Part 4: Smilow's Database SQL Final Implementation

Team 2: Anushka Nath, Sam Ziessler, Karl Chavez, Christina Duong

Intr Database Sys (7194)

April 2023

Part 1: Updates (EERD, relational schema, and relational algebra)

Changes for Relational Schema:

The text in italics is our previous work, and the text in bold is the corrected work.

Address (AddressID, StreetAddress, Zipcode, State, City, Country, MemberID (FK))

- Transitive Dependency: {Zipcode} -> {City, State}
    - These attributes (City, State) are fully dependent on Zipcode, so they're removed from the Address table. Instead, we've created two new relations, as shown below
- We also removed 'Country' as it is not necessary and complicates normalizations.

Address (AddressID, StreetAddress, Zipcode, MemberID (FK))

Zip(ZipCode (PK), City, State, Address (FK))

Invoice(InvoiceID (PK), BillingDate, Amount)

- *Dependency: {InvoiceID} -> {BillingDate, Amount}*
- *No partial or transitive dependency, determinants are candidate keys, BCNF*

Invoice(InvoiceID (PK), BillingDate, Amount, ApptID (FK), MedicalID(FK))

- **We added the foreign key to Appointment and Employee back in based on our EERD.**
- **Dependency: {InvoiceID} -> {BillingDate, Amount, ApptID (FK), MedicalID(FK)}**

Appointment (ApptID, ApptDate, Status, ReasonForVisit, PatientID (FK), MedicalID (FK), InvoiceID (FK))

- *Dependency: {ApptID} -> {ApptDate, Status, ReasonForVisit, PatientID, MedicalID, InvoiceID}*
- *No partial or transitive dependency, determinants are candidate keys, BCNF*

Appointment (ApptID, ApptDate, Status, ReasonForVisit, PatientID (FK), MedicalID (FK))

- **We removed the foreign key to Invoice based on our EERD**
- **Dependency: {ApptID} -> {ApptDate, Status, ReasonForVisit, PatientID, MedicalID}**

InsurancePlan (PlanID, InsurancePlanType, ActiveDate, MemberID (FK), PatientID (FK))

- *Dependency: {PlanID} -> {InsurancePlanType, ActiveDate, MemberID}*
- *No partial or transitive dependency, determinants are candidate keys, BCNF*

InsurancePlan (PlanID, InsurancePlanType, ActiveDate, GroupNumber, CompanyID (FK), PatientID (FK))

- **Added GroupNumber based on feedback.**
- **Renamed MemberID to CompanyID based on feedback.**

InsuranceCompany (MemberID, CompanyName, CompanyEmail, GroupNumber, CompanyPhone, AddressID)

- *Dependency: {MemberID} -> {CompanyNmae, CompanyEmail, GroupNumber, CompanyPhone, AddressID}*
- *No partial or transitive dependency, determinants are candidate keys, BCNF*

InsuranceCompany (CompanyID, CompanyName, CompanyEmail, CompanyPhone, AddressID)

- **Renamed MemberID to CompanyID based on feedback.**
- **Removed GroupNumber based on feedback.**

CheckInfo(AccountNo (FK), CheckNo, RoutingNo)

CheckInfo(AccountNo (FK), RoutingNo)

- **Removed CheckNo because AccountNo does not determine CheckNo.**

CheckingAccount(PaymentID (FK), AccountNo, CheckNo)

- **Added CheckNo into CheckingAccount.**

Changes for EERD:

We renamed attributes reflecting the feedback, such as “MemberID” changed to “CompanyID”. We also added attributes that reflect the tables, such as adding “CompanyName” to the Company entity.

Final Relational Algebra:

a. Create a list of patients and the medications they currently take
   i. Patient_Person ← $\Pi_{(PatientIID, FName, LName)}$(Person $\bowtie_{PersonID = PatientID}$ Patient)
   ii. Current_Medication ← $\sigma_{PrescripStart<TODAY\ (PrescipEnd > TODAY\ OR\ PrescripEnd = NULL)}$(Patient_Medication)
   iii. Medication_Info ← $\Pi_{(PatientID, DrugName)}$(Current_Medication * Medication)
   iv. **Patient_Medication ← $\Pi_{(FName, LName, DrugName)}$ (Patient_Person * Medication_Info)**

b. Display Patient information for patients who currently have Delta Dental insurance policy
   i. Patient_Person ←(Person $\bowtie_{PersonID = PatientID}$ Patient)
   ii. Plan_Company ← $\Pi_{(PatientID, CompanyName)}$ (InsurancePlan * InsuranceCompany)
   iii. **DeltaDental_Patients ← $\sigma_{CompanyName="Delta Dental"}$ (Patient_Person * Plan_Company)**

