CS669 Term Project – Health Version

Database Schema Design for Multi-Specialty Ambulatory Health Center

Submitted By: Archana Balachandran

Dec 14, 2016

Table of Contents:

1. Entities

A comprehensive list of entities are listed below. For limiting the scope of the term project, only the most relevant entities will be selected for depicting the complete BUAHC system.

- Facility
- Building
- Physicians
- Patients
- Insurance Plans
- Wait_List
- Department
- Ambulance
- Medical Record
- Prescription
- Bill
- Slots
- Appointment
- Medical_Exams

2. Structural Business Rules: Relationships, Optionality, Plurality

In this section, the relationships between the entities, the optionality and plurality of these relationships are explored.

Facility and Building have a one-to-many relationship. .(mandatory)
A Facility must have at least one Building. A building belongs to only one Facility.

Building and Department share a one-to-many relationship. (optional)
A building may belong to one or more departments (Optional relationship). A department belongs to one Building.

Facility and Medical Records share a one-to-many relationship.(mandatory)
A facility must contain at least one medical record. A medical record belongs to one Facility.

Facility and Insurance plans share a many-to-many relationship. (optional)
A Facility may be covered by one or multiple insurance plans. An insurance plan may cover multiple facilities. (Optional relationships)

Insurance plan and Patient share a many-to-many relationship. (optional)
An insurance plan convers at least one patient. A patient may be enrolled in one or multiple insurance plans.

Patient and Appointment share a one-to-many relationship. (mandatory)
A patient books at least one appointment. An appointment is assigned to only one patient.

Appointment and Phycisians share a many-to-one relationship. (optional)
A physician may have multiple appointments. An appointment is assigned to only one Physician.

Department and Physician has a one-to-many relationship. (mandatory)
Department has at least one physician. A physician belongs to only one building.

Physician and Wait_list has a one-to-one relationship. (mandatory)
A physician can have only one wait list and a wait list belongs to one physician.

Appointment and Prescription has a one-to-one relationship(optional)
An appointment may result in at most one prescription. A prescription belongs to one appointment.

Appointment and Medical_exam share a one-to-many relationship(optional)

An appointment may result in multiple medical_exams(optional). A medical exam belongs t only one Appointment.

Appointment and Bill has a one-to-one optional relationship. .(mandatory)
An appointment generates a bill. A bill belongs to one appointment.

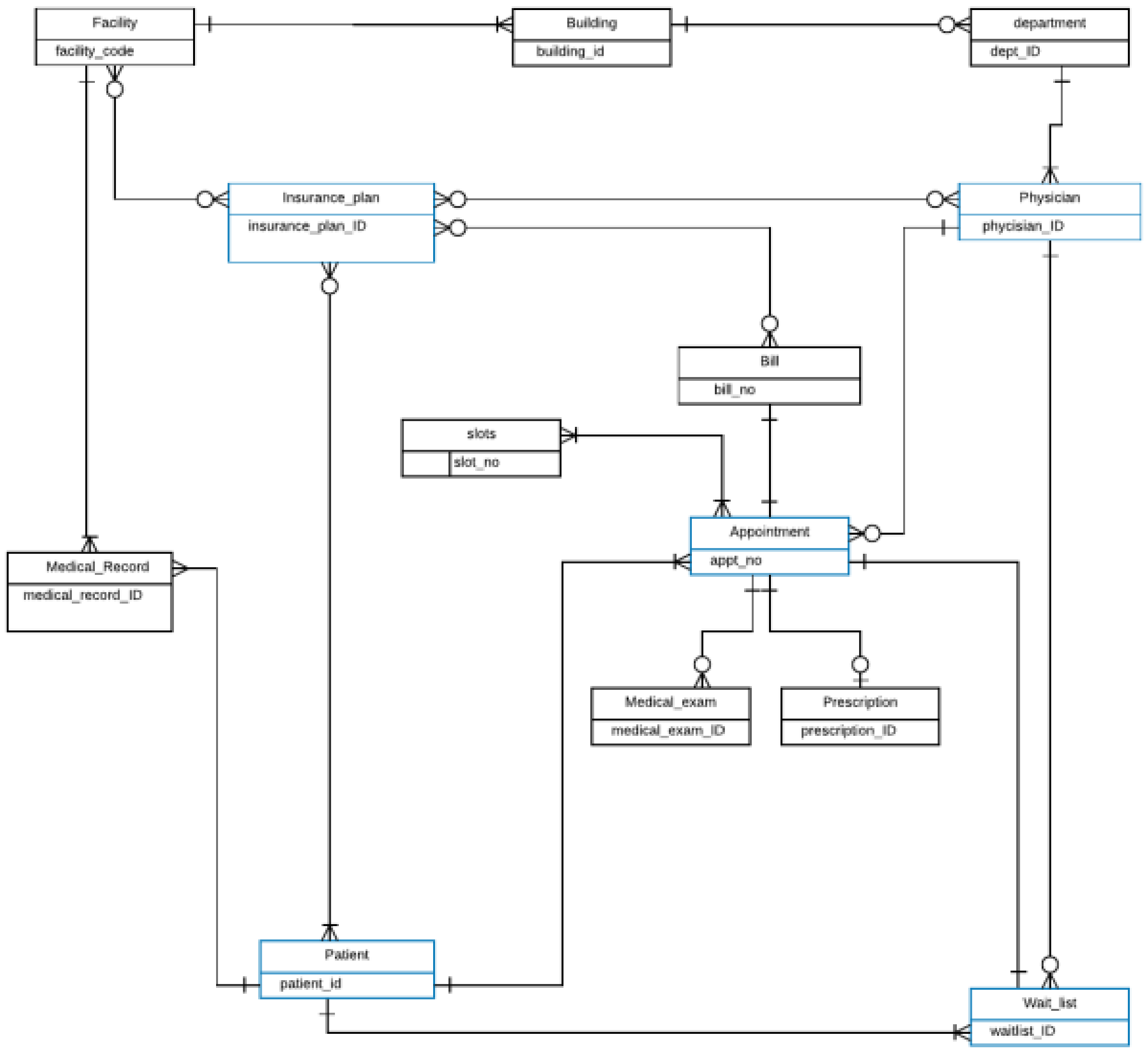
Patient and Medical_records share a one-to-many relationship.(mandatory)
A patient has at least one medical record with a doctor. A medical record belongs to one-patient.

A **patient** can have only one insurance plan
An **insurance plan** can have many patients enrolled.

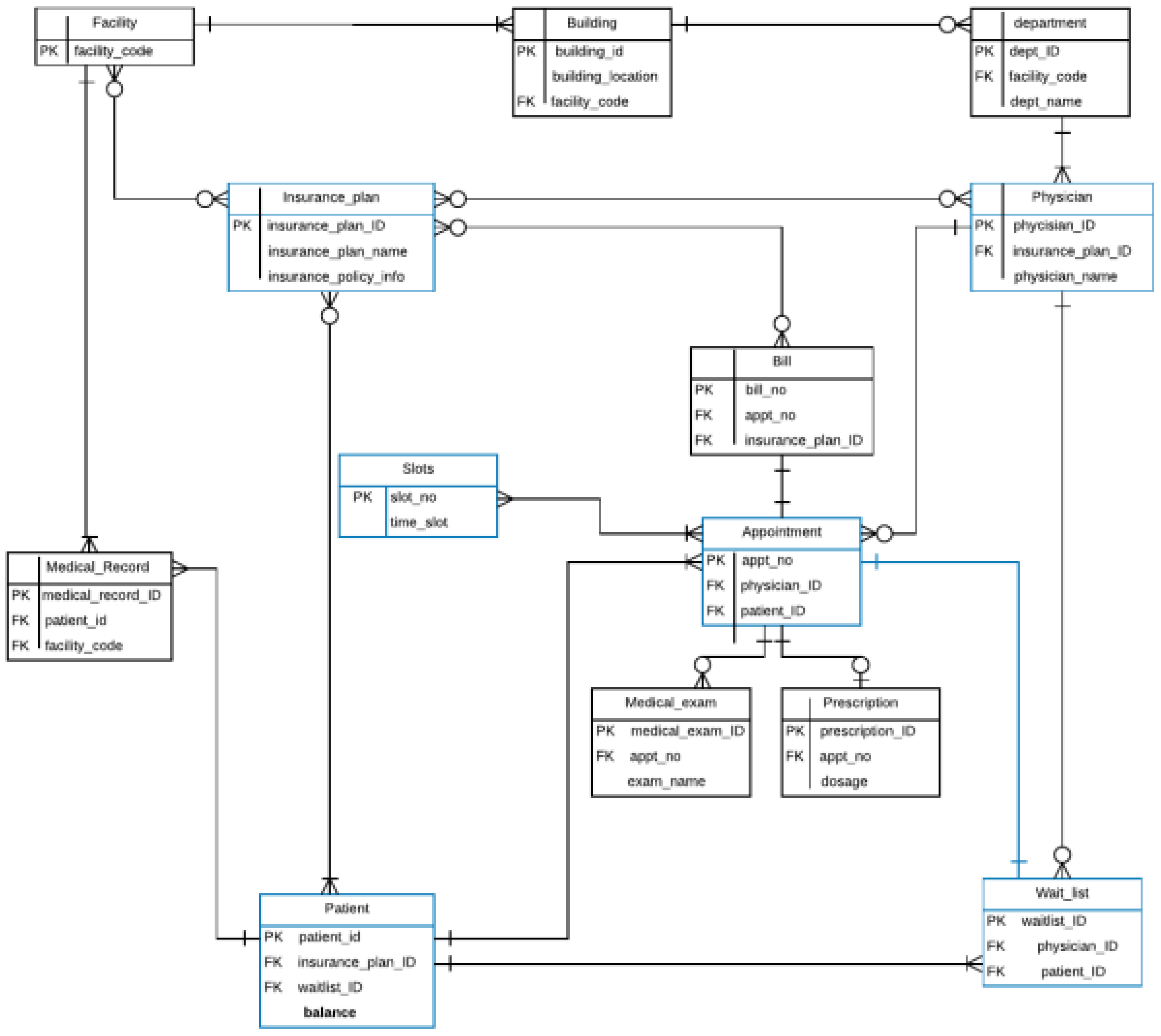
Each appointment can create one waiting_list
Each waiting_list entry belongs to one appointment.

Each appointment has one slot.
Each slot may be assigned to multiple assignments.

3. Conceptual Entity Relationship Diagram



4. Logical Entity Relationship Diagram



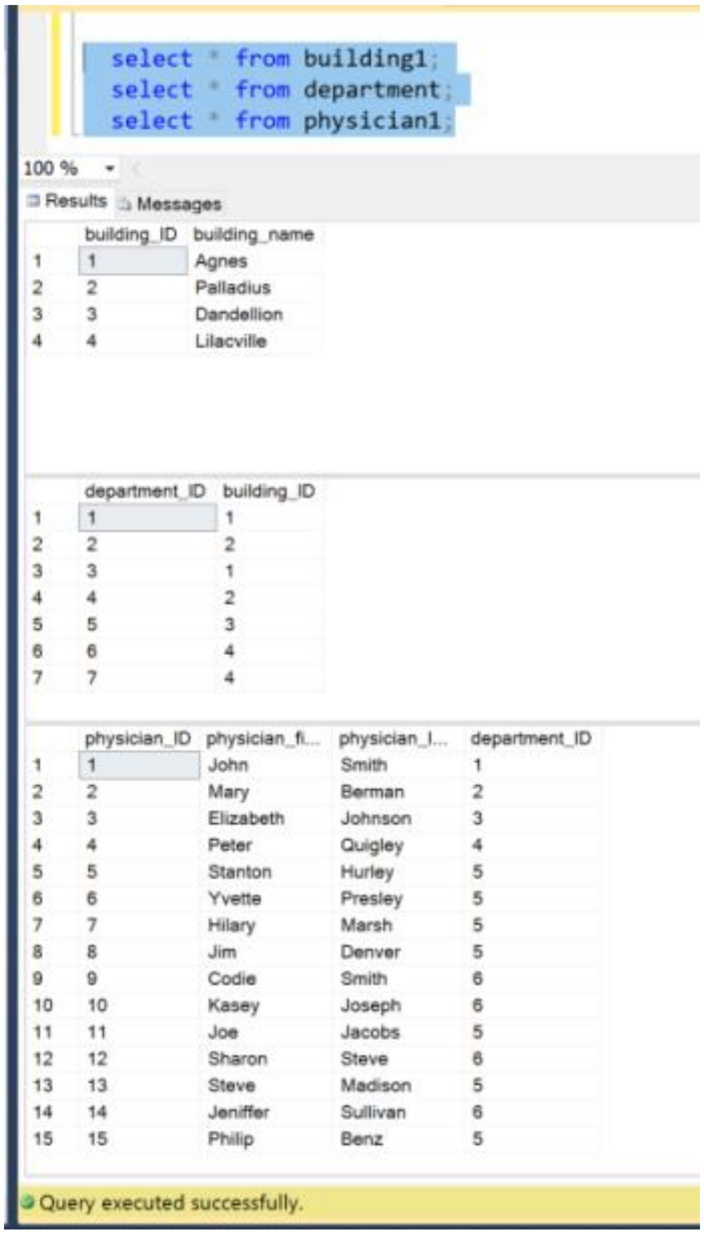
5. Use case Execution

Tables used to satisfy each Use case:

Use case	**Table**	**Columns**
#1	Building, physician	Building_ID, building_name Physician_first,physician_last
#2	Insurance, patient	Insurance_ID, insurance_name Insurance_ID, patient_first, patient_last
#3	Appointment waitlist	Waitlist_ID, physician_ID, patient_ID
#4	Waitlist Appointment physician	waitlist_ID, patient_ID, physician_ID appointment_ID, patient_ID, physician_ID, waitlist_ID physician_ID, physician_first, physician_last
#5	patient	balance, patient_ID
#6	patient	patient_ID, insurance_ID
#7	Physician Appointment	Physician_ID, physician_first, physician_last, max_capacity Appt_date, physician_ID
#8	Insurance Patient	Insurance_ID, insurance_name, copay_amt, number_of_enrollees Insurance_ID
#9	Patient Physician Appointment	Patient_first,patient_last,physician_ID Physician_first, physician_last,physician_ID Physician_ID,patient_ID
#10	appointment	Physician_ID,patient_ID

Use Case #1

Requirement: Health center management requests the first and last names of all **physicians** that work in the ”Agnes” or “Palladius” **buildings**. Write a single query that retrieves this information for management.   The table structures for building, department and physician:



```
select * from building1;
select * from department;
select * from physician1;
```

