Assessment

Advanced Databases

M.Sc Data Science

Database Design & SQL for Data Analysis

Task 1: Hospital GP Management Database Design Proposal and Implementation in Microsoft SQL Server Management Studio using T-SQL

Task 2: Database Creation, Schema Design, and Query Implementation for Food Service Company in Microsoft SQL Server Management Studio using T-SQL

Name : Remya Vellakkadaparambu Karthikeyan

Student ID: 00715519

Date: 24 th April 2024

Contents

Task 1 Part 1: Hospital GP Management Database Design Proposal and Implementation in Microsoft SQL Server Management Studio using T-SQL

1.1 Introduction

General Practice (GP) can be considered as the initial point of contact of a patient with a hospital unless it is an emergency. GP plays a significant role in providing continuing care to the patients registered with them. To improve the ease and quality of GP provided to the patients, the client hospital is taking the initiative to develop a browser-based GP management which is supported by the patient portal (Poon et al., 2003). The online GP management application aims to improve the experience of the patients in aspects such as booking, canceling, rebooking, and giving feedback. The patient portal linked with the GP registration also permits storing the patient details and updating them whenever required. Even though the patient leaves the hospital service, the hospital plans to retain the data collected.

The client hospital (hereafter mentioned as 'client' or 'hospital' or 'stakeholder') identifies patients, doctors, medical records, appointments, and departments as the primary entities in the database. The client hospital describes the entity 'patient' (also mentioned as 'user' or 'application user' interchangeably within this report) including the attributes name, address, date of birth, insurance, username, password, date left, email address, and telephone number. Similarly, the few attributes for other entities are also given by the client. Figure 1.1 illustrates the conceptual Entity-Relationship (E-R) diagram developed from the description given by the client (Elmasri & Navathe, 2016). Apart from the entities and related attributes, the database developing team is given the liberty to add entities as well as attributes under each entity, if required.

The database development team proposes an **Online Transaction Processing** (**OLTP**) database design as it involves managing and processing transactions such as patient registration, appointments, patient record

updation, medical prescription entry, patient feedback recording, etc (Elmasri & Navathe, 2016).

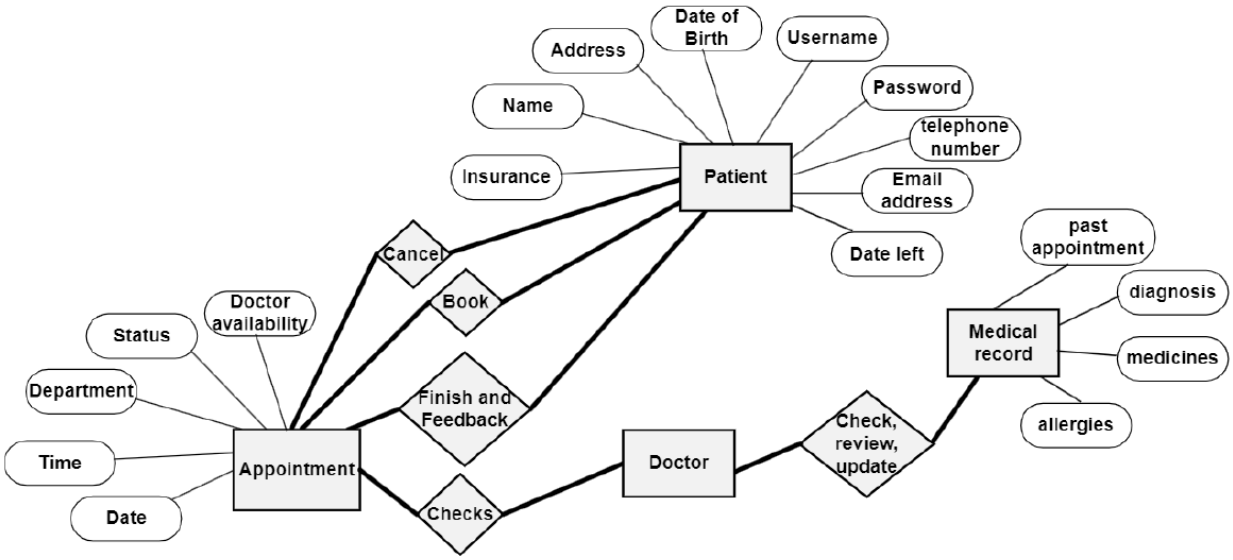


Figure 1.1 Conceptual E-R diagram

For realizing efficient and reliable GP registration applications and patient portals, the database design takes into account the client requirements, and the same is implemented using T-SQL statements in Microsoft SQL Server Management Studio. The database is also intended to support the doctors in the GP by providing the medical history of the patients which helps in quick diagnosis of the disease and medical prescription. The report outlines the database design process, algorithm, and its implementation in Microsoft SQL server management studio using T-SQL statements.

1.1.1 Database Design Proposal Overview

The client hospital aims at developing a user application program or system based on a database system as shown in Figure 1.2. For reliable, flexible, and functional transactions through the application program, it is necessary to have a well-developed database. The goal of OLTP database design for hospital GP management revolves around constructing an appropriate database schema for patient registration, appointments, patient details storage, medical history

storage, user feedback entry, etc according to the client requirements, establishing a database, and the GP registration application, so that the entire system can effectively collect or define, store, maintain, process and control or manage data to meet the user requirements (Huawei Technologies Co., Ltd., 2023).

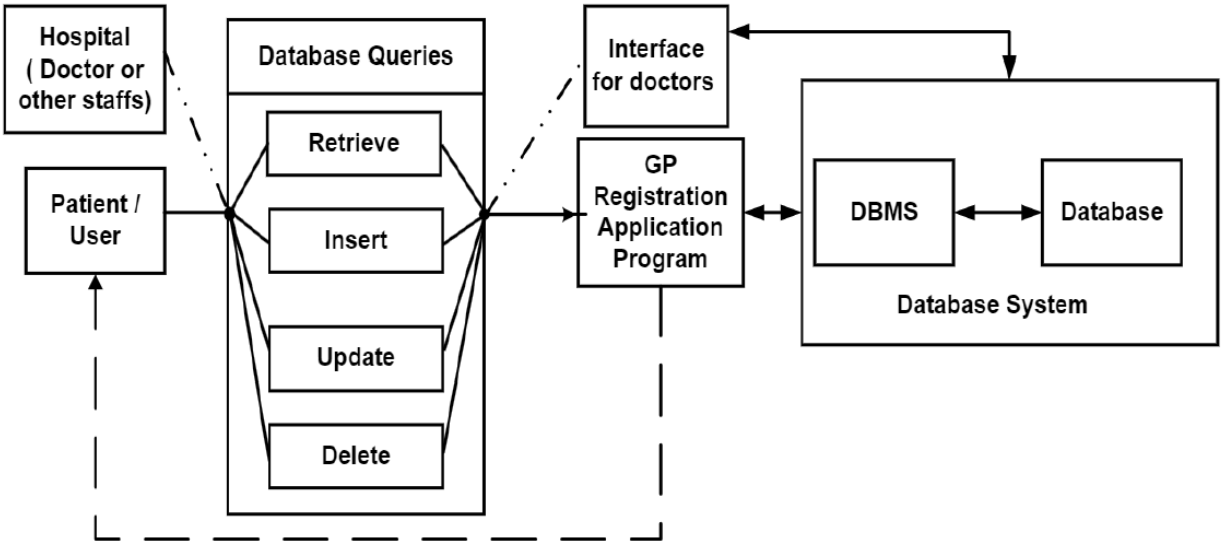


Figure 1.2 GP registration application program based on OLTP database.

1.1.2 OLTP Database Design Approach Overview

As the hospital GP management database design is intended to showcase efficient data access, data storage, database system operation, and management, the hybrid approach - the most effective approach (Elmasri & Navathe, 2016; Huawei Technologies Co., Ltd., 2023) is recommended. The hybrid approach blends the top-down approach based on the E-R diagram and the bottom-up approach based on functional dependency and normalization respectively. The flowchart given in Figure 1.3 depicts the database design flow proposed for the hospital GP management database design.

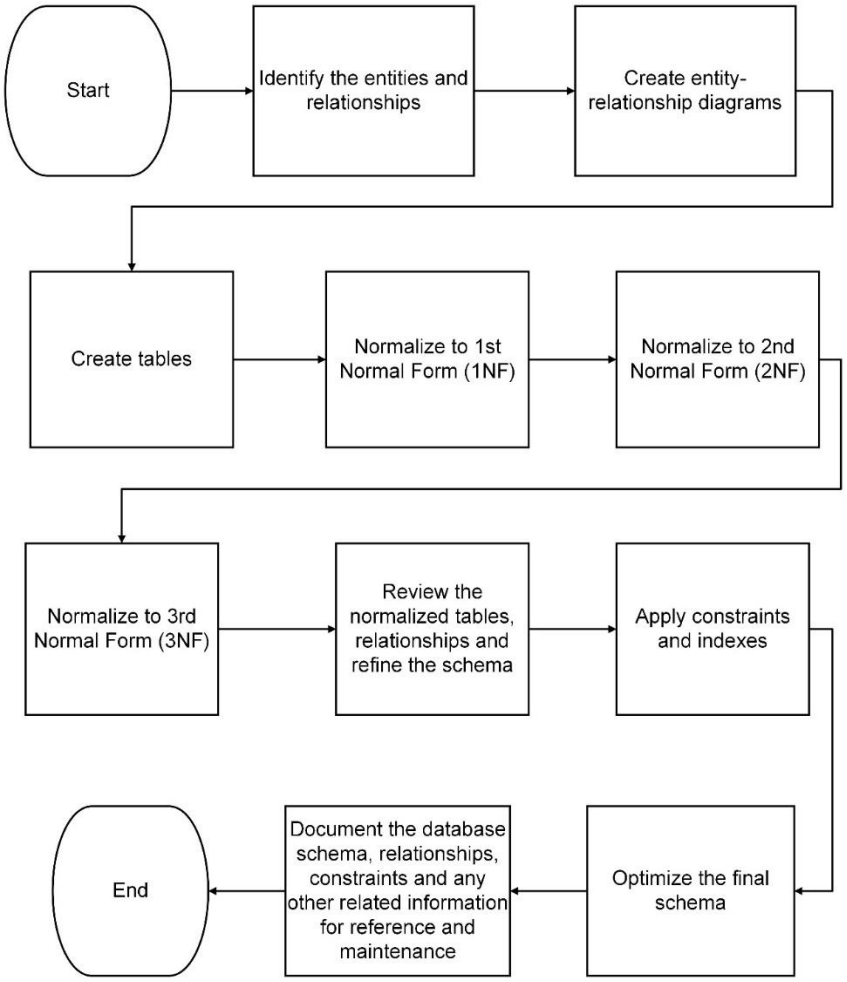


Figure 1.3 Database design approach flowchart

Part 1: Designing and Normalizing a Database into 3NF: A Comprehensive Approach with T-SQL Implementation in Microsoft SQL Server Management Studio

1.2 Identifying the entities and relationships

From the thorough study of the client requirements, the entities identified are given in Table 1.1 with their attributes.

Table 1.1 Identification of entities, attributes, and additional attributes

Entity	**Attributes**	**Additional attributes added by the database design team**
Patient	Full name, address, date of birth, insurance, username, password, email (optional), telephone number (optional), date_left	Patient id, Gender Blood group
Appointment	Date, time, department, status (pending, cancelled , completed)	Appointment id **Department** → [specialization, building name, telephone number] **Status** → [booked, available]
Doctor	Doctor's availability	Doctor id, name, sub-specialization, room no
Medical Record	Past appointment, diagnosis, medicines, allergies	Medical record id
Review/ Feedback		Review id, review text, rating

Table 1.1 Identification of entities, attributes, and additional attributes

The relationships between the entities can be described as :

- **Patient registers with the Hospital**

  The patient provides their information to the hospital system for registration.
- **Patient books an Appointment**

  The patient can book multiple appointments.
- **Appointment is assigned to a Doctor**

  Each appointment is assigned to only one doctor.
- **Doctor reviews Patient's Medical Record**

  Each doctor can review multiple medical records.
- **Medical Record is associated with an Appointment**

  Each medical record corresponds to one appointment.
- **Patient provides Review/Feedback for Doctor**

  Each patient can provide multiple reviews/feedback.
- **Patient cancels/finishes Appointment**

  Each appointment can be associated with only one patient.
- **Hospital retains Patient Information**

  The hospital retains patient information even after the patient leaves.

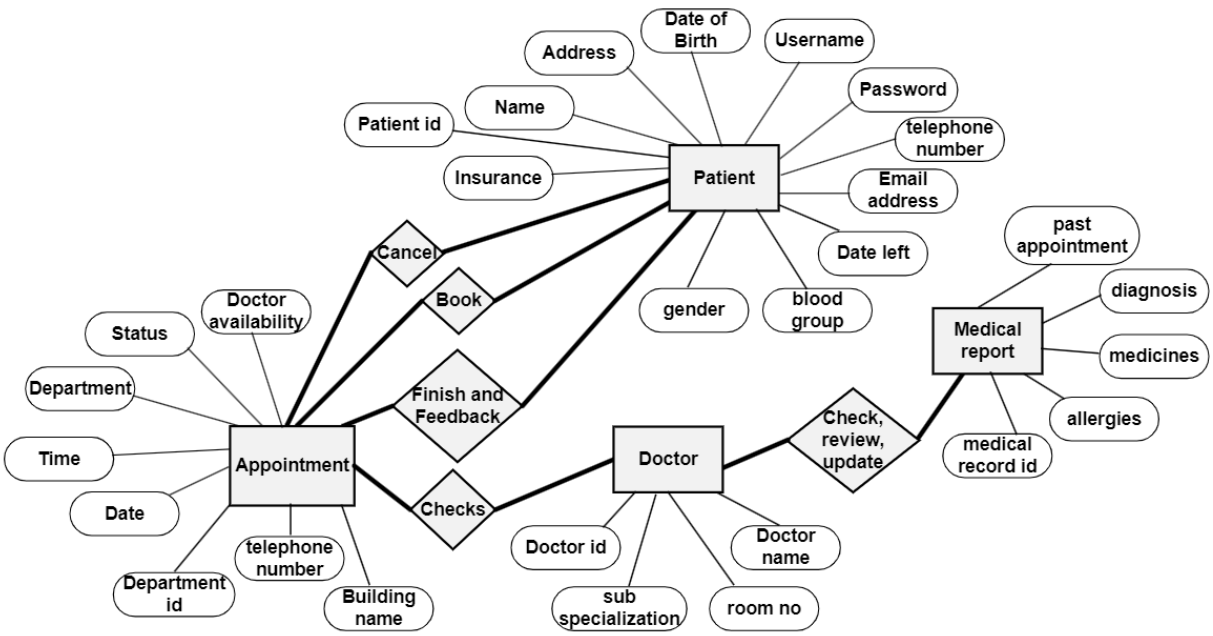


Figure 1.4 Updated E-R diagram with all attributes

1.3 Creating entity-relationship diagrams with cardinality

In the database design, the hospital is not taken in as an entity, as the design is focused on GP management. The E-R diagram with cardinality can be depicted as given in Figure 1.5

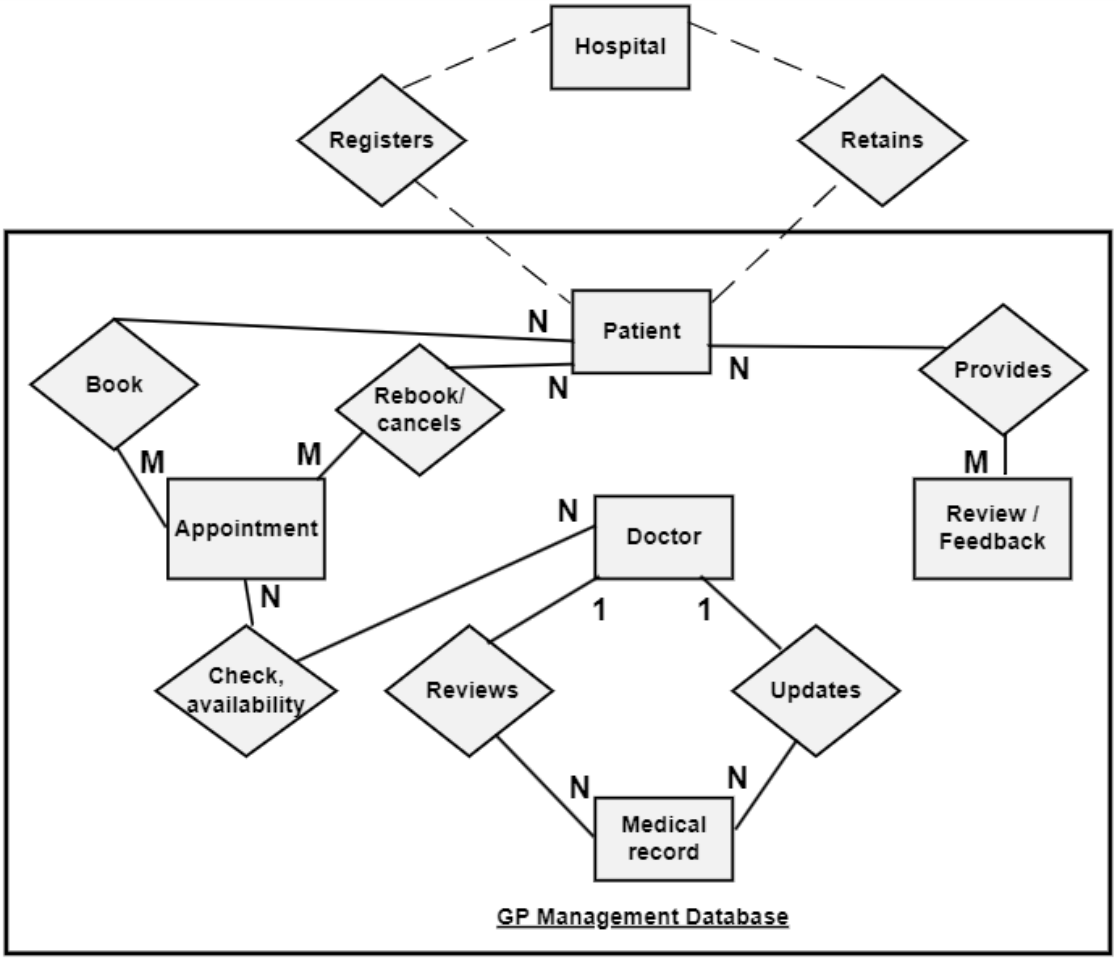


Figure 1.5 E-R diagram with cardinality

1.4 Creating tables from entity-relationship diagram

The first step is to convert the relationships into tables. The primary key and foreign keys of the tables from the E-R diagram are clearly shown in the illustration of the tables below.

Table: Patient

Patient_id (primary key)	**Name** (composite)	**Address** (multi-valued)	**DOB**	**Date left**	**Insurance**	**gender**
Blood group	**Username**	**Password**	**Email address**	**Telephone number**		

The attribute **Name** is separated into first name, middle name, and last name.

The attribute **Address** is multi-valued and is stored in a separate table and foreign key **address_id** is included in the Patient table.

Table: Address

Address_id (primary key)	Address1	Address2	City	County	Postcode	Patient_id

Foreign key **patient_id**, shows patient-address relationship

Table: Appointment

Appointment_id (primary key)	**date**	**time**	**status**	**Doctor and Doctor availability** (multi-valued)	**Department** (multi-valued)

The attributes **Department, Doctor, DoctorSchedule** are multi-valued and are stored in separate tables.

Table: Department

Department_id (primary key)	Building name	Specialization	Telephone number

Table: Doctor

Doctor_id (primary key)	Doctor name (composite)	Sub-specialization	Room no	Department_id (foreign key)

Foreign key **department_id**, shows doctor-department relationship

The attribute Doctor **Name** is separated into first name, middle name and last name.

Table: DoctorSchedule

Schedule_id (primary key)	Day of the week	Bookings remaining	Start time	End time	Doctor_id (foreign key)

Foreign key **doctor_id**, shows doctor-doctor schedule relationship

Table: Medical Record

Foreign key **patient_id**, shows record-patient relationship

Medical record_id (primary key)	Past appointment (multi valued)	Diagnosis	Medicine	Allergy	Patient_id

Table: ReviewFeedback

review_id (primary key)	Day of the week	Bookings remaining	Start time	End time	Doctor_id (Foreign key)

Foreign key **doctor_id**, shows review-doctor relationship

1.5 Normalizing the tables to First Normal Form (1NF)

The attribute Fullname in the **Patient** table is decomposed to first name, middle name, and last name to ensure atomicity. The **Address** table is split from the Patient table to remove multi-valued attributes from the latter and **Address** table formation can be explained in terms of normalization to 1NF as well.

Table: Patient to 1NF

Patient_id (primary key)	**First Name**	**Middle Name**	**Last Name**	**DOB**	**Date left**	**Insurance**	**gender**
Blood group	**Username**	**Password**	**Email address**	**Telephone number**			

1.6 Normalizing the tables to Second Normal Form (2NF)

The attribute username in the **Patient** table is unique for each patient and it violates the condition that all non-prime keys should be fully dependent on the primary key. A new table **PatientPortalInfo** is created as shown below.

Username (primary key)	**Password**	**Email address**	**Telephone number**	**Patient_id** (Foreign key)

Foreign key **patient_id**, shows patient portal info-patient relationship

Table: Medical Record

Medical record_id (primary key)	Past appointment (multi-valued)	Diagnosis	Medicine	Allergy	Patient_id (foreign key)

The attribute past appointment in the **Medical record** table is not fully dependent on the **medical record_id** (primary key) as an appointment id is generated for all patients in the Appointment table. To remove the violation of full dependency on the primary key, the **PastAppointment** table is created. The table takes all the columns from the Appointment table created above and ensures the records correspond to the past based on the system date and time.

Table: PastAppointment

PastAppointment_id (primary key)	**Patient_id** (foreign key)	**date**	**time**	**status**	**Doctor_id** (foreign key)	**Department_id** (foreign key)

Foreign keys **patient_id**, **doctor_id**, and **department_id** show past appointment relationships with the patient, doctor, and department respectively.

The **Medicalrecord** table takes Past appointment_id as the foreign key to relate to the past appointment.

Table: Medicalrecord

Medical record_id (primary key)	Past appointment_id (foreign key)	Diagnosis	Medicine	Allergy	Patient_id (foreign key)

Foreign key **pastappointment_id** shows medical record-past appointment relationships.

As the **PastAppointment** table is exclusively created for past appointments, the Appointment table is renamed to **CurrentAppointment**.

Table: CurrentAppointment

CurrentAppointment_id (primary key)	Patient_id (foreign key)	date	time	status	Doctor_id (foreign key)	Department_id (foreign key)

1.7 Normalizing the tables to Third Normal Form (3NF)

In the **Medicalrecord** table, the diagnosis, medicine, and allergy are transitively dependent on the primary key through the past appointment id. The table violates the condition for 3NF and leads to splitting the table into three different tables namely **Diagnosis**, **Medicine**, and **Allergy**. Other foreign keys are also included to maintain the relationship that the diagnosis, medicine, or allergy is having with the patient, doctor, or both. Instead of making the determinant of transitive dependency the primary key of the new table, **record_id** is added as the foreign key to the **Diagnosis**, **Medicine,** and **Allergy** table, considering there will be multiple diagnoses, medicine, allergies for a single patient based on different departments they take appointments with.

Table: Diagnosis

Diagnosis_id (primary key)	Record_id (foreign key)	Diagnosis	Doctor_id (foreign key)

Table: Medicine

medicine_id (primary key)	Record_id (foreign key)	Medicine name	Doctor_id (foreign key)

Table: Allergy

allergy_id (primary key)	Record_id (foreign key)	allergy	Doctor_id (foreign key)	Patient_id (foreign key)

There is an assumption taken that the diagnosis, or medicine data will be inserted into the database by the respective departments based on the doctors' appointments. The relationship of the diagnosis, medicine, and patient can be deduced by joining the tables. However, a doctor needs to identify the allergies of the patient before giving a prescription, and this leads to the inclusion of patient_id in the allergy table.

1.8 Review of Normalized Tables, Relationships, and Refinement of Schema

The review of the normalized tables and relationships and refinement of the schema are done separately in sections 1.8.1 and 1.8.2 respectively.

1.8.1 Review of Normalized Tables and Relationships

In the hospital GP management database design, there are 13 tables in total. Each table in the database has a primary key that uniquely identifies each record within the table. It is through primary keys; that the tables satisfy the First Normal Form. The atomicity requirement of the table is fulfilled by the unique identifier or primary key in each table.

The partial dependencies brought by composite or non-prime keys are removed using the foreign keys. Foreign keys establish relationships between tables by referencing primary keys from other tables.

By enforcing primary and foreign key constraints, the database schema ensures that each table is in the Third Normal Form (3NF) by eliminating transitive dependencies and maintaining relationships between entities through proper normalization. This design promotes data integrity, efficiency, and scalability in managing patient and medical information.

Table 1.2 reviews the normalized tables in the hospital GP management database.

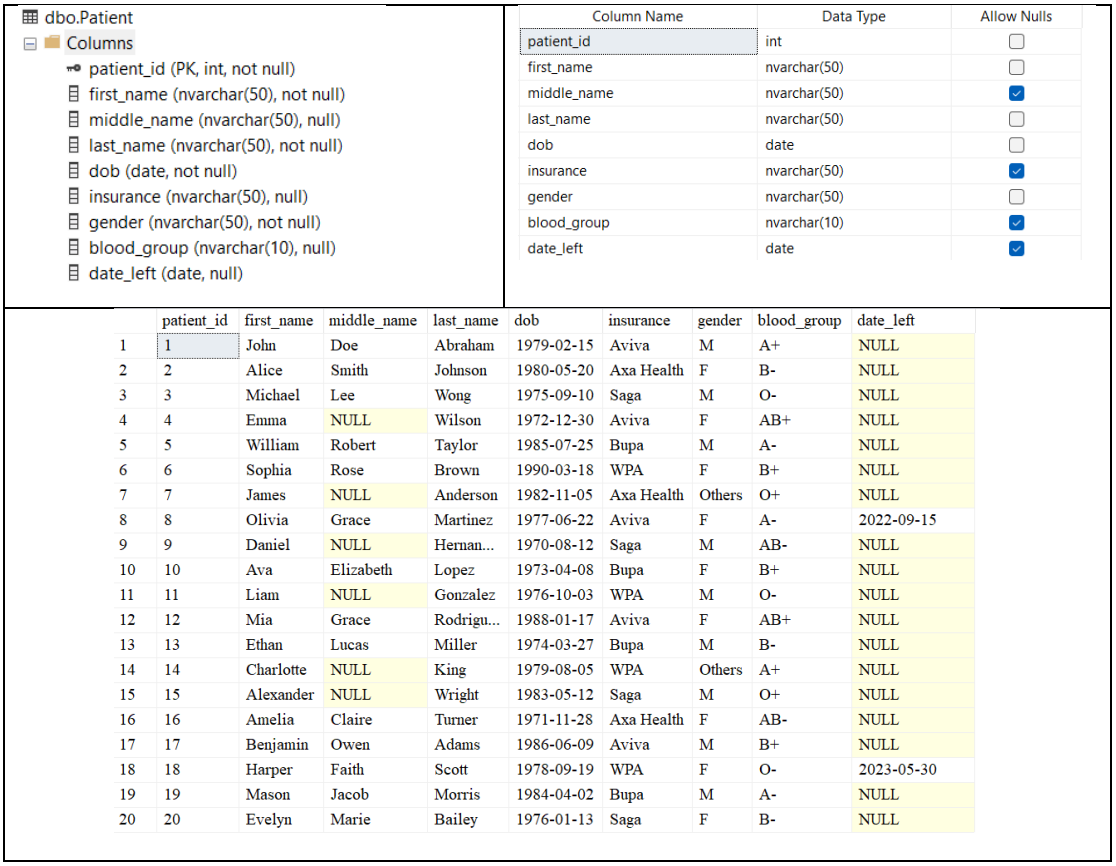
Refer to Appendix A for the SQL codes creating the tables.

Table 1.2 Review of normalized tables and relationship references

Sl No	Table	Primary keys	Foreign Keys	Relationship references from tables
1	**Patient**	patient_id	Nil	Nil
2	**Address**	address_id	patient_id	**Patient** (patient_id))
3	**PatientPortalInfo**	username	patient_id	**Patient** (patient_id))
4	**Department**	department_id	Nil	Nil
5	**Doctor**	doctor_id	department_id	**Department** (department_id)
6	**DoctorSchedule**	schedule_id	doctor_id	**Doctor**(doctor_id)
7	**PastAppointment**	pastappointment_id	patient_id, doctor_id, department_id	**Patient**(patient_id), **Doctor**(doctor_id), **Department**(department_id)
8	**MedicalRecord**	record_id	patient_id	**Patient**(patient_id)
9	**Diagnosis**	diagnosis_id	record_id, doctor_id	**MedicalRecord** (record_id), **Doctor** (doctor_id)
10	**Medicine**	medicine_id	record_id, doctor_id	**MedicalRecord** (record_id), **Doctor** (doctor_id)
11	**Allergy**	allergy_id	record_id, patient_id, doctor_id	**MedicalRecord** (record_id), **Patient** (patient_id), **Doctor** (doctor_id))
12	**ReviewFeedback**	review_id	patient_id, doctor_id, pastappointment_id	**Patient** (patient_id), **Doctor** (doctor_id)) **PastAppointment** (pastappointment_id)
13	**CurrentAppointment**	currentappointment_id	patient_id, doctor_id, department_id	**Patient** (patient_id), **Doctor** (doctor_id), **Department** (department_id))

1.8.2 Refinement of Schema

The refined schema for the GP management database design is shown in Table 1.3 under the default schema dbo. Refer to section 1.13.3 for creating specific schemas for database security. The schema can be viewed in either way as given in Figure 1.6 for table **Patient.**



Column Name	Data Type	Allow Nulls
patient_id	int	☐
first_name	nvarchar(50)	☐
middle_name	nvarchar(50)	☑
last_name	nvarchar(50)	☐
dob	date	☐
insurance	nvarchar(50)	☑
gender	nvarchar(50)	☐
blood_group	nvarchar(10)	☑
date_left	date	☑

	patient_id	first_name	middle_name	last_name	dob	insurance	gender	blood_group	date_left
1	1	John	Doe	Abraham	1979-02-15	Aviva	M	A+	NULL
2	2	Alice	Smith	Johnson	1980-05-20	Axa Health	F	B-	NULL
3	3	Michael	Lee	Wong	1975-09-10	Saga	M	O-	NULL
4	4	Emma	NULL	Wilson	1972-12-30	Aviva	F	AB+	NULL
5	5	William	Robert	Taylor	1985-07-25	Bupa	M	A-	NULL
6	6	Sophia	Rose	Brown	1990-03-18	WPA	F	B+	NULL
7	7	James	NULL	Anderson	1982-11-05	Axa Health	Others	O+	NULL
8	8	Olivia	Grace	Martinez	1977-06-22	Aviva	F	A-	2022-09-15
9	9	Daniel	NULL	Hernan...	1970-08-12	Saga	M	AB-	NULL
10	10	Ava	Elizabeth	Lopez	1973-04-08	Bupa	F	B+	NULL
11	11	Liam	NULL	Gonzalez	1976-10-03	WPA	M	O-	NULL
12	12	Mia	Grace	Rodrigu...	1988-01-17	Aviva	F	AB+	NULL
13	13	Ethan	Lucas	Miller	1974-03-27	Bupa	M	B-	NULL
14	14	Charlotte	NULL	King	1979-08-05	WPA	Others	A+	NULL
15	15	Alexander	NULL	Wright	1983-05-12	Saga	M	O+	NULL
16	16	Amelia	Claire	Turner	1971-11-28	Axa Health	F	AB-	NULL
17	17	Benjamin	Owen	Adams	1986-06-09	Aviva	M	B+	NULL
18	18	Harper	Faith	Scott	1978-09-19	WPA	F	O-	2023-05-30
19	19	Mason	Jacob	Morris	1984-04-02	Bupa	M	A-	NULL
20	20	Evelyn	Marie	Bailey	1976-01-13	Saga	F	B-	NULL

Figure 1.6 Patient table schema and sample data

Table 1.3 Refined table-wise schema and initial data populated

Table-wise schema	Sample data
Address dbo.Address Columns address_id (PK, int, not null) address1 (nvarchar(50), not null) address2 (nvarchar(50), null) city (nvarchar(50), null) county (nvarchar(50), null) postcode (nvarchar(10), not null) patient_id (FK, int, null)	

	address_id	address1	address2	city	county	postcode	patient_id
1	1	123 Main Street	NULL	London	Greater London	E1 6AN	1
2	2	456 Elm Street	Apt 101	Manc...	Greater Manc...	M1 1AB	2
3	3	789 Oak Street	NULL	Birm...	West Midlands	B1 2CD	3
4	4	101 Pine Street	NULL	Leeds	West Yorkshire	LS1 1...	4
5	5	111 Maple Str...	Suite 2...	Glas...	Glasgow City	G1 1YZ	5
6	6	222 Cedar Str...	NULL	Liver...	Merseyside	L1 1ZW	6
7	7	333 Birch Str...	NULL	Bristol	Bristol	BS1 1...	7
8	8	444 Walnut St...	Flat 3B	Sheff...	South Yorkshire	S1 1WB	8
9	9	555 Cherry St...	NULL	Edin...	City of Edinb...	EH1 1...	9
10	10	666 Willow S...	Unit 5	New...	Tyne and Wear	NE1 1...	10
11	11	777 Spruce St...	NULL	Notti...	Nottinghamsh...	NG1 1...	11
12	12	888 Oakwood...	NULL	Leice...	Leicestershire	LE1 1...	12
13	13	999 Ash Street	Floor 2	Brigh...	East Sussex	BN1 1...	13
14	14	123 Birchwo...	NULL	Cam...	Cambridgeshi...	CB1 1...	14
15	15	456 Beechwo...	NULL	Oxford	Oxfordshire	OX1 1...	15
16	16	789 Elmwood...	Room 10	York	North Yorkshi...	YO1 1...	16
17	17	101 Cedarwo...	NULL	Cardiff	Cardiff	CF1 1...	17
18	18	111 Willoww...	Suite 1...	Belfast	Belfast	BT1 1...	18
19	19	222 Pinecrest ...	NULL	Dublin	Dublin	D1 1WT	19
20	20	333 Maplewo...	NULL	Aber...	Aberdeenshire	AB1 1...	20

PatientPortalInfo

- dbo.PatientPortalInfo
  - Columns
    - username (PK, nvarchar(50), not null)
    - patient_id (FK, int, null)
    - pass_word (nvarchar(50), null)
    - email_address (nvarchar(100), null)
    - telephone_number (nvarchar(20), null)

	username	patient_id	pass_word	email_address	telephone_number
1	AJohnson	2	[illegible]	AJohnson@example.com	NULL
2	ALopez	10	[illegible]	ALopez@example.com	07000000710
3	ATurner	16	[illegible]	ATurner@example.com	07000000716
4	AWright	15	[illegible]	AWright@example.com	07000000715
5	BAdams	17	[illegible]	BAdams@example.com	07000000717
6	CKing	14	[illegible]	CKing@example.com	NULL
7	DHernandez	9	[illegible]	DHernandez@example.com	07000000079
8	EBailey	20	[illegible]	EBailey@example.com	07000000720
9	EMiller	13	[illegible]	EMiller@example.com	07000000713
10	EWilson	4	[illegible]	EWilson@example.com	07000000074
11	HScott	18	[illegible]	HScott@example.com	07000000718
12	JAbraham	1	[illegible]	NULL	07000000071
13	JAnderson	7	[illegible]	JAnderson@example.com	NULL
14	LGonzalez	11	[illegible]	LGonzalez@example.com	07000000711
15	MMorris	19	[illegible]	MMorris@example.com	NULL
16	MRodriguez	12	[illegible]	MRodriguez@example.com	07000000712
17	MWong	3	[illegible]	MWong@example.com	07000000073
18	OMartinez	8	[illegible]	OMartinez@example.com	07000000078
19	SBrown	6	[illegible]	SBrown@example.com	07000000076
20	WTaylor	5	[illegible]	WTaylor@example.com	07000000075

