

## Project Direction Overview

I am interested in designing a centralized database and corresponding (secure) online client for electronic medical records in Canada. Accessibility and uniformity of patient records has been a longstanding glaring issue in Canadian healthcare, despite the nation's universal system. As a relative of many healthcare professionals, I regularly hear complaints about the bottleneck this creates in the provision of service to patients, and it consistently surprises me that such an issue exists in such perpetuity.

In the current state of affairs, there is little to no standardization of record-taking when a patient interfaces with the healthcare system. This makes it quite difficult for a patient to seamlessly visit a new doctor or have a prescription filled at a different pharmacy, as the new healthcare provider needs to request, have transferred, and reinterpret all the patient's records prior to administering care. The database that I will be designing will contain information regarding the patient's personal information (name, date of birth, province of residence, health card number, etc.), their medical history detailing the specialties of the physicians who have examined them and their corresponding clinical notes, and any active prescriptions they may have. There may be some potential for standardization of communication regarding commonly occurring conditions in each specialty, though this will take further research to confirm.

I intend for this database to be primarily of use to practicing healthcare professionals (physicians, nurses, pharmacists, etc.) to make the provision of care more seamless. With its implementation, patients will be able to travel to different areas of the country and access care as needed without the tedious and time-consuming process of transferring records. It will also cultivate a network for physicians through which they can share information consult other physicians and easily get second opinions for more effective care. Patients will ideally also be able to go to any other pharmacy in the country and have their prescriptions refilled without the same record transfer process that is currently required.

## Use Cases and Fields (revised to include Patient addition use case)

I can currently envision four use cases for this database, the first of which is the *addition of new Patients to the database*.

1. **A patient visits a physician (of whichever specialty and for whatever reason).**
2. **The physician conducts the appointment and records Notes as needed.**
3. **Upon completion of the appointment, the Physician enters the Patient's recorded Notes and Family History into the database and commits the changes under a new Patient record.**
4. **The new Patient now has a permanent record in the database primed for future access.**

The second use case is the *quick and easy accessing of records by different physicians*.

1. The patient visits a different physician (of whichever specialty and for whatever reason) than who they would normally see (due to travel, etc.)
2. The physician accesses the database through the secure online client and is instantly apprised of the patient's medical history.
3. The physician then provides whatever care is necessary in accordance with the patient's complaints and history.

4. The physician enters the updated information into the client which is immediately synced with the database and is readily accessible to any other healthcare professional that the patient may see.

The third use case is the *networking between physicians to make care more efficient and thorough*.

1. The attending physician reviews the patient's file for, including but not limited to, diagnostic purposes.
2. If the physician deems there is a need for a second opinion, they can request the assistance of another physician (abiding of course by confidentiality logistics).
3. The second physician can then access the patient's records through the client, quickly be apprised of the situation, and formulate a second opinion on how their care should proceed.
4. If a consensus is reached between the physicians, a corresponding note can be made in the client and pushed to the database.

And the fourth and (so far) final use case is the *quick accessibility of prescription refills at different pharmacies*.

1. The patient requests a refill at a different pharmacy than they would normally visit.
2. The attending pharmacist then accesses the client and sees the patient's active prescriptions.
3. The pharmacist can then easily fill the patient's prescription and update the database with how many refills are remaining.

The significant fields that will be stored in the database for the Patient entity are listed below alongside their descriptions and relevance.

Field	What it Stores	Why it's Needed
Name	The patient's name	Aids in identifying the patient and elevating bedside manner
DOB	The patient's date of birth	Provides more specific insight into the level of care needed, as age plays a major factor in healthcare
Gender	The patient's sex	Provides more specific insight into the care to be provided, as there are many gender-specific conditions requiring differing care
ResidentProvince	The province of which the patient is a resident	Can provide further insight into the level of care that has been provided and aid in advising further care in accordance with what is accessible in each province, as well as identifying the insurer to be billed
HealthCardNumber	The patient's health card number	As each patient's number is unique, this can be used as the

		primary identifier when accessing the patient's records through the client
Prescriptions	Any prescriptions the patient has had (with differentiation between active and former)	Provides potentially crucial information regarding how the patient may react to treatment, as well as aiding pharmacists in refilling prescriptions
GeneralNotes	Patient history (and relevant family history) and information that is not easily standardizable	Apprises the accessing healthcare professional of highly important information to aid and facilitate care

For the physician entity:

Field	What it Stores	Why it's Needed
PhysicianID	An identification number or code unique to each physician	Acts as a "signature" to identify whose notes belong to whom on after the record has been updated
Specialty	The physician's medical specialization	Can ensure credibility of comments on the patients' conditions and provide some insight into the situation before reading the entire record
Province	The province in which the physician practices	Provides insight into the capacity in which the physician can provide care.

And for the pharmacy entity:

Field	What it Stores	Why it's Needed
PharmacyID	An identification number or code unique to each pharmacy	Identifies the pharmacy or pharmacies from which the person fills a prescription
Province	The province in which the pharmacy is located	Provides insight into the prescriptions the pharmacy can fill given varying healthcare coverage between provinces

### Structural Database Rules

We can begin with the first use case. I will not restate it since it is outlined above, and I have not revised it. From this use case, we can identify 2 out of the 3 (so far) relevant entities in our database (despite the depth of information that this database will hold, I can only thus far envision 3 separate entities). The first is, of course, the patient's record. It should be noted that a person may not necessarily have a record if they have never seen a doctor, but since we will not be making a comprehensive database of all

Canadians, we will only consider those who have seen doctors. This makes the situation such that there is a one-to-one relationship between person and record and vice versa, and thus we need only to create the record entity.

The second entity we can identify from the first use case is the physician. A physician can see 0 or several records, depending on their clientele, but a record must always be associated with at least one physician. Through the definition of these two entities, we arrive at our first database rule:

1. A record is associated with and edited by 1 or more physicians; a physician can be associated with and edit 0 or more records.

It should also be noted that a physician can view a record, and thus be associated with it, but not edit it, as would be the case with physicians asking their colleagues for recommendations on a certain patient's situation. This accounts for our second use case, in which we find no new entities or relevant database relationships. We could define a relationship from the physician entity to itself to account for when a physician asks for a second opinion, but since viewing the patient's record is effectively the same as the patient seeing the physician themselves, we will just incorporate that relationship into the record-physician relationship.

This brings us to the third and final use case. That is: the patient visiting a new pharmacy. We can identify a new entity here: the pharmacy. Since it is entirely possible that a patient does not have any active prescriptions, each record need not necessarily have an associated pharmacy. Though, if the patient has active prescriptions and is travelling, their record may be associated with several pharmacies depending on when they need refills of their prescriptions. As such, we can define our second, and thus far final, database rule:

2. A record is associated with and can be updated by 0 or more pharmacies; a pharmacy can be associated with and update 0 or more records.

### Specialization-Generalization Structural Database Rules

After examining my use cases listed above, I can conclude, for now, that we can construct two specialization-generalization structural database rules. Both rules can be derived from a slightly updated version of my first use case: the one pertaining to the *quick and easy accessing of records by different physicians*.