100 %

Results Messages

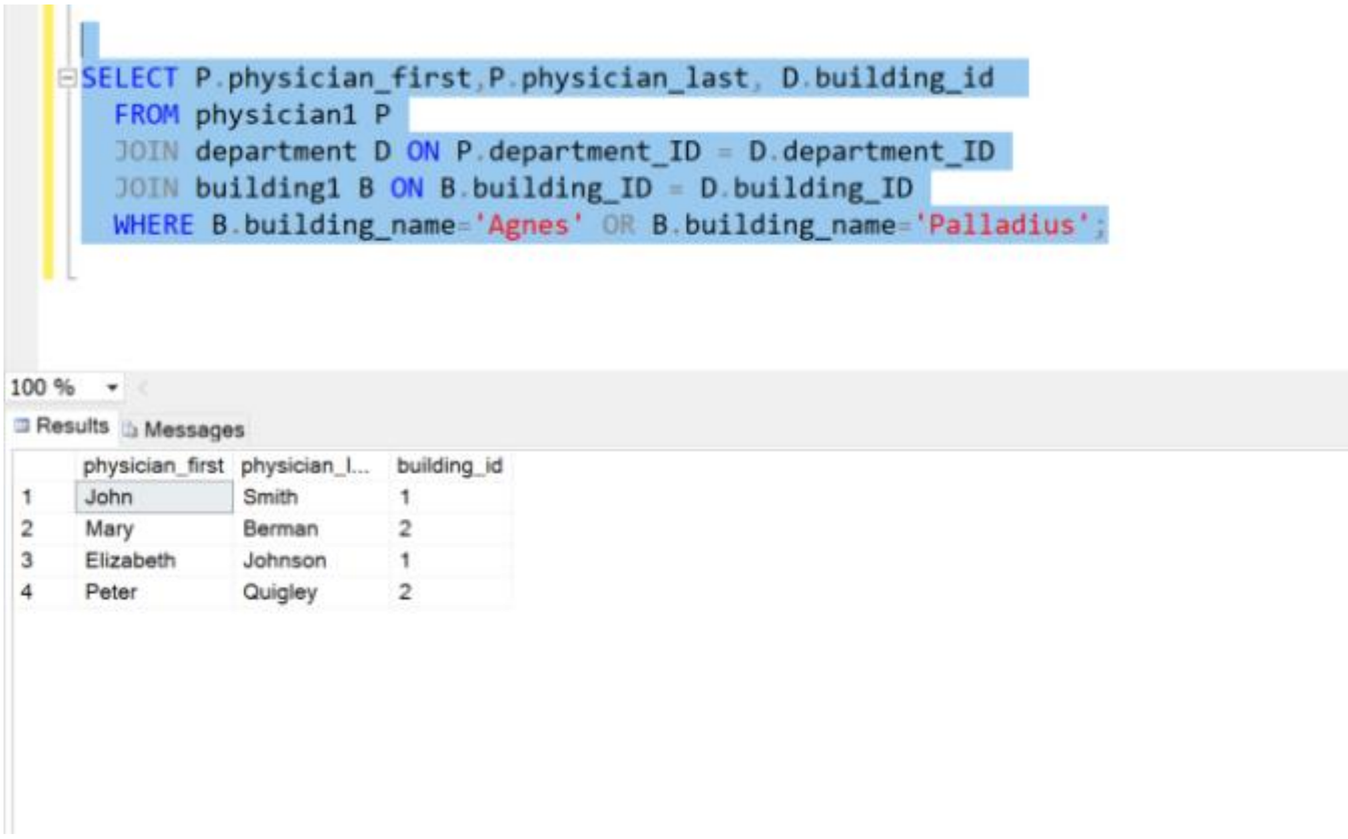
	building_ID	building_name
1	1	Agnes
2	2	Palladius
3	3	Dandellion
4	4	Lilacville

	department_ID	building_ID
1	1	1
2	2	2
3	3	1
4	4	2
5	5	3
6	6	4
7	7	4

	physician_ID	physician_fi...	physician_l...	department_ID
1	1	John	Smith	1
2	2	Mary	Berman	2
3	3	Elizabeth	Johnson	3
4	4	Peter	Quigley	4
5	5	Stanton	Hurley	5
6	6	Yvette	Presley	5
7	7	Hilary	Marsh	5
8	8	Jim	Denver	5
9	9	Codie	Smith	6
10	10	Kasey	Joseph	6
11	11	Joe	Jacobs	5
12	12	Sharon	Steve	6
13	13	Steve	Madison	5
14	14	Jeniffer	Sullivan	6
15	15	Philip	Benz	5

Query executed successfully.

Satisfying Use case 1:



```
SELECT P.physician_first,P.physician_last, D.building_id
  FROM physician1 P
  JOIN department D ON P.department_ID = D.department_ID
  JOIN building1 B ON B.building_ID = D.building_ID
  WHERE B.building_name='Agnes' OR B.building_name='Palladius';
```

100 %

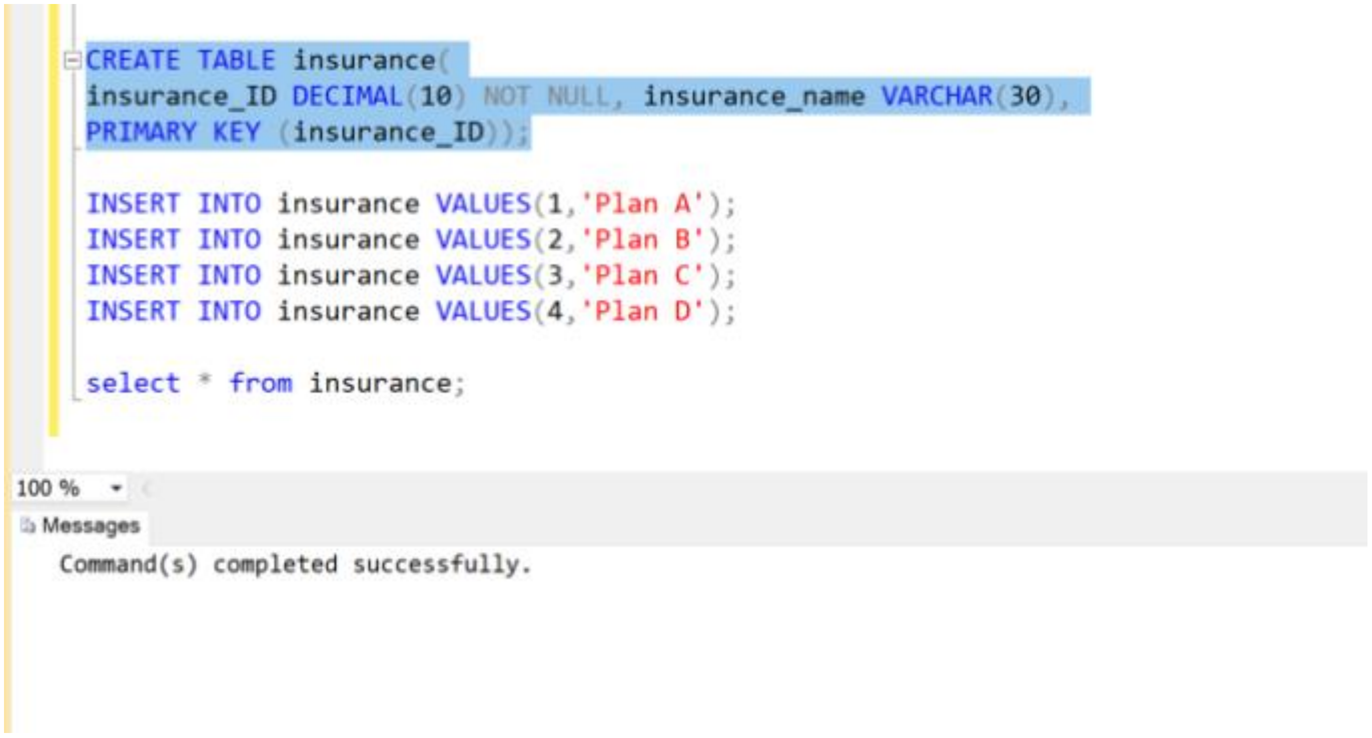
Results Messages

	physician_first	physician_l...	building_id
1	John	Smith	1
2	Mary	Berman	2
3	Elizabeth	Johnson	1
4	Peter	Quigley	2

Use Case # 2

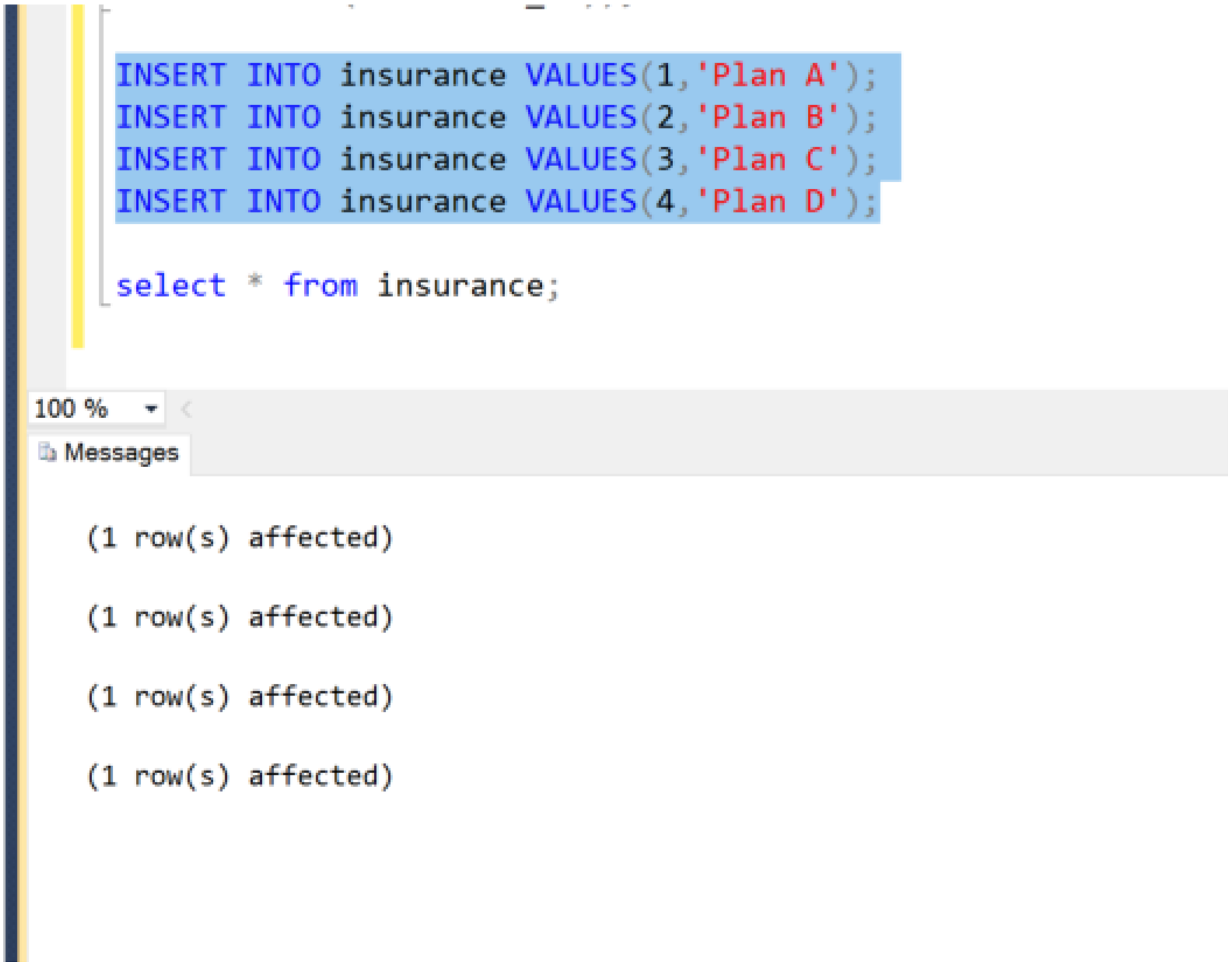
Requirement: Auditors request the names of all patients that currently have insurance, as well as the name of their current insurance plan. Write a single query that retrieves this information for the auditors.

Tables Patient,Insurance_plans – check whether there is an insurance_id NOT null, if so, for those NOT NULL values, refer insurance_plans and list the insurance_plan_name. insurance_id connects the tables.



```
CREATE TABLE insurance(
insurance_ID DECIMAL(10) NOT NULL, insurance_name VARCHAR(30),
PRIMARY KEY (insurance_ID));

INSERT INTO insurance VALUES(1,'Plan A');
INSERT INTO insurance VALUES(2,'Plan B');
INSERT INTO insurance VALUES(3,'Plan C');
INSERT INTO insurance VALUES(4,'Plan D');

select * from insurance;
```

```
Command(s) completed successfully.
```

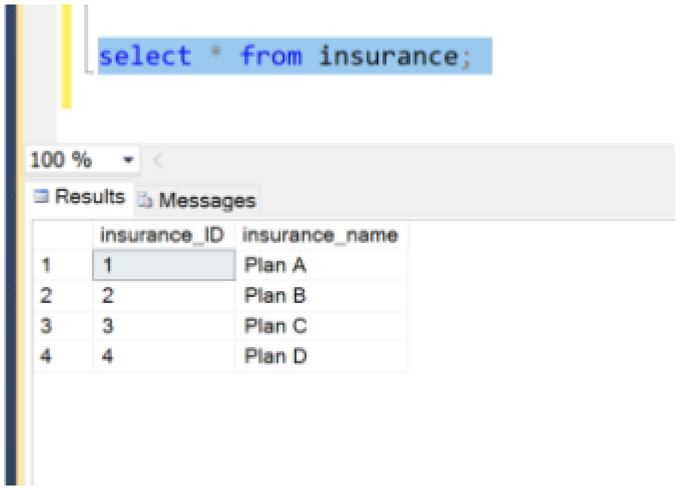


```
INSERT INTO insurance VALUES(1,'Plan A');
INSERT INTO insurance VALUES(2,'Plan B');
INSERT INTO insurance VALUES(3,'Plan C');
INSERT INTO insurance VALUES(4,'Plan D');

select * from insurance;
```

100 %

Messages

```
(1 row(s) affected)

(1 row(s) affected)

(1 row(s) affected)

(1 row(s) affected)
```