Department

- dbo.Department
  - Columns
    - specialization (nvarchar(100), null)
    - building_name (nvarchar(50), null)
    - telephone_no (varchar(20), null)
    - department_id (PK, tinyint, not null)

	specialization	building_name	telephone_no	department_id
1	Oncology	Main Building	123-456-7890	16
2	Gastroenterology	East Wing	234-567-8901	17
3	Cardiology	West Wing	345-678-9012	18
4	Neurology	North Wing	456-789-0123	19
5	Orthopedics	South Wing	567-890-1234	20
6	Pediatrics	Children's Hospital	678-901-2345	21
7	Obstetrics and Gynecology	Specialty Clinic	789-012-3456	22
8	Pulmonology	Clinic Building	890-123-4567	23
9	Dermatology	Dermatology Cen...	901-234-5678	24
10	Ophthalmology	Eye Center	012-345-6789	25
11	Urology	Urology Clinic	123-456-7890	26
12	ENT	ENT Clinic	234-567-8901	27
13	Psychiatry	Psychiatry Center	345-678-9012	28
14	Radiology	Radiology Depar...	456-789-0123	29
15	Anesthesiology	Anesthesiology U...	567-890-1234	30

Doctor

(76 doctor details populated, remaining details in Appendix B)

- dbo.Doctor
  - Columns
    - doctor_id (PK, int, not null)
    - first_name (nvarchar(50), not null)
    - middle_name (nvarchar(50), null)
    - last_name (nvarchar(50), not null)
    - sub_specialization (nvarchar(100), null)
    - room_no (nvarchar(10), null)
    - department_id (FK, tinyint, null)

	doctor_id	first_name	middle_name	last_name	sub_specialization	room_no	department_id
1	1	John	Smith	Ava	medical oncology	101	16
2	2	Alice	Daniel	Johnson	radiation oncology	102	16
3	3	Michael	Long	Lee	surgical oncology	103	16
4	4	Emma	Adams	Wilson	surgical oncology	104	16
5	5	William	Adams	Taylor	radiation oncology	105	16
6	6	Sophia	Addison	Brown	Gastrointestinal ...	231	17
7	7	James	Adrian	Martinez	Pancreatology	232	17
8	8	Olivia	Alexander	Hernan...	Gastrointestinal ...	233	17
9	9	Daniel	Alice	Lopez	Hepatology	234	17
10	10	Ava	Allen	Gonzalez	Gastrointestinal ...	235	17
11	11	Ethan	Allison	Walker	Interventional C...	502	18
12	12	Mia	Amelia	Perez	Electrophysiology	503	18
13	13	Benjamin	Amelia	Hall	Electrophysiology	504	18
14	14	Isabella	Andrews	Young	Heart Failure	505	18
15	15	Jacob	Aria	Allen	Heart Failure	506	18
16	16	Amelia	Aubrey	Lewis	Clinical Neurop...	100	19
17	17	Alexander	Aurora	King	Neurocritical Care	101	19
18	18	Charlotte	Avery	Wright	Neuroimmunology	102	19
19	19	William	Baker	Hill	Clinical Neurop...	103	19
20	20	Sophia	Barnes	Green	Neurocritical Care	104	19
21	21	Matthew	Bell	Adams	Sports Medicine	367	20
22	22	Emily	Benjamin	Russell	Joint Replaceme...	368	20
23	23	Daniel	Blackburn	Hughes	Joint Replaceme...	369	20

DoctorSchedule

(90 doctor schedule details populated, remaining details in Appendix B)

- dbo.DoctorSchedule
  - Columns
    - schedule_id (PK, int, not null)
    - doctor_id (FK, int, null)
    - day_of_week (nvarchar(10), null)
    - bookings_remaining (tinyint, null)
    - start_time (time(7), null)
    - end_time (time(7), null)

	schedule_id	doctor_id	day_of_week	bookings_remaining	start_time	end_time
1	1	1	Monday	10	09:00:00.0000000	12:00:00.0000000
2	2	1	Tuesday	8	08:30:00.0000000	11:30:00.0000000
3	3	2	Monday	12	10:00:00.0000000	13:00:00.0000000
4	4	2	Tuesday	7	09:30:00.0000000	12:30:00.0000000
5	5	3	Wednesday	9	08:00:00.0000000	11:00:00.0000000
6	6	3	Thursday	11	10:30:00.0000000	13:30:00.0000000
7	7	4	Thursday	6	08:30:00.0000000	11:30:00.0000000
8	8	4	Friday	14	09:00:00.0000000	12:00:00.0000000
9	9	5	Friday	5	11:00:00.0000000	14:00:00.0000000
10	10	5	Monday	13	09:30:00.0000000	12:30:00.0000000
11	11	6	Tuesday	8	10:00:00.0000000	13:00:00.0000000
12	12	6	Wednesday	10	08:00:00.0000000	11:00:00.0000000
13	13	7	Thursday	9	09:30:00.0000000	12:30:00.0000000
14	14	7	Friday	11	10:30:00.0000000	13:30:00.0000000
15	15	8	Monday	7	08:30:00.0000000	11:30:00.0000000
16	16	8	Tuesday	13	09:00:00.0000000	12:00:00.0000000
17	17	9	Wednesday	6	10:00:00.0000000	13:00:00.0000000
18	18	9	Thursday	12	08:00:00.0000000	11:00:00.0000000
19	19	10	Friday	8	09:00:00.0000000	12:00:00.0000000
20	20	10	Monday	10	10:30:00.0000000	13:30:00.0000000
21	21	11	Tuesday	9	08:30:00.0000000	11:30:00.0000000
22	22	11	Wednesday	7	10:00:00.0000000	13:00:00.0000000
23	23	12	Thursday	11	08:00:00.0000000	11:00:00.0000000

PastAppointment

- dbo.PastAppointment
  - Columns
    - pastappointment_id (PK, int, not null)
    - patient_id (FK, int, null)
    - date (date, not null)
    - time (time(7), not null)
    - department_id (FK, tinyint, null)
    - doctor_id (FK, int, null)
    - status (nvarchar(20), null)

	pastappointment_id	patient_id	date	time	department_id	doctor_id	status
1	1	1	2023-09-04	10:00:00.0000000	16	1	completed
2	2	19	2023-09-05	10:00:00.0000000	17	6	completed
3	3	2	2023-10-10	09:00:00.0000000	18	11	completed
4	4	13	2023-10-16	09:00:00.0000000	17	8	completed
5	5	15	2023-10-23	10:00:00.0000000	16	2	completed
6	6	1	2023-11-07	10:00:00.0000000	16	2	completed
7	7	9	2023-11-14	11:00:00.0000000	19	16	completed
8	8	7	2023-11-15	10:00:00.0000000	19	19	completed
9	9	15	2023-12-07	09:00:00.0000000	18	14	completed
10	10	1	2023-12-12	09:00:00.0000000	16	1	completed
11	11	15	2023-12-18	11:00:00.0000000	16	2	completed
12	12	6	2023-12-20	10:00:00.0000000	18	11	completed
13	13	3	2023-12-27	08:00:00.0000000	17	6	completed
14	14	5	2024-01-03	11:00:00.0000000	19	16	completed
15	15	15	2024-01-08	10:00:00.0000000	16	1	completed
16	16	10	2024-01-11	10:00:00.0000000	18	14	completed
17	17	4	2024-02-05	09:00:00.0000000	17	8	completed
18	18	10	2024-02-07	11:00:00.0000000	19	16	completed
19	19	17	2024-02-09	11:00:00.0000000	20	25	completed
20	20	3	2024-03-01	09:00:00.0000000	17	10	completed

MedicalRecord

- dbo.MedicalRecord
  - Columns
    - record_id (PK, int, not null)
    - patient_id (FK, int, null)
    - pastappointment_id (FK, int, null)

	record_id	patient_id	pastappointment_id
1	1	1	1
2	2	19	2
3	3	2	3
4	4	13	4
5	5	15	5
6	6	1	6
7	7	9	7
8	8	7	8
9	9	15	9
10	10	1	10
11	11	15	11
12	12	6	12
13	13	3	13
14	14	5	14
15	15	15	15
16	16	10	16
17	17	4	17
18	18	10	18
19	19	17	19
20	20	3	20

Diagnosis

- dbo.Diagnosis
  - Columns
    - diagnosis_id (PK, int, not null)
    - record_id (FK, int, null)
    - diagnosis (nvarchar(100), null)
    - doctor_id (FK, int, null)

	diagnosis_id	record_id	diagnosis	doctor_id
1	1	1	Cancer	1
2	2	2	Abdominal pain	6
3	3	3	Chest pain	11
4	4	4	Heartburn	8
5	5	5	Cancer	2
6	6	6	Cancer	2
7	7	7	Balance problem	16
8	8	8	Slurred speech	19
9	9	9	Heart problem	14
10	10	10	Cancer	1
11	11	11	Cancer	2
12	12	12	Heart problem	11
13	13	13	Nausea	6
14	14	14	Balance problem	16
15	15	15	Cancer	1
16	16	16	Chest pain	14
17	17	17	Abdominal pain	8
18	18	18	Slurred speech	16
19	19	19	Joint pain	25
20	20	20	Nausea	10

Medicine

- dbo.Medicine
  - Columns
    - medicine_id (PK, int, not null)
    - record_id (FK, int, null)
    - medicine_name (nvarchar(100), null)
    - doctor_id (FK, int, null)

	medicine_id	record_id	medicine_name	doctor_id
1	1	1	Paclitaxel	1
2	2	2	Acetaminophen	6
3	3	3	Acetaminophen	11
4	4	4	Omeprazole	8
5	5	5	Paclitaxel	2
6	6	6	Paclitaxel	2
7	7	7	Carbamazepine	16
8	8	8	Carbamazepine	19
9	9	9	Omeprazole	14
10	10	10	Paclitaxel	1
11	11	11	Paclitaxel	2
12	12	12	Omeprazole	11
13	13	13	Metoclopramide	6
14	14	14	Carbamazepine	16
15	15	15	Paclitaxel	1
16	16	16	Acetaminophen	14
17	17	17	Acetaminophen	8
18	18	18	Carbamazepine	16
19	19	19	Acetaminophen	25
20	20	20	Metoclopramide	10

Allergy

- dbo.Allergy
  - Columns
    - allergy_id (PK, int, not null)
    - record_id (FK, int, null)
    - patient_id (FK, int, null)
    - allergy (nvarchar(100), null)
    - doctor_id (FK, int, null)

	allergy_id	record_id	patient_id	allergy	doctor_id
1	1	1	1	carboplatin	1
2	2	2	19	sulfasalazine	6
3	3	3	2	amoxicillin	11
4	4	4	13	amoxicillin	8
5	5	5	15	carboplatin	2
6	6	6	1	carboplatin	2
7	7	7	9	morphine	16
8	8	8	7	morphine	19
9	9	9	15	morphine	14
10	10	10	1	carboplatin	1
11	11	11	15	carboplatin	2
12	12	12	6	morphine	11
13	13	13	3	morphine	6
14	14	14	5	ampicillin	16
15	15	15	15	carboplatin	1
16	16	16	10	amoxicillin	14
17	17	17	4	sulfasalazine	8
18	18	18	10	ampicillin	16
19	19	19	17	amoxicillin	25
20	20	20	3	morphine	10

ReviewFeedback

- dbo.ReviewFeedback
  - Columns
    - review_id (PK, int, not null)
    - patient_id (FK, int, null)
    - doctor_id (FK, int, null)
    - review_text (nvarchar(1000), null)
    - rating (int, null)
    - pastappointment_id (FK, int, null)

	review_id	patient_id	doctor_id	review_text	rating	pastappointment_i
1	1	1	1	Attentive, thorough, and compassionate	4	1
2	2	19	6	Highly recommend!	4	2
3	3	2	11	NULL	1	3
4	4	13	8	Attentive, knowledgeable, and caring	4	4
5	5	15	2	Grateful for the positive experience	4	5
6	6	1	2	NULL	NULL	6
7	7	9	16	Professional, knowledgeable, and empathetic	4	7
8	8	7	19	NULL	NULL	8
9	9	15	14	Approachable, patient, and understanding.	4	9
10	10	1	1	NULL	NULL	10
11	11	15	2	Excellent care from Dr	5	11
12	12	6	11	NULL	NULL	12
13	13	3	6	Made me feel comfortable and well-informed.	4	13
14	14	5	16	Dr. provided exceptional care	5	14
15	15	15	1	NULL	2	15
16	16	10	14	Friendly, informative, and genuinely concerned	5	16
17	17	4	8	NULL	NULL	17
18	18	10	16	Fantastic!	5	18
19	19	17	25	A truly wonderful doctor	5	19
20	20	3	10	NULL	1	20

CurrentAppointment

- dbo.CurrentAppointment
  - Columns
    - currentappointment_id (PK, int, not null)
    - patient_id (FK, int, null)
    - date (date, not null)
    - time (time(7), not null)
    - department_id (FK, tinyint, null)
    - doctor_id (FK, int, null)
    - status (varchar(20), null)

	currentappointment_id	patient_id	date	time	department_id	doctor_id	status
1	1	1	2024-04-02	10:00:00.0000000	16	1	booked
2	2	3	2024-04-02	10:30:00.0000000	16	1	booked
3	3	6	2024-04-02	11:00:00.0000000	16	1	booked
4	4	7	2024-04-02	11:00:00.0000000	16	1	pending
5	5	15	2024-04-02	10:00:00.0000000	16	2	booked
6	6	9	2024-04-09	10:00:00.0000000	19	16	booked
7	7	13	2024-04-09	10:00:00.0000000	19	16	booked
8	8	12	2024-04-02	11:00:00.0000000	16	1	pending
9	9	1	2024-04-09	10:00:00.0000000	19	16	pending
10	10	15	2024-04-10	09:00:00.0000000	18	14	booked
11	11	2	2024-04-08	09:00:00.0000000	16	1	booked
12	12	10	2024-04-02	10:00:00.0000000	19	6	booked
13	13	6	2024-04-03	10:00:00.0000000	18	11	booked
14	14	5	2024-04-03	10:00:00.0000000	19	19	booked
15	15	3	2024-04-03	10:00:00.0000000	18	14	booked
16	16	4	2024-04-04	11:00:00.0000000	19	19	booked
17	17	9	2024-04-04	11:00:00.0000000	19	19	booked
18	18	17	2024-04-04	10:00:00.0000000	18	14	booked
19	19	13	2024-04-05	11:00:00.0000000	20	25	booked
20	20	10	2024-04-05	11:00:00.0000000	17	10	booked

To refine the schema, the following aspects are considered in the database design.

a) Normalization

In the hospital GP management database design, the schema appears to be well-normalized, with 13 tables organized to reduce redundancy and ensure data integrity. Each table is designed to satisfy the requirements of the 1NF to 3NF thereby minimizing data duplication and dependency issues.

b) Indexing

Refer to section 1.9.2 for indexes.

c) Data Types

The data types such as INT, TINYINT, VARCHAR, NVARCHAR, DATE, TIME, etc are appropriately chosen. However, some of the data types are changed afterward to improve storage use. For example, the data type of the password in **PatientPortalInfo** is changed from NVARCHAR(50) to BINARY (64) for more efficient storage of hashed passwords as shown in Figure 1.7.

```
-- Alter the table to change the pass_word column data type
ALTER TABLE PatientPortalInfo
ALTER COLUMN pass_word BINARY(64);

-- Convert existing passwords from NVARCHAR to NVARBINARY
UPDATE PatientPortalInfo
SET pass_word = CONVERT(BINARY(64), pass_word);
```

Figure 1.7 Changing datatype from NVARCHAR to BINARY in PatientPortalInfo table

d) Constraints

Refer to section 1.9.1 for constraints.

e) Naming Conventions

Throughout the hospital GP management database design, consistent and descriptive naming conventions are adopted for tables, columns, constraints, stored procedures, triggers, views, functions, etc to enhance the readability and maintainability of the schema.

f) Referential Integrity

The relationships between the entities developed from the client requirement are maintained using foreign keys. The foreign keys in the tables ensure referential integrity and prevent data inconsistency.

g) Partitioning and Archiving

Partitioning and archiving are performed in the database design for appointment data. The initial **Appointment** table is bifurcated into two –

CurrentAppointment and **PastAppointment** tables (based on dates) to improve manageability and query performance. When the appointment date in the **CurrentAppointment** table becomes one from the past, the corresponding records are deleted from the same and archived into the **PastAppointment** table. The **CurrentAppointment** table is maintained as a smaller dataset for optimizing the performance.

h) Optimization for Specific Queries

In the hospital GP management database, there are frequently performed queries such as patient details collection, doctor's appointments, medicines prescribed, etc. The schema is developed by considering these specific queries and appropriate indexes as mentioned in the section are created to improve the performance.

1.9 Applying Constraints and Indexes

Sections 1.9.1 and 1.9.2 explain the constraints and indexes used in the hospital GP management database design respectively.

1.9.1 Applying Constraints

The GP management database uses various constraints (such as NOT NULL, UNIQUE, and CHECK constraints) to validate and enforce data integrity and thereby ensures only valid data is entered into the database. The constraints discussed below help to maintain data consistency and prevent incorrect or inconsistent data. Figure 1.8 gives one example showing NOT NULL and UNIQUE constraints employed in the table **PatientPortalInfo**. The NOT NULL constraint ensures the primary key is not null and UNIQUE ensures the user email address is unique.

```
-- Create PatientPortalInfo Table
CREATE TABLE PatientPortalInfo (
    username NVARCHAR(50) NOT NULL PRIMARY KEY,
    patient_id INT FOREIGN KEY REFERENCES Patient(patient_id),
    pass_word NVARCHAR(50),
    email_address NVARCHAR(100) UNIQUE,
    telephone_number NVARCHAR(20),
    CONSTRAINT CHK_EmailOrPhoneNumber CHECK (
        (email_address IS NOT NULL AND email_address LIKE '%_@_%._%')
        OR
        (telephone_number IS NOT NULL)
    )
);
```

Figure 1.8 NOT NULL and UNIQUE constraints in PatientPortalInfo table

Table 1.4 Table and Constraints

Table : Patient

```
CONSTRAINT CHK_Gender CHECK (Gender IN ('M', 'F' , 'Others')),
CONSTRAINT CHK_BloodGroup CHECK (blood_group IN ('A+', 'A-', 'B+', 'B-', 'AB+', 'AB-', 'O+', 'O-', 'I dont know'))
```

Constraint: CHK_ BloodGroup checks and ensures the data belongs to the list of blood groups presented as menus in the browser-based GP registration portal.

Constraint: CHK_ Gender checks and ensures the data belongs to the list of genders presented as menus in the GP registration application or website.

Table : PatientPortalInfo

```
CONSTRAINT CHK_EmailOrPhoneNumber CHECK (
  (email_address IS NOT NULL AND email_address LIKE '%_@_%._%')
  OR
  (telephone_number IS NOT NULL)
)
```

Constraint: CHK_ EmailOrPhoneNumber checks whether a valid email address or telephone number is entered as it is important to send the user One-Time Passwords (OTP) or inform/ remind about the appointment.

Table: PastAppointment

```
-- Add Constraint to Check Past Appointment Date is in the Past and Status is Completed
ALTER TABLE PastAppointment
ADD CONSTRAINT CHK_PastAppointmentStatus CHECK (date < CAST(GETDATE() AS DATE) AND status = 'completed');
```

Constraint: CHK_ PastAppointmentStatus checks the data records corresponding to a past date with the status as ‘completed’.

Table: ReviewFeedback

```
CONSTRAINT CK_RatingRange CHECK (rating BETWEEN 1 AND 5) -- Constraint for rating range
```

Constraint: CK_ RatingRange checks and ensures the rating given to a doctor by a user is within the range listed in the menus of the GP registration application or website.

Table: CurrentAppointment

```
ALTER TABLE CurrentAppointment
ADD CONSTRAINT CheckStatus
CHECK (status IN ('booked', 'pending', 'available', 'cancelled', 'completed'));
```

Constraint: CK_ CheckStatus checks the status of the current appointment is one from the list (‘booked’, ‘pending’,’cancelled,’completed’)

The constraint CheckFutureDate on Table CurrentAppointment is explained in section 2.2. Other constraints are shown in Table 1.4

1.9.2 Applying Indexes

In the hospital GP management database, most of the queries are related to patients, appointments, doctors, departments, diagnoses, medicine, allergies, etc. Considering the nature of the queries in the database, appropriate clustered indexes are added to columns (primary keys) such as patient_id, doctor_id, department_id, etc for fast search and retrieval operations. Indexing can improve query performance by facilitating faster data retrieval. Non-clustered indexes are created on the hospital GP management database to improve the query processing time. Refer to section 1.13.4 for non-clustered indexes.

1.9.3 Additional Stored Procedures in the Database

Figures 1.9 and 1.10 give the additional stored procedures employed in the database.

```
--Retrieve patient history
CREATE PROCEDURE sp_GetPatientHistory
@patient_id INT
AS
BEGIN
    SET NOCOUNT ON;

    -- Retrieve past appointments
    SELECT pa.date AS appointment_date, pa.time AS appointment_time, d.department_id, d.specialization AS department, doc.doctor_id, CONCAT(doc.first_name, ' ', doc.last_name) AS doctor_name, pa.status
    FROM PastAppointment pa
    INNER JOIN Department d ON pa.department_id = d.department_id
    INNER JOIN Doctor doc ON pa.doctor_id = doc.doctor_id
    WHERE pa.patient_id = @patient_id;

    -- Retrieve diagnosis
    SELECT d.diagnosis_id, d.diagnosis, doc.doctor_id, CONCAT(doc.first_name, ' ', doc.last_name) AS doctor_name
    FROM Diagnosis d
    INNER JOIN Doctor doc ON d.doctor_id = doc.doctor_id
    WHERE d.record_id IN (SELECT record_id FROM MedicalRecord WHERE patient_id = @patient_id);

    -- Retrieve prescribed medicines
    SELECT m.medicine_name, doc.doctor_id, CONCAT(doc.first_name, ' ', doc.last_name) AS doctor_name
    FROM Medicine m
    INNER JOIN Doctor doc ON m.doctor_id = doc.doctor_id
    WHERE m.record_id IN (SELECT record_id FROM MedicalRecord WHERE patient_id = @patient_id);

    -- Retrieve allergies
    SELECT a.allergy, doc.doctor_id, CONCAT(doc.first_name, ' ', doc.last_name) AS doctor_name
    FROM Allergy a
    INNER JOIN Doctor doc ON a.doctor_id = doc.doctor_id
    WHERE a.record_id IN (SELECT record_id FROM MedicalRecord WHERE patient_id = @patient_id);

    -- Retrieve review feedback
    SELECT rf.review_id, rf.review_text, rf.rating, doc.doctor_id, CONCAT(doc.first_name, ' ', doc.last_name) AS doctor_name
    FROM ReviewFeedback rf
    INNER JOIN Doctor doc ON rf.doctor_id = doc.doctor_id
    WHERE rf.patient_id = @patient_id;
END


EXEC sp_GetPatientHistory @patient_id = 1;
```

	appointment_date	appointment_time	department_id	department	doctor_id	doctor_name	status
1	2023-09-04	10:00:00.0000000	16	Oncology	1	John Ava	completed
2	2023-11-07	10:00:00.0000000	16	Oncology	2	Alice Johnson	completed
3	2023-12-12	09:00:00.0000000	16	Oncology	1	John Ava	completed
4	2024-04-02	10:00:00.0000000	16	Oncology	1	John Ava	completed

	diagnosis_id	diagnosis	doctor_id	doctor_name
1	1	Cancer	1	John Ava
2	6	Cancer	2	Alice Johnson
3	10	Cancer	1	John Ava

	medicine_name	doctor_id	doctor_name
1	Paclitaxel	1	John Ava
2	Paclitaxel	2	Alice Johnson
3	Paclitaxel	1	John Ava

	allergy	doctor_id	doctor_name
1	carboplatin	1	John Ava
2	carboplatin	2	Alice Johnson
3	carboplatin	1	John Ava

	review_id	review_text	rating	doctor_id	doctor_name
1	1	Attentive, thorough, and compassionate	4	1	John Ava
2	6	NULL	NULL	2	Alice Johnson
3	10	NULL	NULL	1	John Ava

The stored procedure **sp_GetPatientHistory** retrieves the history of the patient namely the appointments, diagnosis, medicines, allergies, and review given by the patient to the doctor.

Figure 1. 9 Stored procedure **sp_GetPatientHistory** and result

```
--retrieve patients of a particular doctor
CREATE PROCEDURE sp_GetDoctorPatients
    @doctor_id INT
AS
BEGIN
    SELECT DISTINCT p.*
    FROM Patient p
    INNER JOIN MedicalRecord mr ON p.patient_id = mr.patient_id
    INNER JOIN PastAppointment pa ON pa.patient_id = p.patient_id
    WHERE pa.doctor_id = @doctor_id;
END;

EXEC sp_GetDoctorPatients @doctor_id = 1;
```

	patient_id	first_name	middle_name	last_name	dob	insurance	gender	blood_group	date_left
1	1	John	Doe	Abraham	1979-02-15	Aviva	M	A+	NULL
2	3	Michael	Lee	Wong	1975-09-10	Saga	M	O-	NULL
3	6	Sophia	Rose	Brown	1990-03-18	WPA	F	B+	NULL
4	7	James	NULL	Anderson	1982-11-05	Axa He...	Others	O+	NULL
5	15	Alexander	NULL	Wright	1983-05-12	Saga	M	O+	NULL

The stored procedure **sp_GetDoctorPatients** retrieves all the patients of the given doctor.

Figure 1.10 Stored procedure **sp_GetDoctorPatients** and result

1.9.4 System Functions, Joins, Sub Queries, and Additional User-Defined Functions in the Database

The system functions are utilized throughout the database design. Table 1.5 gives some of the system functions, joins, and sub-queries used in the database design.

Table 1.5 Some of the system functions, joins, and sub-queries in the database design

Code Area	System Functions	Joins and or Sub Queries
Constraint: **CheckFutureDate**	GETDATE()	-
Listing all patients older than 40 years and have Cancer in diagnosis.	GETDATE(), DATEDIFF()	INNER JOIN
Stored procedure/ Function **SearchMedicineByName**	COALESCE()	INNER JOINs
Stored procedure/ Function **GetDiagnosisAndAllergiesForCurrentPatient**	GETDATE(), CONCAT(), ISNULL(), and CAST()	LEFT JOINs
Stored procedure **UpdateDoctorDetails**	ISNULL()	subquery to check the existence of records in the **DoctorSchedule** table
Stored procedure **CompleteAppointmentsAndMoveToPastAppointment**	GETDATE(), CAST(), and TIME	subqueries in the **INSERT INTO** statement to select completed appointments.
Stored procedure **AllDoctorsAppointmentDetailsView**	CONCAT(), ISNULL().	LEFT JOINs subquery to combine appointment details from **CurrentAppointment** and **PastAppointment** tables.
Trigger **trg_CancelledAppointment**	-	INNER JOINs to join the **CurrentAppointment** table with the **inserted** table.
Select a query that allows the hospital to identify the number of completed appointments with the specialty of doctors as 'Gastroenterologists'.	CONCAT(), ISNULL(), COUNT(), and PRINT().	INNER JOINs are used to join the **PastAppointment**, **Doctor**, and **Department** tables.

Figures 1.11 and 1.12 give the additional user-defined functions employed in the database.

```
CREATE FUNCTION fn_GetPatientsVisitedByMonthYearDeptDate
(
   @year INT,
   @month INT
)
RETURNS TABLE
AS
RETURN
(
   SELECT DISTINCT P.*, PA.date AS visit_date, D.specialization AS department, CONCAT(DO.first_name, ' ', DO.last_name) AS doctor
   FROM PastAppointment PA
   INNER JOIN Patient P ON P.patient_id = PA.patient_id
   INNER JOIN Doctor DO ON PA.doctor_id = DO.doctor_id
   INNER JOIN Department D ON PA.department_id = D.department_id
   WHERE YEAR(PA.date) = @year AND MONTH(PA.date) = @month
);

SELECT * FROM fn_GetPatientsVisitedByMonthYearDeptDate(2023, 10);
```

	patient_id	first_name	middle_name	last_name	dob	insurance	gender	blood_group	date_left	visit_date	department	doctor
1	2	Alice	Smith	Johnson	1980-05-20	Axa Health	F	B-	NULL	2023-10-10	Cardiology	Ethan Walker
2	13	Ethan	Lucas	Miller	1974-03-27	Bupa	M	B-	NULL	2023-10-16	Gastroen...	Olivia Herna...
3	15	Alexander	NULL	Wright	1983-05-12	Saga	M	O+	NULL	2023-10-23	Oncology	Alice Johnson

The function **fn_GetPatientsVisitedByMonthYearDeptDate** retrieves the patients who visited the GP of the hospital in a specific year and month with the department and doctor visited.

Figure 1.11 Function **fn_GetPatientsVisitedByMonthYearDeptDate(2023,10)** and result

```
--Retrieve the list of patients who left the hospital
CREATE FUNCTION fn_GetPatientsLeftHospitalWithDetails
(
)
RETURNS TABLE
AS
RETURN
(
   SELECT P.*, A.address1, A.address2, A.city, A.county, A.postcode,
        COALESCE(PPI.email_address, PPI.telephone_number) AS contact_info
   FROM Patient P
   INNER JOIN Address A ON P.patient_id = A.patient_id
   LEFT JOIN PatientPortalInfo PPI ON P.patient_id = PPI.patient_id
   WHERE P.date_left IS NOT NULL
);

-- Execute the function
SELECT * FROM fn_GetPatientsLeftHospitalWithDetails();
```

	patient_id	first_name	middle_name	last_name	dob	insurance	gender	blood_group	date_left	address1	address2	city
1	8	Olivia	Grace	Martinez	1977-06-22	Aviva	F	A-	2022-09-15	444 Walnut Street	Flat 3B	Sheffield
2	18	Harper	Faith	Scott	1978-09-19	WPA	F	O-	2023-05-30	111 Willowwo...	Suite 1...	Belfast

county	postcode	contact_info
South Yorkshire	S1 1WB	OMartinez@example.com
Belfast	BT1 1...	HScott@example.com

The function **fn_GetPatientsLeftHospitalWithDetails** gives the details of the patient who has left the hospital.

Figure 1.12 Function **fn_GetPatientsLeftHospitalWithDetails** and result

1.9.5 Additional Views in the Database

Figure 1.13 gives the additional views employed in the database.

```
-- Review text and rating of a specific doctor
CREATE VIEW DoctorReviewTexts AS
SELECT
    R.review_id,
    R.review_text,
    R.rating,
    P.patient_id,
    CONCAT(P.first_name, ' ', COALESCE(P.middle_name, ''), ' ', P.last_name) AS patient_name,
    R.doctor_id
FROM
    ReviewFeedback R
INNER JOIN
    Patient P ON R.patient_id = P.patient_id;

    SELECT *
FROM DoctorReviewTexts
WHERE doctor_id = 6;
```

	review_id	review_text	rating	patient_id	patient_name	doctor_id
1	2	Highly recommend!	4	19	Mason Jacob Morris	6
2	13	Made me feel comfortable and well-informed.	4	3	Michael Lee Wong	6

The view **DoctorReviewTexts** gives the review texts and ratings given by patients to a specific doctor.

Figure 1.13 View **DoctorReviewTexts** and result for doctor_id =6

1.9.6 Additional Triggers in the Database

Figure 1.14 gives the additional views employed in the database.

```
-- A trigger that keeps the patients left to a new table 'RemovedPatients'
CREATE TABLE RemovedPatients (
    patient_id INT PRIMARY KEY,
    first_name NVARCHAR(50),
    middle_name NVARCHAR(50),
    last_name NVARCHAR(50),
    dob DATE,
    insurance NVARCHAR(50),
    gender NVARCHAR(50),
    blood_group NVARCHAR(10),
    date_left DATE
);
CREATE TRIGGER trg_PatientRemoval
ON Patient
AFTER UPDATE
AS
BEGIN
    SET NOCOUNT ON;

    IF UPDATE(date_left)
    BEGIN
        INSERT INTO RemovedPatients (patient_id, first_name, middle_name, last_name, dob, insurance, gender, blood_group, date_left)
        SELECT patient_id, first_name, middle_name, last_name, dob, insurance, gender, blood_group, date_left
        FROM inserted
        WHERE date_left IS NOT NULL;
    END
END;

UPDATE Patient
SET date_left = '2022-09-15'
WHERE patient_id = 8;
UPDATE Patient
SET date_left = '2023-05-30'
WHERE patient_id = 18;
```

	patient_id	first_name	middle_name	last_name	dob	insurance	gender	blood_group	date_left
1	8	Olivia	Grace	Martinez	1977-06-22	Aviva	F	A-	2022-09-15
2	18	Harper	Faith	Scott	1978-09-19	WPA	F	O-	2023-05-30

A table **'RemovedPatients'** is created in the database. Whenever a patient record is updated with a non-null value in the date left column, the patient record is added to the **RemovedPatients** table. This table **RemovedPatients** does not have any relationship with any of the other tables and is not shown in the database diagram. After the execution of update commands and trigger **trg_PatientRemoval**, the data in the **RemovedPatients** table is as given above.

Figure 1.13 Trigger **trg_PatientRemoval** and result

1.10 Optimizing the Final Schema

Along with all the aspects of database design explained in Sections 1.8.2 and 1.9, regular maintenance, testing, and monitoring are involved with the optimization of the final schema. It is required to regularly maintain the database for maintaining database integrity, consistency, and performance over time. Thorough testing of the optimized schema and regular monitoring of the database performance is important for improving the performance as well as meeting the client requirements without fail.

1.11 Documentation

This document describes all the details associated with the hospital GP management database design. The schema design, including table structures, relationships, constraints, and indexing is well documented for facilitating. understanding and maintenance of the database by the client.

1.12 Hospital GP Management Database Diagram

The hospital GP management database diagram given in Figure 1.15 represents the structure and relationships of database entities relevant to managing GP services within the client hospital. Figure 1.15 provides a visual representation of the database schema and relationships between different entities useful for developers, administrators, and stakeholders to understand the structure and functionality of the database system.

