patients TO AdministratorRole;

GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Patient\_Treatments TO AdministratorRole;

GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Payment TO AdministratorRole;

GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Treatments TO AdministratorRole;

GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Rooms TO AdministratorRole;

GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Room\_Bookings TO AdministratorRole;

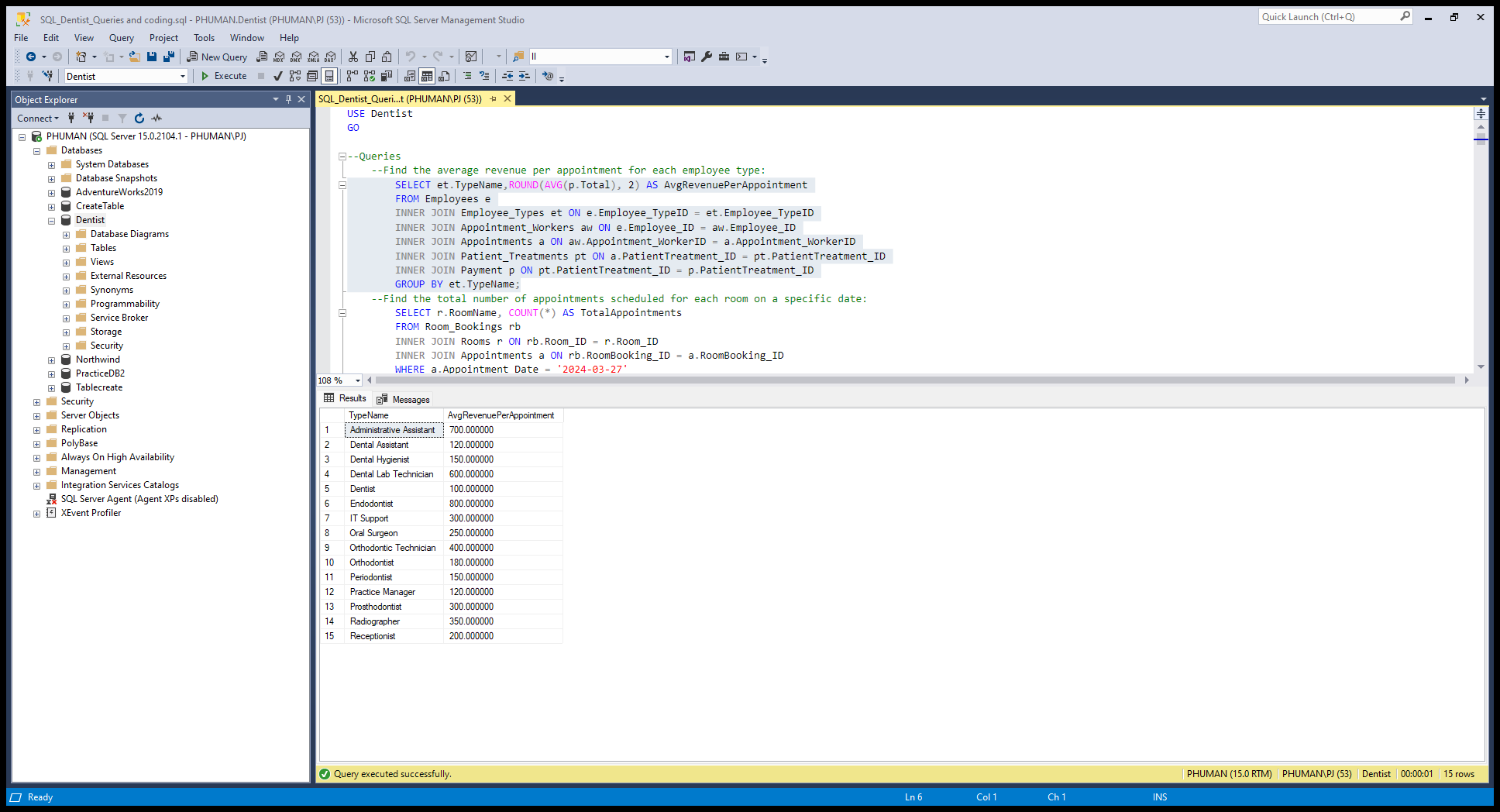
GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Employee\_Types TO AdministratorRole;

GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Employees TO AdministratorRole;

# Physical design – database environment, hardware, software ,security, performance requirements

# Questions that can be answered with the database.

1. Find the average revenue per appointment for each employee type.

SELECT et.TypeName,ROUND(AVG(p.Total), 2) AS AvgRevenuePerAppointment

FROM Employees e

INNER JOIN Employee\_Types et ON e.Employee\_TypeID = et.Employee\_TypeID

INNER JOIN Appointment\_Workers aw ON e.Employee\_ID = aw.Employee\_ID

INNER JOIN Appointments a ON aw.Appointment\_WorkerID = a.Appointment\_WorkerID

INNER JOIN Patient\_Treatments pt ON a.PatientTreatment\_ID = pt.PatientTreatment\_ID

INNER JOIN Payment p ON pt.PatientTreatment\_ID = p.PatientTreatment\_ID

GROUP BY et.TypeName;

1. Show all details about all appointments along with patient, employee, room, and treatment information.

SELECT A.Appointment\_ID, A.Appointment\_Date, A.Appointment\_time, P.Patient\_ID,P.Patient\_FirstName, P.Patient\_LastName, E.Employee\_ID, E.Employee\_FirstName, E.Employee\_LastName, R.Room\_ID, R.RoomName, T.Treatment\_ID, T.Treatment, T.Price