c. Generate a list of procedures and service dates performed by Dr. Smillow
   i. Medical_Employee ← $\sigma_{MedicalFlag = "True"}$(Employee)
   ii. Medical_Person ← Medical_Employee $\bowtie_{PersonID = EmployeeID}$ Person
   iii. Dr_Smillow ← $\sigma_{Lname = "Smillow"}$ Medical_Person
   iv. Smillow_LicenseNo ← Dr_Smillow * Licensure_Procedure
   v. **Smillow_Procedures ← Smillow_LicenseNo * Procedure**
   vi. Name_Date ← $\Pi_{ProcedureID, ProcedureType, ProcedureDate}$(Procedure)
   vii. **Result ← Smillow_Procedures * Name_Date**

d. Print out a list of due invoices with patient contact info.
   i. Past_Due ← $\sigma_{(Amount > 10)\ AND\ (TODAY\ >= BillingDate + 30)}$ Invoice

ii. PastDue_Invoices ← Invoice * Past_Due

iii. Patient_Info ← $\pi_{PhoneNumber}$(Patient $\bowtie_{PatientID = PersonID}$ Person)

iv. **Result ← PastDue_Invoices $\bowtie_{PersonID = PatientID}$ Patient_Info**

e. Find the patients who brought the most revenue in the past year

i. Patient_Person ← Patient $\bowtie_{PatientID = PersonID}$ Person

ii. Invoices ← Patient_Person * Invoice

iii. Current_Invoices ← $\sigma_{BillingDate >= '2022\text{-}01\text{-}01' AND BillingDate < '2023\text{-}01\text{-}01'}$(Invoices)

iv. **Current_Invoice_Sums ← $_{(Fname, Lname)}\mathcal{F}_{SUM\ Amount}$(Current_Invoices)**

v. *SQL features can now be used to sort the top X number of patients in Current_Invoice_Sums.*

f. Create a list of doctors who performed less than 5 procedures this year.

i. Medical_Employee ← $\sigma_{MedicalFlag = \text{“True”}}$(Employee)

ii. Medical_Person ← Medical_Employee $\bowtie_{EmployeeID = PersonID}$ Person

iii. LicenseNos ← Medical_Person * Licensure_Procedure

iv. Procedures ← LicenseNos * Procedure

v. Current_Procedures ← $\sigma_{ProcedureDate > '2023\text{-}01\text{-}01'}$(Procedures)

vi. Current_Procedures_Count ← $_{(Fname, Lname)}\mathcal{F}_{COUNT\ ProcedureID}$(Current_Procedures)

vii. **Result ← $\pi_{Fname, Lname}(\sigma_{Count\_procedureid < 5}$(Current_Procedures_Count))**

g. Find the highest paying procedures, procedure price, and the total number of those procedures performed.

i. Procedure_Info ← $\pi_{ProcedureID, ProcedureType, UnitCharge}$ (Procedure)

ii. Procedure_Count ← $_{ProcedureType}\mathcal{F}_{COUNT\ ProcedureID}$ (Procedure_Info)

iii. Procedure_Price ← $\pi_{ProcedureType, UnitCharge, COUNT\_ProcedureID}$(Procedure_Count * Procedure_Info)

iv. *SQL features can now be used to sort the top X highest paying procedures in Procedure_Price.*

h. Create a list of all payment types accepted, number of times each of them was used, and total amount charged to that type of payment.

i. Payment_Type_Amount ← $\pi_{PaymentType, Amount}$(Payment)

ii. Sum_Count ← $_{PaymentType}\mathcal{F}_{COUNT\ PaymentType, SUM\ Amount}$ (Payment_Type_Amount)

iii. **Result ← $\pi_{PaymentType, COUNT\ Amount, SUM\ Amount}$(Sum_Count)**

i. List ids and names of insurance plans ever used by patients and how many patients have that plan.

i. Plan_Description ← $\rho_{PlanID, PlanName, ActiveDate, MemberID, PatientID}$(InsurancePlan)

ii. Id_Name ← $\pi_{PlanID, PlanName}$(Plan_Description)

iii. Id_PlanName_Count ← $_{\text{PlanID, PlanName}}\mathcal{F}_{\text{COUNT PlanName}}$(Id_Name)

iv. **Result ← $\pi_{\text{PlanID, PlanName, COUNT PlanName}}$(Id_PlanName_Count)**

Additional Interesting Queries

a. Outerjoins: List all the employee names, and if they have a license, display their license number and expiration date. List all employee names regardless if they have a license.

i. Employee_Person ← π(EmployeeID, FName, LName)(Person $\bowtie_{\text{PersonID = EmployeeID}}$ Employee)

ii. Employee_License ← (Employee_Person $⟕_{\text{(EmployeeID=MedicalID)}}$ Licensure)

iii. **EmployeeInfo_License ← π(FName, LName, LicenseNumber, ExpirationDate)(Employee_License)**

b. Aggregate Functions: Find the total number of allergies grouped by patient.