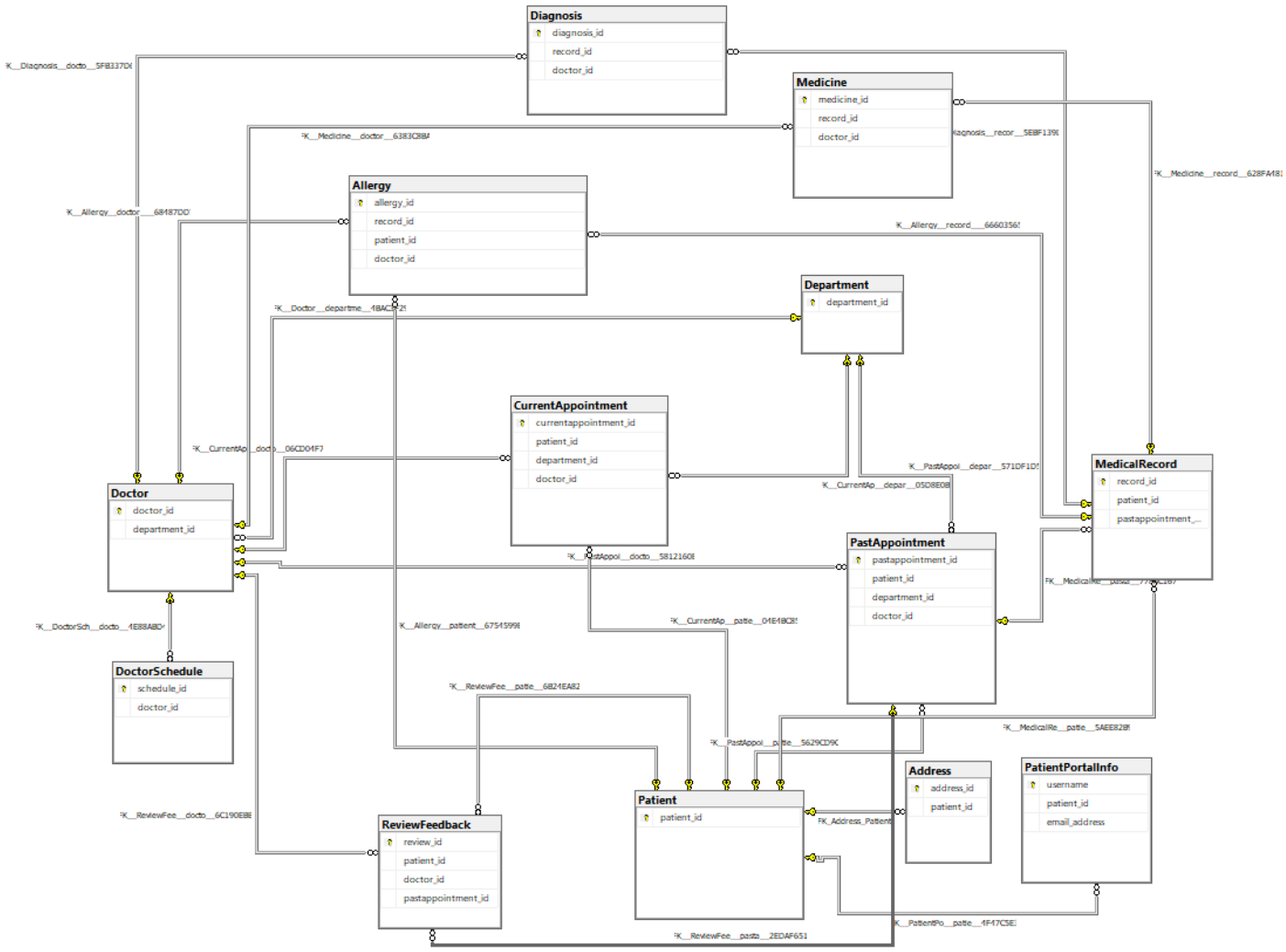


Figure 1.15 Hospital GP management database diagram

1.13 Advice and Guidance to Client

Various measures are taken in the database design to ensure data integrity, concurrency, security, backup, and recovery. Sections 1.13.1 to 1.13.4 explain each one of them respectively.

1.13.1 Advice and Guidance to Client on Data Integrity

In the entire GP management database design, the data integrity is ensured in different ways as listed in Table 1.6.

Table 1.6 Data integrity measures employed in database design

Parameter	**Implementation Method**	**Example from the database design**
Entity integrity	All Primary keys	patient_id (**Patient**) doctor_id (**Doctor**) username (**PatientPortalInfo**)
Domain integrity	Data types, Constraints such as UNIQUE	appointmentdate (DATE) appointmenttime (TIME) emailaddress (UNIQUE-**PatientPortalInfo**)
Referential integrity	All foreign keys	patient_id (**Address**) department_id (**Doctor**) doctor_id (**DoctorSchedule**)
User-defined integrity	Constraints	CheckFutureDate (**CurrentAppointment**) CHK_EmailOrPhoneNumber (**PatientPortalInfo**)
Transaction management	BEGIN TRANSACTION, COMMIT TRANSACTION, and ROLLBACK TRANSACTION	Stored Procedure (**UpdateDoctorDetails**)
Role assignment	GRANT and REVOKE commands	GRANT permission (**UpdateDoctorDetails**)

The advice and guidance to the client on data integrity are:

1. Conduct regular data quality checks to ensure data stored in the database is accurate, complete, and consistent.
2. Develop procedures for validating and verifying data integrity periodically.
3. Hire trained data entry operators or train the hospital staff on standard data entry procedures and how to maintain data integrity.

4. Educate the patients on booking GP through the website by installing kiosks or computers at various departments with GP consultation.
5. Prevent access to the database by unauthorized hospital staff by defining roles based on job responsibilities
6. Schedule periodic database maintenance to optimize database performance and ensure data integrity.
7. Establish procedures for promptly addressing and resolving data integrity issues as they arise.

1.13.2 Advice and Guidance to Client on Data Concurrency

As the GP management database is an OLTP design, it is important to ensure data concurrency. One such concurrency control implemented in the database design is setting the transaction isolation level to the doctor details updation (stored procedure **UpdateDoctorDetails1**) as given in Figure 1.16.

```
AS
BEGIN
    SET TRANSACTION ISOLATION LEVEL SERIALIZABLE; -- Setting transaction isolation level

    BEGIN TRY
        BEGIN TRANSACTION;
        -- Update doctor's room number and sub-specialization
        IF @roomNo IS NOT NULL OR @subSpecialization IS NOT NULL
        BEGIN
            UPDATE Doctor
            SET
                room_no = ISNULL(@roomNo, room_no),
                sub_specialization = ISNULL(@subSpecialization, sub_specialization)
            WHERE
                doctor_id = @doctorId;
        END
```

Figure 1.16 Setting transaction isolation level in stored procedure **UpdateDoctorDetails1**

The following are the recommendations given to the client for concurrency control.

1. To prevent multiple users or patients from simultaneously booking GP for the same doctor on the same day locking mechanisms such as row-level locking, table-level locking, or optimistic locking can be implemented for concurrency control.

2. Setting an isolation level for the transactions in the GP management database is appropriate as it controls the visibility of changes made by concurrent GP appointment bookings.
3. For efficiently managing concurrent access of data and data manipulation by hospital staff or doctors, different concurrency algorithms such as timestamp ordering, two-phase locking, and multi-version concurrency control (MVCC), etc can be employed.
4. As the GP management database is web-browser based, caching can be used to store frequently accessed data such as doctor schedules (**DoctorSchedule**), department details(**Department**), etc in memory. Replication is a technique that helps in the distribution of GP management data across multiple servers.

1.13.3 Advice and Guidance to Client on Database Security

Access control, inference control, flow control, and data encryption are the different data security control methods to be implemented in a database (Elmasri & Navathe, 2016). One of the data securities implemented in the database design is password encryption using the hashing method **Secure Hash Algorithm 1** (SHA1) is shown in Figure 1.17.

```
-- Update passwords in PatientPortalInfo table
UPDATE PatientPortalInfo
SET pass_word = SUBSTRING(
    CONVERT(NVARCHAR(100),HASHBYTES('SHA1', pass_word)),
    1,
    10
)
WHERE pass_word LIKE '%[^a-zA-Z0-9$#]%';
```

Figure 1. 17 Hashing the password in **PatientPortalInfo** table

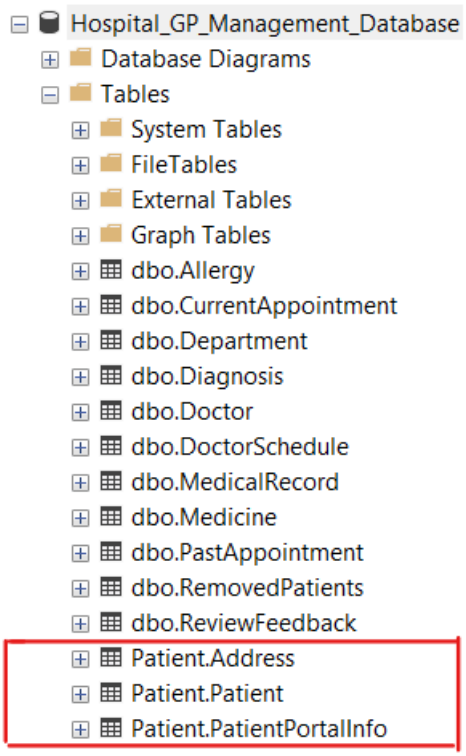


Figure 1.18 Creating schema Patient with tables **Patient**, **Address,** and **PatientPortalInfo** tables allocated to it.

```
--creating schema, and granting, revoking privileges to login users namely hospital_user and non_hospital_user respectively
CREATE SCHEMA Patient;
GO
ALTER SCHEMA Patient TRANSFER dbo.Patient
ALTER SCHEMA Patient TRANSFER dbo.Address
ALTER SCHEMA Patient TRANSFER dbo.PatientPortalInfo

CREATE LOGIN hospital_user WITH PASSWORD = 'pass123word';

CREATE USER hospital_user FOR LOGIN hospital_user;

GRANT SELECT ON SCHEMA :: Patient TO hospital_user;

GRANT SELECT, INSERT, DELETE, UPDATE ON Patient.Address
TO hospital_user WITH GRANT OPTION;

CREATE LOGIN non_hospital_user WITH PASSWORD = 'pass321word';

CREATE USER non_hospital_user FOR LOGIN non_hospital_user;

GRANT SELECT ON SCHEMA :: Patient TO non_hospital_user;

REVOKE INSERT, DELETE, UPDATE ON Patient.Address TO non_hospital_user;
```

Figure 1.19 Granting and Revoking privileges to users and roles

It is possible to establish security boundaries by creating schemas and allocating tables, views, functions, stored procedures, etc to the schemas created. Figure 1.18 shows allocating the tables **Patient**, **Address**, and **PatientPortalInfo** to the Patient schema. (The schema Patient is dropped from the database later). Figure 1.19 shows the creation of users and roles and granting and revoking privileges.

The following are the recommendations given to the client for ensuring database security.

1. Create user accounts and passwords to control the login to the GP management database.
2. Ensure strong password policies on the GP appointment booking website or application.
3. Encrypt the passwords corresponding to usernames in the **PatientPortal Info** to prevent data breaches.
4. Grant privileges to perform data manipulation only to certain logins given, based on the hospital staff roles.
5. Revoke privilege to perform certain actions such as some views, updation, insertion, etc to certain logins given, based on the hospital staff roles.
6. Prevent unauthorized hospital staff from retrieving sensitive individual data such as gender, blood group, medical records, etc.
7. Create schemas and assign tables and other database objects to schemas for improved security.

1.13.4 Advice and Guidance to Client on Database Backup and Recovery

Non-clustered indexes enforce referential integrity by allowing the database engine to quickly locate related rows and improve data integrity. As the GP management database is a multiuser relational database, indexing also ensures smooth concurrency control and performance by optimizing query plans. Considering database backup and recovery, the non-clustered indexes introduced in the GP management database design lead to faster backup and recovery times by reducing the duration of data access operations. The non-

clustered indexes introduced in the database are dob (**Patient**), medicine_name (**Medicine**), specialization (**Department**), and status (**CurrentAppointment**) as shown inside red boxes in Figure 1.20.

C	TableName	IndexName	IndexType	ColumnName	ColumnOrder
1	backup_metadata_store	IX_backup_metadata_type_finish_date	NONCLUSTERED	backup_type	1
2	backup_metadata_store	IX_backup_metadata_type_finish_date	NONCLUSTERED	backup_finish_date	2
3	CurrentAppointment	NonClusteredIndex-status	NONCLUSTERED	status	1
4	db_ledger_transactions	nci_transaction_id	NONCLUSTERED	transaction_id	1
5	Department	NonClusteredIndex-specialization	NONCLUSTERED	specialization	1
6	filestream_tombstone_2073058421	FSTSNCIdx	NONCLUSTERED	file_id	1
7	filestream_tombstone_2073058421	FSTSNCIdx	NONCLUSTERED	rowset_guid	2
8	filestream_tombstone_2073058421	FSTSNCIdx	NONCLUSTERED	column_guid	3
9	filestream_tombstone_2073058421	FSTSNCIdx	NONCLUSTERED	oplsn_fseqno	4
10	filestream_tombstone_2073058421	FSTSNCIdx	NONCLUSTERED	oplsn_bOffset	5
11	filestream_tombstone_2073058421	FSTSNCIdx	NONCLUSTERED	oplsn_slotid	6
12	Medicine	NonClusteredIndex-medicinename	NONCLUSTERED	medicine_name	1
13	Patient	NonClusteredIndex-dob	NONCLUSTERED	dob	1
14	PatientPortalInfo	UQ__PatientP__20C6DFF563C035...	NONCLUSTERED	email_address	1
15	PatientPortalInfo	UQ_Username	NONCLUSTERED	username	1
16	plan_persist_plan	plan_persist_plan_idx1	NONCLUSTERED	query_id	1
17	plan_persist_plan_feedback	plan_feedback_idx1	NONCLUSTERED	plan_id	1
18	plan_persist_plan_feedback	plan_feedback_idx1	NONCLUSTERED	feature_id	2
19	plan_persist_plan_forcing_locati...	plan_persist_plan_force_idx1	NONCLUSTERED	plan_forcing_loc...	1
20	plan_persist_query	plan_persist_query_idx1	NONCLUSTERED	query_text_id	1
21	plan_persist_query	plan_persist_query_idx1	NONCLUSTERED	context_settings_id	2
22	plan_persist_query_hints	plan_persist_query_hints_idx1	NONCLUSTERED	query_id	1
23	plan_persist_query_template_par...	plan_persist_query_template_param...	NONCLUSTERED	query_template_...	1

1.20 List of non-clustered indexes in the GP management database

The following are the recommendations given to the client for ensuring database backup and recovery. (Refer to Appendix C for the backup and recovery operation performed by the database design team).

1. Establish a regular backup schedule daily for the GP maintenance database. The schedule should include full backup, differential backup, and transaction log backups for efficient recovery.
2. Automate the backup process, review backup logs, and generate and monitor reports on the backup.
3. Implement reliable cloud storage or a secure offsite location for storing the backup copies of the GP management database to ensure recovery from loss of data caused by disasters such as floods, fire, earthquakes, theft, etc.
4. Test the backup and recovery procedures periodically to verify the integrity of the backup files, and thereby validate that the backup and recovery processes are on point.

5. Regularly maintain the GP management database environment and optimize the performance and storage resources for reliable backup and recovery operation.

Task 1 Part 2: Implementing Advanced Database Functionality and Performance Optimization: Constraints, Stored Procedures, Views, Triggers, and Data Administration Strategies in Hospital GP Management Database System

2.1 Introduction

For building a robust hospital GP management database system, advanced functionalities and optimization techniques are essential. The database design is based on the various tables created in Part 1. The Part 2 section focuses on integrating constraints, stored procedures, views, triggers, and sophisticated querying methods into the database design to enhance data integrity, security, and accessibility.

The specific requirements of a hospital such as ensuring the validity of the current appointments by checking the dates, listing patients above a certain age having a particular diagnosis, retrieving patient information based on complex criteria, and updating the doctor's details, etc are addressed through constraints, stored procedures, views, triggers, and sophisticated querying methods. All these functionalities improve the application efficiency and user experience. The data administration aspects such as database integrity, concurrency control, security measures, backup/ or recovery strategies, etc are also tailored into the database design to give a holistic solution to the client. The upcoming section addresses specific queries requested by the client hospital.

2.2 Implementing Date Constraint to Ensure Future Appointment Dates

Task 1: Part 2:

1. Add the constraint to check that the appointment date is not in the past.

A user can log in to the GP registration application using the credentials and password given in the **PatientInfoPortal** table. Once logged in, the user can book the appointment and appointments appear in the **CurrentAppointment**

table. The user is not permitted to choose a date from the past while booking through the application.

In the database design, a constraint can be added to the **CurrentAppointment** table to ensure that the appointment date is not in the past as illustrated in Figure 2.1.

```
-- Create CurrentAppointment Table
CREATE TABLE CurrentAppointment (
    currentappointment_id INT IDENTITY (1,1) NOT NULL PRIMARY KEY,
    patient_id INT FOREIGN KEY REFERENCES Patient(patient_id),
    date DATE,
    time TIME,
    department_id TINYINT FOREIGN KEY REFERENCES Department(department_id),
    doctor_id INT FOREIGN KEY REFERENCES Doctor(doctor_id),
    status VARCHAR(20)
);

-- Add constraint to check that the appointment date is not in the past
ALTER TABLE CurrentAppointment
ADD CONSTRAINT CheckFutureDate
CHECK (date >= CAST(GETDATE() AS DATE));
```

Figure 2.1 T-SQL statements to add the constraint to check that the appointment date is not in the past

The constraint **CheckFutureDate** primarily pertains to data integrity and ensures the accuracy and consistency of the data by preventing invalid appointment dates from users. It ensures that appointments are scheduled only for future dates, which aligns with the expected behavior of the system and enhances the overall reliability of the database. Even though the constraint indirectly contributes to concurrency by potentially reducing conflicts arising from invalid data entries, its primary goal is to maintain data integrity in the appointment scheduling process.

2.2.1 Checking the data integrity in the appointment scheduling process.

The constraint **CheckFutureDate** does not allow insertion of past appointment dates into the **CurrentApointment** table. The attempt to insert past

appointment date using codes as shown in Figure 2.2 results in an error message given in Figure 2.3.

```
INSERT INTO CurrentAppointment (patient_id, date, time, department_id, doctor_id, status)
VALUES
(1, '2022-06-02', '10:00:00', 18, 20, 'booked')
```

Figure 2.2 Inserting past dates to CurrentAppointment table

```
Msg 547, Level 16, State 0, Line 756
The INSERT statement conflicted with the CHECK constraint "CheckFutureDate".
The conflict occurred in database "Hospital_GP_Management_Database",
table "dbo.CurrentAppointment", column 'date'.
The statement has been terminated.
```

Figure 2.3 Error message when inserting past dates to CurrentAppointment table

2.3 Identifying Patients Over 40 with a Diagnosis of Cancer

Task 1: Part 2:

2. List all the patients with older than 40 and have Cancer in diagnosis.

In hospital GP management database design, it is crucial to identify and manage patients with specific medical conditions. Identifying patients with specific health conditions enables targeted care, treatment planning, and ongoing monitoring. Moreover, this contributes to effective patient management and delivery of quality healthcare services within the hospital's GP department.

Here the task is to extract relevant patient data for individuals above the age of 40 and diagnosed with cancer. The data from tables **Patient**, **MedicalRecord**, and **Diagnosis** are combined to retrieve comprehensive patient information and associated diagnoses. The result provides crucial diagnostic information, highlighting patients who have been diagnosed with cancer along with patient identifiers such as ID, name, and demographic details like age and gender.

```
SELECT DISTINCT
    p.patient_id,
    CONCAT(p.first_name, ' ', COALESCE(p.middle_name, ''), ' ', p.last_name) AS fullname,
    DATEDIFF(YEAR, p.dob, GETDATE()) AS age, p.gender,
    d.diagnosis
FROM
    Patient p
JOIN
    MedicalRecord mr ON p.patient_id = mr.patient_id
JOIN
    Diagnosis d ON mr.record_id = d.record_id
WHERE
    DATEDIFF(YEAR, p.dob, GETDATE()) > 40 AND
    d.diagnosis = 'Cancer';
```

Figure 2.4 T-SQL statements to list all the patients older than 40 and have Cancer in diagnosis

Results Messages

	patient_id	fullname	age	gender	diagnosis
1	1	John Doe Abraham	45	M	Cancer
2	15	Alexander Wright	41	M	Cancer

Figure 2.5 List all the patients older than 40 and have Cancer in the diagnosis

2.4 Implementing Stored Procedures and User-Defined Functions

2.4.1 Implementing Search Functionality for Medicines with Sorting by Prescription Date

Task 1: Part 2:

4 a . Search the database of the hospital for matching character strings by name of medicine. Results should be sorted with the most recent medicine prescribed date first.

The search for medicines by name is a common task performed by medical professionals, pharmacists, and administrators in the hospital GP management database. Here, the task involves querying the database to find medicines that match a given search string, along with associated patient information and prescription dates with the most recent prescription listed first.

Both the stored procedure **SearchMedicineByName** and the table-valued function **SearchMedicineByNameFunction** can perform the same task of searching for medicines by name. The data from tables (**Patient**, **MedicalRecord**, **PastAppointment**, and **Medicine**) are utilized. The result retrieves patient details, including the patient's full name and prescription date, along with the medicine name. The results are sorted by the most recent prescription date, enabling users to easily identify the latest prescriptions. The stored procedure and function developed are illustrated in Figures 2.6 and 2.8 respectively. The results for the search '**Omeprazole**' are given in Figures 2.7 and 2.9 respectively. Figure 2.10 gives the results for matching character string '**et**'.

```
CREATE PROCEDURE SearchMedicineByName
   @medicineName NVARCHAR(100)
AS
BEGIN
   SET NOCOUNT ON;

   SELECT
      p.patient_id,
      p.first_name + ' ' + COALESCE(p.middle_name, '') + ' ' + p.last_name AS FullName,
      m.medicine_name,
      pa.date AS PrescriptionDate
   FROM
      Patient p
   JOIN
      MedicalRecord mr ON p.patient_id = mr.patient_id
   JOIN
      PastAppointment pa ON mr.pastappointment_id = pa.pastappointment_id
   JOIN
      Medicine m ON mr.record_id = m.record_id
   WHERE
     m.medicine_name LIKE '%' + @medicineName + '%'
   ORDER BY
      pa.date DESC;
END;

EXEC SearchMedicineByName 'Omeprazole';
```

Figure 2.6 Stored procedure **SearchMedicineByName**

Results Messages

	patient_id	FullName	medicine_name	PrescriptionDate
1	6	Sophia Rose Brown	Omeprazole	2023-12-20
2	15	Alexander Wright	Omeprazole	2023-12-07
3	13	Ethan Lucas Miller	Omeprazole	2023-10-16

Figure. 2.7 Stored procedure **SearchMedicineByName** results for medicine **'Omeprazole'**

```
CREATE FUNCTION dbo.SearchMedicineByNameFunction(@medicineName NVARCHAR(100))
RETURNS TABLE
AS
RETURN
(
    SELECT
        p.patient_id,
        p.first_name + ' ' + COALESCE(p.middle_name, '') + ' ' + p.last_name AS FullName,
        m.medicine_name,
        pa.date AS PrescriptionDate
    FROM
        Patient p
    JOIN
        MedicalRecord mr ON p.patient_id = mr.patient_id
    JOIN
        PastAppointment pa ON mr.pastappointment_id = pa.pastappointment_id
    JOIN
        Medicine m ON mr.record_id = m.record_id
    WHERE
        m.medicine_name LIKE '%' + @medicineName + '%'
);

SELECT *
FROM dbo.SearchMedicineByNameFunction('Omeprazole')
ORDER BY PrescriptionDate DESC;
```

Figure 2.8 Table-valued function **SearchMedicineByNameFunction**

Results Messages

	patient_id	FullName	medicine_name	PrescriptionDate
1	6	Sophia Rose Brown	Omeprazole	2023-12-20
2	15	Alexander Wright	Omeprazole	2023-12-07
3	13	Ethan Lucas Miller	Omeprazole	2023-10-16

Figure 2.9 Table-valued function **SearchMedicineByNameFunction** results for medicine **'Omeprazole'**

Stored Procedure results

	patient_id	FullName	medicine_name	PrescriptionDate
1	3	Michael Lee Wong	Metoclopramide	2024-03-01
2	17	Benjamin Owen Adams	Acetaminophen	2024-02-09
3	4	Emma Wilson	Acetaminophen	2024-02-05
4	10	Ava Elizabeth Lopez	Acetaminophen	2024-01-11
5	3	Michael Lee Wong	Metoclopramide	2023-12-27
6	2	Alice Smith Johnson	Acetaminophen	2023-10-10
7	19	Mason Jacob Morris	Acetaminophen	2023-09-05

Table-valued Function results

	patient_id	FullName	medicine_name	PrescriptionDate
1	3	Michael Lee Wong	Metoclopramide	2024-03-01
2	17	Benjamin Owen Adams	Acetaminophen	2024-02-09
3	4	Emma Wilson	Acetaminophen	2024-02-05
4	10	Ava Elizabeth Lopez	Acetaminophen	2024-01-11
5	3	Michael Lee Wong	Metoclopramide	2023-12-27
6	2	Alice Smith Johnson	Acetaminophen	2023-10-10
7	19	Mason Jacob Morris	Acetaminophen	2023-09-05

Figure 2.10 Results for matching characters **'et'** in medicine name

As the primary objective is to encapsulate a reusable query for searching medicines, a table-valued function **SearchMedicineByNameFunction** is preferred over the stored procedure.

2.4.2 Retrieve Diagnosis and Allergies for Patients with Appointments Today (run on April 2, 2024)

Task 1: Part 2:

4 b. Return a full list of diagnoses and allergies for a specific patient who has an appointment today (i.e., the system date when the query is run)

There is a common requirement in the hospital GP management database to retrieve the list of diagnoses and allergies for a patient who has an appointment on the current day. Here, the task involves querying patient records, medical history, current appointments, diagnoses, and allergies to provide relevant information to medical professionals. The data from multiple tables (**Patient**, **MedicalRecord**, **Diagnosis**, **Allergy**, and **CurrentAppointment**) are joined to list the results containing patient id, patient name, gender, blood group, current appointment id, diagnosis, and allergy.

Both the stored procedure **GetDiagnosisAndAllergiesForCurrentPatient** and the table-valued function **GetDiagnosisAndAllergiesForCurrentPatientFunction** can provide results for the same task. However, the stored procedure **GetDiagnosisAndAllergiesForCurrentPatient** is preferred as it prints the message "**No appointment for the patient today**" which is not directly given

by the table-valued function. The related codes and results are given in Figure 2.11 to 2.14 respectively.

```
CREATE PROCEDURE GetDiagnosisAndAllergiesForCurrentPatient
   @patientId INT
AS
BEGIN
   DECLARE @appointmentDate DATE = CAST(GETDATE() AS DATE);

   IF EXISTS (SELECT 1 FROM CurrentAppointment WHERE patient_id = @patientId AND date = @appointmentDate)
   BEGIN
      SELECT DISTINCT
         p.patient_id,
         CONCAT(p.first_name, ' ', ISNULL(p.middle_name, ''), ' ', p.last_name) AS PatientName,
         p.gender,
         p.blood_group,
         ca.currentappointment_id AS CurrentAppointmentID,
         d.diagnosis,
         a.allergy
      FROM
         Patient p
      LEFT JOIN
         MedicalRecord mr ON p.patient_id = mr.patient_id
      LEFT JOIN
         Diagnosis d ON mr.record_id = d.record_id
      LEFT JOIN
         Allergy a ON mr.record_id = a.record_id
      LEFT JOIN
         CurrentAppointment ca ON p.patient_id = ca.patient_id
      WHERE
         p.patient_id = @patientId
         AND ca.date = @appointmentDate;
   END
   ELSE
   BEGIN
      PRINT 'No appointment for this patient today';
   END
END;

EXEC GetDiagnosisAndAllergiesForCurrentPatient @patientId = 10;
```

Figure 2.11 Stored procedure **GetDiagnosisAndAllergiesForCurrentPatient**

Patient_id=1

	patient_id	PatientName	gender	blood_group	CurrentAppointmentID	diagnosis	allergy
1	1	John Doe Abraham	M	A+	1	Cancer	carboplatin

Patient_id=10

	patient_id	PatientName	gender	blood_group	CurrentAppointmentID	diagnosis	allergy
1	10	Ava Elizabeth Lopez	F	B+	12	Chest pain	amoxicillin
2	10	Ava Elizabeth Lopez	F	B+	12	Slurred speech	ampicillin

Patient_id=15

	patient_id	PatientName	gender	blood_group	CurrentAppointmentID	diagnosis	allergy
1	15	Alexander Wright	M	O+	5	Cancer	carboplatin
2	15	Alexander Wright	M	O+	5	Heart problem	morphine

Patient_id=19

Messages

```
No appointment for this patient today

Completion time: 2024-04-02T22:37:32.3127948+01:00
```

Figure 2.12 Results for stored procedure

GetDiagnosisAndAllergiesForCurrentPatient

```
CREATE FUNCTION GetDiagnosisAndAllergiesForCurrentPatientFunction
(
    @patientId INT
)
RETURNS TABLE
AS
RETURN
(
    SELECT DISTINCT
        p.patient_id,
        CONCAT(p.first_name, ' ', ISNULL(p.middle_name, ''), ' ', p.last_name) AS PatientName,
        p.gender,
        p.blood_group,
        ca.currentappointment_id AS CurrentAppointmentID,
        d.diagnosis,
        a.allergy
    FROM
        Patient p
    LEFT JOIN
        MedicalRecord mr ON p.patient_id = mr.patient_id
    LEFT JOIN
        Diagnosis d ON mr.record_id = d.record_id
    LEFT JOIN
        Allergy a ON mr.record_id = a.record_id
    LEFT JOIN
        CurrentAppointment ca ON p.patient_id = ca.patient_id
    WHERE
        p.patient_id = @patientId
        AND ca.date = CAST(GETDATE() AS DATE)
);
SELECT * FROM GetDiagnosisAndAllergiesForCurrentPatientFunction(15);
```

Figure 2.13 Table-valued function

GetDiagnosisAndAllergiesForCurrentPatientFunction

Patient_id=1

	patient_id	PatientName	gender	blood_group	CurrentAppointmentID	diagnosis	allergy
1	1	John Doe Abraham	M	A+	1	Cancer	carboplatin

Patient_id=10

	patient_id	PatientName	gender	blood_group	CurrentAppointmentID	diagnosis	allergy
1	10	Ava Elizabeth Lopez	F	B+	12	Chest pain	amoxicillin
2	10	Ava Elizabeth Lopez	F	B+	12	Slurred speech	ampicillin

Patient_id=15

	patient_id	PatientName	gender	blood_group	CurrentAppointmentID	diagnosis	allergy
1	15	Alexander Wright	M	O+	5	Cancer	carboplatin
2	15	Alexander Wright	M	O+	5	Heart problem	morphine

Patient_id=19

patient_id	PatientName	gender	blood_group	CurrentAppointmentID	diagnosis	allergy

Figure 2.14 Results for table-valued function

GetDiagnosisAndAllergiesForCurrentPatientFunction

2.4.3 Update the details for an existing doctor

Task 1: Part 2:

4 c. Update the details for an existing doctor

In hospitals, it is required to update the doctor's details for easy GP practice. Updating the details of existing doctors involves modifying various attributes such as room number, sub-specialization, department, and schedule in tables **Doctor** and **DoctorSchedule**. As it is data manipulation, it is advised to perform the updation using a stored procedure as the user-defined functions cannot directly modify data. The provided stored procedure **UpdateDoctorDetails** performs the updation efficiently and reliably.

Doctor_id: 45 details before updation

45	45	Zoe	Ethan	Thomps...	Mohs Surgery	560	24
46	46	Nathan	Evan	Bell	Pediatric Derma...	561	24
47	47	Ella	Evans	Harrison	Cornea and Refr...	247	25
48	48	Landon	Ferguson	Gibson	Cornea and Refr...	248	25
49	49	Levi	Fisher	Fuller	Cornea and Refr...	249	25
50	50	Mia	Fletcher	Ford	Vitreoretinal Sur...	250	25
51	51	Lincoln	Ford	Fletcher	Oculoplastics	251	25
52	52	Layla	Foster	Fowler	Female Urology	441	26
53	53	Jayden	Fowler	Griffin	Andrology	442	26
54	54	Peyton	Fuller	Wheeler	Urologic Oncolo...	443	26
55	55	Aubrey	Gibson	Kennedy	Urologic Oncolo...	444	26
56	56	Evan	Gomez	Porter	Female Urology	445	26

Doctor Schedule of Doctor_id: 45 before updation

86	86	43	Tuesday	9	08:00:00.0000000	11:00:00.0000000
87	87	44	Wednesday	6	09:30:00.0000000	12:30:00.0000000
88	88	44	Thursday	12	10:30:00.0000000	13:30:00.0000000
89	89	45	Friday	8	08:30:00.0000000	11:30:00.0000000
90	90	45	Monday	10	09:00:00.0000000	12:00:00.0000000

Figure 2.15 Doctor details and Doctor schedule of doctor_id =45 before updation

```
CREATE PROCEDURE UpdateDoctorDetails
    @doctorId INT,
    @roomNo NVARCHAR(10) = NULL,
    @subSpecialization NVARCHAR(100) = NULL,
    @departmentId TINYINT = NULL,
    @dayOfWeek NVARCHAR(10) = NULL,
    @startTime TIME = NULL,
    @endTime TIME = NULL
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION;
        -- Update doctor's room number and sub-specialization
        IF @roomNo IS NOT NULL OR @subSpecialization IS NOT NULL
        BEGIN
            UPDATE Doctor
            SET
                room_no = ISNULL(@roomNo, room_no),
                sub_specialization = ISNULL(@subSpecialization, sub_specialization)
            WHERE
                doctor_id = @doctorId;
        END
```

```
    -- Update doctor's department
    IF @departmentId IS NOT NULL
    BEGIN
      UPDATE Doctor
      SET
        department_id = @departmentId
      WHERE
        doctor_id = @doctorId;
    END
    -- Update doctor's schedule
    IF @dayOfWeek IS NOT NULL AND @startTime IS NOT NULL AND @endTime IS NOT NULL
    BEGIN
      IF EXISTS (SELECT 1 FROM DoctorSchedule WHERE doctor_id = @doctorId AND day_of_week = @dayOfWeek)
      BEGIN
        UPDATE DoctorSchedule
        SET
          start_time = @startTime,
          end_time = @endTime
        WHERE
          doctor_id = @doctorId
          AND day_of_week = @dayOfWeek;
      END

       ELSE
       BEGIN
         INSERT INTO DoctorSchedule (doctor_id, day_of_week, start_time, end_time)
         VALUES (@doctorId, @dayOfWeek, @startTime, @endTime);
       END
     END

     COMMIT TRANSACTION;
   END TRY
   BEGIN CATCH
     ROLLBACK TRANSACTION;
     THROW;
   END CATCH
END;

EXEC UpdateDoctorDetails
@doctorId = 45,
@roomNo = '563',
@subSpecialization = 'Mohs Micrographic Surgery ',
@departmentId = 30,
@dayOfWeek = 'Friday',
@startTime = '09:00:00',
@endTime = '12:00:00';
```

Figure 2.16 Doctor details and Doctor schedule of doctor_id =45 before updation

Figures 2.15 – 2.17 show the T-SQL statements and results related to the task : update doctor details.