1. The patient (*male, female, or non-binary*) visits a different physician (*a general practitioner, internist, pediatrician*) than who they would normally see (due to travel, etc.)
2. The physician accesses the database through the secure online client and is instantly apprised of the patient's medical history.
3. The physician then provides whatever care is necessary in accordance with the patient's complaints and history.
4. The physician enters the updated information into the client which is immediately synced with the database and is readily accessible to any other healthcare professional that the patient may see.

In bold font, I have indicated the subtypes of the patient and physician entities. The subtypes given for the patient entity are an exhaustive list (as it pertains to medical treatment outside of psychiatry) and therefore the S-G relationship is totally complete. A third structural database rule can be then written as follows:

3. A patient is a male, female, or non-binary person.

On the other hand, the subtypes I have provided for the physician entity do not comprise an exhaustive list. For a real-world implementation of a database such as this, I would more than likely need to define a subtype for each recognized medical specialty in Canada. However, the completeness of this list is ostensibly irrelevant to the scope of this project, so I have chosen to define this S-G relationship as partially complete and only define subtypes for the three most common specialties. If this is an erroneous judgment call on my part, I will change it to be more comprehensive in future iterations, but for now, we arrive at our fourth and (so far) final structural database rule:

4. A physician is a general practitioner, an internist, a pediatrician, or none of these.

### Attributes and Full DBMS Physical ERD

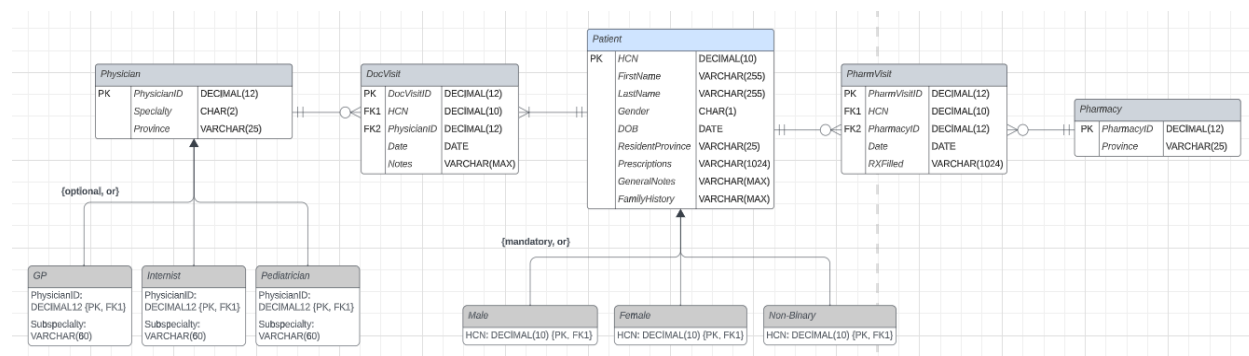
In the process of reading through the Iteration 4 document, I realized that I had already included most of the relevant attributes in my Initial DBMS Physical ERD for Iteration 3. For the, albeit brief, descriptions of the keyed attributes, please refer to that section of my last submission. Here, I will revise and list (more comprehensively than in the Use Cases and Fields section) the non-keyed attributes along with the specifics of what they will store. I also have in mind one additional attribute that I will list and subsequently add to my amended DBMS Physical ERD.

Table	Attribute	Datatype	Reasoning
Physician	Specialty	VARCHAR(32)	This attribute is a subtype discriminator to indicate the specialty of practice of the physician. I use VARCHAR(32) so that I can specify the name of the specialty explicitly.
Physician	FirstName	VARCHAR(255)	The first name of the physician.
Physician	LastName	VARCHAR(255)	The last name of the physician.
Physician	Province	VARCHAR(25)	The Canadian province with the longest name is Newfoundland and Labrador, so we must allow enough characters for that.
DocVisit	Date	DATE	The date on which the patient visits the physician.
DocVisit	Notes	VARCHAR(MAX)	Medical notes can situationally be quite long, so I am trying to

			allow for maximum flexibility.
Patient	FirstName	VARCHAR(255)	The first name of the patient.
Patient	LastName	VARCHAR(255)	The last name of the patient.
Patient	Gender	CHAR(1)	Subtype discriminator indicating the gender (male, female, or non-binary/intersex) of the patient.
Patient	DOB	DATE	The date of birth of the patient.
Patient	ResidentProvince	VARCHAR(25)	The reasoning here is the same as for the Physician's Province attribute.
Patient	Prescriptions	VARCHAR(1024)	Some patients also have a long list of prescribed medications. We allot 1024 characters to list each of them.
Patient	GeneralNotes	VARCHAR(MAX)	The general summary of the patient would likely be at least as long as the notes taken on any particular visit, so we also allot this attribute the maximum character limit.
Patient	FamilyHistory	VARCHAR(MAX)	I cannot anticipate the entirety of hereditary ailments that may affect a given patient, so we allow for the maximum character limit.
PharmVisit	Date	DATE	The date on which the patient visits the pharmacy
PharmVisit	RXFilled	VARCHAR(1024)	The patient will get, at most, all their prescriptions filled during a PharmVisit.
Pharmacy	Province	VARCHAR(25)	The reasoning here is the same as for both the Physician's Province attribute and the Patient's ResidentProvince attribute.

I have not added any new attributes to the Gender subtypes, as they are primarily to be used to provide context to the Patient's GeneralNotes. I could, potentially, include attributes discussing some gender-specific predispositions the patient may have, though I think the new FamilyHistory attribute in Patient will adequately encompass these subtleties. I have also made an executive decision to remove the Allergies attribute from Patient for the sake of simplicity. If the patient has relevant allergies, they can be listed in the GeneralNotes section, as it seems this attribute is becoming more of a general summary of the patient rather than a comprehensive account of their ailments. The more specific notes will be kept in the Notes section of DocVisit.

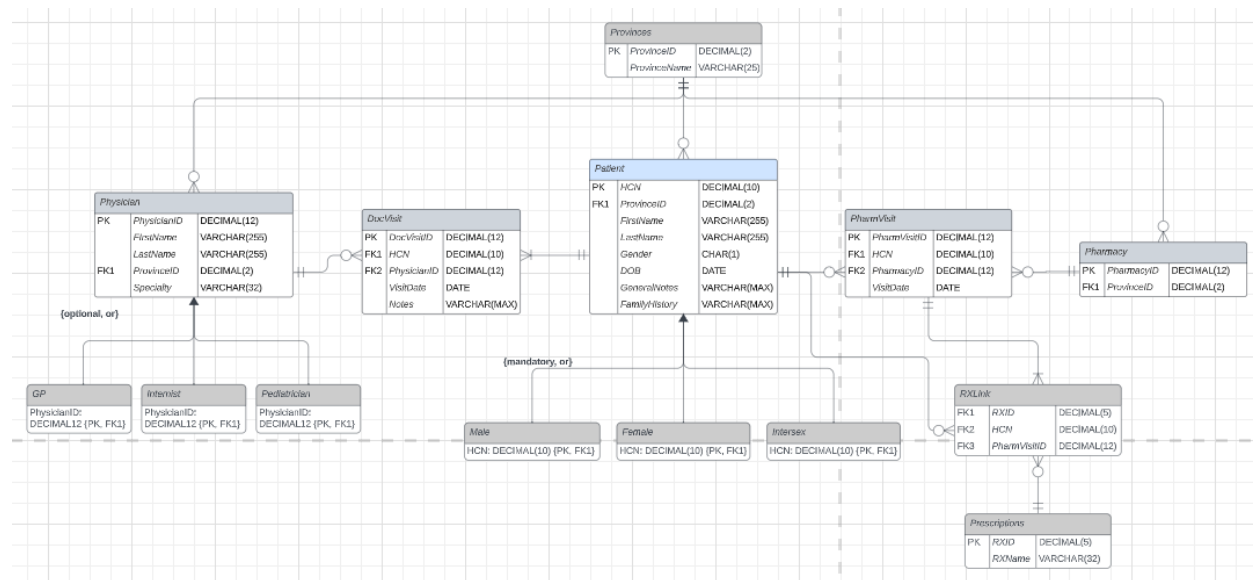
With these attributes more finely detailed, I can now construct my Full DBMS Physical ERD:



Though, as I have alluded to previously, there may be some room for expansion of more subtle details, I believe this ERD sufficiently encapsulates the significant aspects of the database at hand and is robust in its handling of pertinent information. (I removed the Subspecialty attribute from the Physician subtypes after capturing this screenshot and changing the ERD. Please disregard.)

## Normalization

Upon inspecting my physical ERD, I can identify two sources of redundancy: the Province information, and the Prescription/RXFilled information. Many physicians, patients, and pharmacies can be located in the same province; and many patients can be prescribed/can refill the same prescription. To remedy this, we create three new entities: Provinces, Prescriptions, and RXLink (due to the M:N relationship that arises from the new Prescriptions table). The normalized ERD is depicted below.

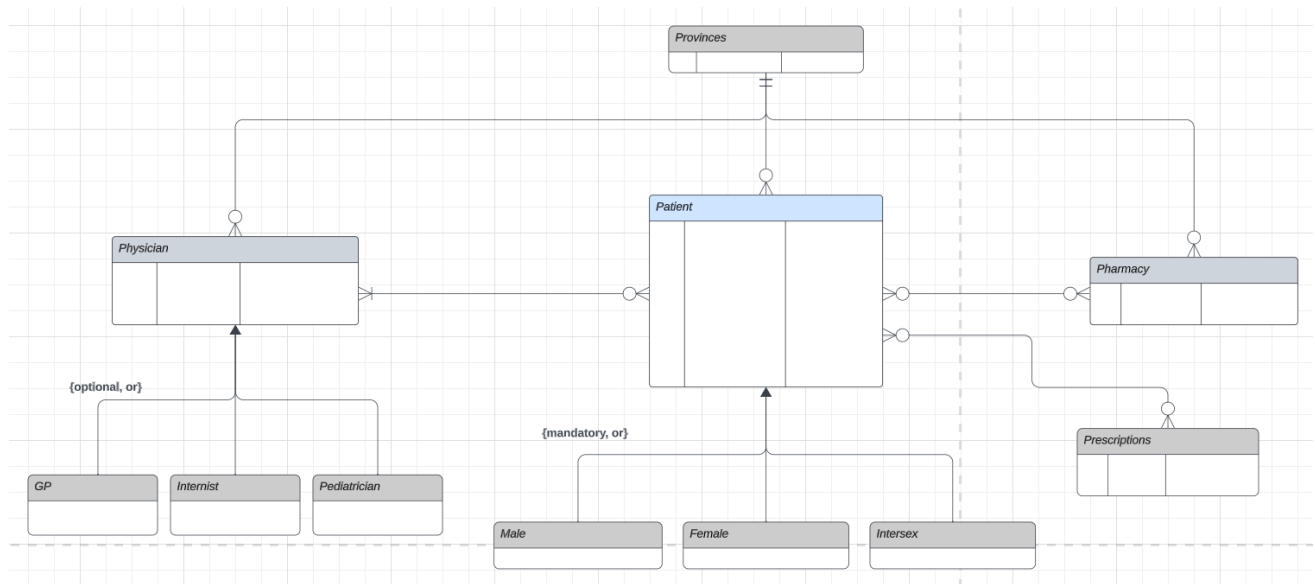


By implementing these changes, we now have entities specifically dedicated to storing the names of the provinces and prescriptions, as well as an entity dedicated to each instance of one patient refilling one prescription. All tables in the database are now in BCNF (according to my assessment), and the new structural database rules, in addition to those listed in previous sections, are:

5. A patient is a resident of one province; a province can be home to 0 to many patients.
6. A physician practices in one province; a province can be the place of practice of 0 to many physicians.
7. A pharmacy is located in one province; a province can contain 0 to many pharmacies.
8. A patient can be prescribed 0 to many prescriptions; a prescription can be prescribed to 0 to many patients.

We can now reconstruct our conceptual ERD to reflect these new rules, shown below.





We now have everything we need to begin building the tables in SQL Server, for which I attach a representative (but not comprehensive) screenshot below.

```
34 CREATE TABLE Pharmacy (  
35   PharmacyID DECIMAL(12) NOT NULL PRIMARY KEY,  
36   ProvinceID DECIMAL(2) NOT NULL,  
37   FOREIGN KEY (ProvinceID) REFERENCES Provinces(ProvinceID));  
38  
39 CREATE TABLE PharmVisit (  
40   PharmVisitID DECIMAL(12) NOT NULL PRIMARY KEY,  
41   HCN DECIMAL(10) NOT NULL,  
42   PharmacyID DECIMAL(12) NOT NULL,  
43   VisitDate DATE NOT NULL,  
44   FOREIGN KEY (HCN) REFERENCES Patient(HCN),  
45   FOREIGN KEY (PharmacyID) REFERENCES Pharmacy(PharmacyID));  
46  
47 CREATE TABLE Prescriptions (  
48   RXID DECIMAL(5) NOT NULL PRIMARY KEY,  
49   RXName VARCHAR(32) NOT NULL);  
50  
51 CREATE TABLE RXLink (  
52   RXID DECIMAL(5) NOT NULL,  
53   HCN DECIMAL(10) NOT NULL,  
54   PharmVisitID DECIMAL(12) NOT NULL,  
55   FOREIGN KEY (RXID) REFERENCES Prescriptions(RXID),  
56   FOREIGN KEY (HCN) REFERENCES Patient(HCN),  
57   FOREIGN KEY (PharmVisitID) REFERENCES PharmVisit(PharmVisitID));  
58  
59 CREATE TABLE GP (  
60   PhysicianID DECIMAL(12) NOT NULL PRIMARY KEY,  
61   FOREIGN KEY (PhysicianID) REFERENCES Physician(PhysicianID));  
62  
63 CREATE TABLE Internist (  
64   PhysicianID DECIMAL(12) NOT NULL PRIMARY KEY,  
65   FOREIGN KEY (PhysicianID) REFERENCES Physician(PhysicianID));  
66  
67 CREATE TABLE Pediatrician (  
68   PhysicianID DECIMAL(12) NOT NULL PRIMARY KEY,  
69   FOREIGN KEY (PhysicianID) REFERENCES Physician(PhysicianID));  
70  
71 CREATE TABLE Male (  
72   HCN DECIMAL(10) NOT NULL PRIMARY KEY,  
73   FOREIGN KEY (HCN) REFERENCES Patient(HCN));  
74  
75 CREATE TABLE Female (  
76   HCN DECIMAL(10) NOT NULL PRIMARY KEY,  
77   FOREIGN KEY (HCN) REFERENCES Patient(HCN));  
78  
79 CREATE TABLE Intersex (  
80   HCN DECIMAL(10) NOT NULL PRIMARY KEY,  
81   FOREIGN KEY (HCN) REFERENCES Patient(HCN));
```

0 %

Messages

Commands completed successfully.