i. Patient_Person ← (Person $\bowtie_{\text{PersonID = PatientID}}$ Patient)

ii. **Result ← $_{\text{PatientID}}\mathcal{F}_{\text{SUM(COUNT AllergyID)}}$ Patient_Person $\bowtie_{\text{PersonID = PatientID}}$ Patient_Allergy**

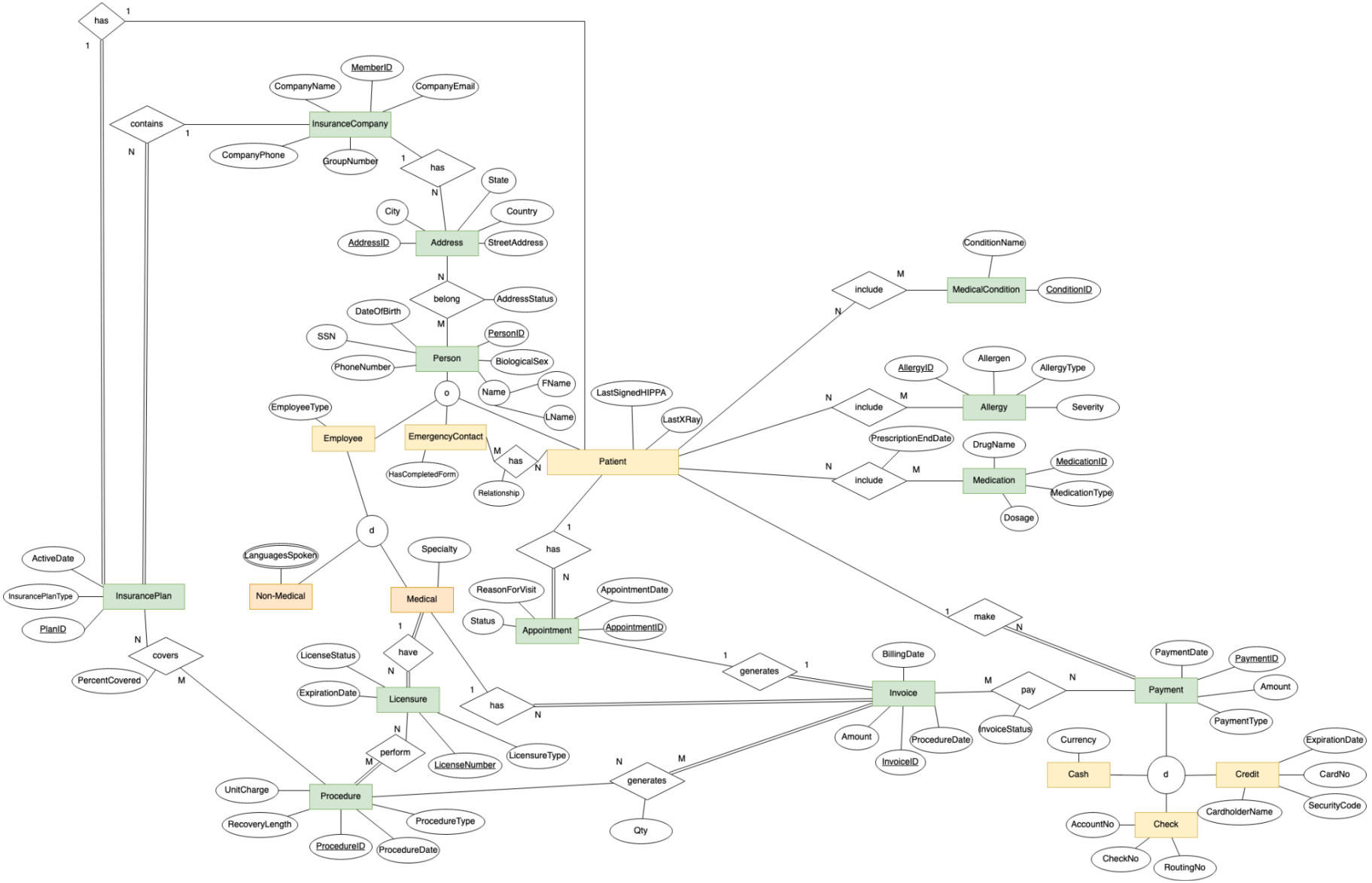
c. Extra Entities from PART 1: List the names of each person and the state of their addresses where the address is valid.

i. P_A ← $\pi_{\text{(FName, LName, AddressID)}}$(Person * Person_Address)