Doctor_id: 45 details after updation

	doctor_id	first_name	middle_name	last_name	sub_specialization	room_no	department_id
34	34	Carter	Daniel	Sanchez	Reproductive Endocrinology	229	22
35	35	Scarlett	David	Morales	Gynecologic Oncology	230	22
36	36	Ryan	Davidson	Foster	Sleep Medicine	301	23
37	37	Sofia	Day	Powell	Interventional Pulmonology	302	23
38	38	Elijah	Dixon	Sullivan	Pulmonary Hypertension	303	23
39	39	Liam	Eli	Butler	Pulmonary Hypertension	304	23
40	40	Aria	Elijah	Gomez	Sleep Medicine	305	23
41	41	Grayson	Elizabeth	Reed	Interventional Pulmonology	306	23
42	42	Amelia	Ella	Cook	Cosmetic Dermatology	557	24
43	43	Lucas	Emily	Morgan	Cosmetic Dermatology	558	24
44	44	Harper	Emma	Fisher	Mohs Surgery	559	24
45	45	Zoe	Ethan	Thomps...	Mohs Micrographic Surgery	563	30
46	46	Nathan	Evan	Bell	Pediatric Dermatology	561	24
47	47	Ella	Evans	Harrison	Cornea and Refractive Surgery	247	25
48	48	Landon	Ferguson	Gibson	Cornea and Refractive Surgery	248	25

Doctor Schedule of Doctor_id: 45 details after updation

	schedule_id	doctor_id	day_of_week	bookings_remaining	start_time	end_time
76	76	38	Tuesday	11	10:30:00.0000000	13:30:00.0000000
77	77	39	Wednesday	7	08:30:00.0000000	11:30:00.0000000
78	78	39	Thursday	9	09:00:00.0000000	12:00:00.0000000
79	79	40	Friday	12	10:00:00.0000000	13:00:00.0000000
80	80	40	Monday	10	08:00:00.0000000	11:00:00.0000000
81	81	41	Tuesday	9	09:30:00.0000000	12:30:00.0000000
82	82	41	Wednesday	7	10:30:00.0000000	13:30:00.0000000
83	83	42	Thursday	12	08:30:00.0000000	11:30:00.0000000
84	84	42	Friday	8	09:00:00.0000000	12:00:00.0000000
85	85	43	Monday	11	10:00:00.0000000	13:00:00.0000000
86	86	43	Tuesday	9	08:00:00.0000000	11:00:00.0000000
87	87	44	Wednesday	6	09:30:00.0000000	12:30:00.0000000
88	88	44	Thursday	12	10:30:00.0000000	13:30:00.0000000
89	89	45	Friday	8	09:00:00.0000000	12:00:00.0000000
90	90	45	Monday	10	09:00:00.0000000	12:00:00.0000000

Figure 2.17 Doctor details and Doctor schedule of doctor_id =45 after updation

The stored procedure **UpdateDoctorDetails** also offers transaction management and control, allowing for atomicity and ensuring that updates are either committed or rolled back entirely. The '**TRY ..CATCH**' block in this stored procedure also allows a robust error-handling mechanism. The database security can be ensured by granting permission for specific users to update the **Doctor** and **DoctorSchedule** details as illustrated in Figure 2.18.

```
CREATE LOGIN hospital_user WITH PASSWORD = 'pass123word';

    -- Create a new user
CREATE USER hospital_user FOR LOGIN hospital_user;

-- Create a new role
CREATE ROLE hospital_staff;

-- Grant SELECT permission on the Doctor table to the role
GRANT SELECT ON dbo.Doctor TO hospital_staff;

-- Grant EXECUTE permission on the UpdateDoctorDetails stored procedure to the role
GRANT EXECUTE ON dbo.UpdateDoctorDetails TO hospital_staff;

-- Add the user to the role
ALTER ROLE hospital_staff ADD MEMBER hospital_user;
```

Figure 2.18 Transaction management, security, and control by granting permission to specific users to update Doctor details.

2.4.4 Streamlining Data Management: Deleting Completed Appointments (run on April 3, 2024)

Task 1: Part 2:

4d. Delete the appointment whose status is already completed

This task is modified as the database table **CurrentAppointment** is designed with the **CheckStatus** constraint that allows the status to be either ‘booked’, ‘pending’, ‘available’, ‘completed’, or ‘cancelled’ as given in Figure 2.19. The table explains what each status indicates.

```
ALTER TABLE CurrentAppointment
ADD CONSTRAINT CheckStatus
CHECK (status IN ('booked', 'pending', 'available', 'cancelled', 'completed'));
```

Figure 2.19 **CheckStatus** constraint

Sl No	Status	Description
1	booked	The appointment is confirmed.
2	pending	The appointment is not booked, as the doctor's booking is over.
3	available	The appointment is available as the doctor has some bookings remaining. This status is for users who have received pending status earlier.
4	cancelled	The appointment is cancelled by the user.
5	completed	The appointment is completed

Table 2.1 Description of status in **CurrentAppointment** table

Modified task: Update the status to 'completed' based on the current date and time. Insert the completed appointments into the **PastAppointment** table. Delete completed appointments from the **CurrentAppointment** table. Figure 2.20 -2.23 gives the T-SQL statements and results for the stored procedure **CurrentAppointmentsAndMoveToPastAppointment** performing the modified task.

```
--Stored procedure to delete the appointment whose status is already completed in CurrentAppointment table
CREATE PROCEDURE CompleteAppointmentsAndMoveToPastAppointment
AS
BEGIN
   DECLARE @CurrentDate DATE = CAST(GETDATE() AS DATE);
   DECLARE @CurrentTime TIME = CAST(GETDATE() AS TIME);

   -- Update appointments in CurrentAppointment table to 'completed'
   UPDATE CurrentAppointment
   SET status = 'completed'
   WHERE date <= @CurrentDate
   AND (date < @CurrentDate OR time <= @CurrentTime);

   -- Insert completed appointments into PastAppointment table
   INSERT INTO PastAppointment (patient_id, date, time, department_id, doctor_id, status)
   SELECT patient_id, date, time, department_id, doctor_id, 'completed'
   FROM CurrentAppointment
   WHERE status = 'completed'
   AND (date < @CurrentDate OR (date = @CurrentDate AND time <= @CurrentTime));

 -- Delete completed appointments from CurrentAppointment table
   DELETE FROM CurrentAppointment
   WHERE status = 'completed'
   AND (date < @CurrentDate OR (date = @CurrentDate AND time <= @CurrentTime));
END

EXEC CompleteAppointmentsAndMoveToPastAppointment;
```

Figure 2.20 Stored procedure **CurrentAppointmentsAndMoveToPastAppointment**

The tables **CurrentAppointment** and **PastAppointment** are utilized for executing this task. The permission to execute the **CurrentAppointmentsAndMoveToPastAppointment** is granted to specific users as shown below in Figure 2.21

```
GRANT EXECUTE ON dbo.CompleteAppointmentsAndMoveToPastAppointment TO hospital_user;
```

Figure 2.21 Granting permission to stored procedure **CurrentAppointmentsAndMoveToPastAppointment**

CurrentAppointment table before executing the stored procedure (check records corresponding to date '**2024-04-02'**)

	currentappointment_id	patient_id	date	time	department_id	doctor_id	status
1	1	1	2024-04-02	10:00:00.0000000	16	1	booked
2	2	3	2024-04-02	10:30:00.0000000	16	1	booked
3	3	6	2024-04-02	11:00:00.0000000	16	1	booked
4	4	7	2024-04-02	11:00:00.0000000	16	1	pending
5	5	15	2024-04-02	10:00:00.0000000	16	2	booked
6	6	9	2024-04-09	10:00:00.0000000	19	16	booked
7	7	13	2024-04-09	10:00:00.0000000	19	16	booked
8	8	12	2024-04-02	11:00:00.0000000	16	1	pending
9	9	1	2024-04-09	10:00:00.0000000	19	16	pending
10	10	15	2024-04-10	09:00:00.0000000	18	14	booked
11	11	2	2024-04-08	09:00:00.0000000	16	1	booked
12	12	10	2024-04-02	10:00:00.0000000	19	6	booked
13	13	6	2024-04-03	10:00:00.0000000	18	11	booked
14	14	5	2024-04-03	10:00:00.0000000	19	19	booked
15	15	3	2024-04-03	10:00:00.0000000	18	14	booked
16	16	4	2024-04-04	11:00:00.0000000	19	19	booked
17	17	9	2024-04-04	11:00:00.0000000	19	19	booked
18	18	17	2024-04-04	10:00:00.0000000	18	14	booked
19	19	13	2024-04-05	11:00:00.0000000	20	25	booked
20	20	10	2024-04-05	11:00:00.0000000	17	10	booked

CurrentAppointment table after executing the stored procedure. (No records corresponding to date '**2024-04-02'** present**)**

	currentappointment_id	patient_id	date	time	department_id	doctor_id	status
1	6	9	2024-04-09	10:00:00.0000000	19	16	booked
2	7	13	2024-04-09	10:00:00.0000000	19	16	booked
3	9	1	2024-04-09	10:00:00.0000000	19	16	pending
4	10	15	2024-04-10	09:00:00.0000000	18	14	booked
5	11	2	2024-04-08	09:00:00.0000000	16	1	booked
6	13	6	2024-04-03	10:00:00.0000000	18	11	booked
7	14	5	2024-04-03	10:00:00.0000000	19	19	booked
8	15	3	2024-04-03	10:00:00.0000000	18	14	booked
9	16	4	2024-04-04	11:00:00.0000000	19	19	booked
10	17	9	2024-04-04	11:00:00.0000000	19	19	booked
11	18	17	2024-04-04	10:00:00.0000000	18	14	booked
12	19	13	2024-04-05	11:00:00.0000000	20	25	booked
13	20	10	2024-04-05	11:00:00.0000000	17	10	booked

Figure 2.22 **CurrentAppointment** table data before and after execution of stored procedure **CurrentAppointmentsAndMoveToPastAppointment**

PastAppointment table before executing the stored procedure

	pastappointment_id	patient_id	date	time	department_id	doctor_id	status
1	1	1	2023-09-04	10:00:00.0000000	16	1	completed
2	2	19	2023-09-05	10:00:00.0000000	17	6	completed
3	3	2	2023-10-10	09:00:00.0000000	18	11	completed
4	4	13	2023-10-16	09:00:00.0000000	17	8	completed
5	5	15	2023-10-23	10:00:00.0000000	16	2	completed
6	6	1	2023-11-07	10:00:00.0000000	16	2	completed
7	7	9	2023-11-14	11:00:00.0000000	19	16	completed
8	8	7	2023-11-15	10:00:00.0000000	19	19	completed
9	9	15	2023-12-07	09:00:00.0000000	18	14	completed
10	10	1	2023-12-12	09:00:00.0000000	16	1	completed
11	11	15	2023-12-18	11:00:00.0000000	16	2	completed
12	12	6	2023-12-20	10:00:00.0000000	18	11	completed
13	13	3	2023-12-27	08:00:00.0000000	17	6	completed
14	14	5	2024-01-03	11:00:00.0000000	19	16	completed
15	15	15	2024-01-08	10:00:00.0000000	16	1	completed
16	16	10	2024-01-11	10:00:00.0000000	18	14	completed
17	17	4	2024-02-05	09:00:00.0000000	17	8	completed
18	18	10	2024-02-07	11:00:00.0000000	19	16	completed
19	19	17	2024-02-09	11:00:00.0000000	20	25	completed
20	20	3	2024-03-01	09:00:00.0000000	17	10	completed

PastAppointment table after executing the stored procedure (newly appended records corresponding to date '**2024-04-02' highlighted**

	pastappointment_id	patient_id	date	time	department_id	doctor_id	status
6	6	1	2023-11-07	10:00:00.0000000	16	2	completed
7	7	9	2023-11-14	11:00:00.0000000	19	16	completed
8	8	7	2023-11-15	10:00:00.0000000	19	19	completed
9	9	15	2023-12-07	09:00:00.0000000	18	14	completed
10	10	1	2023-12-12	09:00:00.0000000	16	1	completed
11	11	15	2023-12-18	11:00:00.0000000	16	2	completed
12	12	6	2023-12-20	10:00:00.0000000	18	11	completed
13	13	3	2023-12-27	08:00:00.0000000	17	6	completed
14	14	5	2024-01-03	11:00:00.0000000	19	16	completed
15	15	15	2024-01-08	10:00:00.0000000	16	1	completed
16	16	10	2024-01-11	10:00:00.0000000	18	14	completed
17	17	4	2024-02-05	09:00:00.0000000	17	8	completed
18	18	10	2024-02-07	11:00:00.0000000	19	16	completed
19	19	17	2024-02-09	11:00:00.0000000	20	25	completed
20	20	3	2024-03-01	09:00:00.0000000	17	10	completed
21	21	1	2024-04-02	10:00:00.0000000	16	1	completed
22	22	3	2024-04-02	10:30:00.0000000	16	1	completed
23	23	6	2024-04-02	11:00:00.0000000	16	1	completed
24	24	7	2024-04-02	11:00:00.0000000	16	1	completed
25	25	15	2024-04-02	10:00:00.0000000	16	2	completed
26	26	12	2024-04-02	11:00:00.0000000	16	1	completed
27	27	10	2024-04-02	10:00:00.0000000	19	6	completed

Figure 2.23 **PastAppointment** table data before and after execution of stored procedure **CurrentAppointmentsAndMoveToPastAppointment**

2.5 Comprehensive View: Appointment History for All Doctors with Department, Specialty, and Feedback Details

Task 1: Part 2:

5 The hospital wants to view the appointment date and time, showing all previous and current appointments for all doctors, and including details of the department (the doctor is associated with), doctor's specialty, and any associate review/feedback given for a doctor. You should create a view containing all the required information.

In a hospital GP management database design, it's often necessary to provide A comprehensive view of all doctor appointments, including details such as the doctor's name, department, specialty, appointment date and time, and any associated reviews or feedback. Three different views based on **Doctor**, **Department**, **CurrentAppointment**, **PastAppointment**, and **ReviewFeedback** tables are created to give the required information, only one is presented here, and the other two are included in Appendix D.

```
DROP VIEW IF EXISTS AllDoctorsAppointmentDetailsView;
CREATE VIEW AllDoctorsAppointmentDetailsView AS
SELECT
    d.doctor_id,
    CONCAT(d.first_name, ' ', ISNULL(d.middle_name, ''), ' ', d.last_name) AS DoctorName,
    dep.specialization AS Department,
    d.sub_specialization AS DoctorSubSpecialization,
    appt.date AS AppointmentDate,
    appt.time AS AppointmentTime,
    CASE WHEN pa.pastappointment_id IS NOT NULL THEN rf.review_text ELSE NULL END AS ReviewText,
    CASE WHEN pa.pastappointment_id IS NOT NULL THEN rf.rating ELSE NULL END AS Rating
FROM
    Doctor d
LEFT JOIN
    Department dep ON d.department_id = dep.department_id
LEFT JOIN (
    SELECT doctor_id, date, time
    FROM CurrentAppointment
    UNION ALL
    SELECT doctor_id, date, time
    FROM PastAppointment
) AS appt ON d.doctor_id = appt.doctor_id
LEFT JOIN
    ReviewFeedback rf ON d.doctor_id = rf.doctor_id
LEFT JOIN
    PastAppointment pa ON d.doctor_id = pa.doctor_id AND pa.date = appt.date
WHERE
    (pa.pastappointment_id = rf.pastappointment_id OR pa.pastappointment_id IS NULL)
    AND appt.date IS NOT NULL
    AND appt.time IS NOT NULL;

SELECT * FROM AllDoctorsAppointmentDetailsView;
```

Figure 2.24 T-SQL statements for View **AllDoctorsAppointmentDetailsView**

In this view, the doctor without any appointments so far is not included. Such views are listed in Appendix D. The related figures are Figure 2.24 and 2.25 respectively.

	doctor_id	DoctorName	Department	DoctorSubSpecialization	AppointmentDate	Appoi...	ReviewText	Rating
1	1	John Smith Ava	Oncology	medical oncology	2024-04-02	10:00...	NULL	NULL
2	1	John Smith Ava	Oncology	medical oncology	2024-04-02	10:00...	NULL	NULL
3	1	John Smith Ava	Oncology	medical oncology	2024-04-02	10:00...	NULL	NULL
4	1	John Smith Ava	Oncology	medical oncology	2024-04-02	10:30...	NULL	NULL
5	1	John Smith Ava	Oncology	medical oncology	2024-04-02	10:30...	NULL	NULL
6	1	John Smith Ava	Oncology	medical oncology	2024-04-02	10:30...	NULL	NULL
7	1	John Smith Ava	Oncology	medical oncology	2024-04-02	11:00...	NULL	NULL
8	1	John Smith Ava	Oncology	medical oncology	2024-04-02	11:00...	NULL	NULL
9	1	John Smith Ava	Oncology	medical oncology	2024-04-02	11:00...	NULL	NULL
10	1	John Smith Ava	Oncology	medical oncology	2024-04-02	11:00...	NULL	NULL
11	1	John Smith Ava	Oncology	medical oncology	2024-04-02	11:00...	NULL	NULL
12	1	John Smith Ava	Oncology	medical oncology	2024-04-02	11:00...	NULL	NULL
13	2	Alice Daniel ...	Oncology	radiation oncology	2024-04-02	10:00...	NULL	NULL
14	2	Alice Daniel ...	Oncology	radiation oncology	2024-04-02	10:00...	NULL	NULL
15	2	Alice Daniel ...	Oncology	radiation oncology	2024-04-02	10:00...	NULL	NULL
16	16	Amelia Aubr...	Neurology	Clinical Neurophysiol...	2024-04-09	10:00...	NULL	NULL
17	16	Amelia Aubr...	Neurology	Clinical Neurophysiol...	2024-04-09	10:00...	NULL	NULL
18	16	Amelia Aubr...	Neurology	Clinical Neurophysiol...	2024-04-09	10:00...	NULL	NULL
19	16	Amelia Aubr...	Neurology	Clinical Neurophysiol...	2024-04-09	10:00...	NULL	NULL
20	16	Amelia Aubr...	Neurology	Clinical Neurophysiol...	2024-04-09	10:00...	NULL	NULL
21	16	Amelia Aubr...	Neurology	Clinical Neurophysiol...	2024-04-09	10:00...	NULL	NULL
22	1	John Smith Ava	Oncology	medical oncology	2024-04-02	11:00...	NULL	NULL
23	1	John Smith Ava	Oncology	medical oncology	2024-04-02	11:00...	NULL	NULL

	doctor_id	DoctorName	Department	DoctorSubSpecialization	AppointmentDate	Appoi...	ReviewText	Rating
24	1	John Smith Ava	Oncology	medical oncology	2024-04-02	11:00...	NULL	NULL
25	16	Amelia Aubr...	Neurology	Clinical Neurophysiol...	2024-04-09	10:00...	NULL	NULL
26	16	Amelia Aubr...	Neurology	Clinical Neurophysiol...	2024-04-09	10:00...	NULL	NULL
27	16	Amelia Aubr...	Neurology	Clinical Neurophysiol...	2024-04-09	10:00...	NULL	NULL
28	14	Isabella Andr...	Cardiology	Heart Failure	2024-04-10	09:00...	NULL	NULL
29	14	Isabella Andr...	Cardiology	Heart Failure	2024-04-10	09:00...	NULL	NULL
30	1	John Smith Ava	Oncology	medical oncology	2024-04-08	09:00...	NULL	NULL
31	1	John Smith Ava	Oncology	medical oncology	2024-04-08	09:00...	NULL	NULL
32	1	John Smith Ava	Oncology	medical oncology	2024-04-08	09:00...	NULL	NULL
33	6	Sophia Addis...	Gastroent...	Gastrointestinal Endos...	2024-04-02	10:00...	NULL	NULL
34	6	Sophia Addis...	Gastroent...	Gastrointestinal Endos...	2024-04-02	10:00...	NULL	NULL
35	11	Ethan Allison...	Cardiology	Interventional Cardiol...	2024-04-03	10:00...	NULL	NULL
36	11	Ethan Allison...	Cardiology	Interventional Cardiol...	2024-04-03	10:00...	NULL	NULL
37	19	William Bak...	Neurology	Clinical Neurophysiol...	2024-04-03	10:00...	NULL	NULL
38	14	Isabella Andr...	Cardiology	Heart Failure	2024-04-03	10:00...	NULL	NULL
39	14	Isabella Andr...	Cardiology	Heart Failure	2024-04-03	10:00...	NULL	NULL
40	19	William Bak...	Neurology	Clinical Neurophysiol...	2024-04-04	11:00...	NULL	NULL
41	19	William Bak...	Neurology	Clinical Neurophysiol...	2024-04-04	11:00...	NULL	NULL
42	14	Isabella Andr...	Cardiology	Heart Failure	2024-04-04	10:00...	NULL	NULL

	doctor_id	DoctorName	Department	DoctorSubSpecialization	AppointmentDate	Appoi...	ReviewText	Rating
43	14	Isabella Andr...	Cardiology	Heart Failure	2024-04-04	10:00...	NULL	NULL
44	25	Mason Brook...	Orthoped...	Orthopedic Trauma	2024-04-05	11:00...	NULL	NULL
45	10	Ava Allen G...	Gastroent...	Gastrointestinal Endos...	2024-04-05	11:00...	NULL	NULL
46	1	John Smith Ava	Oncology	medical oncology	2023-09-04	10:00...	Attentive, thorough, and compassionate	4
47	6	Sophia Addis...	Gastroent...	Gastrointestinal Endos...	2023-09-05	10:00...	Highly recommend!	4
48	11	Ethan Allison...	Cardiology	Interventional Cardiol...	2023-10-10	09:00...	NULL	1
49	8	Olivia Alexa...	Gastroent...	Gastrointestinal Endos...	2023-10-16	09:00...	Attentive, knowledgeable, and caring	4
50	2	Alice Daniel ...	Oncology	radiation oncology	2023-10-23	10:00...	Grateful for the positive experience	4
51	2	Alice Daniel ...	Oncology	radiation oncology	2023-11-07	10:00...	NULL	NULL
52	16	Amelia Aubr...	Neurology	Clinical Neurophysiol...	2023-11-14	11:00...	Professional, knowledgeable, and empathetic	4
53	19	William Bak...	Neurology	Clinical Neurophysiol...	2023-11-15	10:00...	NULL	NULL
54	14	Isabella Andr...	Cardiology	Heart Failure	2023-12-07	09:00...	Approachable, patient, and understanding.	4
55	1	John Smith Ava	Oncology	medical oncology	2023-12-12	09:00...	NULL	NULL
56	2	Alice Daniel ...	Oncology	radiation oncology	2023-12-18	11:00...	Excellent care from Dr	5
57	11	Ethan Allison...	Cardiology	Interventional Cardiol...	2023-12-20	10:00...	NULL	NULL
58	6	Sophia Addis...	Gastroent...	Gastrointestinal Endos...	2023-12-27	08:00...	Made me feel comfortable and well-informed.	4
59	16	Amelia Aubr...	Neurology	Clinical Neurophysiol...	2024-01-03	11:00...	Dr. provided exceptional care	5
60	1	John Smith Ava	Oncology	medical oncology	2024-01-08	10:00...	NULL	2
61	14	Isabella Andr...	Cardiology	Heart Failure	2024-01-11	10:00...	Friendly, informative, and genuinely concerned	5
62	8	Olivia Alexa...	Gastroent...	Gastrointestinal Endos...	2024-02-05	09:00...	NULL	NULL
63	16	Amelia Aubr...	Neurology	Clinical Neurophysiol...	2024-02-07	11:00...	Fantastic!	5
64	25	Mason Brook...	Orthoped...	Orthopedic Trauma	2024-02-09	11:00...	A truly wonderful doctor	5
65	10	Ava Allen G...	Gastroent...	Gastrointestinal Endos...	2024-03-01	09:00...	NULL	1

Figure 2.25 Result for View **AllDoctorsAppointmentDetailsView**

2.6 Trigger Implementation for Appointment Cancellation State Transition

Task 1: Part 2:

6 Create a trigger so that the current state of an appointment can be changed to available when it is cancelled.

Data integrity and consistency are critical when it comes to appointment data in the hospital GP management database design. It is important to automatically update the status of appointments when they are canceled, ensuring that any pending appointments for the same doctor, department, and day are appropriately adjusted.

```
--Update the 'booked' status to 'cancelled'
UPDATE CurrentAppointment
SET status = 'cancelled'
WHERE currentappointment_id = 6; -- Updating the status of currentappointment_id =6

--Creating trigger to automate the change of 'pending' status of the appointment for same
--doctor from same department on the same day of appointment cancellation

CREATE TRIGGER trg_CancelledAppointment
ON CurrentAppointment
AFTER UPDATE
AS
BEGIN
   -- Check if the status of any appointment has been changed to 'cancelled'
   IF UPDATE(status)
   BEGIN
      -- Update pending appointments for the same doctor, department, and day to 'available'
      UPDATE ca
      SET status = 'available'
      FROM CurrentAppointment ca
      JOIN inserted i ON ca.doctor_id = i.doctor_id
                  AND ca.department_id = i.department_id
                  AND ca.date = i.date
      WHERE ca.status = 'pending';
   END
END;
```

Figure 2.26 Trigger **trg_CancelledAppointment**

To address this requirement, a trigger named **trg_CancelledAppointment** is created on the **CurrentAppointment** table as shown in Figure 2.26. This trigger fires after an update operation on the **status** column of the

CurrentAppointment table. Its purpose is to check if the status of any appointment has been changed to 'cancelled' (say ‘n’ appointments cancelled). If so, it updates immediate ‘n’ pending appointments for the same doctor, from same department, on the same day to 'available'.

CurrentAppointment table before updating the status of currentappointment_id =6 (check the highlighted records)

	currentappointment_id	patient_id	date	time	department_id	doctor_id	status
1	6	9	2024-04-09	10:00:00.0000000	19	16	booked
2	7	13	2024-04-09	10:00:00.0000000	19	16	booked
3	9	1	2024-04-09	10:00:00.0000000	19	16	pending
4	10	15	2024-04-10	09:00:00.0000000	18	14	booked
5	11	2	2024-04-08	09:00:00.0000000	16	1	booked
6	13	6	2024-04-03	10:00:00.0000000	18	11	booked
7	14	5	2024-04-03	10:00:00.0000000	19	19	booked
8	15	3	2024-04-03	10:00:00.0000000	18	14	booked
9	16	4	2024-04-04	11:00:00.0000000	19	19	booked
10	17	9	2024-04-04	11:00:00.0000000	19	19	booked
11	18	17	2024-04-04	10:00:00.0000000	18	14	booked
12	19	13	2024-04-05	11:00:00.0000000	20	25	booked
13	20	10	2024-04-05	11:00:00.0000000	17	10	booked

CurrentAppointment table after updating the status of currentappointment_id=6 as ‘cancelled’ and trigger trg_CancelledAppointment being executed.

	currentappointment_id	patient_id	date	time	department_id	doctor_id	status
1	6	9	2024-04-09	10:00:00.0000000	19	16	cancelled
2	7	13	2024-04-09	10:00:00.0000000	19	16	booked
3	9	1	2024-04-09	10:00:00.0000000	19	16	available
4	10	15	2024-04-10	09:00:00.0000000	18	14	booked
5	11	2	2024-04-08	09:00:00.0000000	16	1	booked
6	13	6	2024-04-03	10:00:00.0000000	18	11	booked
7	14	5	2024-04-03	10:00:00.0000000	19	19	booked
8	15	3	2024-04-03	10:00:00.0000000	18	14	booked
9	16	4	2024-04-04	11:00:00.0000000	19	19	booked
10	17	9	2024-04-04	11:00:00.0000000	19	19	booked
11	18	17	2024-04-04	10:00:00.0000000	18	14	booked
12	19	13	2024-04-05	11:00:00.0000000	20	25	booked
13	20	10	2024-04-05	11:00:00.0000000	17	10	booked

Figure 2.27 Results before and after updating the status of currentappointment_id =6 and execution of trigger **trg_CancelledAppointment**

2.7 Identifying Completed Appointments for Gastroenterologists

Task 1: Part 2:

7 Write a select query that allows the hospital to identify the number of completed appointments with the specialty of doctors as 'Gastroenterologists'.

The provided code performs the task of identifying completed appointments for doctors specializing in Gastroenterology and also calculates the total count of such appointments.

It is essential to monitor and analyze the workload and performance of doctors in the Department of 'Gastroenterology'. It helps the hospital management to assess the efficiency of these specialist doctors and allocate resources accordingly. Various codes are executed to perform the same with results displayed differently. Select queries and a stored procedure is written to perform the task. All these SQL codes utilize the tables **PastAppointment**, **Doctor**, and **Department** tables to retrieve relevant information such as appointment details, doctor names, and department specialization. It filters the results to include only appointments with the status 'completed' and doctors specializing in Gastroenterology. These kinds of queries are significant for effective resource planning, performance evaluation, and identifying areas for improvement.

```
--count and department specialization
    SELECT
    pa.department_id,
    dept.specialization AS DepartmentSpecialization,
    COUNT(*) AS Completed_Appointments
FROM
    PastAppointment pa
JOIN
    Doctor d ON pa.doctor_id = d.doctor_id
JOIN
    Department dept ON d.department_id = dept.department_id
WHERE
    dept.specialization = 'Gastroenterology'
    AND pa.status = 'completed'
GROUP BY
    pa.department_id, dept.specialization;
```

	department_id	DepartmentSpecialization	Completed_Appointments
1	17	Gastroenterology	5

Figure 2.28 Listing out the count of completed appointments of doctors having specialization in Gastroenterology

```sql
-- past appointments for doctors having specialization 'Gastroenterology'
SELECT pa.*
FROM PastAppointment pa
JOIN Doctor d ON pa.doctor_id = d.doctor_id
JOIN Department dept ON d.department_id = dept.department_id
WHERE dept.specialization = 'Gastroenterology'
  AND pa.status = 'completed';
```

	pastappointment_id	patient_id	date	time	department_id	doctor_id	status
1	2	19	2023-09-05	10:00:00.0000000	17	6	completed
2	4	13	2023-10-16	09:00:00.0000000	17	8	completed
3	13	3	2023-12-27	08:00:00.0000000	17	6	completed
4	17	4	2024-02-05	09:00:00.0000000	17	8	completed
5	20	3	2024-03-01	09:00:00.0000000	17	10	completed

Figure 2.29 Listing out all the completed appointments of doctors having specialization in Gastroenterology

```sql
--past appointments corresponding to count with doctors name
SELECT
    pa.pastappointment_id, pa.patient_id, pa.date, pa.department_id, dept.specialization AS DepartmentSpecialization,
    d.doctor_id,
    CONCAT(d.first_name, ' ', ISNULL(d.middle_name, ''), ' ', d.last_name) AS DoctorName,
    d.sub_specialization AS DoctorSubSpecialization,
    pa.status
FROM
    PastAppointment pa
JOIN
    Doctor d ON pa.doctor_id = d.doctor_id
JOIN
    Department dept ON d.department_id = dept.department_id
WHERE
    dept.specialization = 'Gastroenterology'
    AND pa.status = 'completed';
```

	pastappointment_id	patient_id	date	department_id	DepartmentSpecialization	doctor_id	DoctorName	DoctorSubSpecialization	status
1	2	19	2023-09-05	17	Gastroenterology	6	Sophia Addison Brown	Gastrointestinal Endoscopy	completed
2	4	13	2023-10-16	17	Gastroenterology	8	Olivia Alexander Hernandez	Gastrointestinal Endoscopy	completed
3	13	3	2023-12-27	17	Gastroenterology	6	Sophia Addison Brown	Gastrointestinal Endoscopy	completed
4	17	4	2024-02-05	17	Gastroenterology	8	Olivia Alexander Hernandez	Gastrointestinal Endoscopy	completed
5	20	3	2024-03-01	17	Gastroenterology	10	Ava Allen Gonzalez	Gastrointestinal Endoscopy	completed

Figure 2.30 Listing out all the completed appointments of doctors having specialization in Gastroenterology with doctor details

The code illustrated in Figure 2.31 returns the table with completed appointments along with patient_id , date, and doctor details such as doctor_id, doctors' name, and their sub-specialization. A message "The number of completed appointments of doctors having specialization 'Gastroenterology is" printed along with the count.

```
-- count displayed in message, and results in table.
DECLARE @Count INT;

SELECT
   pa.pastappointment_id, pa.patient_id, pa.date, pa.department_id, dept.specialization AS DepartmentSpecialization,
   d.doctor_id,
   CONCAT(d.first_name, ' ', ISNULL(d.middle_name, ''), ' ', d.last_name) AS DoctorName,
   d.sub_specialization AS DoctorSubSpecialization,
   pa.status
FROM
   PastAppointment pa
JOIN
   Doctor d ON pa.doctor_id = d.doctor_id
JOIN
   Department dept ON d.department_id = dept.department_id
WHERE
   dept.specialization = 'Gastroenterology'
   AND pa.status = 'completed';

SELECT @Count = COUNT(*)
FROM
   PastAppointment pa
JOIN
   Doctor d ON pa.doctor_id = d.doctor_id
JOIN
   Department dept ON d.department_id = dept.department_id
WHERE
   dept.specialization = 'Gastroenterology'
   AND pa.status = 'completed';

PRINT 'The number of completed appointments of doctors having the specialization  "Gastroenterology" is ' + CAST(@Count AS VARCHAR);
```

	pastappointment_id	patient_id	date	department_id	DepartmentSpecialization	doctor_id	DoctorName	DoctorSubSpecialization	status
1	2	19	2023-09-05	17	Gastroenterology	6	Sophia Addison Brown	Gastrointestinal Endoscopy	completed
2	4	13	2023-10-16	17	Gastroenterology	8	Olivia Alexander Hernandez	Gastrointestinal Endoscopy	completed
3	13	3	2023-12-27	17	Gastroenterology	6	Sophia Addison Brown	Gastrointestinal Endoscopy	completed
4	17	4	2024-02-05	17	Gastroenterology	8	Olivia Alexander Hernandez	Gastrointestinal Endoscopy	completed
5	20	3	2024-03-01	17	Gastroenterology	10	Ava Allen Gonzalez	Gastrointestinal Endoscopy	completed

```
(5 rows affected)
The number of completed appointments of doctors having the specialization  'Gastroenterology' is 5

Completion time: 2024-04-03T00:23:02.9124837+01:00
```

Figure 2.31 Result showing all the completed appointments of doctors having specialization in Gastroenterology with a message giving the total count

A stored procedure **GetGastroenterologyAppointments** as shown in Figure 2.32 for the same task improves modularity, reusability, and maintainability. In addition to that the stored procedure helps improve security, performance optimization, and centralized management of database operations. Considering these benefits, using a stored procedure for this task is preferred over a simple select query.

```sql
--stored procedure for same
CREATE PROCEDURE GetGastroenterologyAppointments
AS
BEGIN
   DECLARE @Count INT;
      -- Retrieve appointment details for Gastroenterology department
   SELECT
      pa.pastappointment_id, pa.patient_id, pa.date, pa.department_id, dept.specialization AS DepartmentSpecialization,
      d.doctor_id,
      CONCAT(d.first_name, ' ', ISNULL(d.middle_name, ''), ' ', d.last_name) AS DoctorName,
      d.sub_specialization AS DoctorSubSpecialization,
      pa.status
   FROM
      PastAppointment pa
   JOIN
      Doctor d ON pa.doctor_id = d.doctor_id
   JOIN
      Department dept ON d.department_id = dept.department_id
   WHERE
      dept.specialization = 'Gastroenterology'
      AND pa.status = 'completed';

   -- Retrieve count of completed appointments for Gastroenterology department
   SELECT @Count = COUNT(*)
   FROM
      PastAppointment pa
   JOIN
      Doctor d ON pa.doctor_id = d.doctor_id
   JOIN
      Department dept ON d.department_id = dept.department_id
   WHERE
      dept.specialization = 'Gastroenterology'
      AND pa.status = 'completed';
         -- Print the count of completed appointments
   PRINT 'The number of completed appointments of doctors having the specialization "Gastroenterology" is ' + CAST(@Count AS VARCHAR);
END
EXEC GetGastroenterologyAppointments;
```

	pastappointment_id	patient_id	date	department_id	DepartmentSpecialization	doctor_id	DoctorName	DoctorSubSpecialization	status
1	2	19	2023-09-05	17	Gastroenterology	6	Sophia Addison Brown	Gastrointestinal Endoscopy	completed
2	4	13	2023-10-16	17	Gastroenterology	8	Olivia Alexander He...	Gastrointestinal Endoscopy	completed
3	13	3	2023-12-27	17	Gastroenterology	6	Sophia Addison Brown	Gastrointestinal Endoscopy	completed
4	17	4	2024-02-05	17	Gastroenterology	8	Olivia Alexander He...	Gastrointestinal Endoscopy	completed
5	20	3	2024-03-01	17	Gastroenterology	10	Ava Allen Gonzalez	Gastrointestinal Endoscopy	completed

```
(5 rows affected)
The number of completed appointments of doctors having the specialization 'Gastroenterology' is 5

Completion time: 2024-04-03T00:28:18.4474551+01:00
```

Figure 2.32 Stored procedure **GetGastroenterologyAppointments and r**esult showing all the completed appointments of doctors having specialization in Gastroenterology with a message giving the total count

2.8 Conclusions

The hospital GP management database is designed with appropriate and necessary tables, constraints, stored procedures, user-defined functions, views, and triggers to efficiently manage the data associated with the web-based GP registration -appointment booking application.