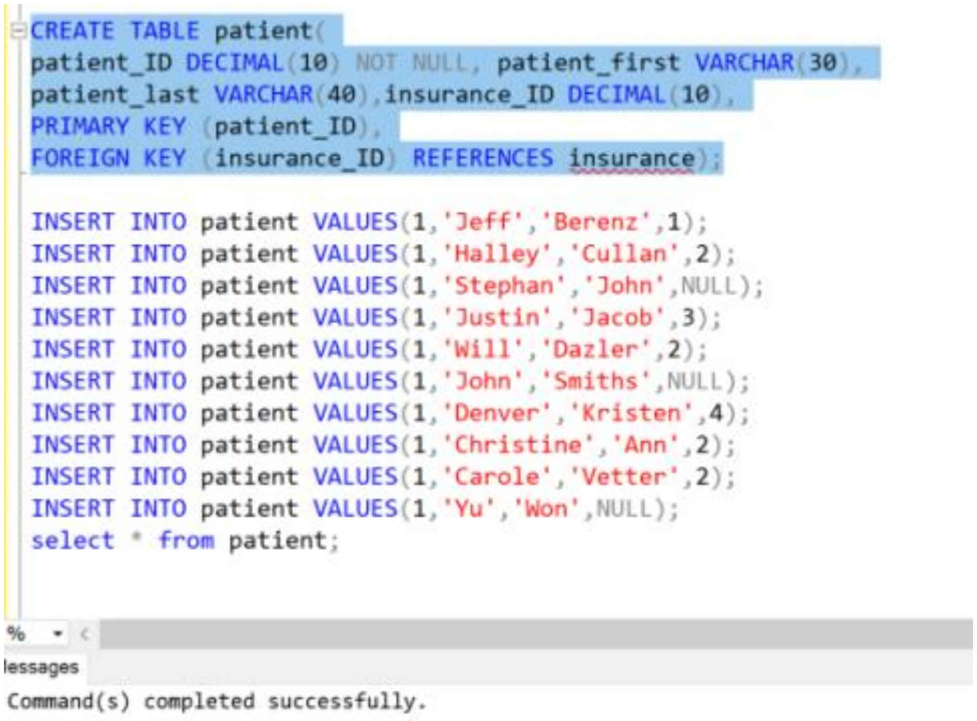


```
select * from insurance;
```

100 %

Results Messages

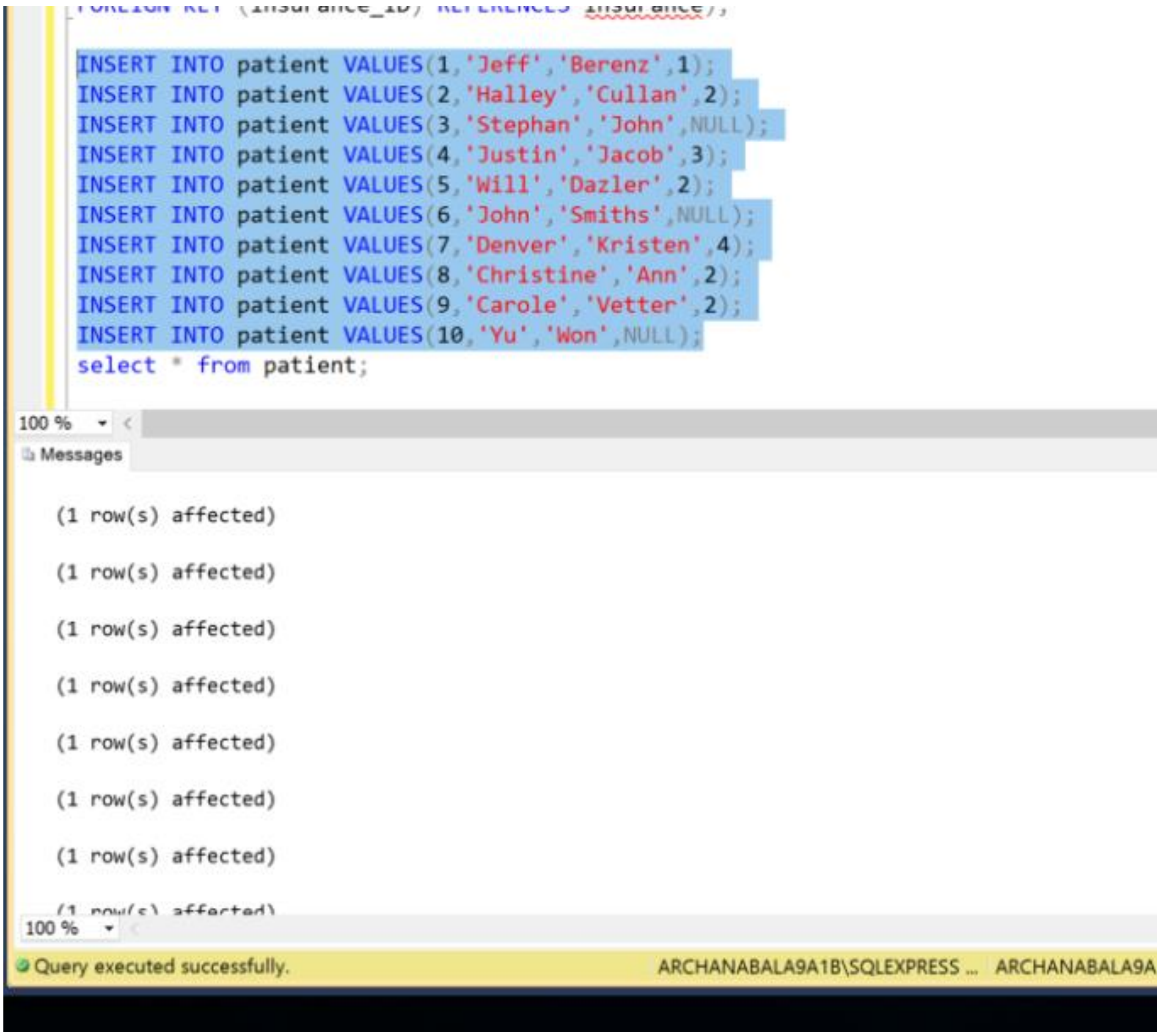
	insurance_ID	insurance_name
1	1	Plan A
2	2	Plan B
3	3	Plan C
4	4	Plan D



```
CREATE TABLE patient(
patient_ID DECIMAL(10) NOT NULL, patient_first VARCHAR(30),
patient_last VARCHAR(40),insurance_ID DECIMAL(10),
PRIMARY KEY (patient_ID),
FOREIGN KEY (insurance_ID) REFERENCES insurance);

INSERT INTO patient VALUES(1,'Jeff','Berenz',1);
INSERT INTO patient VALUES(1,'Halley','Cullan',2);
INSERT INTO patient VALUES(1,'Stephan','John',NULL);
INSERT INTO patient VALUES(1,'Justin','Jacob',3);
INSERT INTO patient VALUES(1,'Will','Dazler',2);
INSERT INTO patient VALUES(1,'John','Smiths',NULL);
INSERT INTO patient VALUES(1,'Denver','Kristen',4);
INSERT INTO patient VALUES(1,'Christine','Ann',2);
INSERT INTO patient VALUES(1,'Carole','Vetter',2);
INSERT INTO patient VALUES(1,'Yu','Won',NULL);
select * from patient;
```

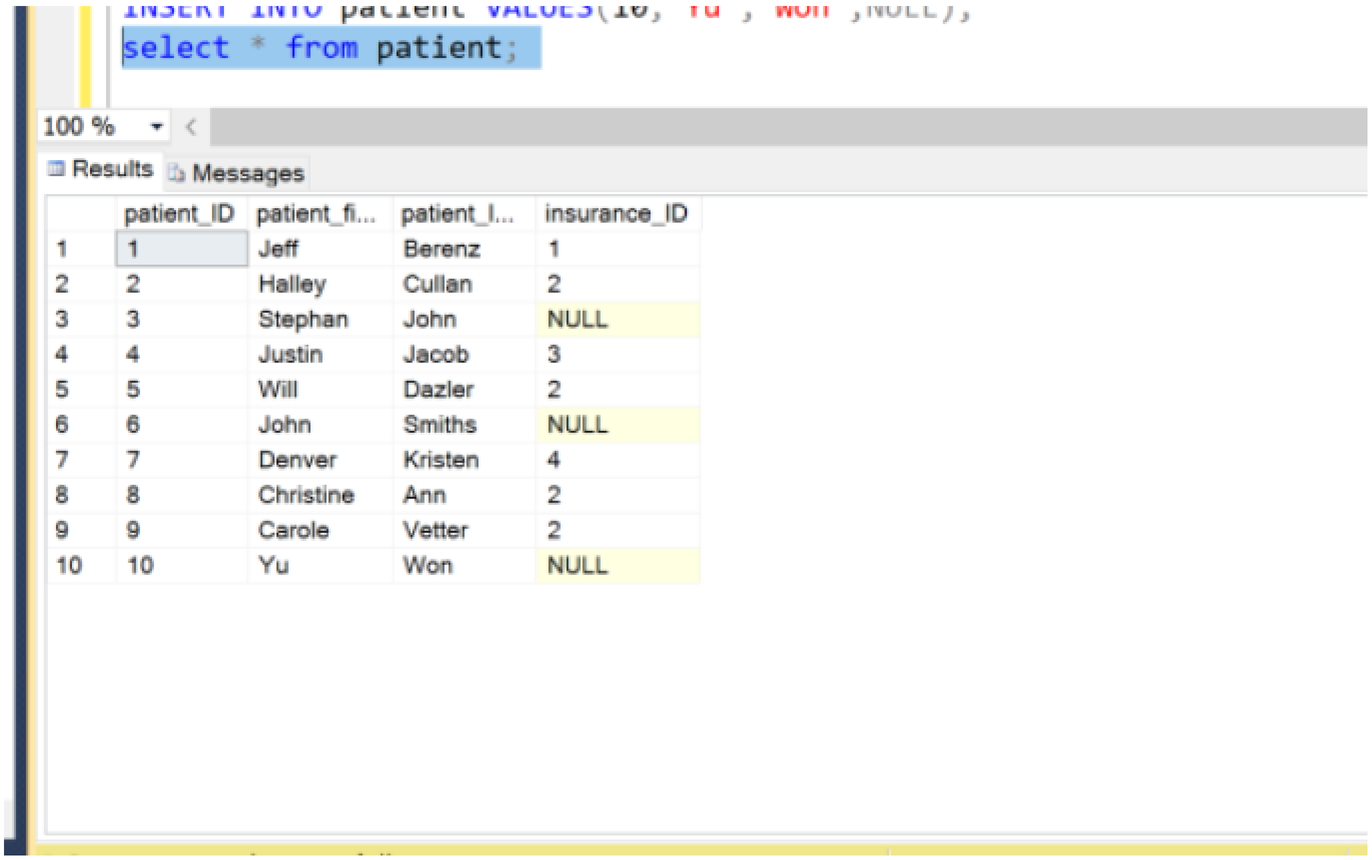
```
Command(s) completed successfully.
```

```
uery executed successfully.                ARCHANABALA9A1B\SQLEXPRESS
```



```
FOREIGN KEY (insurance_ID) REFERENCES insurance);

INSERT INTO patient VALUES(1,'Jeff','Berenz',1);
INSERT INTO patient VALUES(2,'Halley','Cullan',2);
INSERT INTO patient VALUES(3,'Stephan','John',NULL);
INSERT INTO patient VALUES(4,'Justin','Jacob',3);
INSERT INTO patient VALUES(5,'Will','Dazler',2);
INSERT INTO patient VALUES(6,'John','Smiths',NULL);
INSERT INTO patient VALUES(7,'Denver','Kristen',4);
INSERT INTO patient VALUES(8,'Christine','Ann',2);
INSERT INTO patient VALUES(9,'Carole','Vetter',2);
INSERT INTO patient VALUES(10,'Yu','Won',NULL);
select * from patient;
```

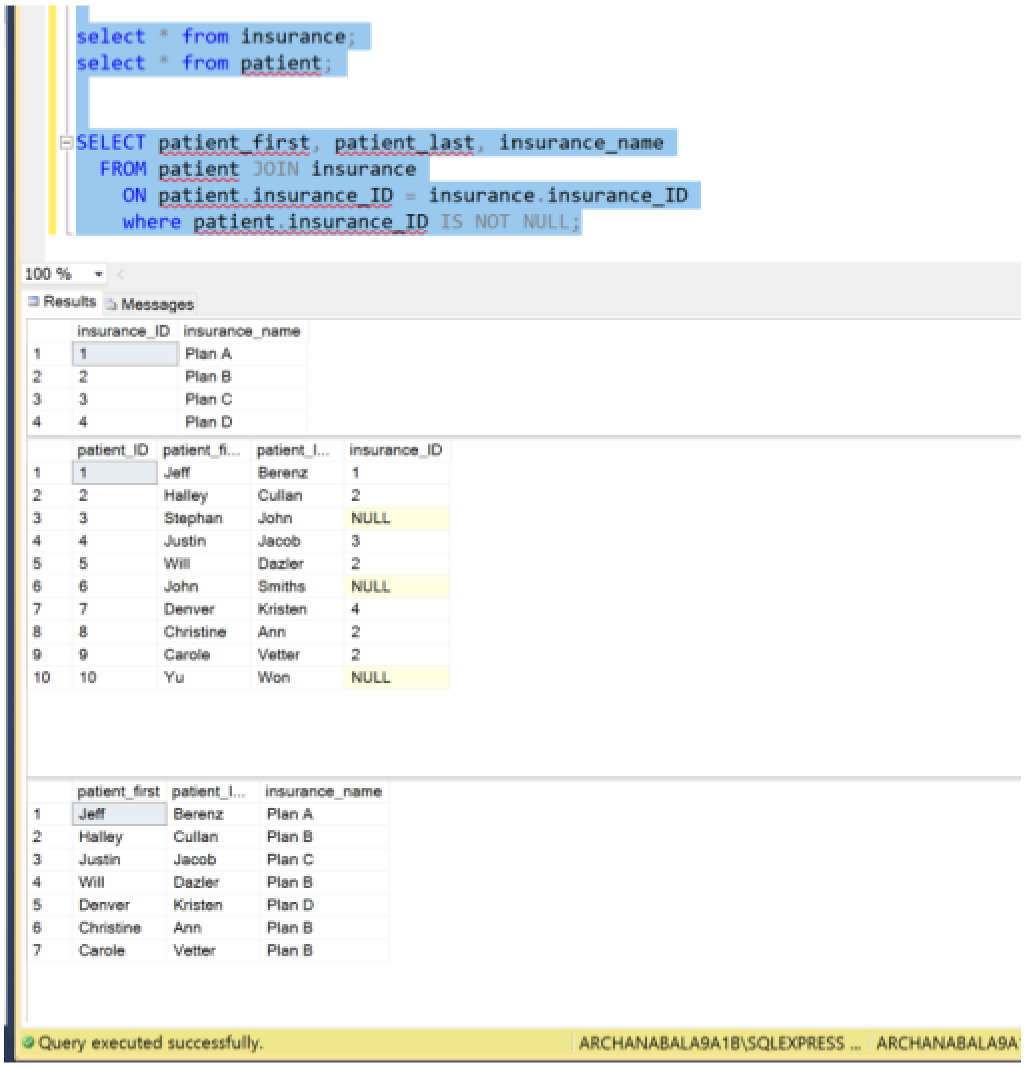
```
(1 row(s) affected)

(1 row(s) affected)

(1 row(s) affected)

(1 row(s) affected)

(1 row(s) affected)

(1 row(s) affected)

(1 row(s) affected)

(1 row(s) affected)
```



```
INSERT INTO patient VALUES(10,'Yu','Won',NULL);
select * from patient;
```

	patient_ID	patient_fi...	patient_l...	insurance_ID
1	1	Jeff	Berenz	1
2	2	Halley	Cullan	2
3	3	Stephan	John	NULL
4	4	Justin	Jacob	3
5	5	Will	Dazler	2
6	6	John	Smiths	NULL
7	7	Denver	Kristen	4
8	8	Christine	Ann	2
9	9	Carole	Vetter	2
10	10	Yu	Won	NULL