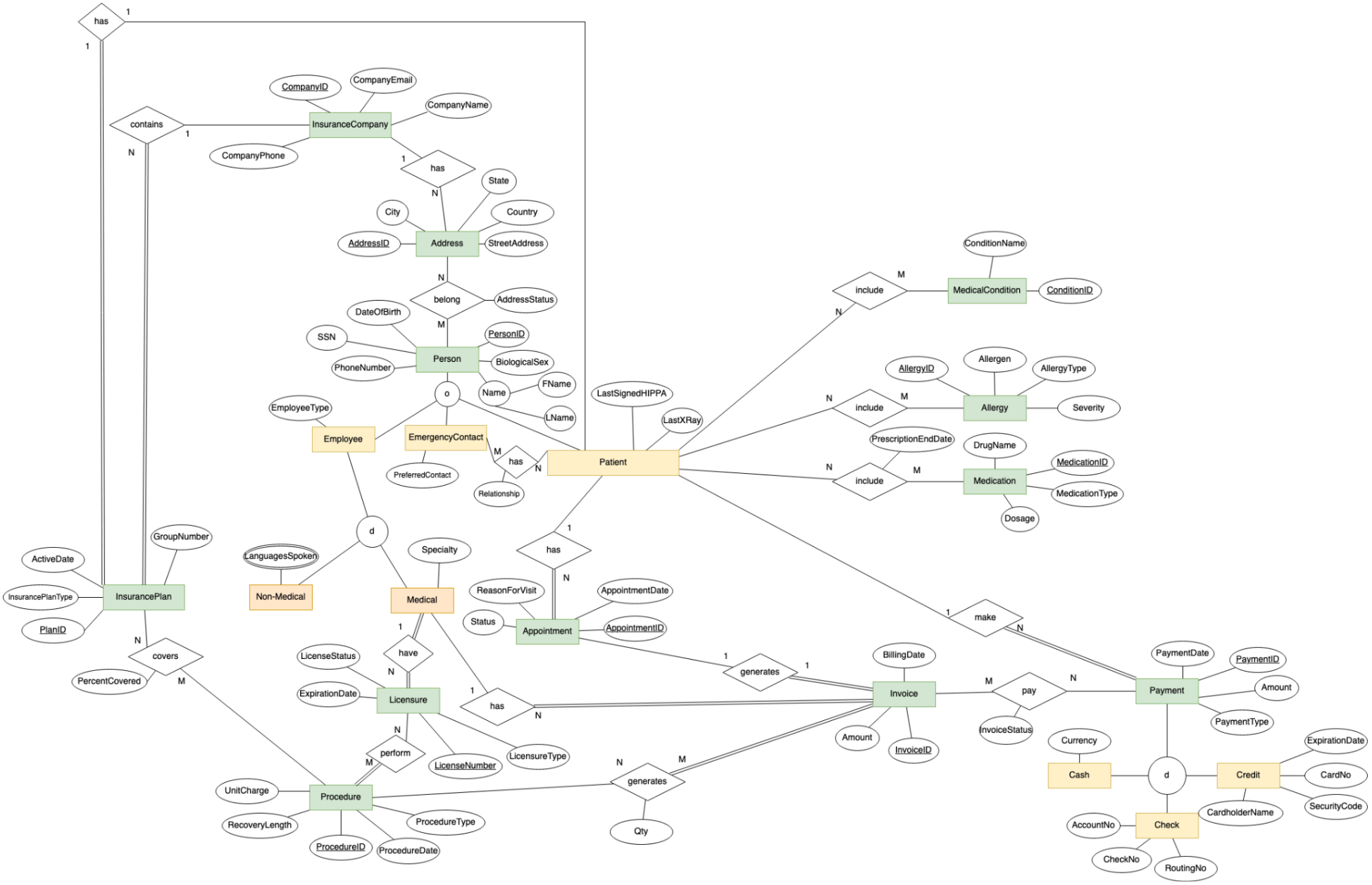
ii. PersonState ← $\pi_{\text{(FName, LName, State, AddressStatus)}}$(P_A * Address)

iii. **Result ← $\pi_{\text{(FName, LName, State)}}$ ($\sigma_{\text{AddressStatus = 'valid'}}$(PersonState))**

Previous EERD:



Part 2 Current ERD and Final Relational Schema



Final Schema

Person (PersonID, FName, LName, SSN, DateOfBirth, BiologicalSex, PhoneNumber)
InsuranceCompany (CompanyID, CompanyName, CompanyEmail, CompanyPhone)
Zip(ZipCode (PK), City, State)
Address (AddressID, StreetAddress, ZipCode (FK), CompanyID (FK))
Patient(PatientID (FK), LastSignedHIPPA, LastXRay)
InsurancePlan (PlanID, InsurancePlanType, ActiveDate, GroupNumber, CompanyID (FK), PatientID (FK))
Employee (EmployeeID (FK), MedicalFlag, Specialty)
Payment (PaymentID, PaymentDate, Amount, PaymentType, Currency, PatientID (FK))
Licensure (LicenseNumber, LicenseType, ExpirationDate, LicenseStatus, MedicalID (FK))
Appointment (ApptID, ApptDate, Status, ReasonForVisit, PatientID (FK))
Invoice(InvoiceID (PK), BillingDate, Amount, ApptID (FK), MedicalID (FK))
CheckingAccount(PaymentID (FK), AccountNo, CheckNo)
CheckInfo(AccountNo (FK), RoutingNo)
CreditAccount(PaymentID (FK), CardNo)
CreditCardInfo(CardNo (FK), CardholderName, SecurityCode, ExpirationDate)
NonMedical_Languages (EmployeeID (FK), LanguagesSpoken)
InsurancePlan_Procedure (PlanID (FK), ProcedureID (FK), PercentCovered)
Licensure_Procedure (LicenseNumber (FK), ProcedureID (FK))
Procedure_Invoice (ProcedureID (FK), InvoiceID (FK), Qty)
Invoice_Payment (InvoiceID (FK), PaymentID (FK), InvoiceStatus)
Patient_Medication (PatientID (FK), MedicationID (FK), PrescripEnd, PrescripStart)
Patient_Allergy (PatientID (FK), AllergyID (FK))
Patient_MedicalCondition (PatientID (FK), ConditionID (FK), DiagnosisDate)
Person_Address (AddressID (FK), PersonID (FK), AddressStatus)
EmergencyContact_Patient (EmergencyPersonID (FK), Relationship, PatientID (FK), ModeOfContact)
Procedure (ProcedureID, UnitCharge, RecoveryLength, ProcedureType, ProcedureDate)
Allergy (AllergyID, Allergen, AllergyType, Severity)
Medication (MedicationID, DrugName MedicationType, Dosage)
MedicalCondition (ConditionID, ConditionName)

Part 3: Verified Scripts

Updated scripts from Part 3 are attached (CreateQueries.txt, InsertQueries.txt, SimpleQueries.txt, ExtraQueries.txt). All of these files were created using SQLite.

Part 4: INSERT and DELETE Scripts

The attached file, “InsertDeleteQueries.txt,” has examples of inserting and deleting data. This file was created using SQLite. Below are the results after running the INSERT and DELETE portions of the script, respectively. This code affects 3 tables.

Results

Figure 4.1 Person Table Before and After

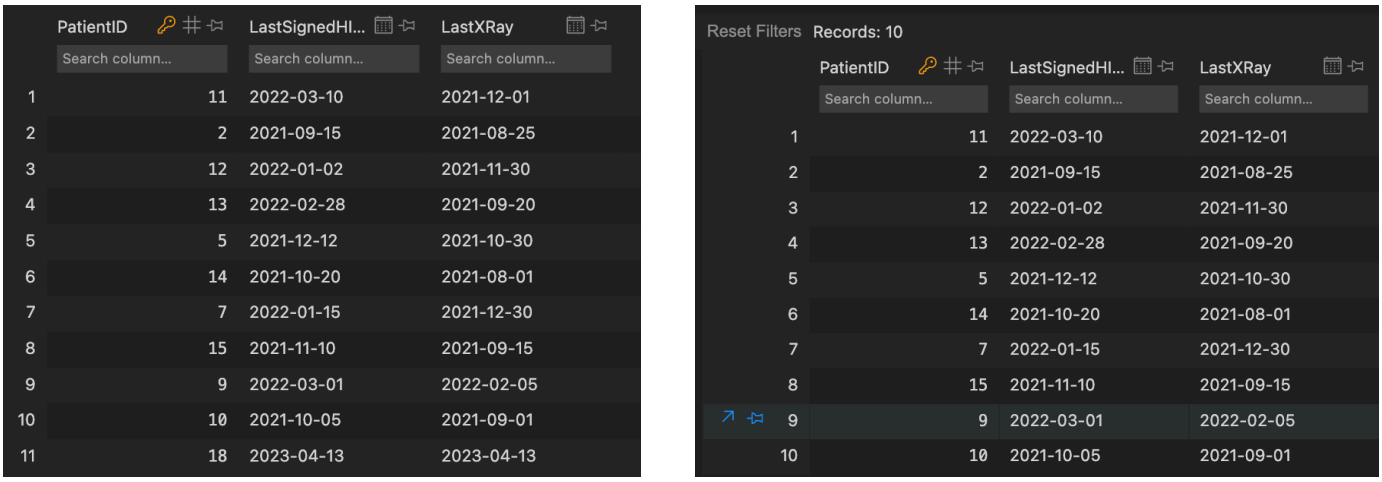
Reset Filters Records: 18

	PersonID	Fname	Lname	SSN	DateOfBirth	BiologicalSex	PhoneNumber
1	1	Anna	Smilow	123456789	2003-12-01	F	9999900000
2	2	John	Smith	999888777	2003-02-11	M	9899900000
3	3	Ronda	Rogers	888777666	1999-12-01	F	9989900000
4	4	Roger	Rogers	133456789	1999-03-12	M	9998900000
5	5	Nicole	Mendez	288072001	2000-01-21	F	9999999999
6	6	Nick	Mendez	334245211	2013-11-21	M	8888884444
7	7	Carla	Adams	323232323	1998-08-15	M	5555555555
8	8	Carl	Adams	424242424	1997-05-11	M	9999980000
9	9	Jane	Doe	222222222	1997-12-01	M	9999900009
10	10	John	Smith	123456789	1997-09-19	M	9999900008
11	11	Emily	Chavez	892134222	1998-10-12	F	6141231234
12	12	Hailey	Brown	892134223	1993-07-12	F	6161231234
13	13	Justin	Bieber	700134222	1998-06-12	M	6141231235
14	14	Ron	Weasley	482134222	1991-10-01	M	6141231236
15	15	Ariana	Grande	182134222	1995-10-12	F	6141231237
16	16	Harry	Potter	333666999	1990-08-12	M	6145991205
17	17	Ginny	Weasley	333666999	1990-03-12	F	6144772603
18	18	Lily	Potter	333666999	2003-08-20	F	6142701186