The significant functionalities included in the design are:

- Efficient management of patient appointments, both current and past, with the ability to update appointment statuses and move completed appointments to the past.
- Detailed tracking of medical records- diagnoses, medicines, and allergies, for each patient.
- Flexible management of department, doctor, and doctor schedule data.
- Ability to retrieve appointment details for specific doctors, departments, and specialties, along with patient feedback and review ratings.
- Inclusion of constraints in various tables to ensure data integrity.
- Setting isolation level in the transactions to ensure data concurrency
- Fine-grained access control and enhanced security through role-based permissions, ensuring data integrity and confidentiality.
- Hashing of passwords to prevent data breaches.
- Executing periodic maintenance, backup, and restoration plans

In general, the hospital GP management database design offers a robust and scalable platform for managing the GP practice in the hospital, facilitating streamlined operations, improved patient experience, and effective decision-making.

Part 3: References and Appendices

References

Elmasri, R., & Navathe, S. (2016). *Fundamentals of database systems* (Seventh edition). Pearson.

Huawei Technologies Co., Ltd. (2023). *Database Principles and Technologies – Based on Huawei GaussDB*. Springer Nature Singapore. https://doi.org/10.1007/978-981-19-3032-4

Poon, E. G., Wang, S. J., Gandhi, T. K., Bates, D. W., & Kuperman, G. J. (2003). Design and implementation of a comprehensive outpatient Results Manager. *Journal of Biomedical Informatics*, *36*(1–2), 80–91. https://doi.org/10.1016/S1532-0464(03)00061-3

Appendix A

1.8.1 Review of Normalized Tables and Relationships

Table 1 SQL codes for creating the tables in the GP management database

Table: Patient

```
-- Create Patient Table
CREATE TABLE Patient (
    patient_id INT IDENTITY(1,1) NOT NULL PRIMARY KEY,
    first_name NVARCHAR(50) NOT NULL,
    middle_name NVARCHAR(50),
    last_name NVARCHAR(50) NOT NULL,
    dob DATE NOT NULL,
    insurance NVARCHAR(50),
    gender NVARCHAR(50) NOT NULL, -- Assuming 'M' for Male ,'F' for Female and 'Others' for Non-binary
    blood_group NVARCHAR (10),
    address_id INT,
    date_left DATE,
    CONSTRAINT CHK_Gender CHECK (Gender IN ('M', 'F' , 'Others')),
    CONSTRAINT CHK_BloodGroup CHECK (blood_group IN ('A+', 'A-', 'B+', 'B-', 'AB+', 'AB-', 'O+', 'O-', 'I dont know'))
);
GO
-- Alter the Patient table to modify the address_id column
ALTER TABLE Patient
DROP COLUMN address_id;
```

Table: Address

```
-- Create Patient Address Table
CREATE TABLE Address (
    address_id INT IDENTITY NOT NULL PRIMARY KEY,
    address1 nvarchar(50) NOT NULL,
    address2 nvarchar(50) NULL,
    city NVARCHAR(50) NULL,
    county NVARCHAR(50) NULL,
    postcode NVARCHAR(10) NOT NULL,
);
GO
-- Step 1: Add the patient_id column to the Address table
ALTER TABLE Address
ADD patient_id INT NULL;

-- Step 2: Add a foreign key constraint to link the patient_id column in the Address table to the patient_id column in the Patient table
ALTER TABLE Address
ADD CONSTRAINT FK_Address_Patient FOREIGN KEY (patient_id) REFERENCES Patient(patient_id);

-- Update the patient_id column with the values from the address_id column
UPDATE Address
SET patient_id = address_id;
```

Table: PatientPortalInfo

```
-- Create PatientPortalInfo Table
CREATE TABLE PatientPortalInfo (
    username NVARCHAR(50) NOT NULL PRIMARY KEY,
    patient_id INT FOREIGN KEY REFERENCES Patient(patient_id),
    pass_word NVARCHAR(50),
    email_address NVARCHAR(100) UNIQUE,
    telephone_number NVARCHAR(20),
    CONSTRAINT CHK_EmailOrPhoneNumber CHECK (
        (email_address IS NOT NULL AND email_address LIKE '%_@_%._%')
        OR
        (telephone_number IS NOT NULL)
    )
);
GO


-- Alter the table to change the pass_word column data type
ALTER TABLE PatientPortalInfo
ALTER COLUMN pass_word BINARY(64);

-- Convert existing passwords from NVARCHAR to NVARBINARY
UPDATE PatientPortalInfo
SET pass_word = CONVERT(BINARY(64), pass_word);
```

Table: Department

```
-- Create Department Table
CREATE TABLE Department (
    department_id TINYINT IDENTITY(1,1) NOT NULL PRIMARY KEY,
    specialization NVARCHAR(100),
    building_name NVARCHAR(10),
    telephone_no VARCHAR(20)
);
GO
```

Table: Doctor

```
-- Create Doctor Table
CREATE TABLE Doctor (
    doctor_id INT IDENTITY (1,1) NOT NULL PRIMARY KEY,
    first_name NVARCHAR(50) NOT NULL,
    middle_name NVARCHAR(50),
    last_name NVARCHAR(50) NOT NULL,
    sub_specialization NVARCHAR(100),
    room_no NVARCHAR(10),
    department_id TINYINT FOREIGN KEY REFERENCES Department(department_id)
);
GO
```

Table: DoctorSchedule

```
-- Create DoctorSchedule Table
CREATE TABLE DoctorSchedule (
    schedule_id INT NOT NULL PRIMARY KEY,
    doctor_id INT FOREIGN KEY REFERENCES Doctor(doctor_id),
    day_of_week NVARCHAR(10),
    available_timing TIME,
    bookings_remaining TINYINT
);
GO

ALTER TABLE DoctorSchedule
DROP COLUMN available_timing;

ALTER TABLE DoctorSchedule
ADD start_time TIME,
    end_time TIME;
```

Table: PastAppointment

```
-- Create PastAppointment Table
CREATE TABLE PastAppointment (
    pastappointment_id INT IDENTITY (1,1) NOT NULL PRIMARY KEY,
    patient_id INT FOREIGN KEY REFERENCES Patient(patient_id),
    date DATE,
    time TIME,
    department_id TINYINT FOREIGN KEY REFERENCES Department(department_id),
    doctor_id INT FOREIGN KEY REFERENCES Doctor(doctor_id),
    status NVARCHAR(20)
);
GO
-- Add Constraint to Check Past Appointment Date is in the Past and Status is Completed
ALTER TABLE PastAppointment
ADD CONSTRAINT CHK_PastAppointmentStatus CHECK (date < CAST(GETDATE() AS DATE) AND status = 'completed');

-- Alter the table to make the date column not null
ALTER TABLE PastAppointment
ALTER COLUMN date DATE NOT NULL;

-- Alter the table to make the time column not null
ALTER TABLE PastAppointment
ALTER COLUMN time TIME NOT NULL;
```

Table: MedicalRecord

```
-- Create MedicalRecord Table
CREATE TABLE MedicalRecord (
    record_id INT IDENTITY (1,1) NOT NULL PRIMARY KEY,
    patient_id INT FOREIGN KEY REFERENCES Patient(patient_id)
);
GO

-- Alter the MedicalRecord table to include a column for pastappointment_id
ALTER TABLE MedicalRecord
ADD pastappointment_id INT FOREIGN KEY REFERENCES PastAppointment(pastappointment_id);

   SELECT *
FROM MedicalRecord;
```

Table: Diagnosis

```
-- Create Diagnosis Table
CREATE TABLE Diagnosis (
    diagnosis_id INT IDENTITY (1,1) NOT NULL PRIMARY KEY,
    record_id INT FOREIGN KEY REFERENCES MedicalRecord(record_id),
    diagnosis NVARCHAR(100),
    doctor_id INT FOREIGN KEY REFERENCES Doctor(doctor_id)
);
GO
```

Table: Medicine

```
-- Create Medicine Table
CREATE TABLE Medicine (
    medicine_id INT IDENTITY (1,1) NOT NULL PRIMARY KEY,
    record_id INT FOREIGN KEY REFERENCES MedicalRecord(record_id),
    medicine_name NVARCHAR(100),
    doctor_id INT FOREIGN KEY REFERENCES Doctor(doctor_id)
);
GO
```

Table: Allergy

```
-- Create Allergy Table
CREATE TABLE Allergy (
    allergy_id INT IDENTITY (1,1) NOT NULL PRIMARY KEY,
    record_id INT FOREIGN KEY REFERENCES MedicalRecord(record_id),
    patient_id INT FOREIGN KEY REFERENCES Patient(patient_id),
    allergy NVARCHAR(100),
    doctor_id INT FOREIGN KEY REFERENCES Doctor(doctor_id)
);
GO
```

Table: ReviewFeedback

```
-- Create ReviewFeedback Table
CREATE TABLE ReviewFeedback (
    review_id INT IDENTITY (1,1) NOT NULL PRIMARY KEY,
    patient_id INT FOREIGN KEY REFERENCES Patient(patient_id),
    doctor_id INT FOREIGN KEY REFERENCES Doctor(doctor_id),
    review_text NVARCHAR(1000),
    rating INT,
    CONSTRAINT CK_RatingRange CHECK (rating BETWEEN 1 AND 5) -- Constraint for rating range
);
GO


    SELECT *
FROM ReviewFeedback;

ALTER TABLE ReviewFeedback
ADD pastappointment_id INT FOREIGN KEY REFERENCES PastAppointment(pastappointment_id);
```

Table:CurrentAppointment

```
-- Create CurrentAppointment Table
CREATE TABLE CurrentAppointment (
    currentappointment_id INT IDENTITY (1,1) NOT NULL PRIMARY KEY,
    patient_id INT FOREIGN KEY REFERENCES Patient(patient_id),
    date DATE,
    time TIME,
    department_id TINYINT FOREIGN KEY REFERENCES Department(department_id),
    doctor_id INT FOREIGN KEY REFERENCES Doctor(doctor_id),
    status VARCHAR(20)
);
-- Alter the table to make the date column not null
ALTER TABLE CurrentAppointment
ALTER COLUMN date DATE NOT NULL;

-- Alter the table to make the time column not null
ALTER TABLE CurrentAppointment
ALTER COLUMN time TIME NOT NULL;
```

Appendix B
1.8.2 Refinement of Schema (Table 1.3)

24	24	Olivia	Bowman	Evans	Sports Medicine	370	20
25	25	Mason	Brooklyn	Cole	Orthopedic Trau...	371	20
26	26	Elizabeth	Brown	Long	Pediatric Neurol...	258	21
27	27	Logan	Butler	Baker	Pediatric Cardio...	259	21
28	28	Grace	Carter	Rivera	Pediatric Neurol...	260	21
29	29	Jackson	Castillo	Parker	Pediatric Oncol...	261	21
30	30	Chloe	Charlotte	Howard	Pediatric Oncol...	262	21
31	31	Sebastian	Chloe	Stewart	Gynecologic On...	226	22
32	32	Avery	Cole	Morris	Reproductive En...	227	22
33	33	Madison	Cook	Scott	Maternal-Fetal ...	228	22
34	34	Carter	Daniel	Sanchez	Reproductive En...	229	22
35	35	Scarlett	David	Morales	Gynecologic On...	230	22
36	36	Ryan	Davidson	Foster	Sleep Medicine	301	23
37	37	Sofia	Day	Powell	Interventional Pu...	302	23
38	38	Elijah	Dixon	Sullivan	Pulmonary Hype...	303	23
39	39	Liam	Eli	Butler	Pulmonary Hype...	304	23
40	40	Aria	Elijah	Gomez	Sleep Medicine	305	23
41	41	Grayson	Elizabeth	Reed	Interventional Pu...	306	23
42	42	Amelia	Ella	Cook	Cosmetic Derma...	557	24
43	43	Lucas	Emily	Morgan	Cosmetic Derma...	558	24
44	44	Harper	Emma	Fisher	Mohs Surgery	559	24
45	45	Zoe	Ethan	Thomps...	Mohs Surgery	560	24
46	46	Nathan	Evan	Bell	Pediatric Derma...	561	24
47	47	Ella	Evans	Harrison	Cornea and Refr...	247	25
48	48	Landon	Ferguson	Gibson	Cornea and Refr...	248	25
49	49	Levi	Fisher	Fuller	Cornea and Refr...	249	25
50	50	Mia	Fletcher	Ford	Vitreoretinal Sur...	250	25
51	51	Lincoln	Ford	Fletcher	Oculoplastics	251	25
52	52	Layla	Foster	Fowler	Female Urology	441	26
53	53	Jayden	Fowler	Griffin	Andrology	442	26
54	54	Peyton	Fuller	Wheeler	Urologic Oncolo...	443	26
55	55	Aubrey	Gibson	Kennedy	Urologic Oncolo...	444	26
56	56	Evan	Gomez	Porter	Female Urology	445	26
57	57	Luna	Gonzalez	Day	Otology	665	27
58	58	Eli	Grace	Bowman	Head and Neck ...	666	27
59	59	Hannah	Grayson	Davidson	Rhinology	667	27
60	60	Zoe	Green	Andrews	Otology	668	27
61	61	Brooklyn	Griffin	Stone	Rhinology	669	27
62	62	Jack	Hall	Hudson	Child and Adole...	854	28
63	63	Addison	Hannah	Dixon	Forensic Psychi...	855	28
64	64	Adrian	Harper	Richard...	Geriatric Psychi...	856	28
65	65	Aurora	Harrison	Barnes	Geriatric Psychi...	857	28

66	66	Luke	Hernandez	Wallace	Forensic Psychi...	858	28
67	67	Isabella	Hess	Castillo	Interventional R...	552	29
68	68	David	Hill	Myers	Diagnostic Radi...	553	29
69	69	Paisley	Howard	Woods	Nuclear Medicine	554	29
70	70	Jonathan	Hudson	Sullivan	Nuclear Medicine	555	29
71	71	Stella	Hughes	Adams	Diagnostic Radi...	556	29
72	72	Julian	Isabella	Jenkins	Perioperative M...	438	30
73	73	Allison	Isabella	Ferguson	Perioperative M...	437	30
74	74	Josiah	Jack	Lawson	Pain Medicine	436	30
75	75	Naomi	Jackson	Blackbu...	Critical Care M...	435	30
76	76	Jaxon	Jacob	Hess	Critical Care M...	434	30

Figure 1 Remaining data from Table Doctor

1.8.2 Refinement of Schema (Table 1.3)

24	24	12	Friday	8	10:30:00.0000000	13:30:00.0000000
25	25	13	Monday	10	09:30:00.0000000	12:30:00.0000000
26	26	13	Tuesday	12	10:30:00.0000000	13:30:00.0000000
27	27	14	Wednesday	7	08:30:00.0000000	11:30:00.0000000
28	28	14	Thursday	9	09:00:00.0000000	12:00:00.0000000
29	29	15	Friday	12	10:00:00.0000000	13:00:00.0000000
30	30	15	Monday	10	08:00:00.0000000	11:00:00.0000000
31	31	16	Tuesday	9	09:30:00.0000000	12:30:00.0000000
32	32	16	Wednesday	7	10:30:00.0000000	13:30:00.0000000
33	33	17	Thursday	12	08:30:00.0000000	11:30:00.0000000
34	34	17	Friday	8	09:00:00.0000000	12:00:00.0000000
35	35	18	Monday	11	10:00:00.0000000	13:00:00.0000000
36	36	18	Tuesday	9	08:00:00.0000000	11:00:00.0000000
37	37	19	Wednesday	6	09:30:00.0000000	12:30:00.0000000
38	38	19	Thursday	12	10:30:00.0000000	13:30:00.0000000
39	39	20	Friday	8	08:30:00.0000000	11:30:00.0000000
40	40	20	Monday	10	09:00:00.0000000	12:00:00.0000000
41	41	21	Tuesday	12	10:00:00.0000000	13:00:00.0000000
42	42	21	Wednesday	9	08:00:00.0000000	11:00:00.0000000
43	43	22	Thursday	7	09:30:00.0000000	12:30:00.0000000
44	44	22	Friday	11	10:30:00.0000000	13:30:00.0000000
45	45	23	Monday	6	08:30:00.0000000	11:30:00.0000000
46	46	23	Tuesday	14	09:00:00.0000000	12:00:00.0000000
47	47	24	Wednesday	5	11:00:00.0000000	14:00:00.0000000
48	48	24	Thursday	13	09:30:00.0000000	12:30:00.0000000
49	49	25	Friday	8	10:00:00.0000000	13:00:00.0000000
50	50	25	Monday	10	08:00:00.0000000	11:00:00.0000000
51	51	26	Tuesday	9	09:30:00.0000000	12:30:00.0000000
52	52	26	Wednesday	11	10:30:00.0000000	13:30:00.0000000
53	53	27	Thursday	7	08:30:00.0000000	11:30:00.0000000
54	54	27	Friday	13	09:00:00.0000000	12:00:00.0000000
55	55	28	Monday	6	10:00:00.0000000	13:00:00.0000000
56	56	28	Tuesday	12	08:00:00.0000000	11:00:00.0000000
57	57	29	Wednesday	8	09:30:00.0000000	12:30:00.0000000
58	58	29	Thursday	10	10:30:00.0000000	13:30:00.0000000
59	59	30	Friday	7	08:30:00.0000000	11:30:00.0000000
60	60	30	Monday	9	09:00:00.0000000	12:00:00.0000000
61	61	31	Tuesday	11	10:00:00.0000000	13:00:00.0000000
62	62	31	Wednesday	8	08:00:00.0000000	11:00:00.0000000
63	63	32	Thursday	10	09:30:00.0000000	12:30:00.0000000
64	64	32	Friday	12	10:30:00.0000000	13:30:00.0000000
65	65	33	Monday	7	08:30:00.0000000	11:30:00.0000000

66	66	33	Tuesday	9	09:00:00.0000000	12:00:00.0000000
67	67	34	Wednesday	6	10:00:00.0000000	13:00:00.0000000
68	68	34	Thursday	11	08:00:00.0000000	11:00:00.0000000
69	69	35	Friday	8	09:30:00.0000000	12:30:00.0000000
70	70	35	Monday	10	10:30:00.0000000	13:30:00.0000000
71	71	36	Tuesday	7	08:30:00.0000000	11:30:00.0000000
72	72	36	Wednesday	13	09:00:00.0000000	12:00:00.0000000
73	73	37	Thursday	6	10:00:00.0000000	13:00:00.0000000
74	74	37	Friday	12	08:00:00.0000000	11:00:00.0000000
75	75	38	Monday	8	09:00:00.0000000	12:00:00.0000000
76	76	38	Tuesday	11	10:30:00.0000000	13:30:00.0000000
77	77	39	Wednesday	7	08:30:00.0000000	11:30:00.0000000
78	78	39	Thursday	9	09:00:00.0000000	12:00:00.0000000
79	79	40	Friday	12	10:00:00.0000000	13:00:00.0000000
80	80	40	Monday	10	08:00:00.0000000	11:00:00.0000000
81	81	41	Tuesday	9	09:30:00.0000000	12:30:00.0000000
82	82	41	Wednesday	7	10:30:00.0000000	13:30:00.0000000
83	83	42	Thursday	12	08:30:00.0000000	11:30:00.0000000
84	84	42	Friday	8	09:00:00.0000000	12:00:00.0000000
85	85	43	Monday	11	10:00:00.0000000	13:00:00.0000000
86	86	43	Tuesday	9	08:00:00.0000000	11:00:00.0000000
87	87	44	Wednesday	6	09:30:00.0000000	12:30:00.0000000
88	88	44	Thursday	12	10:30:00.0000000	13:30:00.0000000
89	89	45	Friday	8	08:30:00.0000000	11:30:00.0000000
90	90	45	Monday	10	09:00:00.0000000	12:00:00.0000000

Figure 2 Remaining data from Table DoctorSchedule

Appendix C

1.13.4 Advice and Guidance to Client on Database Backup and Recovery

1. The database design team successfully performed a compressed full and differential GP management database backup as on date April 7, 2024, and the .bak files will be shared with the client.

2. The database design team successfully restored the GP management database as on date April 7, 2024, and the dialog box is shared in Figure 1 below.

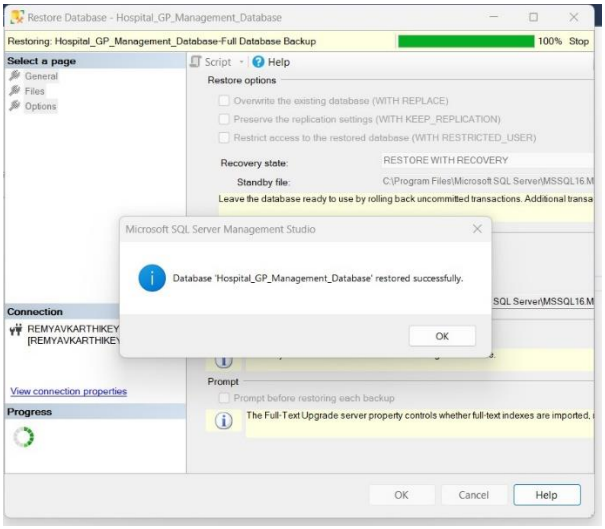


Figure 1 GP management database restore -completed successfully

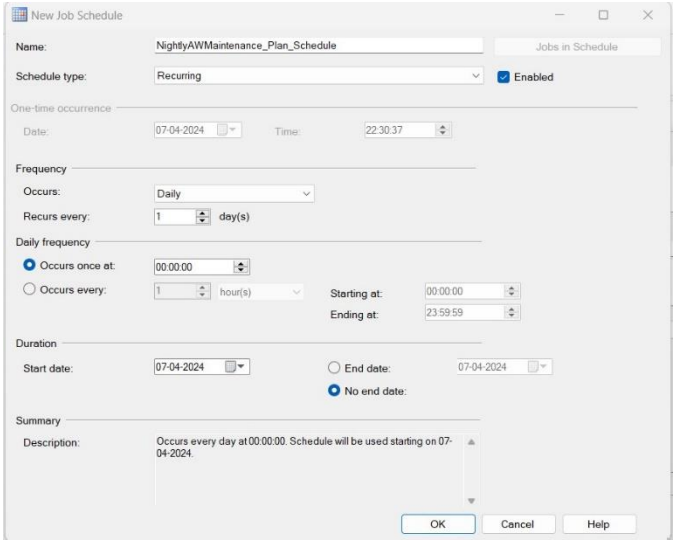


Figure 2 GP management database maintenance plan

3. A maintenance plan as illustrated in Figure 2 is established.
4. In the maintenance plan, the task order is selected as shown in Figure 3.

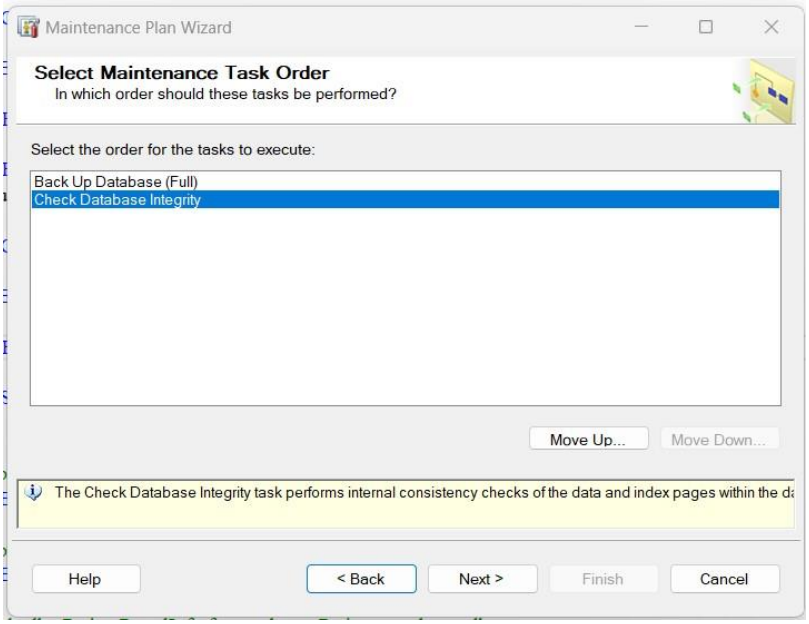


Figure 3 GP management database maintenance plan task order

5. Each of the actions in the GP management database maintenance Plan Wizard is completed successfully as shown in Figure 4.

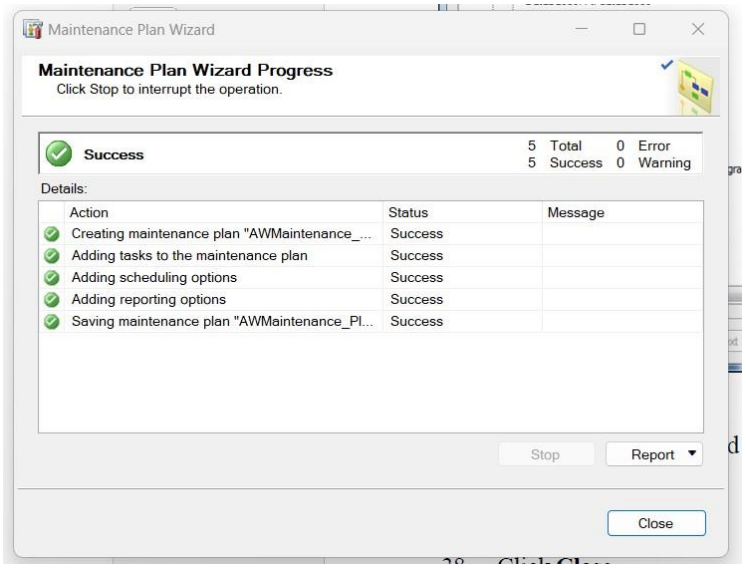


Figure 4 GP management database maintenance plan actions successfully completed

6. Restoring the GP management database to a point in time is performed successfully using the full backup file as shown in Figures 5 and 6 respectively.

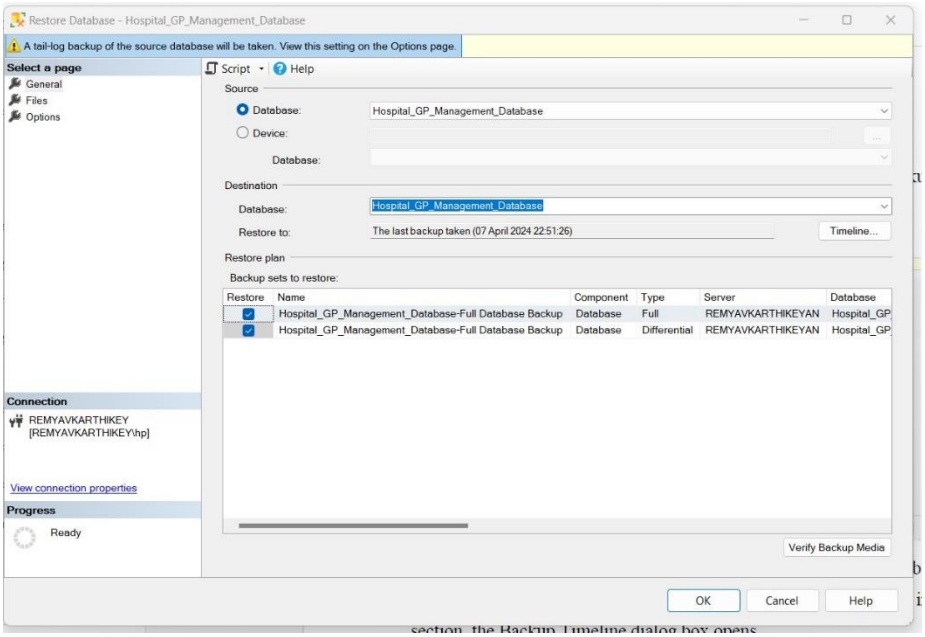


Figure 5 GP management database restoration to a point in time from the full backup file

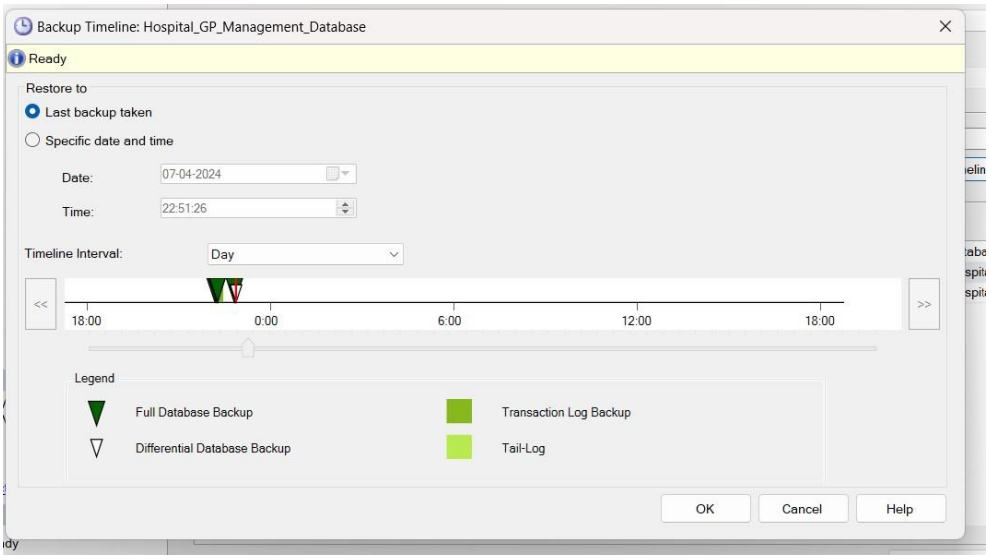


Figure 6 GP management database restoration to a point in time using the specific date and time slider

7. Ensuring the GP management database backup is not corrupt and can restore the database successfully by using the WITH CHECKSUM and RESTOREVERIFY ONLY commands as shown in Figures 7 and 8 respectively.

```
1710 BACKUP DATABASE Hospital_GP_Management_Database
1711  TO DISK ='C:\Hospital_GP_Management_Database_Restore\Hospital_GP_Management_Database_full.bak'
1712  WITH CHECKSUM
```

Messages

```
Processed 1136 pages for database 'Hospital_GP_Management_Database', file 'Hospital_GP_Management_Database' on file 4.
Processed 2 pages for database 'Hospital_GP_Management_Database', file 'Hospital_GP_Management_Database_log' on file 4.
BACKUP DATABASE successfully processed 1138 pages in 0.032 seconds (277.709 MB/sec).

Completion time: 2024-04-08T17:19:28.4548942+01:00
```

Figure 7 Use of WITH CHECKSUM command and result

```
1715 RESTORE VERIFYONLY
1716 FROM DISK ='C:\Hospital_GP_Management_Database_Restore\Hospital_GP_Management_Database_full.bak'
1717 WITH CHECKSUM;
```

Messages

```
The backup set on file 1 is valid.

Completion time: 2024-04-08T17:20:25.9803166+01:00
```

Figure 8 Use of RESTOREVERIFY ONLY command and result

Appendix D

2.5 Comprehensive View: Appointment History for All Doctors with Department, Specialty, and Feedback Details

Task 1 Part 1 Q 5

```
--View showing view the appointment date and time, showing all previous and current appointments for all doctors,
--and including details of the department (the doctor is associated with), doctor's specialty and any associate
--review/feedback given for a doctor.
DROP VIEW IF EXISTS AllDoctorsAppointmentDetailsView_1;
CREATE VIEW AllDoctorsAppointmentDetailsView_1 AS
SELECT
    d.doctor_id,
    CONCAT(d.first_name, ' ', ISNULL(d.middle_name, ''), ' ', d.last_name) AS DoctorName,
    dep.specialization AS Department,
    d.sub_specialization AS DoctorSubSpecialization,
    appt.date AS AppointmentDate,
    appt.time AS AppointmentTime,
    CASE WHEN pa.pastappointment_id IS NOT NULL THEN rf.review_text ELSE NULL END AS ReviewText,
    CASE WHEN pa.pastappointment_id IS NOT NULL THEN rf.rating ELSE NULL END AS Rating
FROM
    Doctor d
LEFT JOIN
    Department dep ON d.department_id = dep.department_id
LEFT JOIN (
    SELECT doctor_id, date, time
    FROM CurrentAppointment
    UNION ALL
    SELECT doctor_id, date, time
    FROM PastAppointment
) AS appt ON d.doctor_id = appt.doctor_id
LEFT JOIN
    ReviewFeedback rf ON d.doctor_id = rf.doctor_id
LEFT JOIN
    PastAppointment pa ON d.doctor_id = pa.doctor_id AND pa.date = appt.date
WHERE
    pa.pastappointment_id = rf.pastappointment_id OR pa.pastappointment_id IS NULL;

    SELECT * FROM AllDoctorsAppointmentDetailsView_1;
```

Figure 1 View: AllDoctorsAppointmentDetailsView_1

	doctor_id	DoctorName	Department	DoctorSubSpecialization	AppointmentDate	AppointmentTime	ReviewText	Rating
1	1	John Smith Ava	Oncology	medical oncology	2024-04-08	09:00:00.0000000	NULL	NULL
2	1	John Smith Ava	Oncology	medical oncology	2024-04-08	09:00:00.0000000	NULL	NULL
3	1	John Smith Ava	Oncology	medical oncology	2024-04-08	09:00:00.0000000	NULL	NULL
4	1	John Smith Ava	Oncology	medical oncology	2023-09-04	10:00:00.0000000	Attentive, ...	4
5	1	John Smith Ava	Oncology	medical oncology	2023-12-12	09:00:00.0000000	NULL	NULL
6	1	John Smith Ava	Oncology	medical oncology	2024-01-08	10:00:00.0000000	NULL	2
7	2	Alice Daniel ...	Oncology	radiation oncology	2023-10-23	10:00:00.0000000	Grateful f...	4
8	2	Alice Daniel ...	Oncology	radiation oncology	2023-11-07	10:00:00.0000000	NULL	NULL
9	2	Alice Daniel ...	Oncology	radiation oncology	2023-12-18	11:00:00.0000000	Excellent ...	5
10	3	Michael Lon...	Oncology	surgical oncology	NULL	NULL	NULL	NULL
11	4	Emma Adams...	Oncology	surgical oncology	NULL	NULL	NULL	NULL
12	5	William Ada...	Oncology	radiation oncology	NULL	NULL	NULL	NULL
13	6	Sophia Addis...	Gastroent...	Gastrointestinal Endos...	2023-09-05	10:00:00.0000000	Highly re...	4
14	6	Sophia Addis...	Gastroent...	Gastrointestinal Endos...	2023-12-27	08:00:00.0000000	Made me ...	4
15	7	James Adrian...	Gastroent...	Pancreatology	NULL	NULL	NULL	NULL
16	8	Olivia Alexa...	Gastroent...	Gastrointestinal Endos...	2023-10-16	09:00:00.0000000	Attentive, ...	4
17	8	Olivia Alexa...	Gastroent...	Gastrointestinal Endos...	2024-02-05	09:00:00.0000000	NULL	NULL
18	9	Daniel Alice ...	Gastroent...	Hepatology	NULL	NULL	NULL	NULL
19	10	Ava Allen G...	Gastroent...	Gastrointestinal Endos...	2024-04-05	11:00:00.0000000	NULL	NULL
20	10	Ava Allen G...	Gastroent...	Gastrointestinal Endos...	2024-03-01	09:00:00.0000000	NULL	1
21	11	Ethan Allison...	Cardiology	Interventional Cardiol...	2024-04-03	10:00:00.0000000	NULL	NULL
22	11	Ethan Allison...	Cardiology	Interventional Cardiol...	2024-04-03	10:00:00.0000000	NULL	NULL
23	11	Ethan Allison...	Cardiology	Interventional Cardiol...	2023-10-10	09:00:00.0000000	NULL	1

Figure 2 Results of View: AllDoctorsAppointmentDetailsView_1

2.5 Comprehensive View: Appointment History for All Doctors with Department, Specialty, and Feedback Details

Task 1 Part 1 Q 5

```
CREATE VIEW AllDoctorsAppointmentDetailsView_2 AS
SELECT
    d.doctor_id,
    CONCAT(d.first_name, ' ', ISNULL(d.middle_name, ''), ' ', d.last_name) AS DoctorName,
    dep.specialization AS Department,
    d.sub_specialization AS DoctorSubSpecialization,
    COUNT(appt.date) AS TotalAppointments,
    MAX(appt.date) AS LatestAppointmentDate,
    MAX(appt.time) AS LatestAppointmentTime,
    AVG(rf.rating) AS AverageRating
FROM
    Doctor d
LEFT JOIN
    Department dep ON d.department_id = dep.department_id
LEFT JOIN (
    SELECT doctor_id, date, time
    FROM CurrentAppointment
    UNION ALL
    SELECT doctor_id, date, time
    FROM PastAppointment
) AS appt ON d.doctor_id = appt.doctor_id
LEFT JOIN
    ReviewFeedback rf ON d.doctor_id = rf.doctor_id
LEFT JOIN
    PastAppointment pa ON d.doctor_id = pa.doctor_id AND pa.date = appt.date
WHERE
    (pa.pastappointment_id = rf.pastappointment_id OR pa.pastappointment_id IS NULL)
    AND appt.date IS NOT NULL
    AND appt.time IS NOT NULL
GROUP BY
    d.doctor_id,
    CONCAT(d.first_name, ' ', ISNULL(d.middle_name, ''), ' ', d.last_name),
    dep.specialization,
    d.sub_specialization;

    SELECT * FROM AllDoctorsAppointmentDetailsView_2;
```

Figure 3 View: AllDoctorsAppointmentDetailsView_2

	doctor_id	DoctorName	Department	DoctorSubSpecialization	TotalAppointments	LatestAppointmentDate	LatestAppointmentTime	AverageRating
1	1	John Smith Ava	Oncology	medical oncology	6	2024-04-08	10:00:00.0000000	3
2	2	Alice Daniel Johnson	Oncology	radiation oncology	3	2023-12-18	11:00:00.0000000	4
3	6	Sophia Addison Brown	Gastroenterology	Gastrointestinal Endoscopy	2	2023-12-27	10:00:00.0000000	4
4	8	Olivia Alexander Hernandez	Gastroenterology	Gastrointestinal Endoscopy	2	2024-02-05	09:00:00.0000000	4
5	10	Ava Allen Gonzalez	Gastroenterology	Gastrointestinal Endoscopy	2	2024-04-05	11:00:00.0000000	1
6	11	Ethan Allison Walker	Cardiology	Interventional Cardiology	4	2024-04-03	10:00:00.0000000	1
7	14	Isabella Andrews Young	Cardiology	Heart Failure	8	2024-04-10	10:00:00.0000000	4
8	16	Amelia Aubrey Lewis	Neurology	Clinical Neurophysiology	12	2024-04-09	11:00:00.0000000	4
9	19	William Baker Hill	Neurology	Clinical Neurophysiology	4	2024-04-04	11:00:00.0000000	NULL
1..	25	Mason Brooklyn Cole	Orthopedics	Orthopedic Trauma	2	2024-04-05	11:00:00.0000000	5

Figure 4 Result of View: AllDoctorsAppointmentDetailsView_2

Task 2 Part 1: Database Creation, Schema Design, and Query Implementation for Food Service Company in Microsoft SQL Server Management Studio using T-SQL

1.1 Introduction

In a food service company, consumers, restaurants, ratings, and cuisines collectively form the backbone of success. Being the heartbeat of the business, consumers' preferences, dietary requirements, and dining habits are crucial for tailoring menus, developing marketing strategies, and ensuring high levels of customer satisfaction. Restaurants, being the physical spaces should provide quality dining experiences. The influence of the restaurant's location, menu planning, staffing, and customer service are very significant in building the food service brand and its reputation. The customers' love for the food service brand, their loyalty, and satisfaction can be visibly seen from the ratings given to them by the consumers. The ratings provide insights into the quality of service, scope for improvements, and guidance to set high standards by the restaurants. The cuisines of the restaurants are also pivotal as consumer choices majorly depend on popular trends and regional preferences.