Satisfying Use case 2:



```
select * from insurance;
select * from patient;


SELECT patient_first, patient_last, insurance_name
  FROM patient JOIN insurance
    ON patient.insurance_ID = insurance.insurance_ID
    where patient.insurance_ID IS NOT NULL;
```

100 %

Results | Messages

	insurance_ID	insurance_name
1	1	Plan A
2	2	Plan B
3	3	Plan C
4	4	Plan D

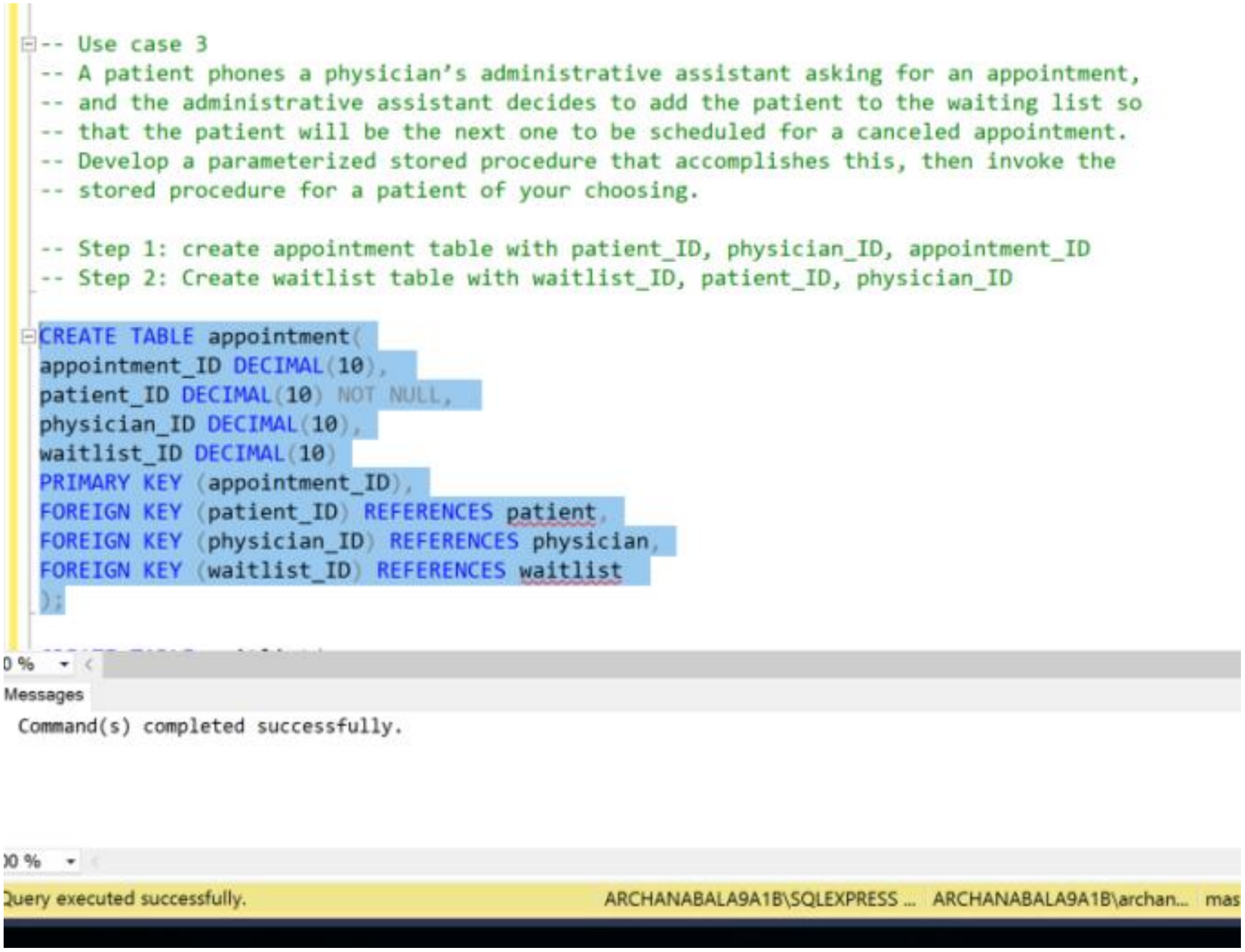
	patient_ID	patient_fi...	patient_l...	insurance_ID
1	1	Jeff	Berenz	1
2	2	Halley	Cullan	2
3	3	Stephan	John	NULL
4	4	Justin	Jacob	3
5	5	Will	Dazler	2
6	6	John	Smiths	NULL
7	7	Denver	Kristen	4
8	8	Christine	Ann	2
9	9	Carole	Vetter	2
10	10	Yu	Won	NULL

	patient_first	patient_l...	insurance_name
1	Jeff	Berenz	Plan A
2	Halley	Cullan	Plan B
3	Justin	Jacob	Plan C
4	Will	Dazler	Plan B
5	Denver	Kristen	Plan D
6	Christine	Ann	Plan B
7	Carole	Vetter	Plan B

Query executed successfully. ARCHANABALA9A1B\SQLEXPRESS ... ARCHANABALA9A

Use Case #3

Requirement: A patient phones a physician's administrative assistant asking for an appointment, and the administrative assistant decides to add the patient to the waiting list so that the patient will be the next one to be scheduled for a canceled appointment. Develop a parameterized stored procedure that accomplishes this, then invoke the stored procedure for a patient of your choosing.

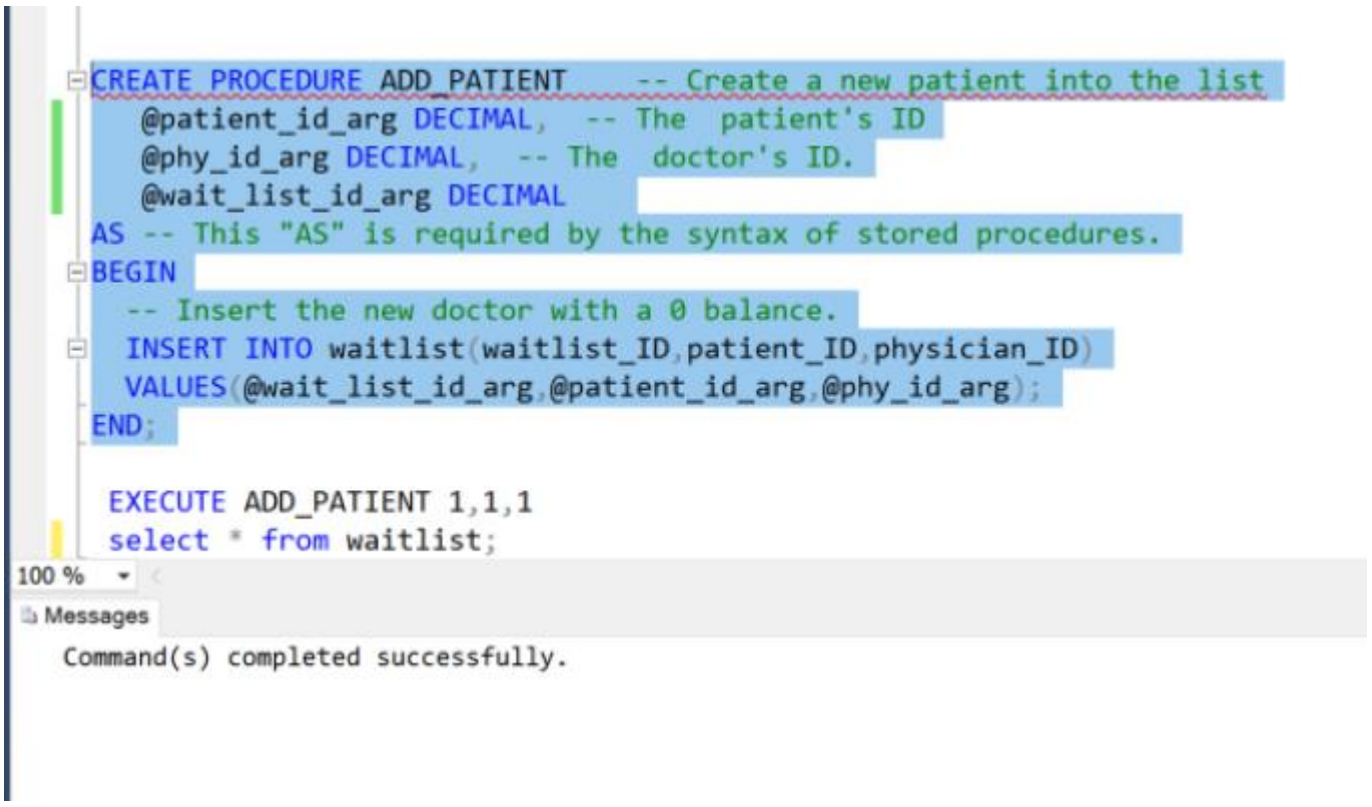


```
-- Use case 3
-- A patient phones a physician's administrative assistant asking for an appointment,
-- and the administrative assistant decides to add the patient to the waiting list so
-- that the patient will be the next one to be scheduled for a canceled appointment.
-- Develop a parameterized stored procedure that accomplishes this, then invoke the
-- stored procedure for a patient of your choosing.

-- Step 1: create appointment table with patient_ID, physician_ID, appointment_ID
-- Step 2: Create waitlist table with waitlist_ID, patient_ID, physician_ID

CREATE TABLE appointment(
appointment_ID DECIMAL(10),
patient_ID DECIMAL(10) NOT NULL,
physician_ID DECIMAL(10),
waitlist_ID DECIMAL(10)
PRIMARY KEY (appointment_ID),
FOREIGN KEY (patient_ID) REFERENCES patient,
FOREIGN KEY (physician_ID) REFERENCES physician,
FOREIGN KEY (waitlist_ID) REFERENCES waitlist
);
```

```
Messages
Command(s) completed successfully.
```

```
Query executed successfully.    ARCHANABALA9A1B\SQLEXPRESS ...   ARCHANABALA9A1B\archan...   mas
```

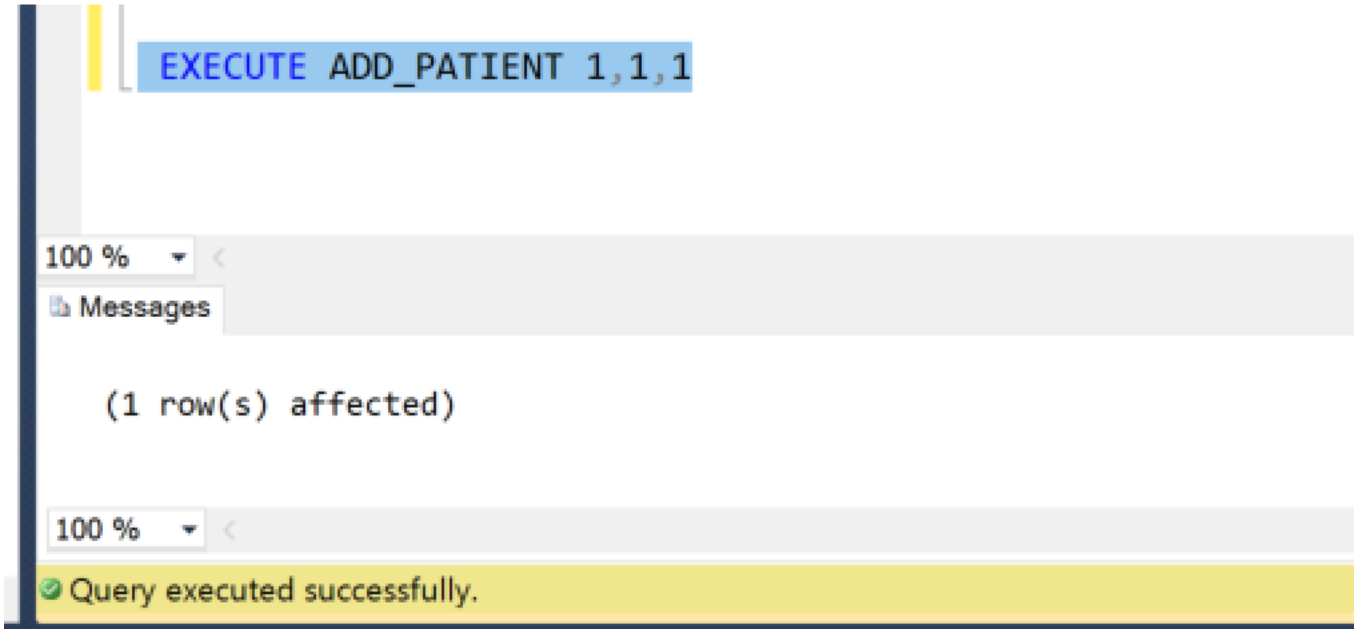
Satisfying Use case 3:


```
CREATE PROCEDURE ADD_PATIENT     -- Create a new patient into the list
    @patient_id_arg DECIMAL,   -- The  patient's ID
    @phy_id_arg DECIMAL,   -- The  doctor's ID.
    @wait_list_id_arg DECIMAL
AS -- This "AS" is required by the syntax of stored procedures.
BEGIN
  -- Insert the new doctor with a 0 balance.
  INSERT INTO waitlist(waitlist_ID,patient_ID,physician_ID)
  VALUES(@wait_list_id_arg,@patient_id_arg,@phy_id_arg);
END;

 EXECUTE ADD_PATIENT 1,1,1
 select * from waitlist;
```

100 %

Messages

```
Command(s) completed successfully.
```



```
EXECUTE ADD_PATIENT 1,1,1
```