## Indexing

The already-indexed primary keys are:

- Province.ProvinceID
- Patient.HCN
- Male.HCN
- Female.HCN
- Intersex.HCN
- Physician.PhysicianID
- GP.PhysicianID
- Internist.PhysicianID
- Pediatrician.PhysicianID
- DocVisit.DocVisitID
- Pharmacy.PharmacyID
- PharmVisit.PharmVisitID
- Prescriptions.RXID

We can now index all the foreign keys. In the table below, we identify each of these foreign keys, whether or not they are unique, and a description of their uniqueness (or lack thereof).

Column	Unique?	Description of Uniqueness
Patient.ProvinceID	No	Not unique because there can be many patients from the same province
Physician.ProvinceID	No	Not unique because there can be many physicians from the same province
DocVisit.HCN	No	Not unique because a patient can visit the same physician multiple times
DocVisit.PhysicianID	No	Not unique because the same physician can conduct many appointments
Pharmacy.ProvinceID	No	Not unique because there can be many pharmacies in the same province
PharmVisit.HCN	No	Not unique because a patient can visit the same pharmacy multiple times
PharmVisit.PharmacyID	No	Not unique because the same pharmacy can serve many patients
RXLink.RXID	No	Not unique because the same prescription can be refilled multiple times
RXLink.HCN	No	Not unique because a patient can refill prescriptions multiple times

RXLink.PharmVisitID	No	Not unique because multiple prescriptions can be refilled during one PharmVisit
---------------------	----	---

I can also immediately identify three other columns that would be well served with indexes, judging by what I assume could be common queries. I list them in the table below in the same format as above.

Column	Unique?	Description of Uniqueness
Physician.Specialty	No	Not unique because there can be multiple physicians that practice the same specialty
DocVisit.VisitDate	No	Not unique because many appointments can happen on the same day
PharmVisit.VisitDate	No	Not unique because there can be many pharmacy visits on the same day

Physician.Specialty is useful to index in case one desires to query for a specific catalog of procedures that are only conducted by specifically specialized physicians. DocVisit.VisitDate and PharmVisit.VisitDate are also useful to index in case there is some known procedural mishap or other event of interest, and one desires to find out all patients who may have been affected.

With these indexes explicitly determined, we can now implement them into our database in SQL Server, a screenshot for which is shown below.

```
--
86 CREATE INDEX PhysicianProvinceIdx
87 ON Physician(ProvinceID);
88
89 CREATE INDEX DocVisitHCNIdx
90 ON DocVisit(HCN);
91
92 CREATE INDEX DocVisitPhysicianIdx
93 ON DocVisit(PhysicianID);
94
95 CREATE INDEX PharmacyProvinceIdx
96 ON Pharmacy(ProvinceID);
97
98 CREATE INDEX PharmVisitHCNIdx
99 ON PharmVisit(HCN);
100
101 CREATE INDEX PharmVisitPharmacyIdx
102 ON PharmVisit(PharmacyID);
103
104 CREATE INDEX RXLinkRXIdx
105 ON RXLink(RXID);
106
107 CREATE INDEX RXLinkHCNIdx
108 ON RXLink(HCN);
109
110 CREATE INDEX RXLinkPharmVisitIdx
111 ON RXLink(PharmVisitID);
112
113 CREATE INDEX PhysicianSpecialtyIdx
114 ON Physician(Specialty);
115
116 CREATE INDEX DocVisitDateIdx
117 ON DocVisit(VisitDate);
118
119 CREATE INDEX PharmVisitDateIdx
120 ON PharmVisit(VisitDate);
```