It will be a remarkable approach to gather information about consumers, restaurants, ratings, and cuisines and subsequently design a database. The food service company can diversify its business plans and proposals through the data insights from the database. The database built on the aforementioned data can help in devising plans to enhance customer satisfaction and loyalty. It can also be a driving force for restaurants to upgrade their standards, service, and facilities to reach the top rating. Overall, the database creation, schema design, and query implementation for deriving data insight or business intelligence can help the competitive food service industry players to grow and succeed.

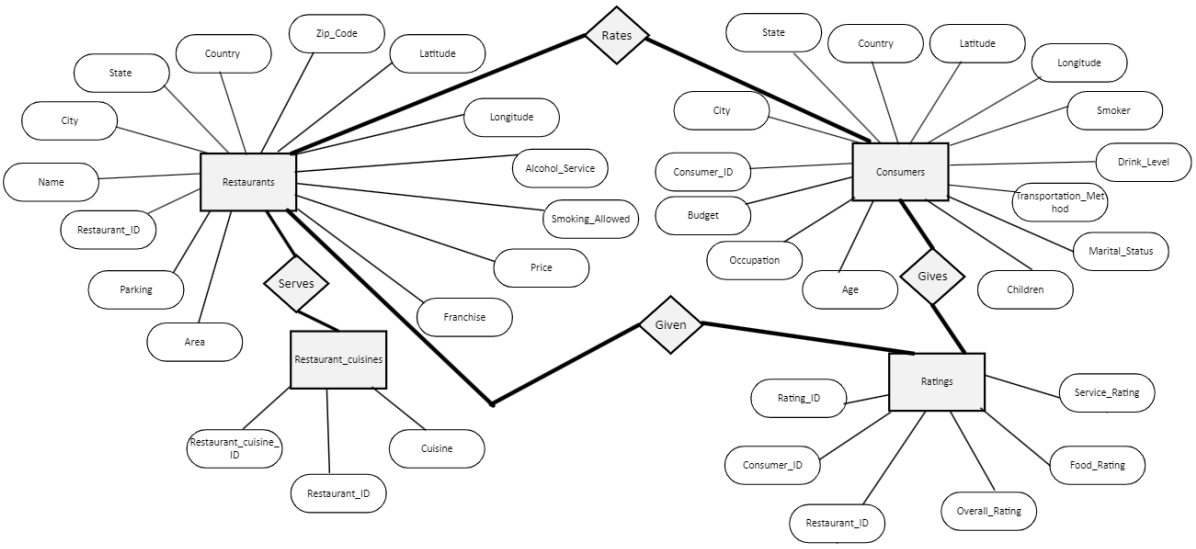


Figure 1.1 Conceptual E-R diagram

For realizing efficient and reliable food service database design, the database design takes into account the consumers, restaurants, ratings and cuisines, and the same is implemented using T-SQL statements in Microsoft SQL Server Management Studio. The report outlines the database design process, algorithm, and its implementation in Microsoft SQL server management studio using T-SQL statements.

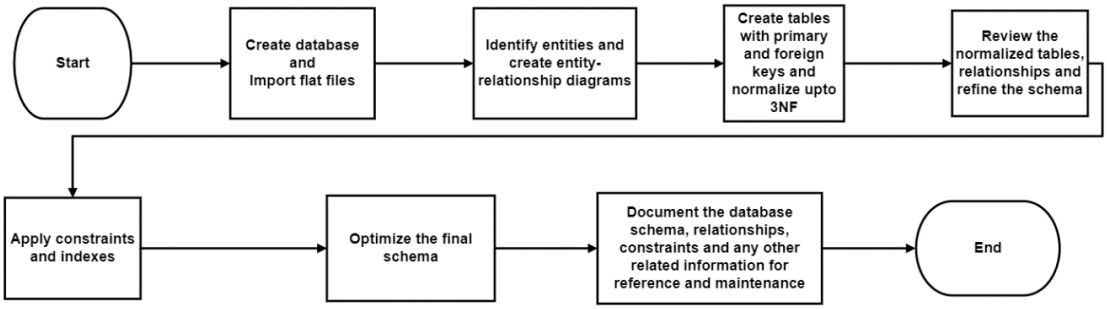


Figure 1.2 Database design flowchart

1.2 Food Service Database Design

From the csv file description given, the database design is formulated according to the conceptual E-R diagram presented in Figure 1.1. The steps of the database design can be illustrated as given in Figure 1.2.

The database design is detailed in the following sections.

1.2.1 Create database and import flat files

The four csv files correspond to the entities **Consumers**, **Restaurants**, **Ratings** and **Restaurant Cuisines**. The flat files are imported into tables within SQL server to the database created under the name **FoodserviceDB** as shown in Figure 1.3.

```
-- Create the database
CREATE DATABASE FoodserviceDB;

--Switch to the created database
USE FoodserviceDB;
GO
```

Figure 1.3 Creating **FoodserviceDB** database

Figures 1.4 to 1.9 explains the different steps involved in importing the flat files to tables (shown for consumers.csv file) into **FoodserviceDB**.

1. For importing the consumer. csv flat file- Right click on the database, select Tasks from the menu and then select Import Flat File as shown in Figure 1.4

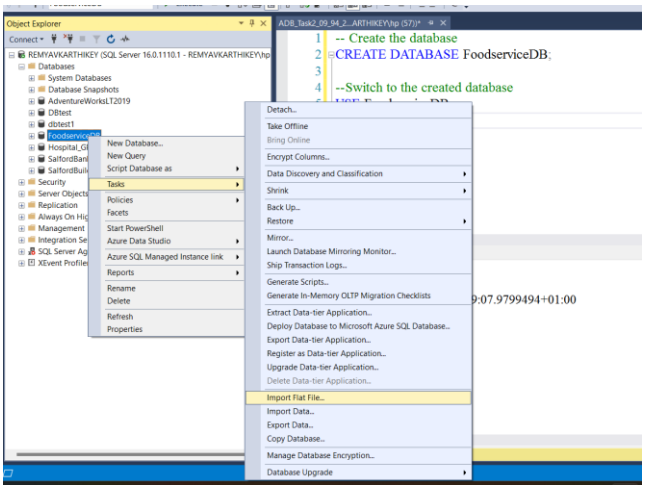


Figure 1.4 Importing flat file

2. Once this step is completed, in the window that opens click on Browse and then navigate to the csv file location.

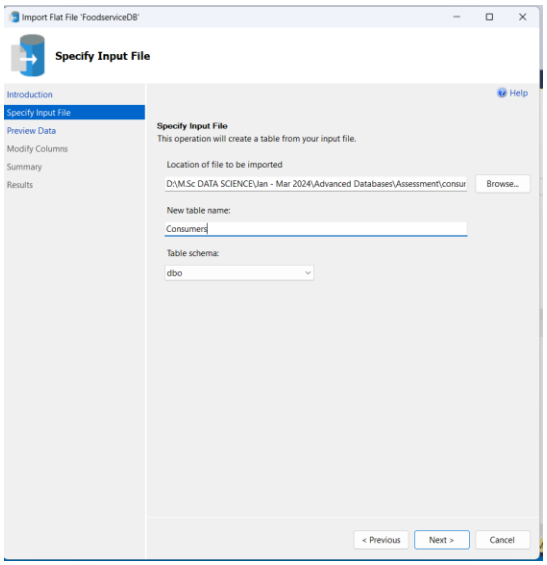


Figure 1.5 Navigating to the flat file location

3. Once the csv file is selected, a preview of the file importing is obtained. Selected 'Use Rich Data Type Detection'

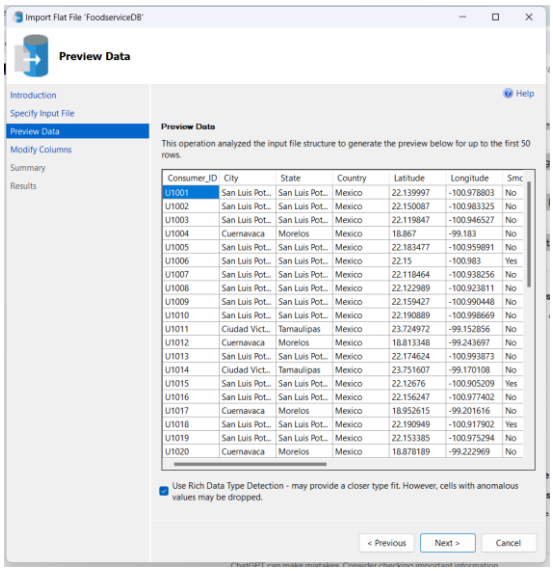


Figure 1.6 Preview of the csv file

4. Select the datatypes for each of the columns in the csv file. Refer Table 1.1 to 1.4 for data type selection.

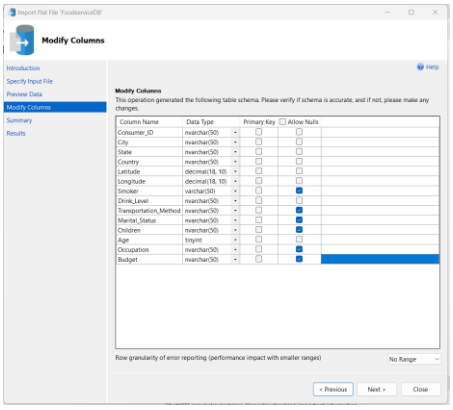


Figure 1.7 Selection of data types.

5. Run the import

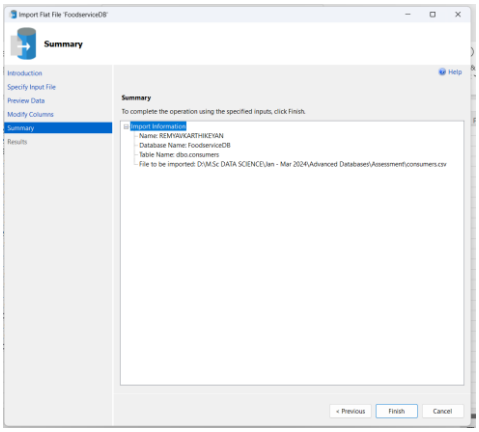


Figure 1.8 Import operation running

6. Import completion notified on screen.

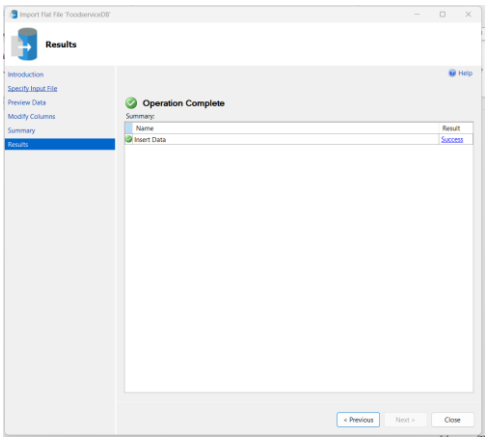


Figure 1.9 Import complete

Table 1.1 details about the selection of data type for the attributes in each flat file namely **Consumers**, **Restaurants**, **Ratings** and **Restaurant Cuisines.**

Table 1.1 Data types of the attributes in the flat files

Consumers

Column Name	Data Type	Allow Nu
Consumer_ID	nvarchar(50)	☐
City	nvarchar(50)	☐
State	nvarchar(50)	☐
Country	nvarchar(50)	☐
Latitude	decimal(18, 10)	☐
Longitude	decimal(18, 10)	☐
Smoker	varchar(50)	☑
Drink_Level	nvarchar(50)	☐
Transportation_Method	nvarchar(50)	☑
Marital_Status	nvarchar(50)	☑
Children	nvarchar(50)	☑
Age	tinyint	☐
Occupation	nvarchar(50)	☑
Budget	nvarchar(50)	☑
		☐

Restaurants

Column Name	Data Type	Allow Null
Restaurant_ID	int	☐
Name	nvarchar(100)	☐
City	nvarchar(50)	☐
State	nvarchar(50)	☐
Country	nvarchar(50)	☐
Zip_Code	nvarchar(50)	☑
Latitude	decimal(10, 7)	☐
Longitude	decimal(10, 7)	☐
Alcohol_Service	nvarchar(50)	☐
Smoking_Allowed	nvarchar(50)	☐
Price	nvarchar(50)	☐
Franchise	nvarchar(3)	☐
Area	nvarchar(50)	☐
Parking	nvarchar(50)	☐

Ratings

Column Name	Data Type	Allow Nulls
Consumer_ID	nvarchar(50)	☐
Restaurant_ID	int	☐
Overall_Rating	tinyint	☐
Food_Rating	tinyint	☐
Service_Rating	tinyint	☐
		☐

Restaurant Cuisines

Column Name	Data Type	Allow Nulls
Restaurant_ID	int	☐
Cuisine	nvarchar(50)	☐
		☐

All the attributes that take text data are given **NVARCHAR** data type.

The latitude and longitude are given **DECIMAL** datatype.

Age is given **TINYINT** data type.

Ratings are given **TINYINT.**

Nulls are allowed based on the flat files.

The datatypes are optimized while creating the tables.

1.2.2 Identify entities and create entity-relationship diagram

According to E-R diagram given in Figure 1.1, the entities and relationships are identified.

Entiities : **Consumers**, **Restaurants**, **Ratings** and **Restaurant Cuisines**

Relationships:

Consumers rate Restaurants.

Restaurants are given Ratings.

Consumer gives Ratings.

Restaurants serves Cuisines.

1.2.3 Create tables with primary and foreign keys and normalize upto 3NF

Prior to table creation, the primary keys and foreign keys considerations are analysed. In order to establish the relationships between the entities, foreign keys are introduced in tables **Ratings** and **Restaurant_Cuisines.** For identifying the specific records of the tables **Ratings** and **Restaurant_Cuisines,** Rating_ID and Restaurant_cuisine_ID are introduced in the newly created tables corresponding to each of the flat file tables. The database is normalized up to 3NF through the introduction of primary and foreign keys. Figure 1. 10 gives the codes for creating the tables.

Consumers

```
--creat table Consumers
CREATE TABLE Consumers (
Consumer_ID nvarchar(10) NOT NULL PRIMARY KEY,
City nvarchar(20) NOT NULL,
State nvarchar(20) NOT NULL,
Country nvarchar(20) NOT NULL,
Latitude decimal(18,10)  NOT NULL,
Longitude decimal(18,10)  NOT NULL,
Smoker varchar(3),
Drink_Level nvarchar(20) NOT NULL,
Transportation_Method nvarchar(20),
Marital_Status nvarchar(50),
Children nvarchar(20),
Age tinyint NOT NULL CHECK (Age BETWEEN 1 AND 100),
Occupation nvarchar(20),
Budget nvarchar(20),
);
```

Restaurants

```
--Create table Restaurant
CREATE TABLE Restaurants (
Restaurant_ID int NOT NULL PRIMARY KEY,
Name nvarchar(100) NOT NULL,
City nvarchar(20) NOT NULL,
State nvarchar(20) NOT NULL,
Country nvarchar(20) NOT NULL,
Zip_Code nvarchar(10),
Latitude decimal(18,10)  NOT NULL,
Longitude decimal(18,10)  NOT NULL,
Alcohol_Service nvarchar(20) NOT NULL,
Smoking_Allowed nvarchar(20) NOT NULL,
Price nvarchar(10) NOT NULL,
Franchise nvarchar(4) NOT NULL,
Area nvarchar(10) NOT NULL,
Parking nvarchar(10) NOT NULL,
);
```

Ratings

```
--Create table Ratings
CREATE TABLE Ratings (
Rating_ID int IDENTITY (1,1) NOT NULL PRIMARY KEY,
Consumer_ID  nvarchar(10) NOT NULL FOREIGN KEY (Consumer_ID) REFERENCES Consumers (Consumer_ID),
Restaurant_ID int NOT NULL FOREIGN KEY (Restaurant_ID) REFERENCES Restaurants (Restaurant_ID),
Overall_Rating tinyint NOT NULL CHECK (Overall_Rating BETWEEN 0 AND 2),
Food_Rating tinyint NOT NULL CHECK (Food_Rating BETWEEN 0 AND 2),
Service_Rating tinyint NOT NULL CHECK (Service_Rating BETWEEN 0 AND 2),
);
```

Restaurant Cuisines

```
--Create table Restaurant_cuisines
CREATE TABLE Restaurant_cuisines (
Restaurant_cuisine_ID int IDENTITY (1,1) NOT NULL PRIMARY KEY,
Restaurant_ID int NOT NULL FOREIGN KEY (Restaurant_ID) REFERENCES Restaurants (Restaurant_ID),
Cuisine nvarchar(20) NOT NULL
);
```

Figure 1.10 Codes for table creation

Table 1.2 shows the how the datatypes are optimized when compared to table 1.1 for saving the memory usage.

Table 1.2 Data types of the attributes in tables created

Consumers

Column Name	Data Type	Allow Nulls
Consumer_ID	nvarchar(10)	☐
City	nvarchar(20)	☐
State	nvarchar(20)	☐
Country	nvarchar(20)	☐
Latitude	decimal(18, 10)	☐
Longitude	decimal(18, 10)	☐
Smoker	varchar(3)	☑
Drink_Level	nvarchar(20)	☐
Transportation_Method	nvarchar(20)	☑
Marital_Status	nvarchar(50)	☑
Children	nvarchar(20)	☑
Age	tinyint	☐
Occupation	nvarchar(20)	☑
Budget	nvarchar(20)	☑
		☐

Restaurants

Column Name	Data Type	Allow Nulls
Restaurant_ID	int	☐
Name	nvarchar(100)	☐
City	nvarchar(20)	☐
State	nvarchar(20)	☐
Country	nvarchar(20)	☐
Zip_Code	nvarchar(10)	☑
Latitude	decimal(18, 10)	☐
Longitude	decimal(18, 10)	☐
Alcohol_Service	nvarchar(20)	☐
Smoking_Allowed	nvarchar(20)	☐
Price	nvarchar(10)	☐
Franchise	nvarchar(4)	☐
Area	nvarchar(10)	☐
Parking	nvarchar(10)	☐
		☐

Ratings

Column Name	Data Type	Allow Nulls
Rating_ID	int	☐
Consumer_ID	nvarchar(10)	☐
Restaurant_ID	int	☐
Overall_Rating	tinyint	☐
Food_Rating	tinyint	☐
Service_Rating	tinyint	☐
		☐

Restaurant Cuisines

Column Name	Data Type	Allow Nulls
Restaurant_cuisine_ID	int	☐
Restaurant_ID	int	☐
Cuisine	nvarchar(20)	☐
		☐

NVARCHAR data types are optimized for efficient use of memory.

1.2.3.1 Normalize the tables to First Normal Form (1NF)

The attributes in tables **Consumers, Restaurants, Ratings** and **Restaurant Cuisines** are atomic, single valued and obeys the conditions of First Normal Form.

Table: Consumers

Consumer_ID (primary key)	City	State	Country	Latitude	Longitude	Smoker	Drink Level

Transportation Method	Marital Status	Children	Age	Occupation	Budget

Table: Restaurants

Restaurant_ID (primary key)	Name	City	State	Country	Zip code	Latitude	Longitude
Alcohol Service	Smoke allowed	Price	Franchise	Area	Parking		

Table: Ratings

Consumer_ID	Restaurant_ID	Overall _rating	Food _Rating	Service _Rating

Table: Resaturant_Cuisines

Restaurant_ID	Cuisine

1.2.3.2 Normalize the tables to Second Normal Form (2NF)

Table: Ratings

Primary key **Rating_id** introduced for full dependency.

Rating_ID (primary key)	**Consumer_ID** (Foreign key)	**Restaurant_ID** (Foreign key)	**Overall _rating**	**Food _Rating**	**Service _Rating**

Table: Restaurant_Cuisines

Primary key **Restaurant_cuisine_id** introduced for full dependency.

Restaurant_cuisine_id (primary key)	**Restaurant_ID** (Foreign key)	Cuisine

After the introduction of Rating_ID and Restaurant_ID in **Ratings** and **Restaurant_Cuisines** tables, all the four tables obey the condition that all non-prime keys should be fully dependent on the primary key. The tables **Consumers, Restaurants, Ratings** and **Restaurant Cuisines** are in Second Normal Form.

1.2.3.3 Normalize the tables to Third Normal Form (3NF)

There is no transitive dependency existing in the tables **Consumers, Restaurants, Ratings** and **Restaurant Cuisines** and they are in Third Normal Form.

1.2.4 Review of Normalized Tables, Relationships, and Refinement of Schema

The review of the normalized tables and relationships is presented in Table 1.3.

Table 1.3 Review of normalized tables and relationship references

Sl No	Table	Primary keys	Foreign Keys	Relationship references from tables
1	**Consumers**	consumer_id	Nil	Nil
2	**Restaurants**	restaurant_id	Nil	Nil
3	**Ratings**	Rating_id	consumer_id, restaurant_id	**Consumer** (consumer_id) **Restaurant** (restaurant_id)
4	**Restaurant_ Cuisines**	restaurant_cuisine_id	restaurant_id	**Restaurant** (restaurant_id)

Table 1.4 Refined table-wise schema of the food service database

Consumers	**Restaurants**
dbo.Consumers Columns Consumer_ID (PK, nvarchar(10), not null) City (nvarchar(20), not null) State (nvarchar(20), not null) Country (nvarchar(20), not null) Latitude (decimal(18,10), not null) Longitude (decimal(18,10), not null) Smoker (varchar(3), null) Drink_Level (nvarchar(20), not null) Transportation_Method (nvarchar(20), null) Marital_Status (nvarchar(50), null) Children (nvarchar(20), null) Age (tinyint, not null) Occupation (nvarchar(20), null) Budget (nvarchar(20), null)	dbo.Restaurants Columns Restaurant_ID (PK, int, not null) Name (nvarchar(100), not null) City (nvarchar(20), not null) State (nvarchar(20), not null) Country (nvarchar(20), not null) Zip_Code (nvarchar(10), null) Latitude (decimal(18,10), not null) Longitude (decimal(18,10), not null) Alcohol_Service (nvarchar(20), not null) Smoking_Allowed (nvarchar(20), not null) Price (nvarchar(10), not null) Franchise (nvarchar(4), not null) Area (nvarchar(10), not null) Parking (nvarchar(10), not null)

Ratings	Restaurant_cuisines
dbo.Ratings Columns Rating_ID (PK, int, not null) Consumer_ID (FK, nvarchar(10), not null) Restaurant_ID (FK, int, not null) Overall_Rating (tinyint, not null) Food_Rating (tinyint, not null) Service_Rating (tinyint, not null)	dbo.Restaurant_cuisines Columns Restaurant_cuisine_ID (PK, int, not null) Restaurant_ID (FK, int, not null) Cuisine (nvarchar(20), not null)

The refined schema for the food service database design is shown in Table 1.3 under the default schema dbo.

1.2.5 Applying Constraints and Indexes

```
--creat table Consumers
CREATE TABLE Consumers (
Consumer_ID nvarchar(10) NOT NULL PRIMARY KEY,
City nvarchar(20) NOT NULL,
State nvarchar(20) NOT NULL,
Country nvarchar(20) NOT NULL,
Latitude decimal(18,10)  NOT NULL,
Longitude decimal(18,10)  NOT NULL,
Smoker varchar(3),
Drink_Level nvarchar(20) NOT NULL,
Transportation_Method nvarchar(20),
Marital_Status nvarchar(50),
Children nvarchar(20),
Age tinyint NOT NULL CHECK (Age BETWEEN 1 AND 100),
Occupation nvarchar(20),
Budget nvarchar(20),
);
```

Figure 1.11 Constraint on age in **Consumers** table

```
--Create table Ratings
CREATE TABLE Ratings (
Rating_ID int IDENTITY (1,1) NOT NULL PRIMARY KEY,
Consumer_ID  nvarchar(10) NOT NULL FOREIGN KEY (Consumer_ID) REFERENCES Consumers (Consumer_ID),
Restaurant_ID int NOT NULL FOREIGN KEY (Restaurant_ID) REFERENCES Restaurants (Restaurant_ID),
Overall_Rating tinyint NOT NULL CHECK (Overall_Rating BETWEEN 0 AND 2),
Food_Rating tinyint NOT NULL CHECK (Food_Rating BETWEEN 0 AND 2),
Service_Rating tinyint NOT NULL CHECK (Service_Rating BETWEEN 0 AND 2),
);
```

Figure 1.12 Constraint on ratings in **Ratings** table

The food service database uses various constraints such as NOT NULL (refer Table 1.4 for NOT NULL constraints), and CHECK constraints to validate and enforce data integrity and thereby ensures only valid data is entered into the database. The constraints presented in Figures 1.11 and 1.12 respectively ensures data integrity by ensuring age is between 1 and 100 and ratings are between 0 and 2.

In the food service database, most of the queries are related to consumers, restaurants, ratings, cuisines etc. Considering the nature of the queries in the

database, appropriate clustered indexes are added to columns (primary keys) such as consumer_id, restaurant_id, rating_id, and restaurant_cuisine_id for fast search and retrieval operations. Non-clustered indexes can be created on the food service database to improve the query processing time.

1.2.6 Optimizing the Final Schema

Regular maintenance, testing, and monitoring are involved with the optimization of the final schema. It is required to regularly maintain the database for maintaining database integrity, consistency, and performance over time. Thorough testing of the optimized schema and regular monitoring of the database performance is important for improving the performance as well as meeting the client requirements without fail.

1.2.7 Documentation

This document describes all the details associated with the food service management database design. The schema design, including table structures, relationships, constraints, and indexing is well documented for facilitating understanding and maintenance of the database by the client.

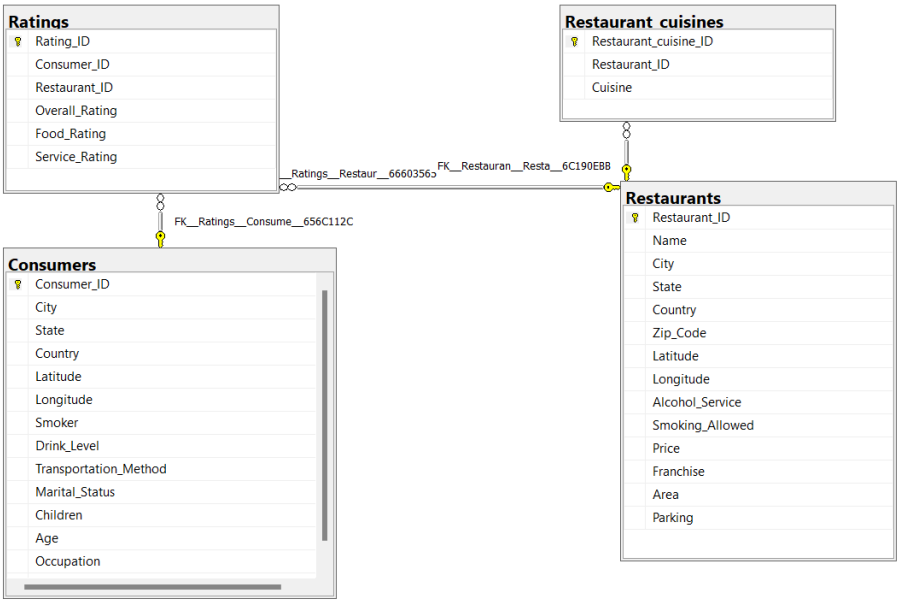


Figure 1.13 Food service database diagram

1.2.8 Food Service Database Diagram

The food service database diagram given in Figure 1.13 represents the structure and relationships of database entities. Figure 1.15 provides a visual representation of the database schema and relationships between different entities useful for developers, administrators, and stakeholders to understand the structure and functionality of the database system.

Task 2 Part 2: Exploratory Data Analysis and Query Implementation based on the Created Food Service Database in Microsoft SQL Server Management Studio using T-SQL

2.1 Introduction

For building a robust food service database system, advanced functionalities and optimization techniques are essential. The database design is based on the various tables created in Part 1. The Part 2 section focuses on exploratory data analysis and query implementation on the created food service database for deriving business intelligence.

The section below deals with the intricacies of the food service industry, where understanding consumer behavior, restaurant performance, rating variations, and culinary trends is paramount. By harnessing the power of SQL queries and optimization techniques, meaningful insights are extracted from the database. Through exploratory data analysis, patterns, correlations, and anomalies within the data are uncovered, enabling informed decision-making for food service companies.

The upcoming section addresses specific queries implemented by the database consultant for the growth and success of the food service company.

2.2 Medium-Priced Restaurants Offering Mexican Cuisine with Open Seating

Task 2: Part 2:

1. Write a query that lists all restaurants with a Medium range price with open area, serving Mexican food.

The query and result given in Figure 2.1 are aimed at providing valuable insights for business intelligence within the food service industry. By listing restaurants with a medium range price offering Mexican cuisine and open seating areas, the query helps to identify specific market segments and consumer preferences. Understanding the demand for Mexican cuisine at medium-priced restaurants with open seating areas can inform strategic

decision-making processes for restaurant owners and managers. It helps in developing targeted marketing campaigns, optimizing menu offerings, and pricing strategies, and even identifying potential locations for new restaurant ventures. Additionally, periodic analysis of this data can provide trends and patterns that can be leveraged to enhance customer satisfaction and overall business performance.

```
--Task 2 Part 1 Q 1
--A query that lists all restaurants with a Medium range price with open area, serving Mexican food.
SELECT R.*
FROM Restaurants R
WHERE R.Price = 'Medium'
AND R.Area = 'Open'
AND EXISTS (
    SELECT 1
    FROM Restaurant_Cuisines RC
    WHERE R.Restaurant_ID = RC.Restaurant_ID
    AND Cuisine = 'Mexican'
);
```

	Restaurant_ID	Name	City	State	Country	Zip_Code	Latitude	Longitude	Alcohol_Service
1	135018	El Oceano Dorado	Cuernavaca	Morelos	Mexico	NULL	18.8598030000	-99.2221640000	Full Bar
2	135106	El Rincón De San Francisco	San Luis Potosi	San Luis Potosi	Mexico	78000	22.1497088000	-100.9760928000	Wine & Beer

Smoking_Allowed	Price	Franchise	Area	Parking
Yes	Medium	No	Open	Yes
Bar Only	Medium	No	Open	None

Figure 2.1 Query listing Medium-Priced Restaurants Offering Mexican Cuisine with Open Seating and result

```
--A query that lists all restaurants with a Medium range price with open area, serving Mexican food. (Using JOIN Comman
SELECT R.*, RC.Cuisine
FROM Restaurants R
JOIN Restaurant_Cuisines RC ON R.Restaurant_ID = RC.Restaurant_ID
WHERE R.Price = 'Medium'
AND R.Area = 'Open'
AND RC.Cuisine = 'Mexican';
```

	Restaurant_ID	Name	City	State	Country	Zip_Code	Latitude	Longitude	Alcohol_Service
1	135018	El Oceano Dorado	Cuernavaca	Morelos	Mexico	NULL	18.8598030000	-99.2221640000	Full Bar
2	135106	El Rincón De San Francisco	San Luis Potosi	San Luis Potosi	Mexico	78000	22.1497088000	-100.9760928000	Wine & Beer

Smoking_Allowed	Price	Franchise	Area	Parking	Cuisine
Yes	Medium	No	Open	Yes	Mexican
Bar Only	Medium	No	Open	None	Mexican

Figure 2.2 Modified query listing Medium-Priced Restaurants Offering Mexican Cuisine with Open Seating and result (cuisine included)

Two restaurants are obtained as results for restaurants with a medium range price with open area, serving Mexican food. To include the cuisine in the results, the query is modified and given in Fig 2.2 with the results. Table 2.1 gives more details of the codes.