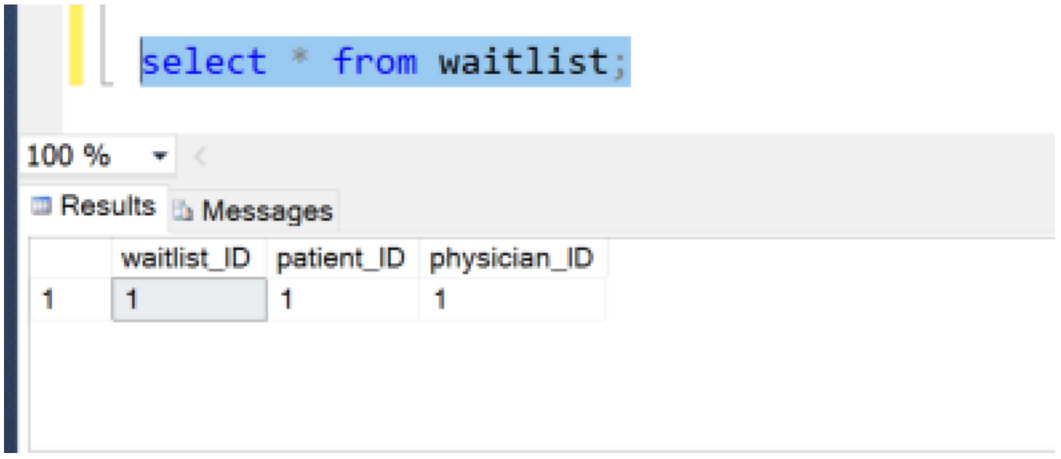
100 %

Messages

```
(1 row(s) affected)
```

100 %

Query executed successfully.



```
select * from waitlist;
```

100 %

Results | Messages

	waitlist_ID	patient_ID	physician_ID
1	1	1	1

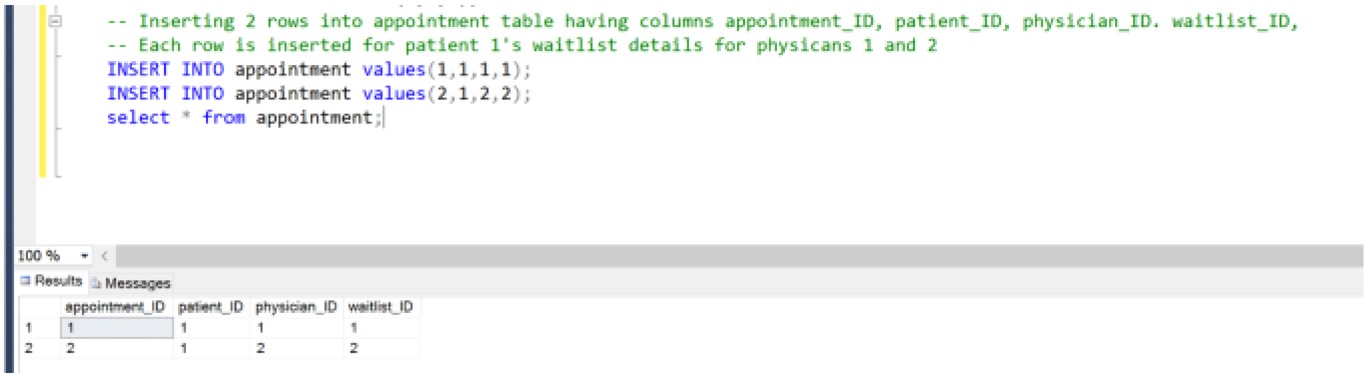
Use case #4

Requirement: A receptionist requests the first and last names of all physicians that a specific patient has *never* visited. Write a single query that retrieves this information for the receptionist.

Full Query:

```
 -- Use case #4
 -- A receptionist requests the first and last names of all physicians that a specific patient
 -- has never visited. Write a single query that retrieves this information for the receptionist.

-- INSERTING DATA INTO TABLES WAITLIST AND APPOINTMENT FOR PATIENT 1'S APPT FOR PHYSICIANS 1 AND 2

    -- Inserting 2 rows into waitlist table having columns waitlist_ID, patient_ID, physician_ID.
    -- Each row is inserted for patient 1's waitlist details for physicans 1 and 2
    INSERT INTO waitlist values(1,1,1);
    INSERT INTO waitlist values(2,1,2);
    select * from waitlist;
    -- Inserting 2 rows into appointment table having columns appointment_ID, patient_ID, physician_ID. waitlist_
    -- Each row is inserted for patient 1's waitlist details for physicans 1 and 2
    INSERT INTO appointment values(1,1,1,1);
    INSERT INTO appointment values(2,1,2,2);
    select * from appointment;

     -- Step 1 : APPOINTMENT table has confirmed physician visit info. Refer APPOINTMENT table
     --          Get physician_IDs for the particular patient into a list. Check PHYSICIAN table for
     --          physican IDs NOT in the array. Refer the PHYSICIAN table and display their first and last names

  select physician_first,physician_last from physician
  where physician_ID NOT IN (select physician_ID from appointment where patient_ID=1);
```

Query Execution in parts:

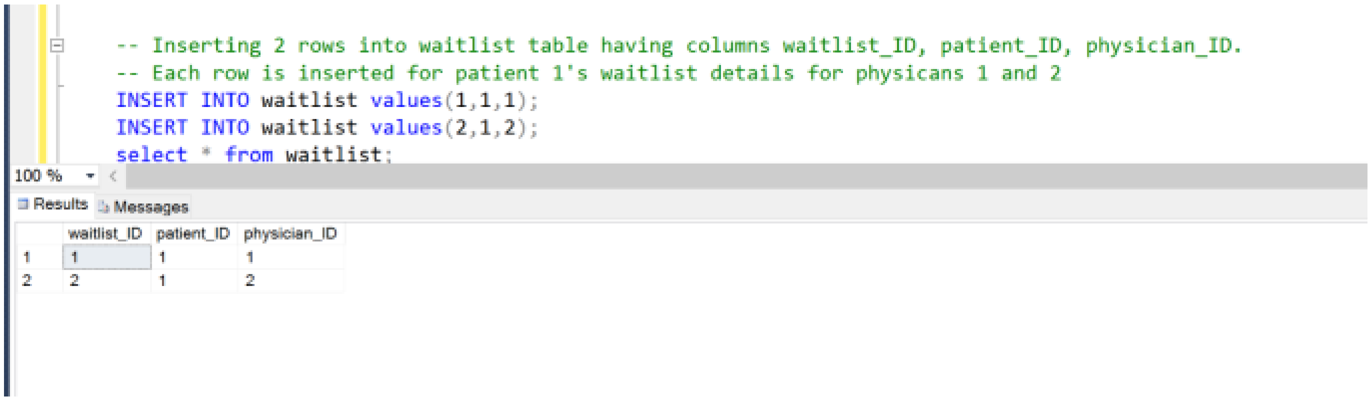


```
-- Inserting 2 rows into appointment table having columns appointment_ID, patient_ID, physician_ID. waitlist_ID,
-- Each row is inserted for patient 1's waitlist details for physicans 1 and 2
INSERT INTO appointment values(1,1,1,1);
INSERT INTO appointment values(2,1,2,2);
select * from appointment;
```

100 %

Results | Messages

	appointment_ID	patient_ID	physician_ID	waitlist_ID
1	1	1	1	1
2	2	1	2	2

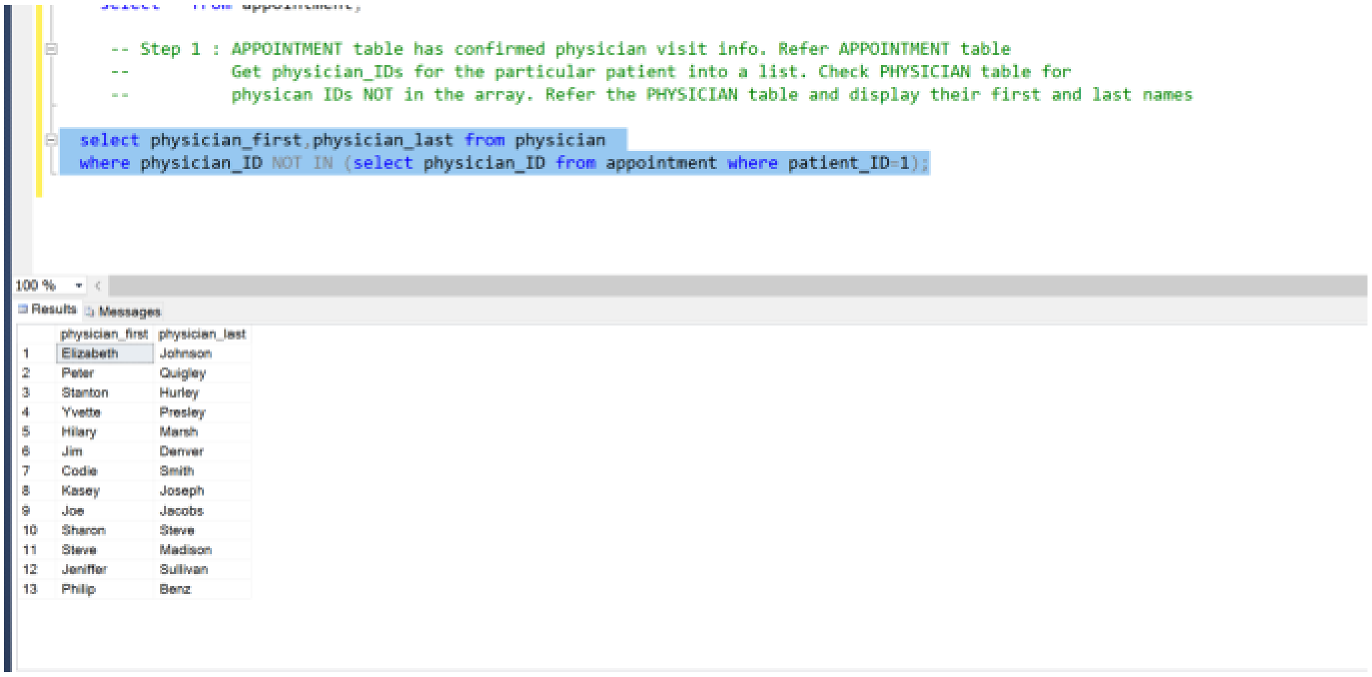


```sql
-- Inserting 2 rows into waitlist table having columns waitlist_ID, patient_ID, physician_ID.
-- Each row is inserted for patient 1's waitlist details for physicans 1 and 2
INSERT INTO waitlist values(1,1,1);
INSERT INTO waitlist values(2,1,2);
select * from waitlist;
```

100 %

Results | Messages

```
   waitlist_ID  patient_ID  physician_ID
1  1            1           1
2  2            1           2
```



```sql
-- Step 1 : APPOINTMENT table has confirmed physician visit info. Refer APPOINTMENT table
--          Get physician_IDs for the particular patient into a list. Check PHYSICIAN table for
--          physican IDs NOT in the array. Refer the PHYSICIAN table and display their first and last names

select physician_first,physician_last from physician
where physician_ID NOT IN (select physician_ID from appointment where patient_ID=1);
```

100 %

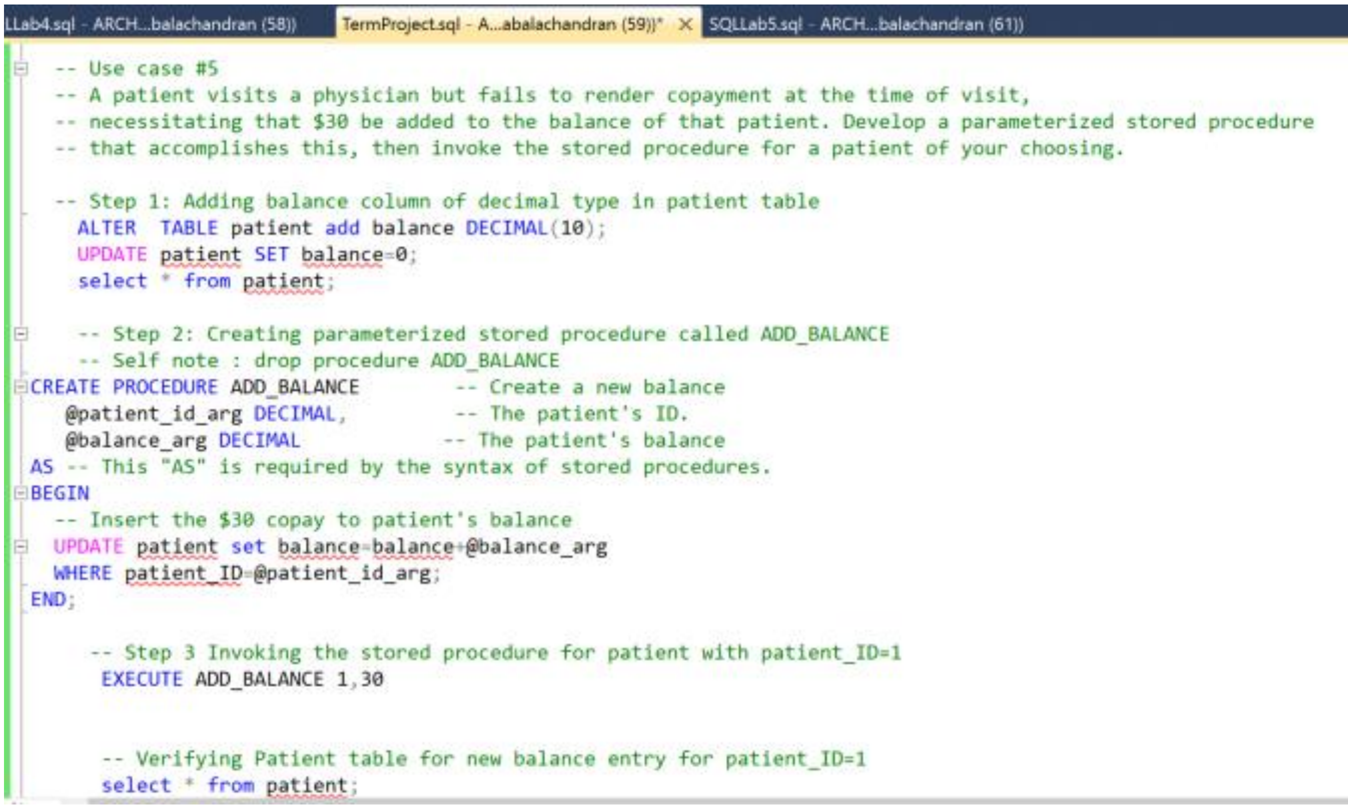
Results | Messages

```
    physician_first  physician_last
1   Elizabeth        Johnson
2   Peter            Quigley
3   Stanton          Hurley
4   Yvette           Presley
5   Hilary           Marsh
6   Jim              Denver
7   Codie            Smith
8   Kasey            Joseph
9   Joe              Jacobs
10  Sharon           Steve
11  Steve            Madison
12  Jeniffer         Sullivan
13  Philip           Benz
```

Use case #5:

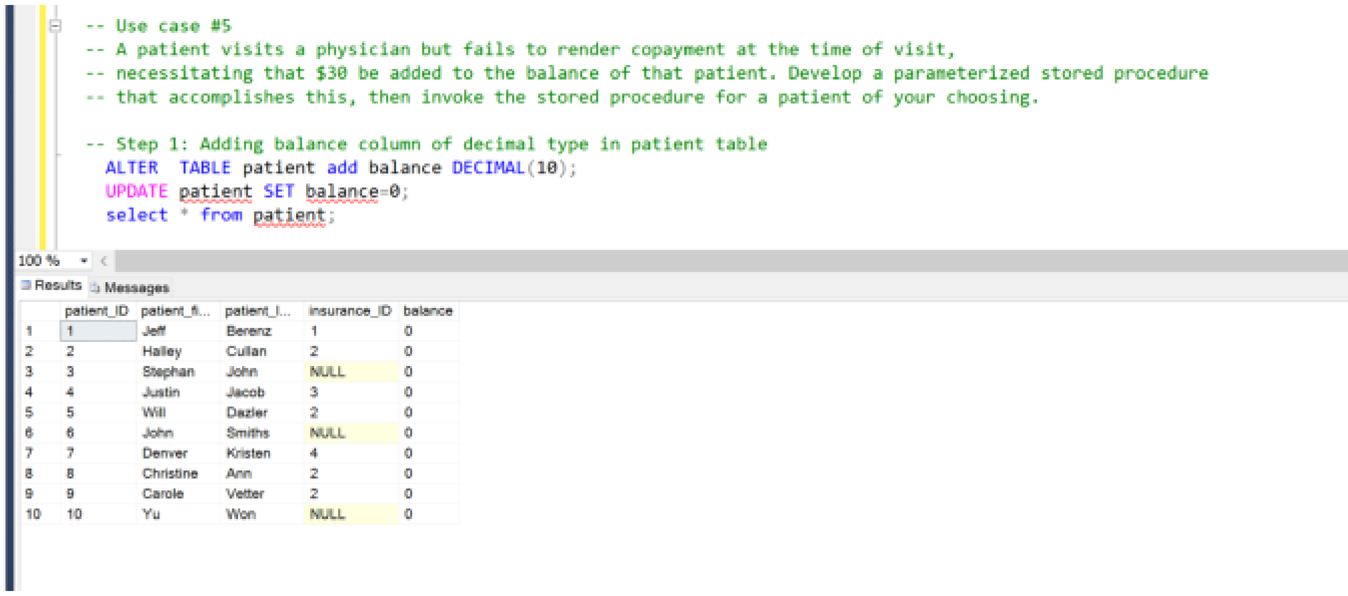
Requirement: A patient visits a physician but fails to render copayment at the time of visit, necessitating that $30 be added to the balance of that patient. Develop a parameterized stored procedure that accomplishes this, then invoke the stored procedure for a patient of your choosing.