FROM Appointments A

INNER JOIN Patients P ON A.Patient\_ID = P.Patient\_ID

INNER JOIN Appointment\_Workers AW ON A.Appointment\_WorkerID = AW.Appointment\_WorkerID

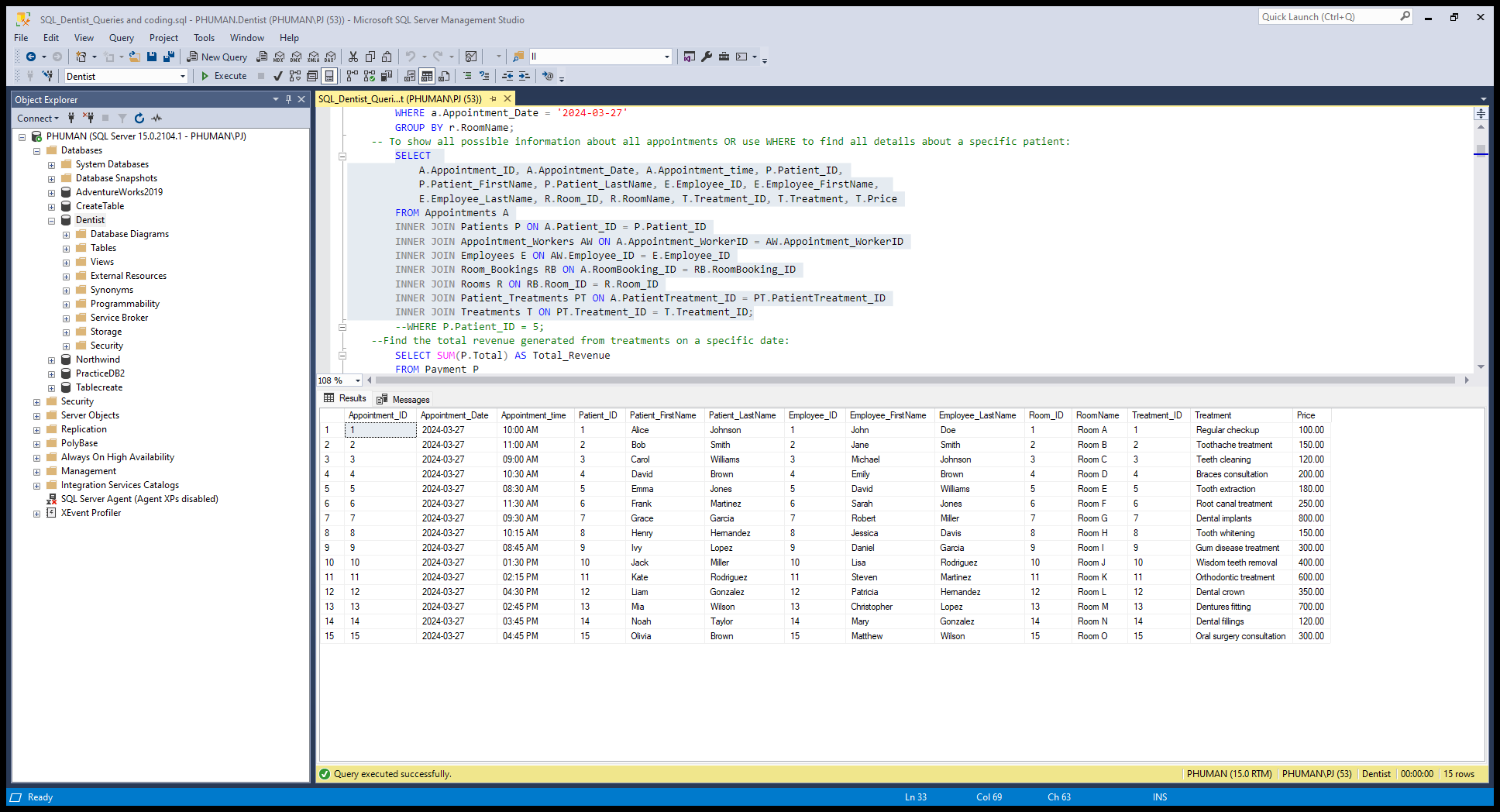
INNER JOIN Employees E ON AW.Employee\_ID = E.Employee\_ID

INNER JOIN Room\_Bookings RB ON A.RoomBooking\_ID = RB.RoomBooking\_ID

INNER JOIN Rooms R ON RB.Room\_ID = R.Room\_ID

INNER JOIN Patient\_Treatments PT ON A.PatientTreatment\_ID = PT.PatientTreatment\_ID

INNER JOIN Treatments T ON PT.Treatment\_ID = T.Treatment\_ID;



1. Stored Procedure: Update Patient Information.Write a stored method named UpdatePatientInfo that updates the records of a patient based on their affected person ID. The stored manner should accept the affected person ID, new first name, new last name, new email, and new phone quantity as parameters and replace the corresponding affected person's information inside the Patients desk.

CREATE PROCEDURE UpdatePatientInfo @PatientID INT,@NewCity VARCHAR(50)

AS

BEGIN

UPDATE Patients

SET City = @NewCity

WHERE Patient\_ID = @PatientID;

END;

-- Execute UpdatePatientInfo stored procedure

EXECUTE UpdatePatientInfo @PatientID = 123, @NewCity = 'New York';

-- Replace 123 with the actual Patient\_ID and 'New York' with the new city value

GO