Table 2.1 List of objects used in Task 2 Part 2 Q 1

Task 2 Part 2 Q 1: Medium-Priced Restaurants Offering Mexican Cuisine with Open Seating		
Objects	Usage and Explanation	
Nested queries-EXISTS	Yes	The EXISTS clause is used to check for the existence of rows in the **Restaurant_Cuisines** table that match the criteria specified in the subquery. It is nested within the WHERE clause.
Nested queries-IN		No
System functions		No
Use of GROUP BY, HAVING, and ORDER BY clauses		No
Joins	Yes (in modified query)	A JOIN operation is used to combine the **Restaurants** and **Restaurant_Cuisines** tables based on the common column Restaurant_ID.

2.3 Comparison of Restaurants with Overall Rating 1 Serving Mexican vs. Italian (Distinct Results Taken)

Task 2: Part 2:

1. Write a query that returns the total number of restaurants that have an overall rating of 1 and are serving Mexican food. Compare the results with the total number of restaurants that have the overall rating as 1 serving Italian food (please give explanations on their comparison)

The query is designed to assess the number of restaurants with an overall rating of 1 that serve Mexican food and compare it with the number of restaurants with an overall rating of 1 serving Italian food. It first filters the restaurants based on the condition that their overall rating is 1 and they serve Mexican food. Then, it counts the total number of such restaurants. Similarly, it applies the same conditions to identify restaurants serving Italian food with an

overall rating of 1 and counts them. Table 2.2 shows the database objectss used.

```
-- Count of restaurants with overall rating as 1 serving Mexican food
SELECT COUNT( DISTINCT RC.Restaurant_ID) AS 'Count of restaurants with overall rating as 1 serving Mexican food'
FROM Restaurant_Cuisines RC
WHERE EXISTS (
   SELECT 1
   FROM Ratings Ra
   WHERE Ra.Restaurant_ID = RC.Restaurant_ID
   AND Ra.Overall_Rating = 1
)
AND RC.Cuisine = 'Mexican';
```

	Count of restaurants with overall rating as 1 serving Mexican food
1	27

```
-- Count of restaurants with overall rating as 1 serving Italian food
SELECT COUNT(DISTINCT RC.Restaurant_ID) AS 'Count of restaurants with overall rating as 1 serving Italian food'
FROM Restaurant_Cuisines RC
WHERE EXISTS (
   SELECT 1
   FROM Ratings Ra
   WHERE Ra.Restaurant_ID = RC.Restaurant_ID
   AND Ra.Overall_Rating = 1
)
AND RC.Cuisine = 'Italian';
```

	Count of restaurants with overall rating as 1 serving Italian food
1	4

Figure 2.3 Comparison of Restaurants with Overall Rating 1 Serving Mexican vs. Italian (Distinct Results Taken)

Table 2.2 List of objects used in Task 2 Part 2 Q 2

Task 2 Part 2 Q 2: Comparison of Restaurants with Overall Rating 1 Serving Mexican vs. Italian (Distinct Results Taken)		
Objects	Usage and Explanation	
Nested queries-EXISTS	Yes	The main query contains a nested EXISTS subquery, which checks for the existence of restaurants with an overall rating of 1 in the **Ratings** table for each restaurant cuisine.
Nested queries-IN, System functions, Use of GROUP BY, HAVING, and ORDER BY clauses		No
Joins	Yes	See Figure 2.4

2.3.1 Explanation of the comparison results

The results of the query reveal that 27 restaurants are offering Mexican cuisine with an overall rating of 1, whereas only 4 restaurants serving Italian cuisine have the same overall rating. This comparison indicates that more consumers are dissatisfied with restaurants serving Mexican food, as represented by the higher number of low ratings in this category. The insight from the result is that there are potential areas for improvement in the quality of Mexican cuisine offerings or service standards within these establishments compared to Italian food joints.

```
-- Count of restaurants with overall rating as 1 serving Mexican food (Using JOIN commands)
SELECT COUNT(DISTINCT RC.Restaurant_ID) AS 'Count of restaurants with overall rating as 1 serving Mexican food'
FROM Ratings Ra
JOIN Restaurant_Cuisines RC ON Ra.Restaurant_ID = RC.Restaurant_ID
JOIN Ratings R ON R.Restaurant_ID = RC.Restaurant_ID
WHERE Ra.Overall_Rating = 1
AND RC.Cuisine = 'Mexican';
```

```
-- Count of restaurants with overall rating as 1 serving Italian food (Using JOIN commands)
SELECT COUNT(DISTINCT RC.Restaurant_ID) AS 'Count of restaurants with overall rating as 1 serving Italian food'
FROM Ratings Ra
JOIN Restaurant_Cuisines RC ON Ra.Restaurant_ID = RC.Restaurant_ID
JOIN Ratings R ON R.Restaurant_ID = RC.Restaurant_ID
WHERE Ra.Overall_Rating = 1
AND RC.Cuisine = 'Italian';
```

Figure 2.4 Comparison of Restaurants with Overall Rating 1 Serving Mexican vs. Italian using JOIN commands (Distinct Results Taken)

The same query results can be established with JOIN commands as given in Figure 2.4.

The 4 Italian cuisine restaurant names and 27 Mexican cuisine restaurant names are listed using the queries given in Figure 2.5 and 2.6 respectively.

```
--Names of restaurants with overall rating as 1 serving Italian food
SELECT DISTINCT RC.Restaurant_ID, R.Name
FROM Restaurant_Cuisines RC
JOIN Ratings Ra ON Ra.Restaurant_ID = RC.Restaurant_ID
JOIN Restaurants R ON R.Restaurant_ID = RC.Restaurant_ID
WHERE Ra.Overall_Rating = 1
AND RC.Cuisine = 'Italian';
```

	Restaurant_ID	Name
1	132626	La Perica Hamburguesa
2	132856	Unicol's Pizza
3	135035	El Mundo De La Pasta
4	135109	Paniroles

Figure 2.5 Names of restaurants with Overall Rating 1 Serving Italian

```
--Names of restaurants with overall rating as 1 serving Mexican food
SELECT DISTINCT RC.Restaurant_ID, R.Name
FROM Restaurant_Cuisines RC
JOIN Ratings Ra ON Ra.Restaurant_ID = RC.Restaurant_ID
JOIN Restaurants R ON R.Restaurant_ID = RC.Restaurant_ID
WHERE Ra.Overall_Rating = 1
AND RC.Cuisine = 'Mexican';
;
```

	Restaurant_ID	Name
5	132630	Palomo Tec
6	132663	Tacos Abi
7	132665	Tacos Correcaminos
8	132668	Tacos El Guero
9	132706	Gorditas Dona Tota
10	132715	Tacos De La Estacion
11	132723	Gordas De Morales
12	132732	Taqueria El Amigo
13	132740	Carreton De Flautas Y Migadas
14	132754	Cabana Huasteca
15	132755	La Estrella De Dimas
16	132773	El Cotorreo
17	132825	Puesto De Tacos
18	132834	Gorditas Doa Gloria
19	132845	Cenaduría El Rincón De Tlaquepa...
20	132925	El Pueblito
21	134976	Log Yin
22	135018	El Oceano Dorado
23	135025	El Rincon De San Francisco
24	135027	Restaurant Orizatlan
25	135028	La Virreina
26	135104	Vips
27	135106	El Rincón De San Francisco

Figure 2.6 Names of restaurants with Overall Rating 1 Serving Mexican

2.4 Average Age of Consumers Giving 0 Service Rating

Task 2: Part 2:
3 . Calculate the average age of consumers who have given a 0 rating to the 'Service_rating' column. (NB: round off the value if it is a decimal)

This query calculates the average age of consumers who have given a 0 rating to the 'Service_rating' column. In the **Consumers** table, the age is given the datatype **TINYINT, NOT NULL**, with a constraint as given below in Fig 2.7. The age column is not allowed to take decimal numbers due to the data type.

```
Age tinyint NOT NULL CHECK (Age BETWEEN 1 AND 100),
```

Figure 2.7 Datatype of Age in **Consumers** Table

The average age results are different when the data types chosen for the column Age in the Consumers table are TINY INT and DECIMAL.

There is a repetition of consumer_IDs in the **Ratings** table. When considering the repetition, there are **160 records** matching the condition of 0 rating to the

'Service_rating' whereas the results reduced to **73 records** on ignoring the repetition.

The average age is **26 years** considering the repetition of consumers in the **Ratings** table whereas the average age becomes **25 years** while ignoring the consumer_ID repetition. Figures 2.8 and 2.9 show the SQL codes for average age results of 26 and 25 years respectively.

```
--Calulating average age of customers (repetition considered)
SELECT  ROUND(AVG(C.Age), 0) AS 'Average age of consumers (repetition considered) who have given a 0 rating to the Service_rating colum
FROM Consumers C
INNER JOIN Ratings Ra ON C.Consumer_ID = Ra.Consumer_ID
WHERE Ra.Service_Rating = 0;

--Calculating the count of customers (repetition considered) who have given a 0 rating to the 'Service_rating' column.
SELECT  COUNT(*) AS 'Count of consumers (repetition considered) who have given a 0 rating to the Service_rating column'
FROM Consumers C
INNER JOIN Ratings Ra ON C.Consumer_ID = Ra.Consumer_ID
WHERE Ra.Service_Rating = 0;

--Listing customers (repetition considered) who have given a 0 rating to the 'Service_rating' column.
SELECT  C.Consumer_ID, C.Age, Ra.Service_Rating
FROM Consumers C
INNER JOIN Ratings Ra ON C.Consumer_ID = Ra.Consumer_ID
WHERE Ra.Service_Rating = 0;
```

Average Age:

	Average age of consumers (repetition considered) who have given a 0 rating to the Service_rating column
1	26

Count:

	Count of consumers (repetition considered) who have given a 0 rating to the Service_rating column
1	160

List :

	Consumer_ID	Age	Service_Rating
1	U1002	22	0
2	U1002	22	0
3	U1005	20	0
4	U1005	20	0
5	U1007	23	0
6	U1008	23	0
7	U1009	21	0
8	U1009	21	0
9	U1009	21	0
10	U1010	25	0
11	U1011	23	0
12	U1013	30	0
13	U1014	22	0
14	U1014	22	0
15	U1014	22	0
16	U1016	21	0
17	U1018	23	0
18	U1019	23	0
19	U1019	23	0
20	U1019	23	0
21	U1019	23	0
22	U1019	23	0
23	U1022	22	0

The red boxes is drawn in the list to highlight few of the repetitions in the consumer_IDs. When considering the consumer_ID repetition, the average age of the consumers who have give 0 rating to the service rating column is 26 years.

(160 rows affected)

Completion time: 2024-04-12T16:27:07.2279319+01:00

Figure 2.8 Average age of consumers giving 0 service rating (repetition of consumer_IDs in **Ratings** table considered- Age data type **TINY INT**)

```
--Calulating average age of customers (repetition ignored)
SELECT ROUND(AVG(C.Age), 0) AS 'Average age of consumers (repetition ignored) who have given a 0 rating to the Service_rating column'
FROM Consumers C
WHERE  C.Consumer_ID IN (SELECT DISTINCT Ra.Consumer_ID FROM Ratings Ra
WHERE Ra.Service_Rating = 0
GROUP BY Consumer_ID);

--Calculating the count of customers  (repetition ignored) who have given a 0 rating to the 'Service_rating' column.
SELECT COUNT(C. Consumer_ID) as 'Count of customers  (repetition ignored) who has given a 0 rating to the Service_rating'
FROM Consumers C
WHERE  C.Consumer_ID IN (SELECT DISTINCT Ra.Consumer_ID FROM Ratings Ra
WHERE Ra.Service_Rating = 0
GROUP BY Consumer_ID);

--Listing customers  (repetition ignored) who have given a 0 rating to the 'Service_rating' column.
SELECT C. Consumer_ID as 'Customer  (repetition ignored) who has given a 0 rating to the Service_rating', C.Age
FROM Consumers C
WHERE  C.Consumer_ID IN (SELECT DISTINCT Ra.Consumer_ID FROM Ratings Ra
WHERE Ra.Service_Rating = 0
GROUP BY Consumer_ID);
```

	Average age of consumers (repetition ignored) who have given a 0 rating to the Service_rating column
1	25

	Count of customers (repetition ignored) who has given a 0 rating to the Service_rating
1	73

	Customer (repetition ignored) who has given a 0 rating to the Service_rating	Age
1	U1002	22
2	U1005	20
3	U1007	23
4	U1008	23
5	U1009	21
6	U1010	25
7	U1011	23
8	U1013	30
9	U1014	22
10	U1016	21
11	U1018	23
12	U1019	23
13	U1022	22
14	U1023	24
15	U1024	82
16	U1025	22
17	U1026	23
18	U1028	23
19	U1029	22
20	U1030	21

(73 rows affected)

Completion time: 2024-04-12T17:04:42.6095068+01:00

When the consumer_ID repetition is ignored, the average age of the consumers who have give 0 rating to the service rating column is 25 years.

Figure 2.9 Average age of consumers giving 0 service rating (repetition of consumer_IDs in **Ratings** table ignored- Age data type **TINY INT**)

```
--Calulating average age (Using EXISTS Command)(repetition ignored)
SELECT ROUND(AVG(C.Age), 0) AS 'Average age of consumers who have given a 0 rating to the Service_rating column'
FROM Consumers C
WHERE EXISTS (
    SELECT 1
    FROM Ratings Ra
    WHERE Ra.Consumer_ID = C.Consumer_ID
    AND Ra.Service_Rating = 0
);

--Calulating average age (Using Sub queries) (repetition ignored)
SELECT ROUND(AVG(C.Age), 0) AS 'Average age of consumers who have given a 0 rating to the Service_rating column'
FROM Consumers C
WHERE  C.Consumer_ID IN (SELECT Ra.Consumer_ID FROM Ratings Ra
WHERE Ra.Service_Rating = 0
GROUP BY Consumer_ID);

--Calulating average age (Using format CTE) (repetition ignored)
WITH ConsumerServiceRatingZero AS (
    SELECT Ra.Consumer_ID
    FROM Ratings Ra
    WHERE Ra.Service_Rating = 0
    GROUP BY Ra.Consumer_ID
)
SELECT ROUND(AVG(C.Age), 0) AS [Average age of consumers who have given a 0 rating to the Service_rating column]
FROM Consumers C
WHERE C.Consumer_ID IN (SELECT Consumer_ID FROM ConsumerServiceRatingZero);
```

Figure 2.10 Additional queries to get average age results

The average age result can be obtained using different codes utilizing the EXISTS command, subqueries, or CTEs as shown in Figure 2.10. Table 2.3 and 2.4 shows the database objectss used in the query.

Table 2.3 List of objects used in Task 2 Part 2 Q 3

Task 2 Part 2 Q 3: Average age of consumers giving 0 service rating (codes where repetition is considered)		
Objects	Usage and Explanation	
System functions	Yes	The query uses the ROUND function to round the average age to the nearest integer, even though the data type is TINYINT.
Nested queries-EXISTS, Nested queries-IN, Use of GROUP BY, HAVING, and ORDER BY clauses		No
Joins	Yes	The query uses an INNER JOIN to join the Consumers and Ratings tables based on the Consumer_ID column.

Table 2.4 List of objects used in Task 2 Part 2 Q 3

Task 2 Part 2 Q 3: Average age of consumers giving 0 service rating (codes where repetition is ignored)		
Objects	Usage and Explanation	
Nested queries-IN	Yes	The query uses the IN clause to filter Consumer_IDs based on the subquery result.
System functions	Yes	ROUND()
Nested queries-EXISTS, Joins, HAVING, and ORDER BY clauses		No
Use of GROUP BY	Yes	The subquery uses the GROUP BY clause to group Consumer_IDs.

When the age data type is changed to decimal, the average age of the consumers giving 0 service ratings is 27 years (repetition considered) and 26 years (repetition ignored) respectively.

The query provides valuable insights into the age demographics of consumers who are dissatisfied with the service provided by restaurants, allowing for targeted improvements or marketing strategies to enhance customer satisfaction.

2.5 Restaurant Rankings by the Youngest Consumer

Task 2: Part 2:

4 Write a query that returns the restaurants ranked by the youngest consumer. You should include the restaurant name and food rating that is given by that customer to the restaurant in your result. Sort the results based on food rating from high to low.

The query shown in Figure 2.11 and 2.12 aims to return restaurant ranks given by the age youngest consumer. There are two approaches taken to run this query

1) **First Approach** - Identify the youngest consumer in the **Ratings** table and return the food rating given by that customer to various restaurants.
2) **Second Approach** -Identify all the distinct restaurants in the **Restaurants** table and return the food rating given to each of these restaurants by the yougest consumers.

The results from both the approaches are then sorted in descending order based on the food rating, from highest to lowest. By executing this query insights into which restaurants attract younger consumers and their satisfaction level with the food offered can be gained, thereby helping the business decisions and marketing strategies.

First Approach

(4 rows affected)

```
--First approach - Listing the restaurants and the food rating given by the youngest consumer in the Ratings table.
SELECT YoungestConsumer.Consumer_ID AS Youngest_Consumer, YoungestConsumer.Age, R.Restaurant_ID,R.Name AS Restaurant_Name, Ra.Food_Rating
FROM Restaurants R
JOIN Ratings Ra ON R.Restaurant_ID = Ra.Restaurant_ID
JOIN (
   SELECT TOP 1 Consumer_ID, Age
   FROM Consumers
   ORDER BY Age ASC
) YoungestConsumer ON Ra.Consumer_ID = YoungestConsumer.Consumer_ID
ORDER BY Ra.Food_Rating DESC;
```

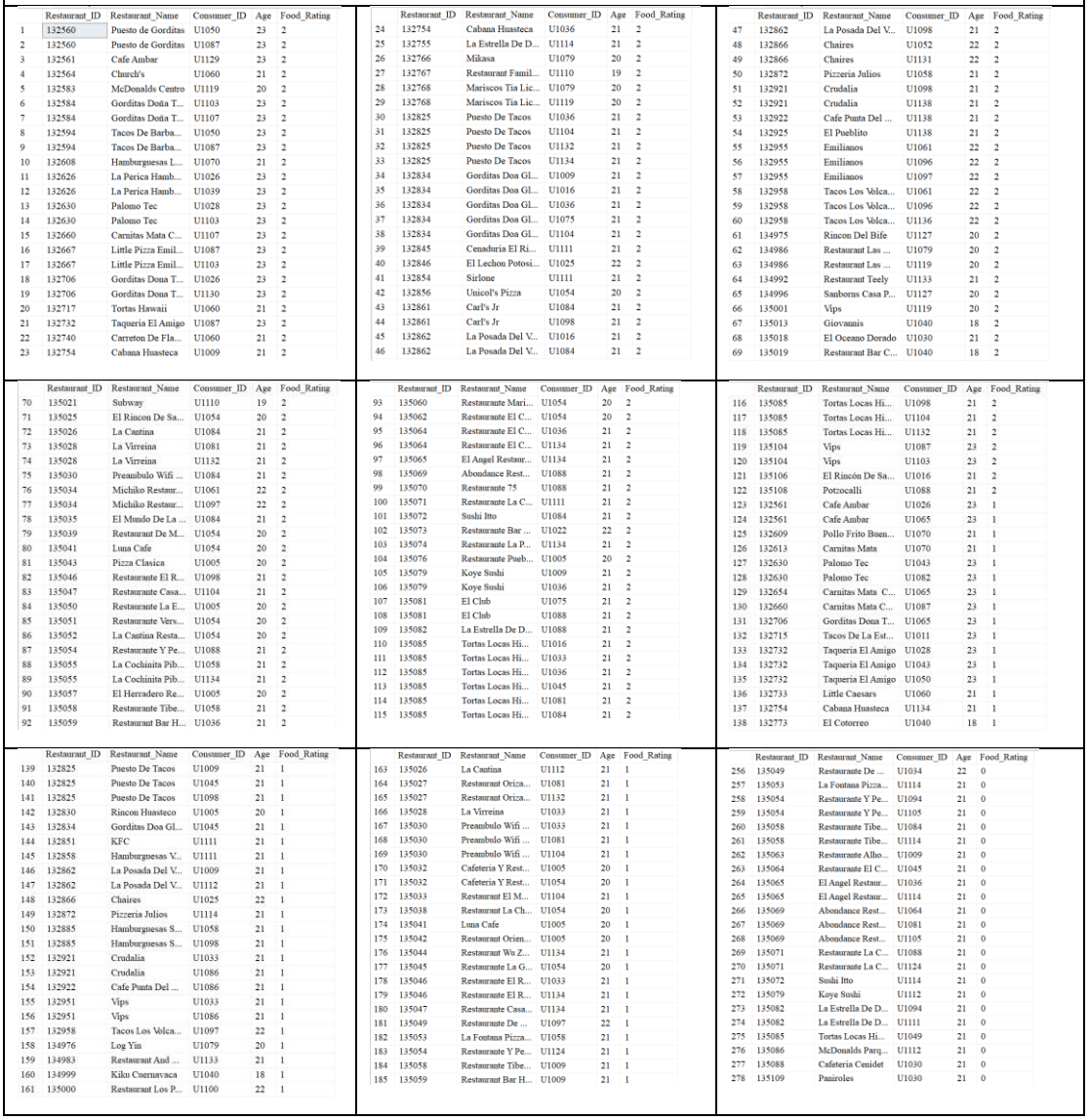
	Youngest_Consumer	Age	Restaurant_ID	Restaurant_Name	Food_Rating
1	U1040	18	135013	Giovannis	2
2	U1040	18	135019	Restaurant Bar Coty Y Pablo	2
3	U1040	18	132773	El Cotorreo	1
4	U1040	18	134999	Kiku Cuernavaca	1

Figure 2.11 Restaurant rankings by the youngest consumer

(278 rows affected)

Second Approach (Rows 186 to 255 skipped)

```
--Second approach - List of all restaurants with food rating given by the youngest consumers from that restaurant
SELECT DISTINCT R.Restaurant_ID, R.Name AS Restaurant_Name,
        C.Consumer_ID, C.Age,
        Ra.Food_Rating
FROM Restaurants R
JOIN Ratings Ra ON R.Restaurant_ID = Ra.Restaurant_ID
JOIN Consumers C ON Ra.Consumer_ID = C.Consumer_ID
JOIN (
   SELECT R.Restaurant_ID, MIN(C.Age) AS Youngest_Age
   FROM Restaurants R
   JOIN Ratings Ra ON R.Restaurant_ID = Ra.Restaurant_ID
   JOIN Consumers C ON Ra.Consumer_ID = C.Consumer_ID
   GROUP BY R.Restaurant_ID
) AS MinAge ON R.Restaurant_ID = MinAge.Restaurant_ID AND C.Age = MinAge.Youngest_Age
ORDER BY Ra.Food_Rating DESC;
```



	Restaurant_ID	Restaurant_Name	Consumer_ID	Age	Food_Rating
1	132560	Puesto de Gorditas	U1050	23	2
2	132560	Puesto de Gorditas	U1087	23	2
3	132561	Cafe Ambar	U1129	23	2
4	132564	Church's	U1060	21	2
5	132583	McDonalds Centro	U1119	20	2
6	132584	Gorditas Doña T...	U1103	23	2
7	132584	Gorditas Doña T...	U1107	23	2
8	132594	Tacos De Barba...	U1050	23	2
9	132594	Tacos De Barba...	U1087	23	2
10	132608	Hamburguesas L...	U1070	21	2
11	132626	La Perica Hamb...	U1026	23	2
12	132626	La Perica Hamb...	U1039	23	2
13	132630	Palomo Tec	U1028	23	2
14	132630	Palomo Tec	U1103	23	2
15	132660	Carnitas Mata C...	U1107	23	2
16	132667	Little Pizza Emil...	U1087	23	2
17	132667	Little Pizza Emil...	U1103	23	2
18	132706	Gorditas Dona T...	U1026	23	2
19	132706	Gorditas Dona T...	U1130	23	2
20	132717	Tortas Hawaii	U1060	21	2
21	132732	Taqueria El Amigo	U1087	23	2
22	132740	Carreton De Fla...	U1060	21	2
23	132754	Cabana Huasteca	U1009	21	2

	Restaurant_ID	Restaurant_Name	Consumer_ID	Age	Food_Rating
24	132754	Cabana Huasteca	U1036	21	2
25	132755	La Estrella De D...	U1114	21	2
26	132766	Mikasa	U1079	20	2
27	132767	Restaurant Famil...	U1110	19	2
28	132768	Mariscos Tia Lic...	U1079	20	2
29	132768	Mariscos Tia Lic...	U1119	20	2
30	132825	Puesto De Tacos	U1036	21	2
31	132825	Puesto De Tacos	U1104	21	2
32	132825	Puesto De Tacos	U1132	21	2
33	132825	Puesto De Tacos	U1134	21	2
34	132834	Gorditas Doa Gl...	U1009	21	2
35	132834	Gorditas Doa Gl...	U1016	21	2
36	132834	Gorditas Doa Gl...	U1036	21	2
37	132834	Gorditas Doa Gl...	U1075	21	2
38	132834	Gorditas Doa Gl...	U1104	21	2
39	132845	Cenaduria El Ri...	U1111	21	2
40	132846	El Lechon Potosi...	U1025	22	2
41	132854	Sirlone	U1111	21	2
42	132856	Unicol's Pizza	U1054	20	2
43	132861	Carl's Jr	U1084	21	2
44	132861	Carl's Jr	U1098	21	2
45	132862	La Posada Del V...	U1016	21	2
46	132862	La Posada Del V...	U1084	21	2

	Restaurant_ID	Restaurant_Name	Consumer_ID	Age	Food_Rating
47	132862	La Posada Del V...	U1098	21	2
48	132866	Chaires	U1052	22	2
49	132866	Chaires	U1131	22	2
50	132872	Pizzeria Julios	U1058	21	2
51	132921	Crudalia	U1098	21	2
52	132921	Crudalia	U1138	21	2
53	132922	Cafe Punta Del ...	U1138	21	2
54	132925	El Pueblito	U1138	21	2
55	132955	Emilianos	U1061	22	2
56	132955	Emilianos	U1096	22	2
57	132955	Emilianos	U1097	22	2
58	132958	Tacos Los Volca...	U1061	22	2
59	132958	Tacos Los Volca...	U1096	22	2
60	132958	Tacos Los Volca...	U1136	22	2
61	134975	Rincon Del Bife	U1127	20	2
62	134986	Restaurant Las ...	U1079	20	2
63	134986	Restaurant Las ...	U1119	20	2
64	134992	Restaurant Teely	U1133	21	2
65	134996	Sanborns Casa P...	U1127	20	2
66	135001	Vips	U1119	20	2
67	135013	Giovannis	U1040	18	2
68	135018	El Oceano Dorado	U1030	21	2
69	135019	Restaurant Bar C...	U1040	18	2

	Restaurant_ID	Restaurant_Name	Consumer_ID	Age	Food_Rating
70	135021	Subway	U1110	19	2
71	135025	El Rincon De Sa...	U1054	20	2
72	135026	La Cautina	U1084	21	2
73	135028	La Virreina	U1081	21	2
74	135028	La Virreina	U1132	21	2
75	135030	Preambulo Wifi ...	U1084	21	2
76	135034	Michiko Restaur...	U1061	22	2
77	135034	Michiko Restaur...	U1097	22	2
78	135035	El Mundo De La ...	U1084	21	2
79	135039	Restaurant De M...	U1054	20	2
80	135041	Luna Cafe	U1054	20	2
81	135043	Pizza Clasica	U1005	20	2
82	135046	Restaurante El R...	U1098	21	2
83	135047	Restaurante Casa...	U1104	21	2
84	135050	Restaurante La E...	U1005	20	2
85	135051	Restaurante Vers...	U1054	20	2
86	135052	La Cautina Resta...	U1054	20	2
87	135054	Restaurante Y Pe...	U1088	21	2
88	135055	La Cochinita Pib...	U1058	21	2
89	135055	La Cochinita Pib...	U1134	21	2
90	135057	El Herradero Re...	U1005	20	2
91	135058	Restaurante Tibe...	U1058	21	2
92	135059	Restaurant Bar H...	U1036	21	2

	Restaurant_ID	Restaurant_Name	Consumer_ID	Age	Food_Rating
93	135060	Restaurante Mari...	U1054	20	2
94	135062	Restaurante El C...	U1054	20	2
95	135064	Restaurante El C...	U1036	21	2
96	135064	Restaurante El C...	U1134	21	2
97	135065	El Angel Restaur...	U1134	21	2
98	135069	Abondance Rest...	U1088	21	2
99	135070	Restaurante 75	U1088	21	2
100	135071	Restaurante La C...	U1111	21	2
101	135072	Sushi Itto	U1084	21	2
102	135073	Restaurante Bar ...	U1022	22	2
103	135074	Restaurante La P...	U1134	21	2
104	135076	Restaurante Pueb...	U1005	20	2
105	135079	Koye Sushi	U1009	21	2
106	135079	Koye Sushi	U1036	21	2
107	135081	El Club	U1075	21	2
108	135081	El Club	U1088	21	2
109	135082	La Estrella De D...	U1088	21	2
110	135085	Tortas Locas Hi...	U1016	21	2
111	135085	Tortas Locas Hi...	U1033	21	2
112	135085	Tortas Locas Hi...	U1036	21	2
113	135085	Tortas Locas Hi...	U1045	21	2
114	135085	Tortas Locas Hi...	U1081	21	2
115	135085	Tortas Locas Hi...	U1084	21	2

	Restaurant_ID	Restaurant_Name	Consumer_ID	Age	Food_Rating
116	135085	Tortas Locas Hi...	U1098	21	2
117	135085	Tortas Locas Hi...	U1104	21	2
118	135085	Tortas Locas Hi...	U1132	21	2
119	135104	Vips	U1087	23	2
120	135104	Vips	U1103	23	2
121	135106	El Rincón De Sa...	U1016	21	2
122	135108	Potzocalli	U1088	21	2
123	132561	Cafe Ambar	U1026	23	1
124	132561	Cafe Ambar	U1065	23	1
125	132609	Pollo Frito Buen...	U1070	21	1
126	132613	Carnitas Mata	U1070	21	1
127	132630	Palomo Tec	U1043	23	1
128	132630	Palomo Tec	U1082	23	1
129	132654	Carnitas Mata C...	U1065	23	1
130	132660	Carnitas Mata C...	U1087	23	1
131	132706	Gorditas Dona T...	U1065	23	1
132	132715	Tacos De La Est...	U1011	23	1
133	132732	Taqueria El Amigo	U1028	23	1
134	132732	Taqueria El Amigo	U1043	23	1
135	132732	Taqueria El Amigo	U1050	23	1
136	132733	Little Caesars	U1060	21	1
137	132754	Cabana Huasteca	U1134	21	1
138	132773	El Cotorreo	U1040	18	1

	Restaurant_ID	Restaurant_Name	Consumer_ID	Age	Food_Rating
139	132825	Puesto De Tacos	U1009	21	1
140	132825	Puesto De Tacos	U1045	21	1
141	132825	Puesto De Tacos	U1098	21	1
142	132830	Rincon Huasteco	U1005	20	1
143	132834	Gorditas Doa Gl...	U1045	21	1
144	132851	KFC	U1111	21	1
145	132858	Hamburguesas V...	U1111	21	1
146	132862	La Posada Del V...	U1009	21	1
147	132862	La Posada Del V...	U1112	21	1
148	132866	Chaires	U1025	22	1
149	132872	Pizzeria Julios	U1114	21	1
150	132885	Hamburguesas S...	U1058	21	1
151	132885	Hamburguesas S...	U1098	21	1
152	132921	Crudalia	U1033	21	1
153	132921	Crudalia	U1086	21	1
154	132922	Cafe Punta Del ...	U1086	21	1
155	132951	Vips	U1033	21	1
156	132951	Vips	U1086	21	1
157	132958	Tacos Los Volca...	U1097	22	1
158	134976	Log Yin	U1079	20	1
159	134983	Restaurant And ...	U1133	21	1
160	134999	Kiku Cuernavaca	U1040	18	1
161	135000	Restaurant Los P...	U1100	22	1

	Restaurant_ID	Restaurant_Name	Consumer_ID	Age	Food_Rating
163	135026	La Cautina	U1112	21	1
164	135027	Restaurant Oriza...	U1081	21	1
165	135027	Restaurant Oriza...	U1132	21	1
166	135028	La Virreina	U1033	21	1
167	135030	Preambulo Wifi ...	U1033	21	1
168	135030	Preambulo Wifi ...	U1081	21	1
169	135030	Preambulo Wifi ...	U1104	21	1
170	135032	Cafeteria Y Rest...	U1005	20	1
171	135032	Cafeteria Y Rest...	U1054	20	1
172	135033	Restaurant El M...	U1104	21	1
173	135038	Restaurant La Ch...	U1054	20	1
174	135041	Luna Cafe	U1005	20	1
175	135042	Restaurant Orien...	U1005	20	1
176	135044	Restaurant Wu Z...	U1134	21	1
177	135045	Restaurante La G...	U1054	20	1
178	135046	Restaurante El R...	U1033	21	1
179	135046	Restaurante El R...	U1134	21	1
180	135047	Restaurante Casa...	U1134	21	1
181	135049	Restaurante De ...	U1097	22	1
182	135053	La Fontana Pizza...	U1058	21	1
183	135054	Restaurante Y Pe...	U1124	21	1
184	135058	Restaurante Tibe...	U1009	21	1
185	135059	Restaurant Bar H...	U1009	21	1

	Restaurant_ID	Restaurant_Name	Consumer_ID	Age	Food_Rating
256	135049	Restaurante De ...	U1034	22	0
257	135053	La Fontana Pizza...	U1114	21	0
258	135054	Restaurante Y Pe...	U1094	21	0
259	135054	Restaurante Y Pe...	U1105	21	0
260	135058	Restaurante Tibe...	U1084	21	0
261	135058	Restaurante Tibe...	U1114	21	0
262	135063	Restaurante Alho...	U1009	21	0
263	135064	Restaurante El C...	U1045	21	0
264	135065	El Angel Restaur...	U1036	21	0
265	135065	El Angel Restaur...	U1114	21	0
266	135069	Abondance Rest...	U1064	21	0
267	135069	Abondance Rest...	U1081	21	0
268	135069	Abondance Rest...	U1105	21	0
269	135071	Restaurante La C...	U1088	21	0
270	135071	Restaurante La C...	U1124	21	0
271	135072	Sushi Itto	U1114	21	0
272	135079	Koye Sushi	U1112	21	0
273	135082	La Estrella De D...	U1094	21	0
274	135082	La Estrella De D...	U1111	21	0
275	135085	Tortas Locas Hi...	U1049	21	0
276	135086	McDonalds Parq...	U1112	21	0
277	135088	Cafeteria Cenidet	U1030	21	0
278	135109	Pamiroles	U1030	21	0

Figure 2.12 Restaurant rankings of all restaurants by their youngest consumer

Table 2.5 List of objects used in Task 2 Part 2 Q 4

Task 2 Part 2 Q 4: Restaurant rankings by the youngest consumer(1st Approach)		
Objects	Usage and Explanation	
Nested queries	Yes	Subqueries present, no EXISTS or IN command used
System functions	Yes	The TOP function is used to select the youngest consumer.
Nested queries-EXISTS, Nested queries-IN , HAVING, and GROUP BY clauses		No
JOINS	Yes	(INNER) JOINs are used to connect the Restaurants, Ratings, and Consumers tables.
Use of ORDER BY	Yes	The ORDER BY clause is used to sort the results.