Full Query:


LLab4.sql - ARCH...balachandran (58))
TermProject.sql - A...abalachandran (59))*
SQLLab5.sql - ARCH...balachandran (61))

```
-- Use case #5
-- A patient visits a physician but fails to render copayment at the time of visit,
-- necessitating that $30 be added to the balance of that patient. Develop a parameterized stored procedure
-- that accomplishes this, then invoke the stored procedure for a patient of your choosing.

-- Step 1: Adding balance column of decimal type in patient table
  ALTER  TABLE patient add balance DECIMAL(10);
  UPDATE patient SET balance=0;
  select * from patient;

  -- Step 2: Creating parameterized stored procedure called ADD_BALANCE
  -- Self note : drop procedure ADD_BALANCE
CREATE PROCEDURE ADD_BALANCE        -- Create a new balance
   @patient_id_arg DECIMAL,         -- The patient's ID.
   @balance_arg DECIMAL            -- The patient's balance
AS -- This "AS" is required by the syntax of stored procedures.
BEGIN
  -- Insert the $30 copay to patient's balance
  UPDATE patient set balance=balance+@balance_arg
  WHERE patient_ID=@patient_id_arg;
END;

     -- Step 3 Invoking the stored procedure for patient with patient_ID=1
      EXECUTE ADD_BALANCE 1,30


      -- Verifying Patient table for new balance entry for patient_ID=1
      select * from patient;
```

Query Execution in parts:

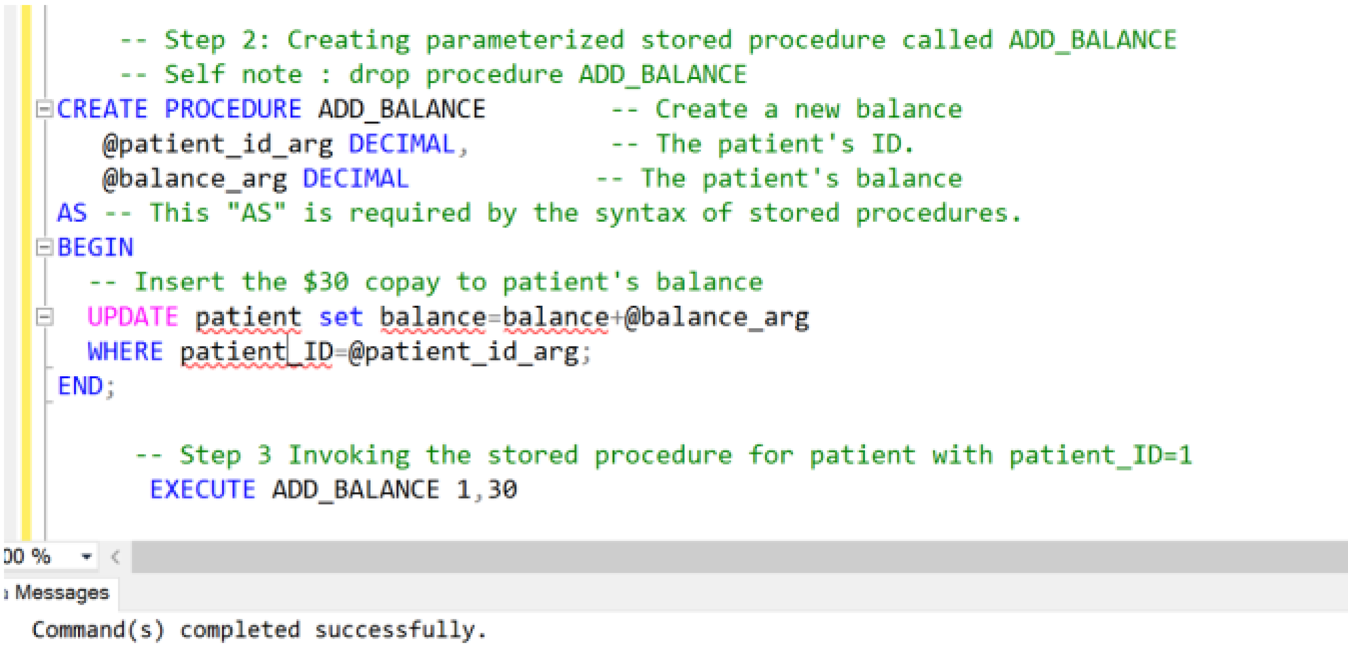


```
-- Use case #5
-- A patient visits a physician but fails to render copayment at the time of visit,
-- necessitating that $30 be added to the balance of that patient. Develop a parameterized stored procedure
-- that accomplishes this, then invoke the stored procedure for a patient of your choosing.

-- Step 1: Adding balance column of decimal type in patient table
  ALTER  TABLE patient add balance DECIMAL(10);
  UPDATE patient SET balance=0;
  select * from patient;
```

100 %

Results | Messages

	patient_ID	patient_fi...	patient_l...	insurance_ID	balance
1	1	Jeff	Berenz	1	0
2	2	Halley	Cullan	2	0
3	3	Stephan	John	NULL	0
4	4	Justin	Jacob	3	0
5	5	Will	Dazler	2	0
6	6	John	Smiths	NULL	0
7	7	Denver	Kristen	4	0
8	8	Christine	Ann	2	0
9	9	Carole	Vetter	2	0
10	10	Yu	Won	NULL	0

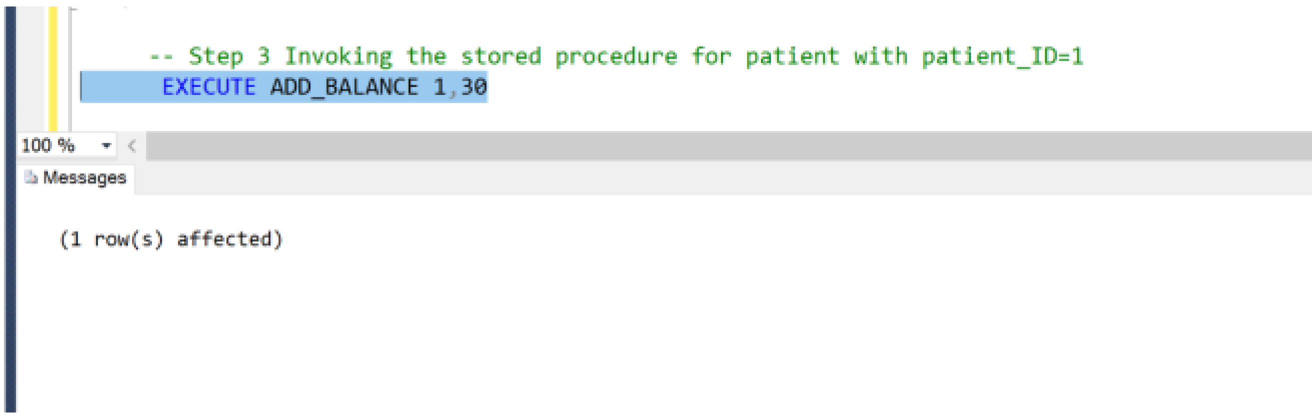


```
    -- Step 2: Creating parameterized stored procedure called ADD_BALANCE
    -- Self note : drop procedure ADD_BALANCE
CREATE PROCEDURE ADD_BALANCE        -- Create a new balance
   @patient_id_arg DECIMAL,         -- The patient's ID.
   @balance_arg DECIMAL            -- The patient's balance
AS -- This "AS" is required by the syntax of stored procedures.
BEGIN
  -- Insert the $30 copay to patient's balance
  UPDATE patient set balance=balance+@balance_arg
  WHERE patient_ID=@patient_id_arg;
END;

     -- Step 3 Invoking the stored procedure for patient with patient_ID=1
      EXECUTE ADD_BALANCE 1,30
```

00 %

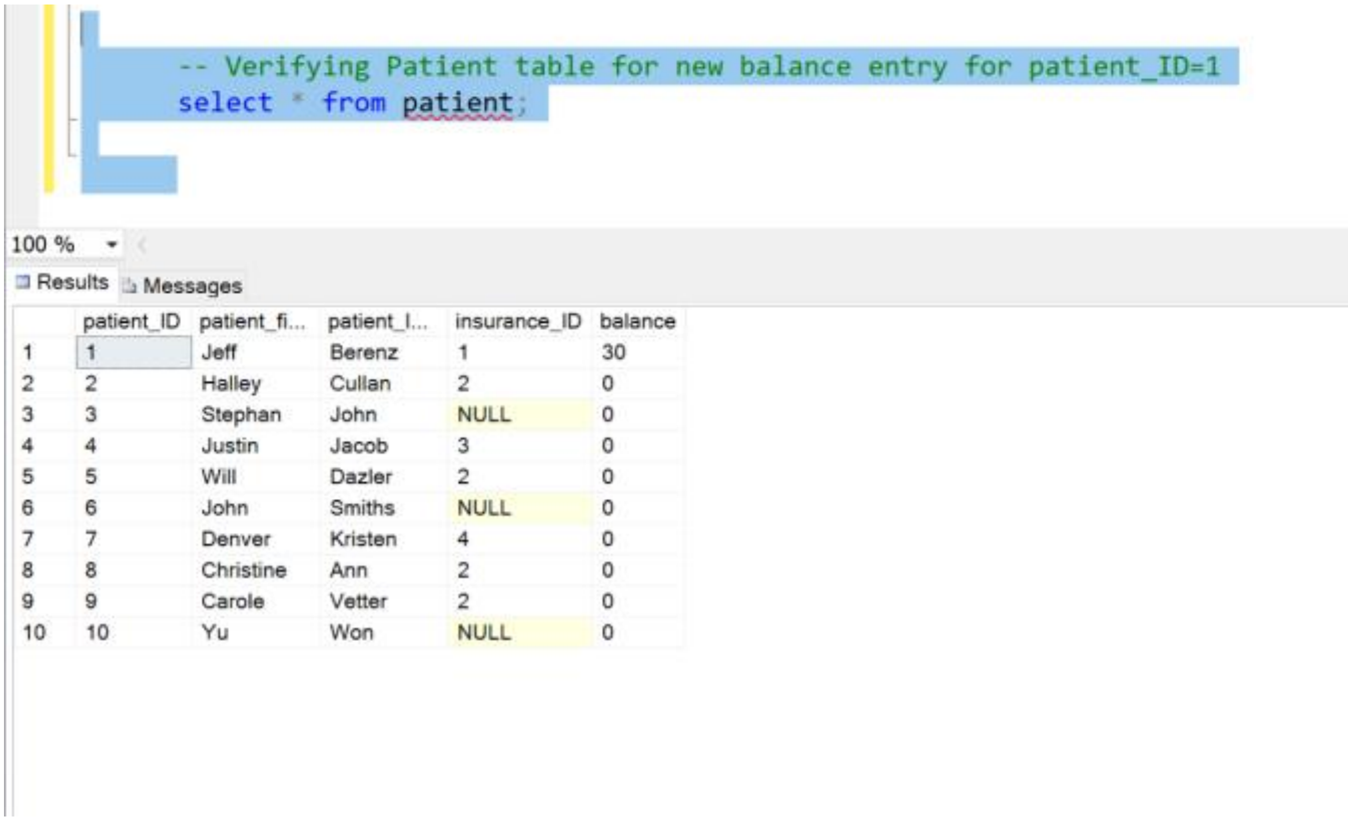
Messages

```
Command(s) completed successfully.
```



```
-- Step 3 Invoking the stored procedure for patient with patient_ID=1
EXECUTE ADD_BALANCE 1,30
```

```
(1 row(s) affected)
```



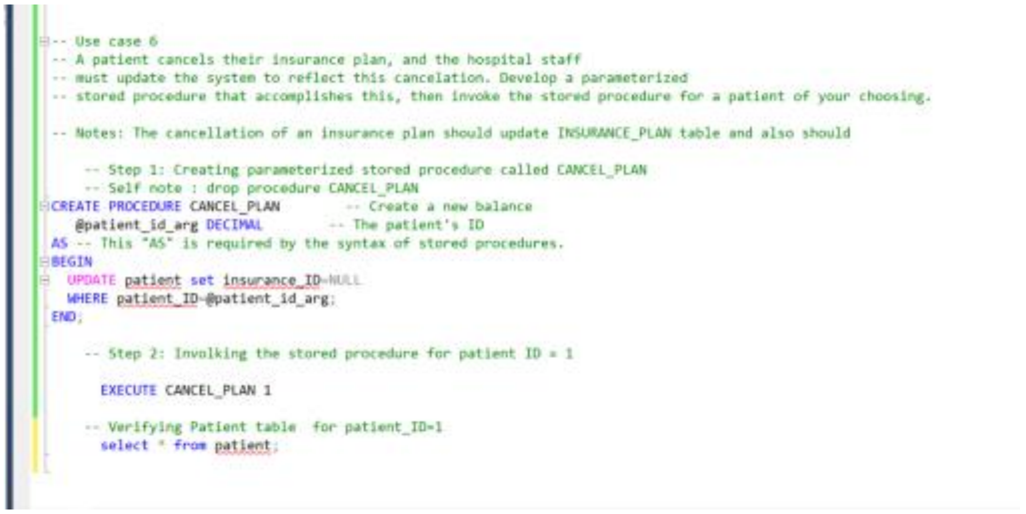
```
-- Verifying Patient table for new balance entry for patient_ID=1
select * from patient;
```

	patient_ID	patient_fi...	patient_l...	insurance_ID	balance
1	1	Jeff	Berenz	1	30
2	2	Halley	Cullan	2	0
3	3	Stephan	John	NULL	0
4	4	Justin	Jacob	3	0
5	5	Will	Dazler	2	0
6	6	John	Smiths	NULL	0
7	7	Denver	Kristen	4	0
8	8	Christine	Ann	2	0
9	9	Carole	Vetter	2	0
10	10	Yu	Won	NULL	0