Reset Filters Records: 15

	PersonID	Fname	Lname	SSN	DateOfBirth	BiologicalSex	PhoneNumber
1	1	Anna	Smilow	123456789	2003-12-01	F	9999900000
2	2	John	Smith	999888777	2003-02-11	M	9899900000
3	3	Ronda	Rogers	888777666	1999-12-01	F	9989900000
4	4	Roger	Rogers	133456789	1999-03-12	M	9998900000
5	5	Nicole	Mendez	288072001	2000-01-21	F	9999999999
6	6	Nick	Mendez	334245211	2013-11-21	M	8888884444
7	7	Carla	Adams	323232323	1998-08-15	M	5555555555
8	8	Carl	Adams	424242424	1997-05-11	M	9999980000
9	9	Jane	Doe	222222222	1997-12-01	M	9999900009
10	10	John	Smith	123456789	1997-09-19	M	9999900008
11	11	Emily	Chavez	892134222	1998-10-12	F	6141231234
12	12	Hailey	Brown	892134223	1993-07-12	F	6161231234
13	13	Justin	Bieber	700134222	1998-06-12	M	6141231235
14	14	Ron	Weasley	482134222	1991-10-01	M	6141231236
15	15	Ariana	Grande	182134222	1995-10-12	F	6141231237

Figure 4.2 Patient Table Before and After

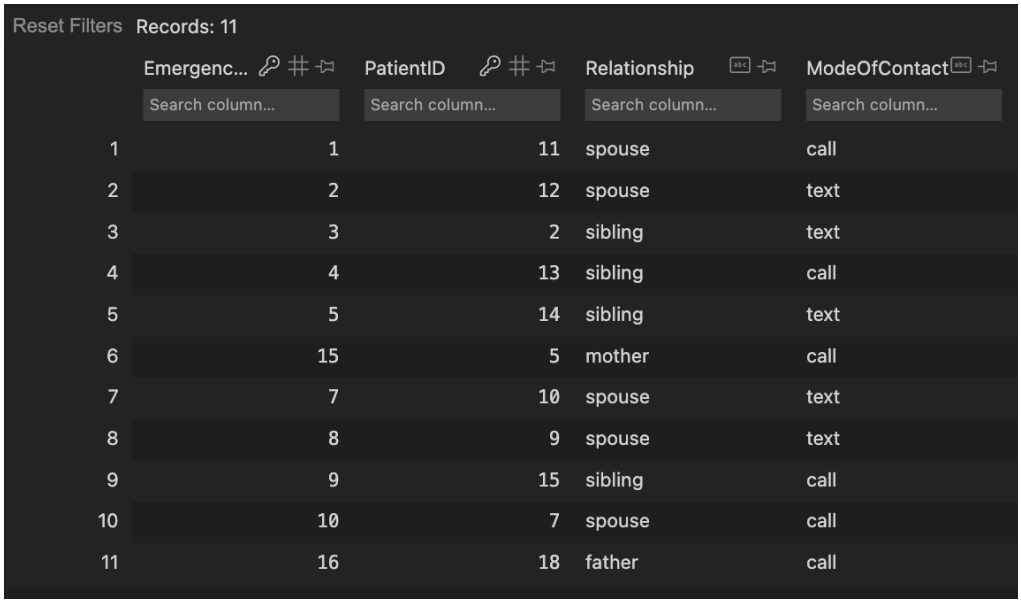


	PatientID	LastSignedHI...	LastXRay
	Search column...	Search column...	Search column...
1	11	2022-03-10	2021-12-01
2	2	2021-09-15	2021-08-25
3	12	2022-01-02	2021-11-30
4	13	2022-02-28	2021-09-20
5	5	2021-12-12	2021-10-30
6	14	2021-10-20	2021-08-01
7	7	2022-01-15	2021-12-30
8	15	2021-11-10	2021-09-15
9	9	2022-03-01	2022-02-05
10	10	2021-10-05	2021-09-01
11	18	2023-04-13	2023-04-13