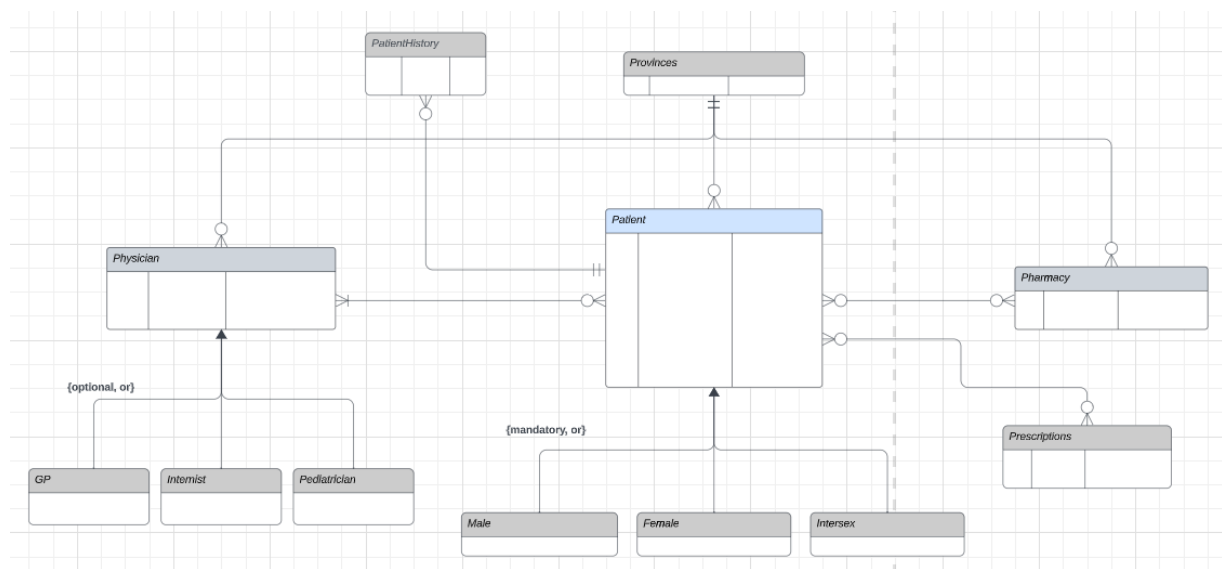
10 %

Messages

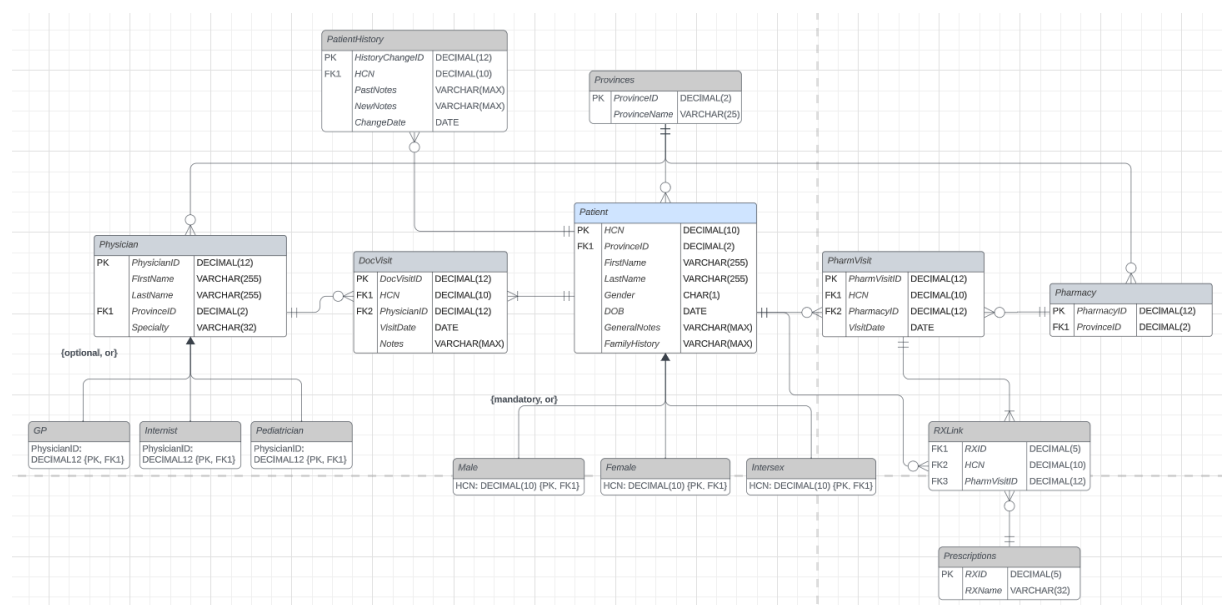
Commands completed successfully.

## History Table and Trigger Creation

The table I can envision incurring the most benefit from a history table is the Patient entity, particularly its GeneralNotes column. After each DocVisit event, the attending Physician will write some notes summarizing the visit and the Patient's condition, and these notes will need to be synced with the Patient's record. However the Patient's past notes may also remain of great importance, so we instantiate the PatientHistory table to keep track of these changes. The trigger associated with this table will activate whenever there is an update to the GeneralNotes column, and the new structural database rule associated with this table is: **each Patient may have their GeneralNotes updated 0 to many times; each GeneralNotes update is associated with one Patient.** To account for this new table and SRD, I include screenshots of my new conceptual ERD:



as well as my updated DBMS physical ERD:



Additionally, I include a table listing the attributes and descriptions thereof for the PatientHistory table.

Attribute	Datatype	Reasoning
HistoryChangeID	DECIMAL(12)	Primary key of the history table. Patients may visit Physicians many times so we use DECIMAL(12)
HCN	DECIMAL(10)	Foreign key referencing the HCN of the Patient whose record is being updated
PastNotes	VARCHAR(MAX)	The notes from the Patient's previous DocVisit. Datatype (necessarily) mirrors that of the Notes column of DocVisit and the GeneralNotes column of Patient
NewNotes	VARCHAR(MAX)	The notes from the Patient's latest DocVisit. Datatype (necessarily) mirrors that of the Notes column of DocVisit and the GeneralNotes column of Patient
ChangeDate	DATE	Date on which the Patient's record is updated. Coincides with the date of the Patient's DocVisit due to the construction of the trigger

I also include a screenshot of the definition of the PatientHistory table and its associated trigger below.

```
85  -- History Table (in the future: maybe add support for RX fills)
86  CREATE TABLE PatientHistory (
87    HistoryChangeID DECIMAL(12) NOT NULL PRIMARY KEY,
88    HCN DECIMAL(10) NOT NULL,
89    PastNotes VARCHAR(MAX) NOT NULL,
90    NewNotes VARCHAR(MAX) NOT NULL,
91    ChangeDate DATE NOT NULL,
92    FOREIGN KEY (HCN) REFERENCES Patient(HCN));
93
94  CREATE TRIGGER PatientHistoryTrigger
95  ON Patient
96  AFTER UPDATE
97  AS
98  BEGIN
99    DECLARE @PastNotes VARCHAR(MAX) = (SELECT GeneralNotes FROM DELETED);
100    DECLARE @NewNotes VARCHAR(MAX) = (SELECT GeneralNotes FROM INSERTED);
101
102    IF UPDATE(GeneralNotes)
103    BEGIN
104      INSERT INTO PatientHistory(HistoryChangeID, HCN, PastNotes, NewNotes, ChangeDate)
105      VALUES(ISNULL((SELECT MAX(HistoryChangeID)+1 FROM PatientHistory), 1), (SELECT HCN FROM INSERTED),
106              @PastNotes, @NewNotes, GETDATE());
107    END;
108
109  Messages
110  Commands completed successfully.
```

In this code, I ensure that the PatientHistory table is only inserted into when the GeneralNotes column of the Patient table is updated, since this is the only aspect of the Patient's record that will change over time. This is done through the use of the IF UPDATE(GeneralNotes) clause. Before this, I declare variables @PastNotes and @NewNotes that take the respective values of the GeneralNotes column from the DELETED and INSERTED pseudo tables, respectively. Using these variables inside my IF clause, I insert into the PatientHistory table a new HistoryChangeID, the HCN of the Patient who has been attended to, and these Past and New Notes, along with the ChangeDate which is the date of the DocVisit.

Updates to the GeneralNotes column only take place when there is a new DocVisit event, so this trigger will only activate through the execution of the AddDocVisit stored procedure, detailed in the next section where I have also included representative screenshots of its efficacy.

## Reusable, Transaction-Oriented Stored Procedures

The first use case for which I will implement a stored procedure is the adding of a new Patient to the database. This is a new use case I have constructed for this iteration because I realized I had not yet accounted for new patient additions, and because I needed an additional data-adding use case to reach the 3 necessary for this step. A screenshot of the stored procedure (for adding a new male patient) definition is included here. (I have revised my Use Cases and Fields section above to include this new case)

```
343 -- Stored Procedure: Adding male patients
344 CREATE PROCEDURE AddMalePatient @HCN DECIMAL(10), @ProvinceID DECIMAL(2), @FirstName VARCHAR(255), @LastName VARCHAR(255),
345 @DOB DATE, @GeneralNotes VARCHAR(MAX), @FamilyHistory VARCHAR(MAX)
346 AS
347 BEGIN
348     INSERT INTO Patient(HCN, ProvinceID, FirstName, LastName, Gender, DOB, GeneralNotes, FamilyHistory)
349     VALUES(@HCN, @ProvinceID, @FirstName, @LastName, 'M', CAST(@DOB AS DATE), @GeneralNotes, @FamilyHistory);
350
351     INSERT INTO Male(HCN)
352     VALUES(@HCN)
353 END;
354 GO
355
356 -- Stored Procedure: Adding female patients
357 CREATE PROCEDURE AddFemalePatient @HCN DECIMAL(10), @ProvinceID DECIMAL(2), @FirstName VARCHAR(255), @LastName VARCHAR(255),
358 @DOB DATE, @GeneralNotes VARCHAR(MAX), @FamilyHistory VARCHAR(MAX)
359 AS
360 BEGIN
361     INSERT INTO Patient(HCN, ProvinceID, FirstName, LastName, Gender, DOB, GeneralNotes, FamilyHistory)
362     VALUES(@HCN, @ProvinceID, @FirstName, @LastName, 'F', CAST(@DOB AS DATE), @GeneralNotes, @FamilyHistory);
363
364     INSERT INTO Female(HCN)
365     VALUES(@HCN)
366 END;
367 GO
368
369 -- Stored Procedure: Adding intersex patients
370 CREATE PROCEDURE AddIntersexPatient @HCN DECIMAL(10), @ProvinceID DECIMAL(2), @FirstName VARCHAR(255), @LastName VARCHAR(255),
371 @DOB DATE, @GeneralNotes VARCHAR(MAX), @FamilyHistory VARCHAR(MAX)
372 AS
373 BEGIN
374     INSERT INTO Patient(HCN, ProvinceID, FirstName, LastName, Gender, DOB, GeneralNotes, FamilyHistory)
375     VALUES(@HCN, @ProvinceID, @FirstName, @LastName, 'I', CAST(@DOB AS DATE), @GeneralNotes, @FamilyHistory);
376
377     INSERT INTO Intersex(HCN)
378     VALUES(@HCN)
379 END;
380 GO
```