Use case #6:

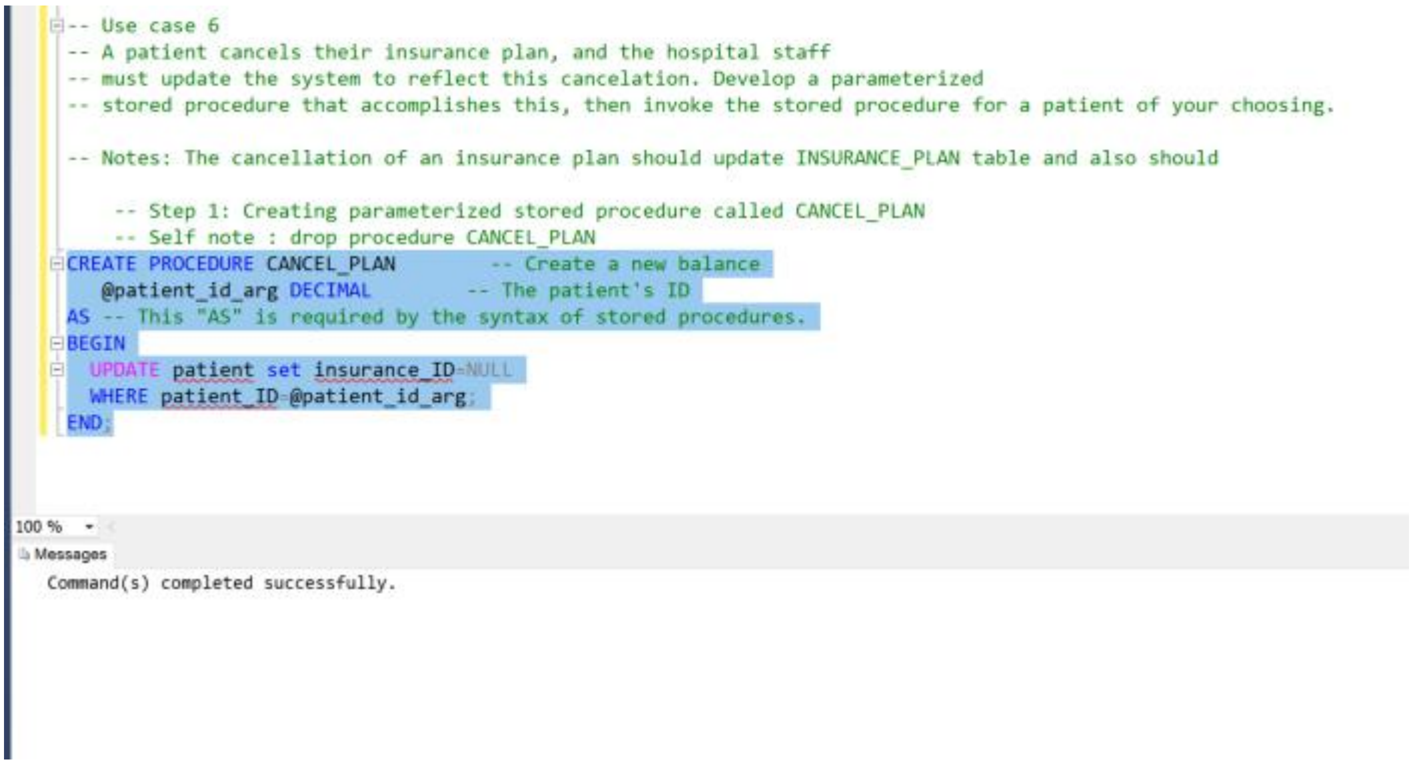
Requirement: A patient cancels their insurance plan, and the hospital staff must update the system to reflect this cancelation. Develop a parameterized stored procedure that accomplishes this, then invoke the stored procedure for a patient of your choosing.

Full Query:



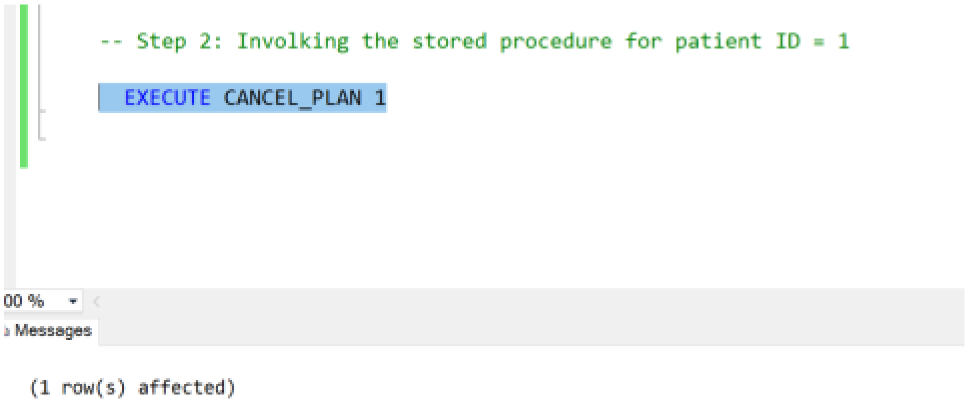
```
-- Use case 6
-- A patient cancels their insurance plan, and the hospital staff
-- must update the system to reflect this cancelation. Develop a parameterized
-- stored procedure that accomplishes this, then invoke the stored procedure for a patient of your choosing.

-- Notes: The cancellation of an insurance plan should update INSURANCE_PLAN table and also should

    -- Step 1: Creating parameterized stored procedure called CANCEL_PLAN
    -- Self note : drop procedure CANCEL_PLAN
CREATE PROCEDURE CANCEL_PLAN          -- Create a new balance
   @patient_id_arg DECIMAL          -- The patient's ID
AS -- This "AS" is required by the syntax of stored procedures.
BEGIN
  UPDATE patient set insurance_ID=NULL
  WHERE patient_ID=@patient_id_arg;
END;

    -- Step 2: Involking the stored procedure for patient ID = 1

      EXECUTE CANCEL_PLAN 1

    -- Verifying Patient table  for patient_ID=1
      select * from patient;
```

Query Execution in parts:



```
-- Use case 6
-- A patient cancels their insurance plan, and the hospital staff
-- must update the system to reflect this cancelation. Develop a parameterized
-- stored procedure that accomplishes this, then invoke the stored procedure for a patient of your choosing.

-- Notes: The cancellation of an insurance plan should update INSURANCE_PLAN table and also should

    -- Step 1: Creating parameterized stored procedure called CANCEL_PLAN
    -- Self note : drop procedure CANCEL_PLAN
CREATE PROCEDURE CANCEL_PLAN          -- Create a new balance
   @patient_id_arg DECIMAL          -- The patient's ID
AS -- This "AS" is required by the syntax of stored procedures.
BEGIN
  UPDATE patient set insurance_ID=NULL
  WHERE patient_ID=@patient_id_arg;
END;
```

```
Command(s) completed successfully.
```

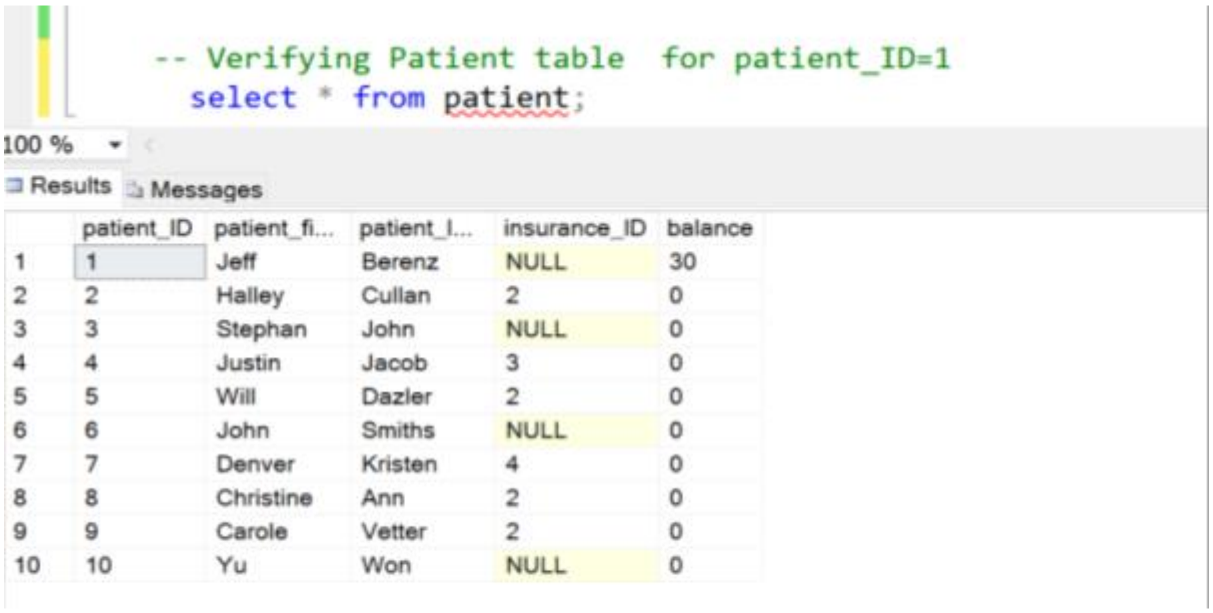


```
-- Step 2: Involking the stored procedure for patient ID = 1

EXECUTE CANCEL_PLAN 1
```

00 %

Messages

```
(1 row(s) affected)
```



```
-- Verifying Patient table  for patient_ID=1
  select * from patient;
```

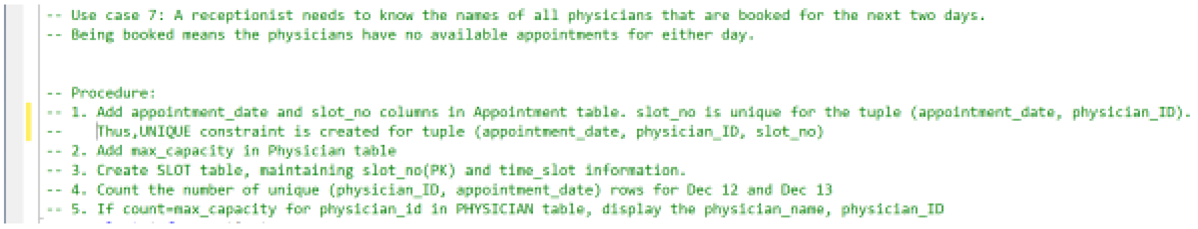
100 %

Results Messages

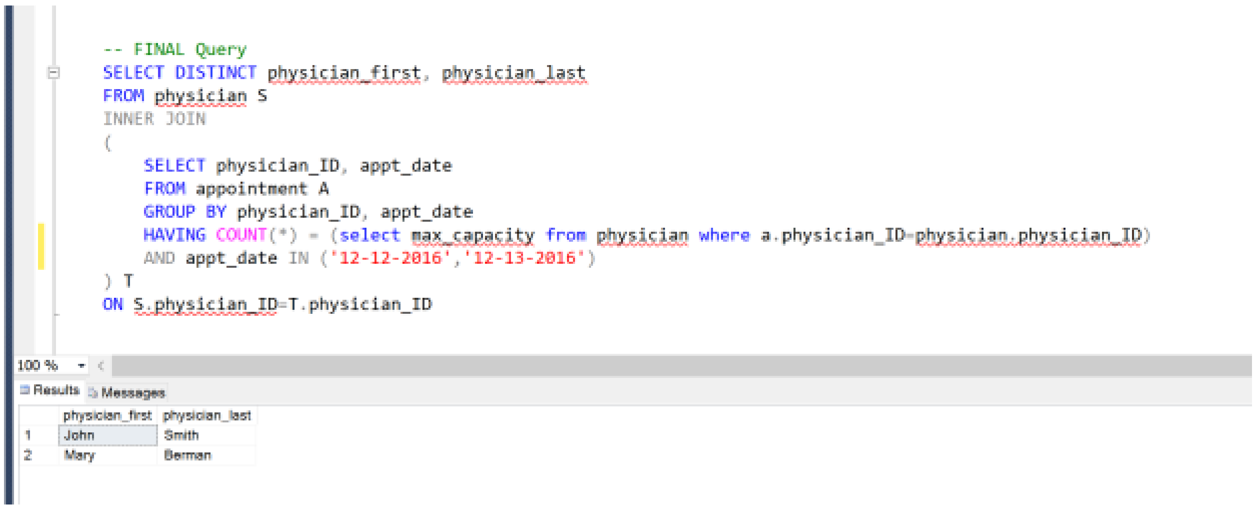
	patient_ID	patient_fi...	patient_l...	insurance_ID	balance
1	1	Jeff	Berenz	NULL	30
2	2	Halley	Cullan	2	0
3	3	Stephan	John	NULL	0
4	4	Justin	Jacob	3	0
5	5	Will	Dazler	2	0
6	6	John	Smiths	NULL	0
7	7	Denver	Kristen	4	0
8	8	Christine	Ann	2	0
9	9	Carole	Vetter	2	0
10	10	Yu	Won	NULL	0

Use case #7:

Requirement: A receptionist needs to know the names of all physicians that are booked for the next two days. Being booked means the physicians have no available appointments for either day. Write a single query that retrieves this information for the receptionist.



```
-- Use case 7: A receptionist needs to know the names of all physicians that are booked for the next two days.
-- Being booked means the physicians have no available appointments for either day.


-- Procedure:
-- 1. Add appointment_date and slot_no columns in Appointment table. slot_no is unique for the tuple (appointment_date, physician_ID).
--    Thus,UNIQUE constraint is created for tuple (appointment_date, physician_ID, slot_no)
-- 2. Add max_capacity in Physician table
-- 3. Create SLOT table, maintaining slot_no(PK) and time_slot information.
-- 4. Count the number of unique (physician_ID, appointment_date) rows for Dec 12 and Dec 13
-- 5. If count=max_capacity for physician_id in PHYSICIAN table, display the physician_name, physician_ID
```



```
-- FINAL Query
SELECT DISTINCT physician_first, physician_last
FROM physician S
INNER JOIN
(
    SELECT physician_ID, appt_date
    FROM appointment A
    GROUP BY physician_ID, appt_date
    HAVING COUNT(*) = (select max_capacity from physician where a.physician_ID=physician.physician_ID)
    AND appt_date IN ('12-12-2016','12-13-2016')
) T
ON S.physician_ID=T.physician_ID
```