Reset Filters Records: 10

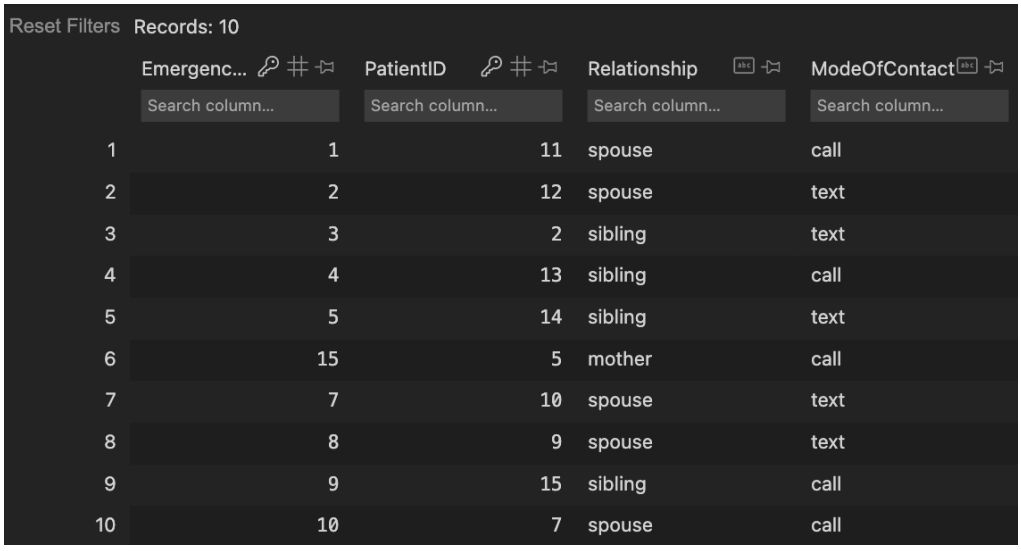
	PatientID	LastSignedHI...	LastXRay
	Search column...	Search column...	Search column...
1	11	2022-03-10	2021-12-01
2	2	2021-09-15	2021-08-25
3	12	2022-01-02	2021-11-30
4	13	2022-02-28	2021-09-20
5	5	2021-12-12	2021-10-30
6	14	2021-10-20	2021-08-01
7	7	2022-01-15	2021-12-30
8	15	2021-11-10	2021-09-15
9	9	2022-03-01	2022-02-05
10	10	2021-10-05	2021-09-01

Figure 4.3 Emergency Contact Table Before and After



Reset Filters Records: 11

	Emergenc...	PatientID	Relationship	ModeOfContact
	Search column...	Search column...	Search column...	Search column...
1	1	11	spouse	call
2	2	12	spouse	text
3	3	2	sibling	text
4	4	13	sibling	call
5	5	14	sibling	text
6	15	5	mother	call
7	7	10	spouse	text
8	8	9	spouse	text
9	9	15	sibling	call
10	10	7	spouse	call
11	16	18	father	call



Reset Filters Records: 10

	Emergenc...	PatientID	Relationship	ModeOfContact
	Search column...	Search column...	Search column...	Search column...
1	1	11	spouse	call
2	2	12	spouse	text
3	3	2	sibling	text
4	4	13	sibling	call
5	5	14	sibling	text
6	15	5	mother	call
7	7	10	spouse	text
8	8	9	spouse	text
9	9	15	sibling	call
10	10	7	spouse	call

Part 5: Indexes

Implementing a Hash-based index on the `InsurancePlanType` column of the `InsurancePlan` table can improve query performance for this table. When dealing with costs, we frequently need to know the type of insurance plan that a patient has. Therefore, it is crucial to create a hash index for quick retrieval. Hash indexes are particularly effective for columns that are frequently searched using equality tests, such as the InsurancePlanType column. This type of index is especially useful for the InsurancePlan table because the number of distinct insurance plan types is neither too large nor too small, allowing for roughly even distribution across the hash buckets. By using the Hash-based index, queries for a specific InsurancePlanType value will only need to search one hash bucket, rather than doing a full table scan on the entire InsurancePlan table.

Code:

```
CREATE INDEX HashIndex_InsurancePlanType
ON InsurancePlan (InsurancePlanType)
USING HASH;
```

Implementing a Tree-based index on the 'UnitCharge' and 'ProcedureType' columns of the 'Procedure' table can improve query performance for this table. When it comes to procedures, it's important to know the type and cost of the procedure. Therefore, it is crucial to create a tree index for quick retrieval. Tree indexes are particularly effective for columns that are frequently searched using range tests, such as the 'UnitCharge' column. This type of index is especially useful because not only can we create a tree in the order of unit charges, but we can also order the tree by the procedure type. By creating the tree with nodes in order of the unit charge, and then by procedure type, we will be able to create a more balanced tree that will in turn have fast searches compared to if the tree was ordered by unit charge alone. The benefit of using a tree index rather than just a table scan is that the worst-case to search the tree is O(log n), meaning as the tree grows, the time it takes to find an element will increase, but not as quickly as the size of the tree itself (assuming that the tree is balanced).