I have also included a sample execution of this procedure to add a male patient, pictured in the screenshot below.

```
150 | BEGIN TRANSACTION AddMalePatient;
151 | EXECUTE AddMalePatient 0000000001, 01, 'John', 'Doe', '04-JUL-1973', 'This patient has asthma and is allergic
152 |     to peanuts.', 'The patients maternal grandfather has Alzheimers Disease.';
153 | COMMIT TRANSACTION AddMalePatient;
```

100 %

Messages

(1 row affected)

(1 row affected)

To successfully implement this procedure, I had to first fully populate the Provinces table due to the foreign key reference to it in the Patient table. I won't include screenshots of this process since it was very straightforward and not very time consuming (Canada only has 13 provinces and territories).

The second use case for which I will define a stored procedure is the adding of a new Physician. As with the procedures adding Patients, I have defined three procedures for this use case: one for each specialty that I am considering in this project.

```
382 --Stored Procedure: Adding Physician (GP)
383 CREATE PROCEDURE AddGP @PhysicianID DECIMAL(12), @FirstName VARCHAR(255), @LastName VARCHAR(255), @ProvinceID DECIMAL(2)
384 AS
385 BEGIN
386     INSERT INTO Physician(PhysicianID, FirstName, LastName, ProvinceID, Specialty)
387     VALUES(@PhysicianID, @FirstName, @LastName, @ProvinceID, 'GP');
388
389     INSERT INTO GP(PhysicianID)
390     VALUES(@PhysicianID)
391 END;
392 GO
393
394 --Stored Procedure: Adding Physician (Internist)
395 CREATE PROCEDURE AddInternist @PhysicianID DECIMAL(12), @FirstName VARCHAR(255), @LastName VARCHAR(255), @ProvinceID DECIMAL(2)
396 AS
397 BEGIN
398     INSERT INTO Physician(PhysicianID, FirstName, LastName, ProvinceID, Specialty)
399     VALUES(@PhysicianID, @FirstName, @LastName, @ProvinceID, 'Internist');
400
401     INSERT INTO Internist(PhysicianID)
402     VALUES(@PhysicianID)
403 END;
404 GO
405
406 --Stored Procedure: Adding Physician (Pediatrician)
407 CREATE PROCEDURE AddPediatrician @PhysicianID DECIMAL(12), @FirstName VARCHAR(255), @LastName VARCHAR(255), @ProvinceID DECIMAL(2)
408 AS
409 BEGIN
410     INSERT INTO Physician(PhysicianID, FirstName, LastName, ProvinceID, Specialty)
411     VALUES(@PhysicianID, @FirstName, @LastName, @ProvinceID, 'Pediatrician');
412
413     INSERT INTO Pediatrician(PhysicianID)
414     VALUES(@PhysicianID)
415 END;
416 GO
```

I also include a sample execution (of the AddGP procedure).



```
433 -- Sample GP addition
434 BEGIN TRANSACTION AddGP;
435 EXECUTE AddGP 101, 'Sebastian', 'Jurga', 9;
436 COMMIT TRANSACTION AddGP;
437
30 %
Messages

(1 row affected)

(1 row affected)
```

Finally, I define a stored procedure for the use case in which a Patient attends an appointment with a Physician (a new DocVisit event). This procedure will also update the GeneralNotes section of the Patient table, which will, in turn, activate the history trigger and insert a new entry into my PatientHistory table. I include a screenshot of this procedure's definition below.

```
441 -- Stored Procedure: Adding DocVisit events
442 CREATE PROCEDURE AddDocVisit @DocVisitID DECIMAL(12), @HCN DECIMAL(10), @PhysicianID DECIMAL(12), @Notes VARCHAR(MAX)
443 AS
444 BEGIN
445     INSERT INTO DocVisit(DocVisitID, HCN, PhysicianID, VisitDate, Notes)
446     VALUES(@DocVisitID, @HCN, @PhysicianID, GETDATE(), @Notes);
447
448     UPDATE Patient
449     SET GeneralNotes = @Notes
450     WHERE HCN = @HCN;
451 END;
452 GO
```

I additionally include screenshots of a couple of sample executions of this procedure, to demonstrate that my trigger and associated PatientHistory table are working as expected.

```
465 -- Sample DocVisit addition 1
466 BEGIN TRANSACTION AddDocVisit;
467 EXECUTE AddDocVisit 1, 1, 1, 'Patient has no further updates on his asthma and should continue normal puffer dosing.';
468 COMMIT TRANSACTION AddDocVisit;
469
30 %
Messages

(1 row affected)

(1 row affected)

(1 row affected)

470 -- Sample DocVisit addition 2
471 BEGIN TRANSACTION AddDocVisit;
472 EXECUTE AddDocVisit 2, 1, 1, 'Patient had an asthma attack immediately after his last visit and came back but is now stable';
473 COMMIT TRANSACTION AddDocVisit;
474

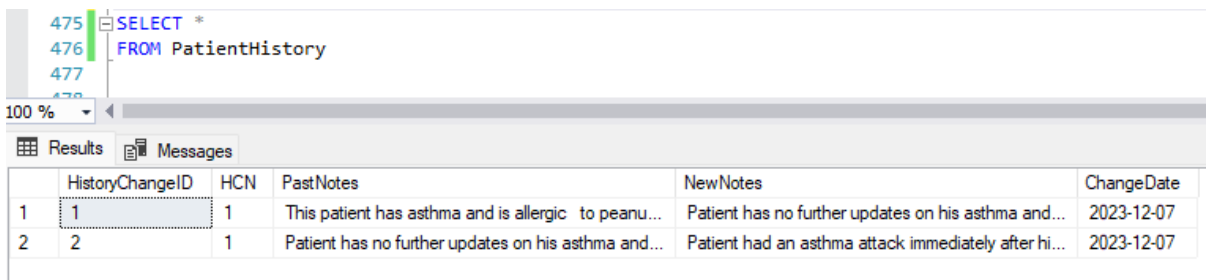
30 %
Messages

(1 row affected)

(1 row affected)

(1 row affected)
```

And lastly, I include a screenshot of my PatientHistory table to conclude the demonstration of its efficacy.