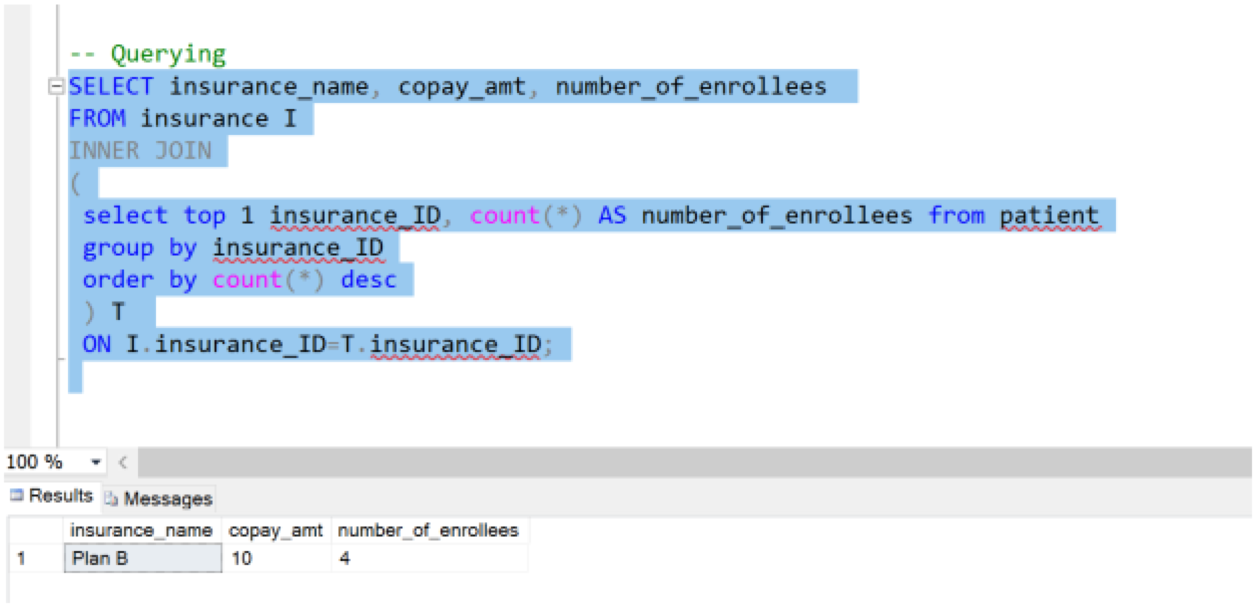
100 %

Results | Messages

	physician_first	physician_last
1	John	Smith
2	Mary	Berman

Use case #8:

Requirement: Health center management requests the insurance plan with the most patient enrollees, and for that plan, its name, required copayment amount, and the number of patient enrollees. Write a single query that retrieves this information for the management.



```
-- Querying
SELECT insurance_name, copay_amt, number_of_enrollees
FROM insurance I
INNER JOIN
(
 select top 1 insurance_ID, count(*) AS number_of_enrollees from patient
 group by insurance_ID
 order by count(*) desc
 ) T
 ON I.insurance_ID=T.insurance_ID;
```

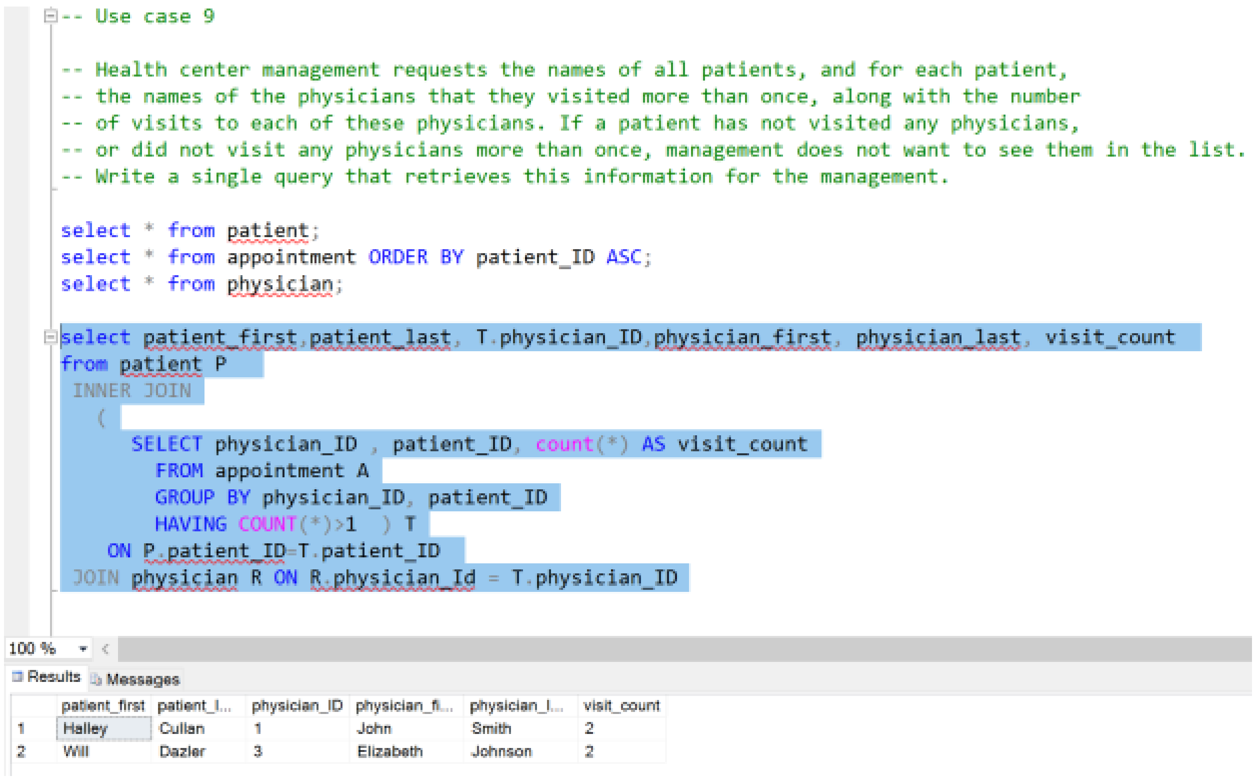
100 %

Results | Messages

```
   insurance_name  copay_amt  number_of_enrollees
1  Plan B          10         4
```

Use case #9:

Requirement: Health center management requests the names of all patients, and for each patient, the names of the physicians that they visited more than once, along with the number of visits to each of these physicians. If a patient has not visited any physicians, or did not visit any physicians more than once, management does not want to see them in the list. Write a single query that retrieves this information for the management.



```
-- Use case 9

-- Health center management requests the names of all patients, and for each patient,
-- the names of the physicians that they visited more than once, along with the number
-- of visits to each of these physicians. If a patient has not visited any physicians,
-- or did not visit any physicians more than once, management does not want to see them in the list.
-- Write a single query that retrieves this information for the management.

select * from patient;
select * from appointment ORDER BY patient_ID ASC;
select * from physician;

select patient_first,patient_last, T.physician_ID,physician_first, physician_last, visit_count
from patient P
 INNER JOIN
   (
      SELECT physician_ID , patient_ID, count(*) AS visit_count
        FROM appointment A
        GROUP BY physician_ID, patient_ID
        HAVING COUNT(*)>1  ) T
    ON P.patient_ID=T.patient_ID
 JOIN physician R ON R.physician_Id = T.physician_ID
```

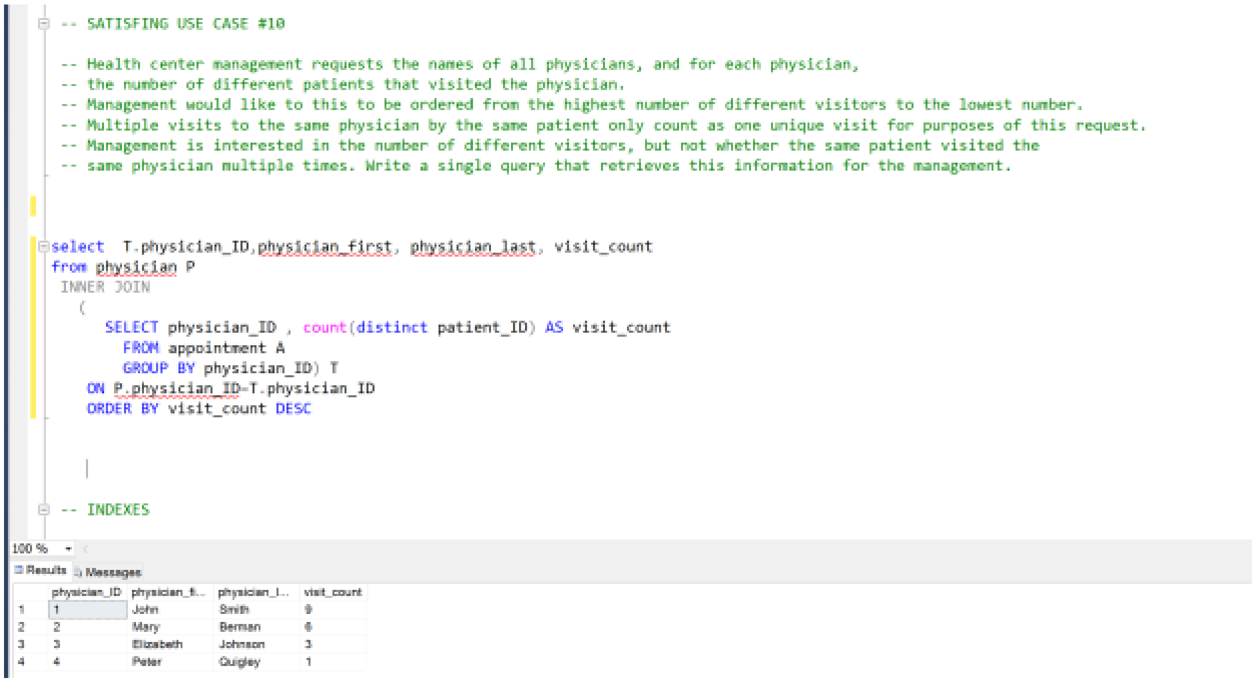
100 %

Results Messages

	patient_first	patient_l...	physician_ID	physician_fi...	physician_l...	visit_count
1	Halley	Cullan	1	John	Smith	2
2	Will	Dazler	3	Elizabeth	Johnson	2

Use case #10:

Requirement: Health center management requests the names of all physicians, and for each physician, the number of different patients that visited the physician. Management would like to this to be ordered from the highest number of different visitors to the lowest number. Multiple visits to the same physician by the same patient only count as one unique visit for purposes of this request. Management is interested in the number of different visitors, but not whether the same patient visited the same physician multiple times. Write a single query that retrieves this information for the management.



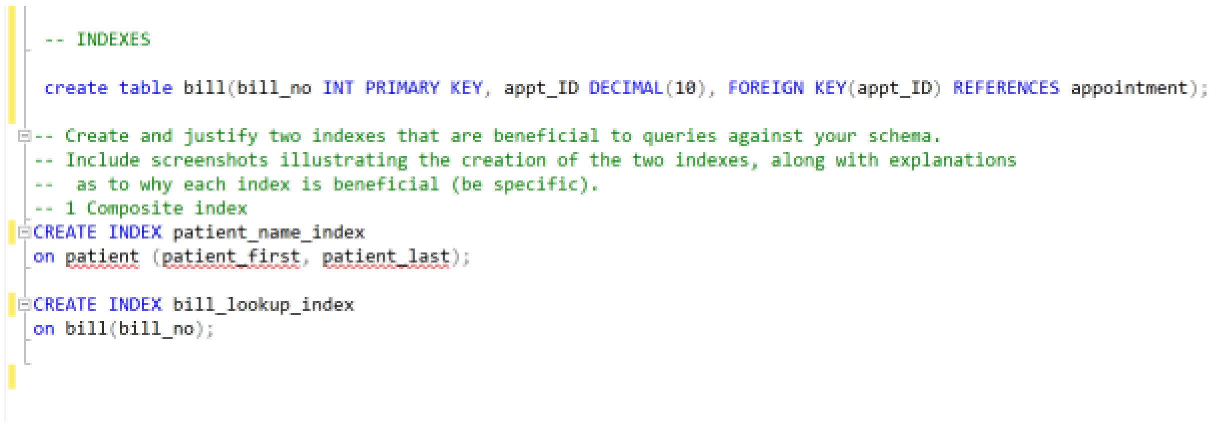
```
-- SATISFING USE CASE #10

-- Health center management requests the names of all physicians, and for each physician,
-- the number of different patients that visited the physician.
-- Management would like to this to be ordered from the highest number of different visitors to the lowest number.
-- Multiple visits to the same physician by the same patient only count as one unique visit for purposes of this request.
-- Management is interested in the number of different visitors, but not whether the same patient visited the
-- same physician multiple times. Write a single query that retrieves this information for the management.


select  T.physician_ID,physician_first, physician_last, visit_count
from physician P
 INNER JOIN
   (
      SELECT physician_ID , count(distinct patient_ID) AS visit_count
        FROM appointment A
        GROUP BY physician_ID) T
    ON P.physician_ID=T.physician_ID
    ORDER BY visit_count DESC



 -- INDEXES
```

100 %

Results Messages

	physician_ID	physician_fi...	physician_l...	visit_count
1	1	John	Smith	9
2	2	Mary	Berman	6
3	3	Elizabeth	Johnson	3
4	4	Peter	Quigley	1

INDEXES



```
-- INDEXES

create table bill(bill_no INT PRIMARY KEY, appt_ID DECIMAL(10), FOREIGN KEY(appt_ID) REFERENCES appointment);

-- Create and justify two indexes that are beneficial to queries against your schema.
-- Include screenshots illustrating the creation of the two indexes, along with explanations
--  as to why each index is beneficial (be specific).
-- 1 Composite index
CREATE INDEX patient_name_index
on patient (patient_first, patient_last);

CREATE INDEX bill_lookup_index
on bill(bill_no);
```

Indexes are created in such a way that query performance is improved, therefore, they are used on large tables, and are avoided on tables with frequent update or insert operations (such as appointment table and waitlist table). Columns that are frequently altered/manipulated are also avoided for enhanced query performance.

The index created in Patient table is a composite index comprising of a patient's first name and last name because typically, if a patient needs to be looked up in the system, their full name is entered to retrieve the best results because this composite index is most likely a unique value, and can boost performance when joins or searches are executed. It is safe to use the index on this table because the patient's personal details such as name will not be changed, and the patient table is considerably large.

Another index is created on the bill table for the column bill_no because this column will not be changed at all since the bill_no values are unique to each appointment table entry. There will not be any update query, and will only have insert operations. This table is going to be a very large table as well, so the indexing will be very helpful.