Table 2.6 List of objects used in Task 2 Part 2 Q 4

Task 2 Part 2 Q 4: Restaurant rankings by the youngest consumer(2nd Approach)		
Objects	Usage and Explanation	
Nested queries	Yes	Subqueries present, no EXISTS or IN command used
System functions	Yes	The MIN function is used in the inner subquery to find the minimum age.
Nested queries-EXISTS, Nested queries-IN , HAVING Clauses		No
JOINS	Yes	INNER JOINs are used to connect the **Restaurants**, **Ratings**, and **Consumers** tables based on their respective keys (Restaurant_ID and Consumer_ID).
Use of GROUP BY, ORDER BY	Yes	GROUP BY is used in the inner subquery to group the results by Restaurant_ID. ORDER BY is used at the end to sort the results by Food_Rating.

Tables 2.5 and 2.6 gives the database objects details of the codes of first and second approach queries.

2.6 Stored Procedure for Updating Service Rating Based on Parking Availability

Task 2: Part 2:

5. Write a stored procedure for the query given as:- Update the Service_rating of all restaurants to '2' if they have parking available, either as 'yes' or 'public'

The stored procedure **UpdateServiceRatingWithParking** updates the Service_rating of **Ratings** table of all restaurants to '2' if they have parking available, either as 'yes' or 'public' in the **Restaurants** table. It provides a convenient and automated way to maintain the Service_rating based on parking availability. Encapsulating this logic in a stored procedure, simplifies the process of updating multiple records in the database, ensuring consistency and accuracy in the data. This procedure can be executed whenever there is a change in parking availability or as part of routine maintenance tasks to keep the database up-to-date.

```
-- Creating stored procedure to
--Update the Service_rating of all restaurants to '2' if they have parking available, either as 'yes' or 'public'
DROP PROCEDURE IF EXISTS UpdateServiceRatingWithParking;
CREATE PROCEDURE UpdateServiceRatingWithParking
AS
BEGIN
  SET NOCOUNT ON;

  BEGIN TRY
    -- Start a transaction
    BEGIN TRANSACTION;

    -- Update the Service_Rating of restaurants with parking available to '2'
    UPDATE Ratings
    SET Service_Rating = 2
    WHERE Restaurant_ID IN (
      SELECT R.Restaurant_ID
      FROM Restaurants R
      WHERE R.Parking IN ('Yes', 'Public')
    );

    -- Commit the transaction if successful
    COMMIT TRANSACTION;
  END TRY
  BEGIN CATCH
    -- Rollback the transaction if an error occurs
    IF @@TRANCOUNT > 0
      ROLLBACK TRANSACTION;

    -- Raise the error
    DECLARE @ErrorMessage NVARCHAR(4000) = ERROR_MESSAGE();
    RAISERROR(@ErrorMessage, 16, 1);
  END CATCH;
END;
EXEC UpdateServiceRatingWithParking;
```

Figure 2.13 Stored procedure **UpdateServiceRatingWithParking** for updating service rating based on parking availability

The stored procedure **UpdateServiceRatingWithParking** utilizes BEGIN TRANSACTION statement to mark the initiation of a database transaction, ensuring that subsequent database operations are treated as a single logical unit of work. The COMMIT TRANSACTION statement finalizes and permanently

applies the changes made during the transaction to the database if all operations are successful, maintaining data integrity. Conversely, the ROLLBACK TRANSACTION statement acts as a safety mechanism, undoing all changes made during the transaction in the event of an error or exception, ensuring that the database returns to its original state and preventing incomplete or inconsistent updates. Together, these transaction control statements provide robustness and reliability to database operations, adhering to the principles of atomicity, consistency, isolation, and durability (ACID properties). The TRY-CATCH block is included for error handling within the stored procedure, raising error message and to maintain data integrity.

A view **ParkingServiceRatingView** is created to view the service rating of the restaurants having parking either as 'yes' or 'public'. Figure 2.14 gives the **ParkingServiceRatingView** code and results before and after updation.

View before updating service rating based on parking availability

	Restaurant_ID	Restaurant_Name	Parking	Serv...
1	135085	Tortas Locas Hipocampo	Public	2
2	132560	Puesto de Gorditas	Public	0
3	135088	Cafeteria Cenidet	Public	2
4	132608	Hamburguesas La Perica	Public	1
5	132717	Tortas Hawaii	Public	0
6	132594	Tacos De Barbacoa Enfre...	Public	0
7	132608	Hamburguesas La Perica	Public	0
8	132560	Puesto de Gorditas	Public	0
9	132717	Tortas Hawaii	Public	2
10	132608	Hamburguesas La Perica	Public	1
11	132723	Gordas De Morales	Public	1
12	135074	Restaurante La Parroquia...	Public	1
13	135085	Tortas Locas Hipocampo	Public	1
14	135032	Cafeteria Y Restaurant El...	Public	2
15	135058	Restaurante Tiberius	Public	1
16	132594	Tacos De Barbacoa Enfre...	Public	1
17	132723	Gordas De Morales	Public	2
18	135088	Cafeteria Cenidet	Public	1
19	135021	Subway	Public	0
20	135021	Subway	Public	1
21	135088	Cafeteria Cenidet	Public	0
22	132706	Gorditas Dona Tota	Public	0
23	132608	Hamburguesas La Perica	Public	2
24	135088	Cafeteria Cenidet	Public	2
25	135021	Subway	Public	0
26	135021	Subway	Public	2
27	135088	Cafeteria Cenidet	Public	0
28	135088	Cafeteria Cenidet	Public	1
29	135074	Restaurante La Parroquia...	Public	2
30	132755	La Estrella De Dimas	Public	2
31	135032	Cafeteria Y Restaurant El...	Public	1
32	135032	Cafeteria Y Restaurant El...	Public	1
33	135085	Tortas Locas Hipocampo	Public	1
34	135032	Cafeteria Y Restaurant El...	Public	2
35	135085	Tortas Locas Hipocampo	Public	2
36	135085	Tortas Locas Hipocampo	Public	2
37	135032	Cafeteria Y Restaurant El...	Public	0
38	135032	Cafeteria Y Restaurant El...	Public	1
39	135074	Restaurante La Parroquia...	Public	2
40	135032	Cafeteria Y Restaurant El...	Public	1
41	135021	Subway	Public	0
42	135032	Cafeteria Y Restaurant El...	Public	2
43	132834	Gorditas Doa Gloria	Public	2
44	135032	Cafeteria Y Restaurant El...	Public	1
45	132834	Gorditas Doa Gloria	Public	1
46	135032	Cafeteria Y Restaurant El...	Public	1
47	132834	Gorditas Doa Gloria	Public	1
48	135032	Cafeteria Y Restaurant El...	Public	2
49	135085	Tortas Locas Hipocampo	Public	2
50	135085	Tortas Locas Hipocampo	Public	2
51	135032	Cafeteria Y Restaurant El...	Public	1
52	135032	Cafeteria Y Restaurant El...	Public	2
53	135085	Tortas Locas Hipocampo	Public	0
54	135032	Cafeteria Y Restaurant El...	Public	1
55	135032	Cafeteria Y Restaurant El...	Public	2
56	135085	Tortas Locas Hipocampo	Public	2
57	135032	Cafeteria Y Restaurant El...	Public	2
58	132834	Gorditas Doa Gloria	Public	1
59	135058	Restaurante Tiberius	Public	2

(571 rows affected)

In total 571 rows are affected, only 59 are shown here.

View after updating service rating based on parking availability

	Restaurant_ID	Restaurant_Name	Parking	Servi…
1	135085	Tortas Locas Hipoca…	Public	2
2	132560	Puesto de Gorditas	Public	2
3	135088	Cafeteria Cenidet	Public	2
4	132608	Hamburguesas La Pe…	Public	2
5	132717	Tortas Hawaii	Public	2
6	132594	Tacos De Barbacoa E…	Public	2
7	132608	Hamburguesas La Pe…	Public	2
8	132560	Puesto de Gorditas	Public	2
9	132717	Tortas Hawaii	Public	2
10	132608	Hamburguesas La Pe…	Public	2
11	132723	Gordas De Morales	Public	2
12	135074	Restaurante La Parro…	Public	2
13	135085	Tortas Locas Hipoca…	Public	2
14	135032	Cafeteria Y Restaura…	Public	2
15	135058	Restaurante Tiberius	Public	2
16	132594	Tacos De Barbacoa E…	Public	2
17	132723	Gordas De Morales	Public	2
18	135088	Cafeteria Cenidet	Public	2
19	135021	Subway	Public	2
20	135021	Subway	Public	2
21	135088	Cafeteria Cenidet	Public	2
22	132706	Gorditas Dona Tota	Public	2
23	132608	Hamburguesas La Pe…	Public	2
24	135088	Cafeteria Cenidet	Public	2
25	135021	Subway	Public	2
26	135021	Subway	Public	2
27	135088	Cafeteria Cenidet	Public	2
28	135088	Cafeteria Cenidet	Public	2
29	135074	Restaurante La Parro…	Public	2
30	132755	La Estrella De Dimas	Public	2
31	135032	Cafeteria Y Restaura…	Public	2
32	135032	Cafeteria Y Restaura…	Public	2
33	135085	Tortas Locas Hipoca…	Public	2
34	135032	Cafeteria Y Restaura…	Public	2
35	135085	Tortas Locas Hipoca…	Public	2
36	135085	Tortas Locas Hipoca…	Public	2
37	135032	Cafeteria Y Restaura…	Public	2
38	135032	Cafeteria Y Restaura…	Public	2
39	135074	Restaurante La Parro…	Public	2
40	135032	Cafeteria Y Restaura…	Public	2
41	135021	Subway	Public	2
42	135032	Cafeteria Y Restaura…	Public	2
43	132834	Gorditas Doa Gloria	Public	2
44	135032	Cafeteria Y Restaura…	Public	2
45	132834	Gorditas Doa Gloria	Public	2
46	135032	Cafeteria Y Restaura…	Public	2
47	132834	Gorditas Doa Gloria	Public	2
48	135032	Cafeteria Y Restaura…	Public	2
49	135085	Tortas Locas Hipoca…	Public	2
50	135085	Tortas Locas Hipoca…	Public	2
51	135032	Cafeteria Y Restaura…	Public	2
52	135032	Cafeteria Y Restaura…	Public	2
53	135085	Tortas Locas Hipoca…	Public	2
54	135032	Cafeteria Y Restaura…	Public	2
55	135032	Cafeteria Y Restaura…	Public	2
56	135085	Tortas Locas Hipoca…	Public	2
57	135032	Cafeteria Y Restaura…	Public	2
58	132834	Gorditas Doa Gloria	Public	2
59	135058	Restaurante Tiberius	Public	2

Figure 2.14 Results of the view **ParkingServiceRatingView** before and after executing stored procedure **UpdateServiceRatingWithParking** for updating service rating based on parking availability

Table 2.7 List of objects used in Task 2 Part 2 Q 5

Task 2 Part 2 Q 5: Stored procedure **UpdateServiceRatingWithParking** for updating service rating based on parking availability		
Objects	Usage and Explanation	
System functions	Yes	BEGIN TRANSACTION, COMMIT TRANSACTION, ROLLBACK TRANSACTION, @@TRANCOUNT, ERROR_MESSAGE(), and RAISERROR..
Nested queries-EXISTS, Nested queries-IN , Use of GROUP BY, HAVING ,ORDER BY Clauses, JOINS		No

2.7 Locating Mexican Restaurants Near Consumer's Location

Task 2: Part 2:

6.1 Create a query that returns the restaurants in the vicinity of the consumer, serving specific Cuisine food, based on latitude and longitude

The query given in Figure 2.15 is designed to retrieve information about restaurants in the vicinity of a consumer, specifically focusing on those serving a specific cuisine, say 'Mexican' as given in the codes. The query filters restaurants based on their latitude and longitude, matching up to zero decimal places, to identify those located near the consumer's location. It utilizes a subquery to ensure that only restaurants serving Mexican cuisine are included in the results. The **ROUND()** function is used to match the latitude and longitude values to the consumer's location, and the results are ordered by the restaurant ID for clarity. Overall, this query facilitates the identification of nearby Mexican restaurants for the consumer.

```
--Task 2 Part 1 Q 6 Query 1
---- Creating query that returns the restaurants in the vicinity of the consumer, serving
--specific Cuisine food, based on latitude and longitude matching up to the zero decimal places:
SELECT R.Restaurant_ID, R.Name AS Restaurant_Name, R.City, R.State, R.Country, R.Zip_Code, R.Latitude, R.Longitude
FROM Restaurants R
WHERE EXISTS (
    SELECT 1
    FROM Restaurant_Cuisines RC
    WHERE RC.Restaurant_ID = R.Restaurant_ID
    AND RC.Cuisine = 'Mexican'
)
AND ROUND(R.Latitude, 0) = ROUND(23.752, 0)
AND ROUND(R.Longitude, 0) = ROUND(-99.166, 0)
GROUP BY R.Restaurant_ID, R.Name, R.City, R.State, R.Country, R.Zip_Code, R.Latitude, R.Longitude
ORDER BY R.Restaurant_ID;
```

	Restaurant_ID	Restaurant_Name	City	State	Country	Zip_Code	Latitude	Longitude
1	132584	Gorditas Doña Tota	Ciudad Victoria	Tamaulipas	Mexico	NULL	23.7523648000	-99.1652879000
2	132594	Tacos De Barbacoa Enfrente Del Tec	Ciudad Victoria	Tamaulipas	Mexico	NULL	23.7521677000	-99.1657090000
3	132608	Hamburguesas La Perica	Ciudad Victoria	Tamaulipas	Mexico	NULL	23.7588052000	-99.1651297000
4	132613	Carnitas Mata	Ciudad Victoria	Tamaulipas	Mexico	NULL	23.7529035000	-99.1650760000
5	132630	Palomo Tec	Ciudad Victoria	Tamaulipas	Mexico	NULL	23.7529305000	-99.1644725000
6	132663	Tacos Abi	Ciudad Victoria	Tamaulipas	Mexico	NULL	23.7525107000	-99.1669536000
7	132665	Tacos Correcaminos	Ciudad Victoria	Tamaulipas	Mexico	NULL	23.7367977000	-99.1342413000
8	132668	Tacos El Guero	Ciudad Victoria	Tamaulipas	Mexico	NULL	23.7382120000	-99.1519547000
9	132706	Gorditas Dona Tota	Ciudad Victoria	Tamaulipas	Mexico	NULL	23.7292162000	-99.1323571000
10	132715	Tacos De La Estacion	Ciudad Victoria	Tamaulipas	Mexico	NULL	23.7324226000	-99.1586602000
11	132732	Taqueria El Amigo	Ciudad Victoria	Tamaulipas	Mexico	87018	23.7543569000	-99.1712880000
12	132740	Carreton De Flautas Y Migadas	Ciudad Victoria	Tamaulipas	Mexico	NULL	23.7521965000	-99.1666317000
13	135104	Vips	Ciudad Victoria	Tamaulipas	Mexico	NULL	23.7529821000	-99.1684341000

Figure 2.15 Locating Mexican restaurants near consumer's location given by latititude, longitude (23.752, -99.166)

Table 2.8 List of objects used in Task 2 Part 2 Q 6.1

Task 2 Part 2 Q 6.1: Locating Mexican restaurants near consumer's location		
Objects	Usage and Explanation	
Nested queries - EXISTS	Yes	The EXISTS clause is used to check for the existence of records in the **Restaurant_Cuisines** table based on the condition specified.
System functions	Yes	ROUND() function is used to round the latitude and longitude values to zero decimal places.
Nested queries-EXISTS, Nested queries-IN , HAVING Clauses, JOINS		No
Use of GROUP BY, ORDER BY	Yes	GROUP BY is used to group the result set by restaurant attributes. ORDER BY is used to sort the result set by Restaurant_ID in ascending order.

2.8 Restaurants Serving American or Japanese Cuisine with Overall Rating Greater Than 0

Task 2: Part 2:

6.2 Creating query that returns the restaurant details restaurant_Id, Name, Cuisine, Average Overall Rating that serves Cuisine - 'American', or 'Japanese' and have the overall rating greater than 0

Figure 2.16 gives the number of restaurants satisfying the condition of overall rating greter than 0 and serving cuisine ‘American’ or ‘Japanese’

SELECT Ra.Restaurant_ID,Ra.Overall_Rating FROM Ratings Ra JOIN Restaurants R ON R.Restaurant_ID = Ra.Restaurant_ID WHERE Ra.Restaurant_ID = R.Restaurant_ID AND Ra.Overall_Rating >0;	SELECT R.Restaurant_ID, R.Name AS Restaurant_Name, RC.Cuisine FROM Restaurants R JOIN Restaurant_Cuisines RC ON R.Restaurant_ID = RC.Restaurant_ID WHERE RC.Cuisine IN ('American', 'Japanese') ORDER BY R.Restaurant_ID DESC;
Listing the restaurants with overall rating greater than 0. The number of rows affected is (907 rows affected)	Listing the restaurants with overall rating greater than 0 and servivg American or Japanese cuisine . The number of rows affected is (10 rows affected)

Figure 2.16 Listing the restaurants satisfying the condition of overall rating greter than 0 and additionally serving cuisine ‘American’ or ‘Japanese’

The query given in Figure 2.16 retrieves restaurant details, including Restaurant ID, Name, Cuisine, and the average Overall Rating, for these 10 restaurants serving 'American' or 'Japanese' cuisine with an overall rating greater than 0 in a single query. It joins the **Restaurants**, **Restaurant_Cuisines**, and **Ratings** tables to gather the necessary information. The **WHERE** clause filters restaurants based on their cuisine, including only those serving American or Japanese food. Additionally, the **HAVING** clause ensures that only restaurants with an overall rating greater than 0 are included in the results. Finally, the results are grouped by Restaurant ID, Name, and Cuisine, and the average overall rating is calculated for each restaurant. Overall, the query helps identify restaurants offering 'American' or 'Japanese' cuisine with high overall ratings.

```
--Task 2 Part 1 Q 6 Query 2
--Creating query that returns the restaurant details
-- Restaurant_Id, Name, Cuisine, Average Overall_Rating
--that serves Cuisine - 'American', or  'Japanese' and have the overall rating greater than 0
SELECT R.Restaurant_ID, R.Name AS Restaurant_Name, RC.Cuisine,AVG(Ra.Overall_Rating) AS Avg_Overall_Rating
FROM Restaurants R
JOIN Restaurant_Cuisines RC ON R.Restaurant_ID = RC.Restaurant_ID
JOIN Ratings Ra ON R.Restaurant_ID = Ra.Restaurant_ID
WHERE RC.Cuisine IN ('American', 'Japanese')
GROUP BY R.Restaurant_ID, R.Name, RC.Cuisine
HAVING EXISTS (
    SELECT 1
    FROM Ratings Ra2
    WHERE Ra2.Restaurant_ID = R.Restaurant_ID
    AND Ra2.Overall_Rating >0
) ORDER BY R.Restaurant_ID;
```

	Restaurant_ID	Restaurant_Name	Cuisine	Avg_Overall_Rating
1	132583	McDonalds Centro	American	1
2	132766	Mikasa	Japanese	0
3	132851	KFC	American	1
4	132872	Pizzeria Julios	American	0
5	132875	Shi Ro Ie	Japanese	1
6	132951	Vips	American	1
7	132958	Tacos Los Volcanes	American	1
8	134999	Kiku Cuernavaca	Japanese	1
9	135034	Michiko Restaurant Japones	Japanese	2
10	135072	Sushi Itto	Japanese	1

Figure 2.17 Restaurants serving American or Japanese cuisine with overall rating greater than 0

Table 2.9 gives the database object details of the query given in Figure 2.17

Table 2.9 List of objects used in Task 2 Part 2 Q 6.2

Task 2 Part 2 Q 6.2: Restaurants serving American or Japanese cuisine with overall rating greater than 0		
Objects	Usage and Explanation	
Nested queries - EXISTS	Yes	The query uses a nested EXISTS subquery to check for the existence of condition – overall rating greater than 0.
Nested queries-IN	Yes	For checking the type of cuisine
System functions	Yes	The AVG function is used to calculate the average overall rating for each restaurant.
JOINS	Yes	The query uses INNER JOINs to join the Restaurants, Restaurant_Cuisines, and Ratings tables based on Restaurant_ID.
Use of GROUP BY, ORDER BY , HAVING clauses	Yes	The query uses the GROUP BY clause to group the results by Restaurant_ID, Name, and Cuisine. The HAVING clause is used to filter the grouped results based on the existence of condition – overall rating greter than 0. ORDER BY is used to sort the result set by Restaurant_ID in ascending order. The ORDER BY clause is used to sort the final result set by Restaurant_ID in ascending order.

2.9 Retrieving Restaurants with Specific Cuisines, Food Ratings, and Price Ranges

Task 2: Part 2:

6.3 Creating query that returns the restaurants with cuisines as ‘bar’, ‘fast food’ or ‘chinese’ and having food rating greater than 0 and price range either ‘low’ or ‘medium’

The query given in Figure 2.18 retrieves restaurants specializing in bar, fast food, or Chinese cuisines, ensuring their food ratings exceed zero and their price

ranges fall within 'Low' or 'Medium' categories. By joining multiple tables, including **Restaurants**, **Restaurant_Cuisines**, **Ratings**, and **Consumers**, it gathers comprehensive data on each restaurant's cuisine, food ratings, and consumer demographics. Aggregating this information allows for insights such as average food rating, total consumer count, and age demographics.

```
--Task 2 Part 1 Q 6 Query 3
--creating query that returns the restaurants with cuisines as bar, fast food or chinese and having food rating greater than 0 and price range either low or medium
SELECT R.Restaurant_ID, R.Name AS Restaurant_Name, RC.Cuisine, R2.Price,
    AVG(Ra.Food_Rating) AS Avg_Food_Rating,
    COUNT(DISTINCT C.Consumer_ID) AS Total_Consumers,
    MAX(C.Age) AS Max_Consumer_Age,
    MIN(C.Age) AS Min_Consumer_Age
FROM Restaurants R
JOIN Restaurant_Cuisines RC ON R.Restaurant_ID = RC.Restaurant_ID
JOIN Ratings Ra ON R.Restaurant_ID = Ra.Restaurant_ID
JOIN Consumers C ON Ra.Consumer_ID = C.Consumer_ID
JOIN Restaurants R2 ON R.Restaurant_ID = R2.Restaurant_ID
WHERE RC.Cuisine IN ('Fast Food', 'Bar', 'Chinese') -- Selecting cuisines
GROUP BY R.Restaurant_ID, R.Name, RC.Cuisine, R2.Price
HAVING EXISTS (
  SELECT 1
  FROM Ratings Ra2
  WHERE Ra2.Restaurant_ID = R.Restaurant_ID
  AND Ra2.Food_Rating > 0
)
AND R2.Price IN ('Low', 'Medium')
ORDER BY R.Restaurant_ID DESC;
```

	Restaurant_ID	Restaurant_Name	Cuisine	Price	Avg_Food_Rating	Total_Consumers	Max_Consumer_Age	Min_Consumer_Age
1	135086	McDonalds Parque Tangamanga	Fast Food	Medium	0	10	29	21
2	135085	Tortas Locas Hipocampo	Fast Food	Medium	1	36	82	21
3	135071	Restaurante La Cantina	Bar	Medium	0	9	23	21
4	135069	Abondance Restaurante Bar	Bar	Low	0	12	29	21
5	135059	Restaurant Bar Hacienda Los Martinez	Bar	Medium	1	9	25	21
6	135057	El Herradero Restaurante And Bar	Bar	Medium	1	15	82	20
7	135046	Restaurante El Reyecito	Fast Food	Medium	1	11	31	21
8	135044	Restaurant Wu Zhuo Yi	Chinese	Medium	1	4	31	21
9	135043	Pizza Clasica	Fast Food	Medium	1	9	24	20
10	135042	Restaurant Oriental Express	Chinese	Medium	1	20	82	20
11	135041	Luna Cafe	Bar	Medium	1	17	82	20
12	135021	Subway	Fast Food	Low	1	9	69	19
13	135019	Restaurant Bar Coty Y Pablo	Bar	Low	1	6	26	18
14	132937	Rockabilly	Bar	Low	1	4	23	21
15	132921	Crudalia	Bar	Low	1	17	30	21
16	132717	Tortas Hawaii	Fast Food	Medium	1	3	24	21
17	132609	Pollo Frito Buenos Aires	Fast Food	Low	0	5	24	21

Figure 2.18 Retrieving Restaurants with Specific Cuisines, Food Ratings, and Price Ranges

Table 2.10 explains the details of the database objects used in query shown in Figure 2.18

Table 2.10 List of objects used in Task 2 Part 2 Q 6.3

Task 2 Part 2 Q 6.3: Retrieving Restaurants with Specific Cuisines, Food Ratings, and Price Ranges		
Objects	Usage and Explanation	
Nested queries - EXISTS	Yes	The query uses a nested EXISTS subquery to check for the existence of condition– overall rating greater than 0.
Nested queries-IN	NO	
System functions	Yes	AVG(), MIN(), MAX()
JOINS	Yes	The query uses INNER JOINs to join the Restaurants, Restaurant_Cuisines, Ratings, Consumers.
Use of GROUP BY, ORDER BY, HAVING clauses	Yes	The query uses the GROUP BY clause to group the results by Restaurant_ID, Name, and Cuisine.. The HAVING clause is used to filter the grouped results based on the existence of specific conditions. The ORDER BY clause is used to sort the final result set by Restaurant_ID in descending order.

2.10 Café Restaurants with Specific Ratings and Cuisines

Task 2: Part 2:

6.4 Write a query that lists the restaurants starting with ‘Café’ with overall rating either 1 or 2 , food rating greater than 0 and cuisine ending with pattern ‘ria’ along with last four characters of the consumer IDs

The query given in Figure 2.19 is designed to retrieve information about restaurants that meet specific criteria: their names start with 'Café', located in a state having patter ‘-an-‘ in the name, their overall ratings are either 1 or 2, and their cuisine names end with the pattern 'ria'. This query gives insights about café restaurants with cuisine ‘Cafeteria’, having overall rating either 1 or 2 and food rating greater than 0. Table 2.11 details the various database objects used in the query presented in Figure 2.19.

```sql
--Task 2 Part 1 Q 6 Query 4
--Listing the consumer ID (Trimmed), restaurant id, Restaurant name(Uppercase),
SELECT SUBSTRING(LTRIM(RTRIM(C.Consumer_ID)), LEN(LTRIM(RTRIM(C.Consumer_ID))) - 3, 4) AS Trimmed_Consumer_ID,
    R.Restaurant_ID, UPPER(R.Name) AS Restaurant_Name_Uppercase, R.State,
    Ra.Overall_Rating, RC.Cuisine
FROM Consumers C
JOIN Ratings Ra ON C.Consumer_ID = Ra.Consumer_ID
JOIN Restaurants R ON Ra.Restaurant_ID = R.Restaurant_ID
JOIN Restaurant_cuisines RC ON R.Restaurant_ID = RC.Restaurant_ID
WHERE (R.State LIKE '%an%' )
 AND R.Name LIKE 'Cafe%'
 AND Ra.Overall_Rating IN (1, 2)
 AND RC.Cuisine IN (
    SELECT RC2.Cuisine
    FROM Restaurant_cuisines RC2
    WHERE PATINDEX('%ria', RC2.Cuisine) > 0
 )
GROUP BY SUBSTRING(LTRIM(RTRIM(C.Consumer_ID)), LEN(LTRIM(RTRIM(C.Consumer_ID))) - 3, 4),
    R.Restaurant_ID, R.Name, R.State, Ra.Overall_Rating, RC.Cuisine
HAVING EXISTS (
   SELECT 1
   FROM Ratings Ra2
   WHERE Ra2.Restaurant_ID = R.Restaurant_ID
   AND Ra2.Food_Rating > 0
)
ORDER BY Ra.Overall_Rating DESC;
```

	Trimmed_Consumer_ID	Restaurant_ID	Restaurant_Name_Uppercase	State	Overall_Rating	Cuisine
1	1003	132922	CAFE PUNTA DEL CIELO	San Luis Potosi	2	Cafeteria
2	1004	135032	CAFETERIA Y RESTAURANT EL PACIFICO	San Luis Potosi	2	Cafeteria
3	1005	135032	CAFETERIA Y RESTAURANT EL PACIFICO	San Luis Potosi	2	Cafeteria
4	1006	132922	CAFE PUNTA DEL CIELO	San Luis Potosi	2	Cafeteria
5	1016	135032	CAFETERIA Y RESTAURANT EL PACIFICO	San Luis Potosi	2	Cafeteria
6	1024	135032	CAFETERIA Y RESTAURANT EL PACIFICO	San Luis Potosi	2	Cafeteria
7	1053	135032	CAFETERIA Y RESTAURANT EL PACIFICO	San Luis Potosi	2	Cafeteria
8	1055	132572	CAFE CHAIRES	San Luis Potosi	2	Cafeteria
9	1071	135032	CAFETERIA Y RESTAURANT EL PACIFICO	San Luis Potosi	2	Cafeteria
10	1083	132572	CAFE CHAIRES	San Luis Potosi	2	Cafeteria
11	1086	132922	CAFE PUNTA DEL CIELO	San Luis Potosi	2	Cafeteria
12	1088	135032	CAFETERIA Y RESTAURANT EL PACIFICO	San Luis Potosi	2	Cafeteria
13	1090	132922	CAFE PUNTA DEL CIELO	San Luis Potosi	2	Cafeteria
14	1091	132572	CAFE CHAIRES	San Luis Potosi	2	Cafeteria
15	1098	135032	CAFETERIA Y RESTAURANT EL PACIFICO	San Luis Potosi	2	Cafeteria
16	1101	135032	CAFETERIA Y RESTAURANT EL PACIFICO	San Luis Potosi	2	Cafeteria
17	1104	135032	CAFETERIA Y RESTAURANT EL PACIFICO	San Luis Potosi	2	Cafeteria
18	1108	135032	CAFETERIA Y RESTAURANT EL PACIFICO	San Luis Potosi	2	Cafeteria
19	1109	135032	CAFETERIA Y RESTAURANT EL PACIFICO	San Luis Potosi	2	Cafeteria
20	1138	132922	CAFE PUNTA DEL CIELO	San Luis Potosi	2	Cafeteria

	Trimmed_Consumer_ID	Restaurant_ID	Restaurant_Name_Uppercase	State	Overall_Rating	Cuisine
20	1138	132922	CAFE PUNTA DEL CIELO	San Luis Potosi	2	Cafeteria
21	1006	132572	CAFE CHAIRES	San Luis Potosi	1	Cafeteria
22	1007	132572	CAFE CHAIRES	San Luis Potosi	1	Cafeteria
23	1007	135032	CAFETERIA Y RESTAURANT EL PACIFICO	San Luis Potosi	1	Cafeteria
24	1013	132572	CAFE CHAIRES	San Luis Potosi	1	Cafeteria
25	1018	135032	CAFETERIA Y RESTAURANT EL PACIFICO	San Luis Potosi	1	Cafeteria
26	1022	135032	CAFETERIA Y RESTAURANT EL PACIFICO	San Luis Potosi	1	Cafeteria
27	1029	132922	CAFE PUNTA DEL CIELO	San Luis Potosi	1	Cafeteria
28	1033	132572	CAFE CHAIRES	San Luis Potosi	1	Cafeteria
29	1046	132572	CAFE CHAIRES	San Luis Potosi	1	Cafeteria
30	1054	135032	CAFETERIA Y RESTAURANT EL PACIFICO	San Luis Potosi	1	Cafeteria
31	1061	132572	CAFE CHAIRES	San Luis Potosi	1	Cafeteria
32	1075	135032	CAFETERIA Y RESTAURANT EL PACIFICO	San Luis Potosi	1	Cafeteria
33	1090	132572	CAFE CHAIRES	San Luis Potosi	1	Cafeteria
34	1092	132572	CAFE CHAIRES	San Luis Potosi	1	Cafeteria
35	1108	132572	CAFE CHAIRES	San Luis Potosi	1	Cafeteria
36	1120	135032	CAFETERIA Y RESTAURANT EL PACIFICO	San Luis Potosi	1	Cafeteria
37	1124	135032	CAFETERIA Y RESTAURANT EL PACIFICO	San Luis Potosi	1	Cafeteria
38	1125	135032	CAFETERIA Y RESTAURANT EL PACIFICO	San Luis Potosi	1	Cafeteria
39	1132	135032	CAFETERIA Y RESTAURANT EL PACIFICO	San Luis Potosi	1	Cafeteria

Figure 2.19 Retrieving Restaurants with Specific Cuisines, Food Ratings, and Price Ranges

Table 2.11 List of objects used in Task 2 Part 2 Q 6.4

Task 2 Part 2 Q 6.4: Retrieving Restaurants with Specific Cuisines, Food Ratings, and Price Ranges		
Objects	Usage and Explanation	
Nested queries - EXISTS	Yes	The query uses a nested EXISTS subquery to check for the existence of condition– food rating greater than 0.
Nested queries-IN	Yes	To check the condition related to cuisine
System functions	Yes	SUBSTRING, LTRIM, RTRIM, UPPER, LEN, and PATINDEX.

JOINS	Yes	The query uses INNER JOINs to join the **Restaurants**, **Restaurant_Cuisines**, **Ratings**, **Consumers**.
Use of GROUP BY, ORDER BY, HAVING clauses	Yes	The GROUP BY clause is used to group the result set by columns trimmed consumer ID, Restaurant_ID, Name, State and Overall Rating. The HAVING clause is used to filter the grouped results based on food rating condition. The ORDER BY clause is used to sort the final result set by overall rating in descending order.

2.11Conclusions

Table 2.12 Key functionalities of the food service database

Key functionality	Explanation
Data Integrity	The database ensures data integrity through the implementation of constraints, relationships, and validation rules, minimizing errors and inconsistencies.
Strategic decision-making	Gaining insights into key metrics such as customer ratings on food, service, and overall performance can help in improving the cuisines, service and other hospitality related facilities.
Geospatial Analysis	Leveraging geospatial data (latitude and longitude), the database allows for geospatial analysis to identify location-based trends, optimize delivery routes, and target marketing efforts to specific geographic areas.
Feedback Mechanism	Analyzing the ratings given by the customers, restaurants identify areas for improvement and address customer concerns promptly.

The food service database allows for the efficient management of restaurant information such as name, location, cuisine, pricing, and amenities like parking availability. It enables the tracking of consumer data including demographics, preferences, and behaviour, facilitating targeted marketing and personalized services. The database implements a rating system where consumers can rate restaurants based on their experience, including overall satisfaction, food quality, and service. It categorizes restaurants based on the cuisines they offer, allowing users to search for restaurants by cuisine type. Based on tables **Consumers, Restaurants, Ratings and Restaurant_Cuisines** , the database is designed to support efficient querying, enabling users to retrieve relevant information quickly and accurately. Indexing, normalization, and proper data structuring contribute to query optimization. The key functionalities of the food service database is given in Table 2.12

It can be concluded that the food service database is designed in such a way that it provides a comprehensive platform for managing all aspects of a food service company, promoting operational efficiency, customer satisfaction, and business growth.