Code:

```
CREATE INDEX TreeIndex_UnitCharge_ProcedureType
ON Procedure (InsurancePlanType, UnitCharge)
USING BTREE;
```

Part 6: Views

The first view is the dentist's personal information and the number of appointments they have performed. This view may provide crucial information to payroll clerks when distributing payroll to dentists across Smilow. To illustrate, dentists may earn commission for meeting certain quotas in appointment bookings. Additionally, total appointments may serve as a way to measure dentist performance for end-of-year employee analyses, general business evaluation, etc. SQL code is below.

```
-- Create a view including personal information regarding dentists,
- -- along with the number of appointments they've performed
CREATE VIEW DentistAppointments AS
  SELECT Person.Fname, Person.Lname, Person.DateOfBirth, Person.SSN
  Employee.Specialty, Person.PersonID, COUNT(Appointment.MedicalID)
  AS Appointments_Performed
  FROM Person, Employee, Appointment
  WHERE Person.PersonID = Employee.EmployeeID
  AND Employee.EmployeeID = Appointment.MedicalID
  AND Employee.Specialty = 'Dentist'
  GROUP BY Employee.EmployeeID;
```

The second view creates a view of how much sales were brought in by different payment streams and if it has been received. This is useful information for accurate accounting, identifying trends, analyzing profitability, and managing cash flow. By tracking sales, Smilow Dentistry can accurately account for their revenue and this is useful for financial reporting and tax purposes. Also, tracking if it has been received is good for managing cash flow. By knowing which payments have been received and which ones are still outstanding can help Smilow Dentistry plan for expenses.

```
-- Create a view of how much of sales were brought in by different
- -- payment streams and determine if total has been received.

CREATE VIEW SalesStatus AS
   SELECT SUM(Payment.Amount) AS Sum_Payments, Payment.PaymentType,
   Invoice_Payment.InvoiceStatus AS Is_Total_Recieved
   FROM Payment, Invoice_Payment
   WHERE Payment.PaymentID = Invoice_Payment.PaymentID
   AND Invoice_Payment.InvoiceStatus = 'paid'
   GROUP BY Payment.PaymentType;
```

Part 7: Transactions

The first transaction adds a payment for a specific pending invoice. It must be executed as a single unit of processing so that it can check for pending invoices and execute writes accordingly at the same time.

```
BEGIN TRANSACTION NEW_PAYMENT;

SELECT COUNT(*) AS PendingPayments
FROM Invoice_Payment
WHERE Invoice_Payment.InvoiceID = 1001
AND Invoice_Payment.InvoiceStatus = 'pending';

IF PendingPayments > 0
  BEGIN
    INSERT OR ROLLBACK INTO Payment VALUES
     (11, '2023-01-01', 50.00, 'Credit Card', 'USD', 1);

     UPDATE Invoice_Payment
     SET Invoice_Payment.InvoiceStatus = 'paid'
     WHERE Invoice_Payment.InvoiceID = 1001;
   END

COMMIT;
END TRANSACTION;
```

The second transaction adds a new medication for a specific patient, as long as the patient is not allergic to it. Checking for allergies and simultaneously adding the medication to the DB makes the transaction necessary to be a single unit of processing.

```
BEGIN TRANSACTION NEW_MEDICATION

SELECT COUNT(*) AS PenicillinAllergies
FROM Allergy, Patient_Allergy
WHERE Allergy.Allergen = 'Penicillin'
AND Allergy.AllergyID = Patient_Allergy.AllergyID
AND Patient_Allergy.PatientID = 1;

IF PenicillinAllergies > 0
  BEGIN
    INSERT OR ROLLBACK INTO Medication VALUES
    (11, 'Penicillin', 'Oral', 1);

     INSERT OR ROLLBACK INTO Patient_Medication VALUES
     (1, 11, '2022-03-09', '2022-04-09');
   END

COMMIT;
END TRANSACTION;
```

Part 8: Team Member Contributions

Christina	Karl	Anushka	Sam	All
Primary Focus: InsertandDelete.txt Created an SQL script that Inserts and Deletes 5 instances of data, affecting 3 tables. This is an example on how to insert and delete queries following referential constraints.	*Primary Focus: Indexing* Created two indexes, one hash and one tree index, while explaining the importance of each index and how they will speed up the program for future queries.	*Primary Focus: CreateViews.txt* Created two separate views focusing on different internal parts of the Smillow dental practice. This included writing working SQL code, along with descriptions of view purposes.	*Primary Focus: Transactions* Created and documented two sample transactions for our database, which included writing SQL code for each and documenting their importance and function.	All team members contributed significantly in all parts of the project. We have two meetings a week to discuss progress, goals, and tasks. Work is divided evenly, and we regularly check each other's work. Communication has been consistent over the semester.

Overall Recommendation: We recommend future teams to establish a routine, and establish your own deadlines for each task in a project. By dividing the project into smaller tasks, the work feels more manageable. Also, communication is really important.