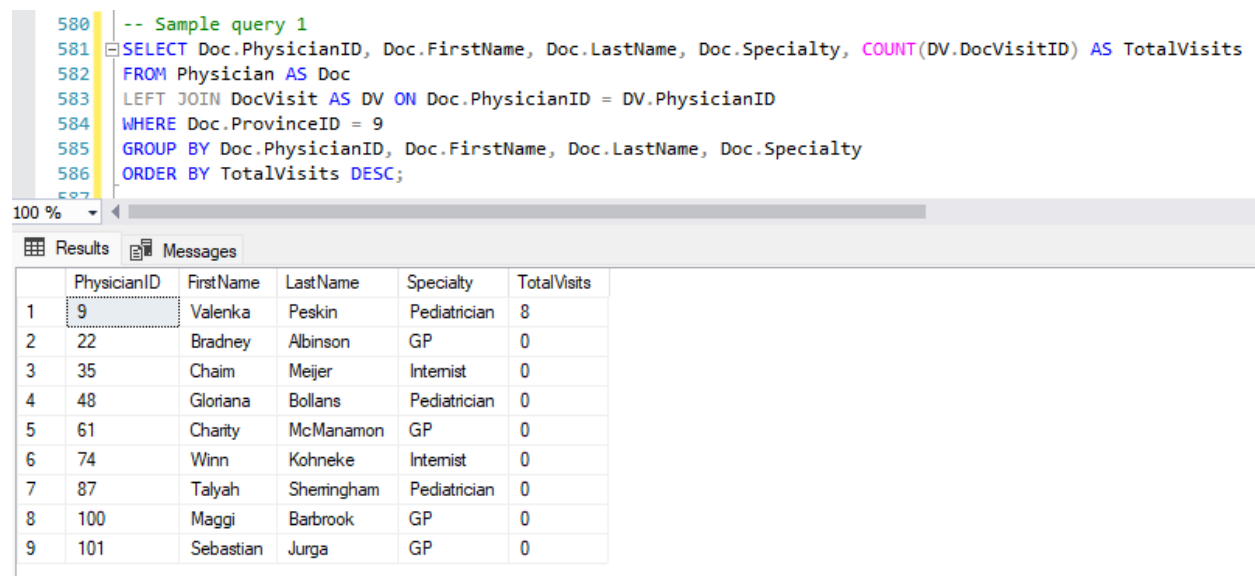
```
475 SELECT *
476 FROM PatientHistory
477
```

	HistoryChangeID	HCN	PastNotes	NewNotes	ChangeDate
1	1	1	This patient has asthma and is allergic to peanu...	Patient has no further updates on his asthma and...	2023-12-07
2	2	1	Patient has no further updates on his asthma and...	Patient had an asthma attack immediately after hi...	2023-12-07

Though this table is worked as intended, we see that this table may be a little redundant in its practical use due to the restatement of the PastNotes with each new entry. I decided to keep it as is for now though.

## Questions and Queries

The first question for which I will compose a query addresses the assessment of productivity of the Physicians in a certain Province. I will retrieve the total number of visits for each Physician in a specific Province, and list them in descending order of visit count. Additionally, this query can help healthcare administrators or analysts understand the workload distribution among Physicians in a specific province. Identifying Physicians with higher visit counts may be useful for resource allocation, scheduling, and overall management of healthcare services. I include a screenshot of the query in action below.

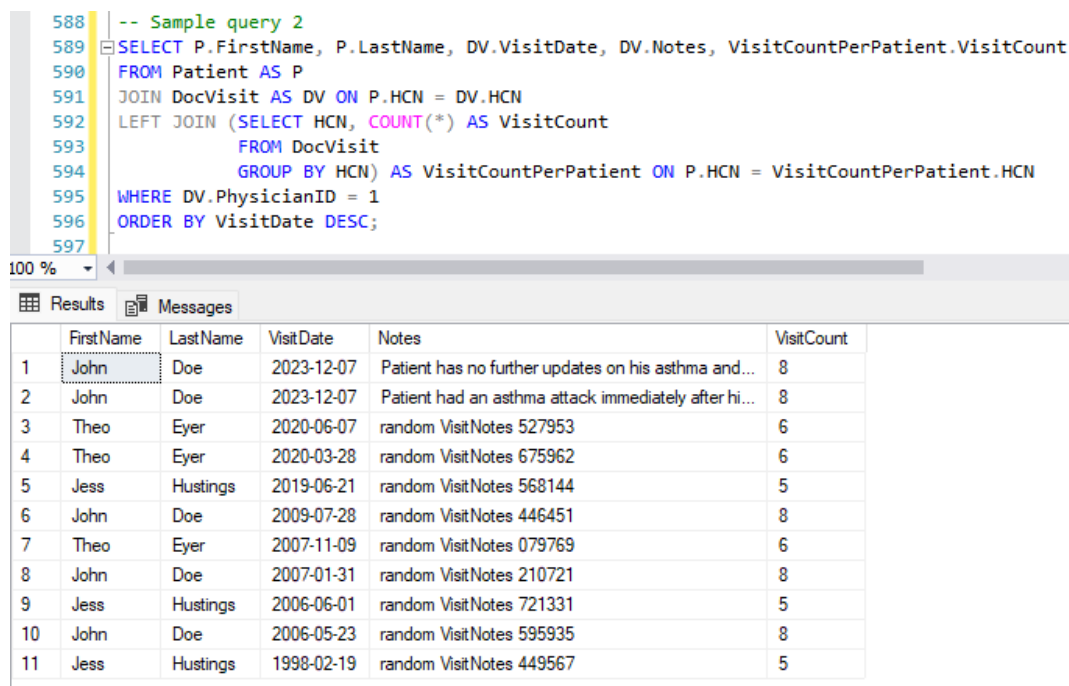


```
580 -- Sample query 1
581 SELECT Doc.PhysicianID, Doc.FirstName, Doc.LastName, Doc.Specialty, COUNT(DV.DocVisitID) AS TotalVisits
582 FROM Physician AS Doc
583 LEFT JOIN DocVisit AS DV ON Doc.PhysicianID = DV.PhysicianID
584 WHERE Doc.ProvinceID = 9
585 GROUP BY Doc.PhysicianID, Doc.FirstName, Doc.LastName, Doc.Specialty
586 ORDER BY TotalVisits DESC;
```

	PhysicianID	FirstName	LastName	Specialty	TotalVisits
1	9	Valenka	Peskin	Pediatrician	8
2	22	Bradney	Albinson	GP	0
3	35	Chaim	Meijer	Internist	0
4	48	Gloriana	Bollans	Pediatrician	0
5	61	Charity	McManamon	GP	0
6	74	Winn	Kohneke	Internist	0
7	87	Talyah	Sherringham	Pediatrician	0
8	100	Maggi	Barbrook	GP	0
9	101	Sebastian	Jurga	GP	0

The table that has been retrieved is comprised of all of the Physicians in Nova Scotia (ProvinceID = 9) ordered from 'busiest' to 'least busy'.

The second use case I will address with a query is that which finds Patients who have visited a specific Physician, along with the number of visits for each patient. This query is beneficial for Physicians or healthcare providers to understand the visit patterns of their patients. By retrieving a count of visits for each Patient who visited a specific Physician, healthcare professionals can gain insights into Patient engagement, frequency of visits, and potentially identify Patients who may require more focused care or follow-ups. It can also aid in optimizing Patient management and resource allocation within the healthcare system. Again, I attach a screenshot of this query's usage.



The screenshot shows a SQL query editor with a query and its results. The query is as follows:

```
-- Sample query 2
SELECT P.FirstName, P.LastName, DV.VisitDate, DV.Notes, VisitCountPerPatient.VisitCount
FROM Patient AS P
JOIN DocVisit AS DV ON P.HCN = DV.HCN
LEFT JOIN (SELECT HCN, COUNT(*) AS VisitCount
FROM DocVisit
GROUP BY HCN) AS VisitCountPerPatient ON P.HCN = VisitCountPerPatient.HCN
WHERE DV.PhysicianID = 1
ORDER BY VisitDate DESC;
```

The results are displayed in a table with the following columns: FirstName, LastName, VisitDate, Notes, and VisitCount. The table contains 11 rows of data, ordered by VisitDate in descending order.

	FirstName	LastName	VisitDate	Notes	VisitCount
1	John	Doe	2023-12-07	Patient has no further updates on his asthma and...	8
2	John	Doe	2023-12-07	Patient had an asthma attack immediately after hi...	8
3	Theo	Eyer	2020-06-07	random VisitNotes 527953	6
4	Theo	Eyer	2020-03-28	random VisitNotes 675962	6
5	Jess	Hustings	2019-06-21	random VisitNotes 568144	5
6	John	Doe	2009-07-28	random VisitNotes 446451	8
7	Theo	Eyer	2007-11-09	random VisitNotes 079769	6
8	John	Doe	2007-01-31	random VisitNotes 210721	8
9	Jess	Hustings	2006-06-01	random VisitNotes 721331	5
10	John	Doe	2006-05-23	random VisitNotes 595935	8
11	Jess	Hustings	1998-02-19	random VisitNotes 449567	5

The output of this query is a table consisting of all of the patients who have visited the Physician in question (PhysicianID = 1), as well as a log of the Physician's notes from each of the Patients' visits. The last column is a count of how many times the Patient has visited any Physician, which could help to indicate the severity of their condition.

The last use case I will address with a query is that which identifies 10 Patients with the highest number of visits in the last 3 months, their attending Physicians, and their changes in condition (indicated by the Physician's notes). This query contributes to the efficient management of Patient care by identifying high-frequency Patients, linking them to their Physicians, and providing insights into historical changes in Patient notes. This information can be valuable for healthcare professionals in optimizing resource allocation, improving Patient outcomes, and delivering patient-centric care. And I lastly attach a screenshot of this query's yield below.

```
598 -- Sample query 3
599 WITH RecentVisitCounts AS (SELECT DV.HCN, DV.PhysicianID, COUNT(*) AS TotalVisits
600 FROM DocVisit AS DV
601 WHERE DV.VisitDate >= DATEADD(MONTH, -3, GETDATE())
602 GROUP BY DV.HCN, DV.PhysicianID)
603 SELECT P.HCN, P.FirstName AS PatientFirstName, P.LastName AS PatientLastName, RVC.PhysicianID,
604 Doc.FirstName AS PhysicianFirstName, Doc.LastName AS PhysicianLastName, RVC.TotalVisits,
605 PH.PastNotes, PH.NewNotes, PH.ChangeDate
606 FROM (SELECT HCN, PhysicianID, TotalVisits, ROW_NUMBER() OVER (ORDER BY TotalVisits DESC) AS VisitRank
607 FROM RecentVisitCounts) AS RVC
608 JOIN Patient AS P ON RVC.HCN = P.HCN
609 JOIN Physician AS Doc ON RVC.PhysicianID = Doc.PhysicianID
610 LEFT JOIN PatientHistory AS PH ON P.HCN = PH.HCN
611 WHERE RVC.VisitRank <= 10;
612
```

HCN	PatientFirstName	PatientLastName	PhysicianID	PhysicianFirstName	PhysicianLastName	TotalVisits	PastNotes	NewNotes	ChangeDate
1	John	Doe	1	Lem	Savil	2	This patient has asthma and is allergic to peanut...	Patient has no further updates on his asthma and...	2023-12-07
2	John	Doe	1	Lem	Savil	2	Patient has no further updates on his asthma and...	Patient had an asthma attack immediately after hi...	2023-12-07

This query yields a table which displays (in this case) the only patient that has visited a Physician multiple times in the last 3 months, as well as the name of the Physician(s) they visited, their number of visits, the notes taken at each visit, and the date of their visit.

### Summary and Reflection (revised to only summarize iteration 5)

This 5th and final (for the purposes of this class) iteration of this project has proven to be quite complex, though it seems this complexity may not be a function of the demands of the iteration itself, but rather a function of how large my database is becoming. Creating such things as stored procedures, triggers, and history tables is not inherently complicated, though it certainly becomes so when there are an ever-growing number of factors to keep track of in the process. I believe I was able to sufficiently map out most of my database in accordance with the project's requirements, however there are certainly some areas that could be improved upon. Particularly the integrability of the Pharmacy-related side of the DBMS. I was not able to coherently construct stored procedures for the RXLink or PharmVisit entities due to the fact that each PharmVisit could be comprised of many RXLink events, nor was I able to devise a solution for updating Patient records in accordance with PharmVisits. This issue may be a function of the fact that I have no reference in my PharmVisit entity to the Prescriptions that are getting filled, though if this is the case, I think I would need to deep significantly further into the foundation of the database to implement a consistent and lasting solution.

I incurred a brief few moments where I considered appending the Notes taken during DocVisits directly to the GeneralNotes section, which would have resulted in a monolith of GeneralNotes in the Patient's record. I ultimately decided against this as I deemed it reasonable for new attending Physicians to simply refer to the PatientHistory table to apprise themselves of past conditions. This decision was further validated when I considered the fact that many Physicians would likely write status quo perpetuations in their Notes, which would end up bloating the GeneralNotes section.

Further, in order to complete the questions and queries section of this iteration, I had to populate all of the relevant tables with mock data using Mockaroo. This was of little immediate consequence, though it brought me to the realization that, in order to adequately populate the PatientHistory and subtype tables, all practical additions to the database would need to be done through the use of the stored procedures, since this is the only way that I have defined insertions into these tables.

I know that this is the terminus of this project, but I have found this experience to be very interesting and informative, and I think I will continue working on the database until I can get it into a fully practical

Sachin Mohandas  
BU ID: U79542832

Project Iteration 5

MET CS 669  
Due: Dec. 2, 2023 (ext. to Dec. 7)

position, and perhaps I can begin to develop a workable client/user interface to go along with it. Thank you for bearing with me throughout!

Notes not explicitly stated above: I ensured that my queries met the stated complexity requirements, and I also added FirstName and LastName columns to my Physician table.