CSE 3241 Project Part 4

Team 7: Jason Fong, Vivian Comer,

Macray Curran, Cameron Erdman

***Code is for SQLiteOnline***

**ERD, Relation schema/Algebra updates**

**ERD/Schema Changes:**

We changed “Allergies” to a strong entity and added the PK “AllergyID” to it. The relationship for “Medication” and “Allergies” both changed from being connected to “MedicalHistory” to “Patients”. “Medication” was also given the PK “MedicationID” We also added the weak entity “Address”. “Address” has a relationship with “People” and has 4 attributes.

The weak entity “Treatment/Procedure” was changed to the strong entity “Treatments”. The PK “TreatmentID” was added to it and the attribute “ProcedureAmount” was changed to “TreatmentAmount”.

FirstName and LastName were taken out of Patients and Employees as attributes.

PatientPayments got the PK “PaymentID”. “PaymentID” became an FK in “Card” and “Money\_Check”. We got rid of InvoiceID being an FK in “Card” and “Money\_Check”

Added “Covers” to the schema (We forgot to before but it existed in the sql code). Changes its name to “CoversInsurance”.

**Relational Algebra Changes:**

a)

Old:

PATIENT ⋈PatientID = PatientID MEDICATION

New:

π P.LastName, M.MedicationName, M.Dose, M.PrescribeDate (γ patientid,(σ p . id = m . patientid (ρ m medications × ρ p people) ) )

b)

Old:

PATIENT\_DELTA←(PATIENT ⋈PatientID =PatientID) (INSURANCE)

RESULT ← σ(InsuranceCompany = Delta Dental Insurance)(PATIENT\_DELTA)

New:

PATIENT\_DELTA←(PATIENT ⋈ID =PatientID) (INSURANCE)

RESULT ← σ(InsuranceCompany = Delta)(PATIENT\_DELTA)

c)

Old:

MEDICAL\_ID ←EMPLOYEES ⋈EmployeeID = EmployeeID (MEDICAL)

SMILLOW\_ID ←σ(LastName = Smillow) (MEDICAL\_ID)

APPOINTMENT\_SMILLOW←APPOINTMENTS⋈ProcedureDoctor = LastName (SMILLOW\_ID)

RESULT ← π(TreatmentType, AppointmentDate)(APPOINTMENT\_SMILLOW))

New:

MEDICAL\_ID ←EMPLOYEES ⋈ID = EmployeeID (MEDICAL)

SMILLOW\_ID ←σ(LastName = Smillow) (MEDICAL\_ID)

APPOINTMENT\_SMILLOW←APPOINTMENTS⋈ProcedureDoctor = LastName (SMILLOW\_ID)

RESULT ← π(TreatmentType, AppointmentDate)(APPOINTMENT\_SMILLOW))

d)

Old:

DUE\_LIST ← σ(RemainingBalance > 10 AND Unpaid > 30)(INVOICE)

PATIENT\_LIST ← PATIENT⋈(PatientID = PatientID)(DUE\_LIST)

RESULT ← π(PatientID, InvoiceID, InvoiceAmount)(PATIENT\_LIST)

New:

DUE\_LIST ← σ(InvoicedAmount > 10 AND DaysPastDue> 30)(INVOICE)

PATIENT\_LIST ← PATIENT⋈(ID = PatientID)(DUE\_LIST)

RESULT ← π(PatientID, InvoiceID, InvoiceAmount)(PATIENT\_LIST)

e)

Old:

RESULT ← PatientID F(SUM PatientPaymentAmount) (Payment)

New:

RESULT ← PatientID F(SUM PatientPaymentAmount) (Payment)

f)

Old:

MEDICAL\_ID ←EMPLOYEES ⋈EmployeeID = EmployeeID (MEDICAL)

APPOINTMENT\_MEDICALS←APPOINTMENTS ⟗ ProcedureDoctor = LastName (MEDICAL\_ID)

RESULT ← σ(AppointmentID\_Count < 5)(APPOINTMENT\_MEDICALS)

New:

MEDICAL\_ID ←EMPLOYEES ⋈ID = EmployeeID (MEDICAL)

APPOINTMENT\_MEDICALS←APPOINTMENTS ⟗ ProcedureDoctor = LastName (MEDICAL\_ID)

RESULT ← σ(AppointmentID\_Count < 5)(APPOINTMENT\_MEDICALS)

g)

Old:

TREATMENT\_NUM ←Invoice ⟗ProcedurePerformed = TreatmentType (Treatment/Procedure)

MAX ←TreatmentType FMAX ProcedureAmount , COUNT TreatmentType (TREATMENT\_NUM)

π (Treatment\_Name, Procedure\_price, Total\_num) (MAX)

New:

TREATMENT\_NUM ←Invoice ⟗ProcedurePerformed = TreatmentType (Treatments)

MAX ←TreatmentType FMAX TreatmentAmount , COUNT TreatmentType (TREATMENT\_NUM)

h)

Old:

LIST ←TypeFCOUNT PaymentDate, SUM PatientPaymentAmount(PatientPayments)

π (PaymentType, Num\_Usage, Total\_Amount) (LIST)

New:

LIST ←TypeFCOUNT PaymentType, SUM PatientPaymentAmount(PatientPayments)

π (PaymentType, Total\_Amount) (LIST)

i)

Old:

INSURANCE\_NUM ←Insurance ⟗AmountCovered = PatientPaymentAmount (PatientPayments)

SUM ←InsuranceCompany FCOUNT PolicyID, COUNT InsuranceCompany, COUNT PatientID(INSURANCE)

π (InsuranceID, InsuranceName, Total\_Patients) (SUM)

New:

INSURANCE\_NUM ←Insurance ⟗AmountCovered = PatientPaymentAmount (PatientPayments)

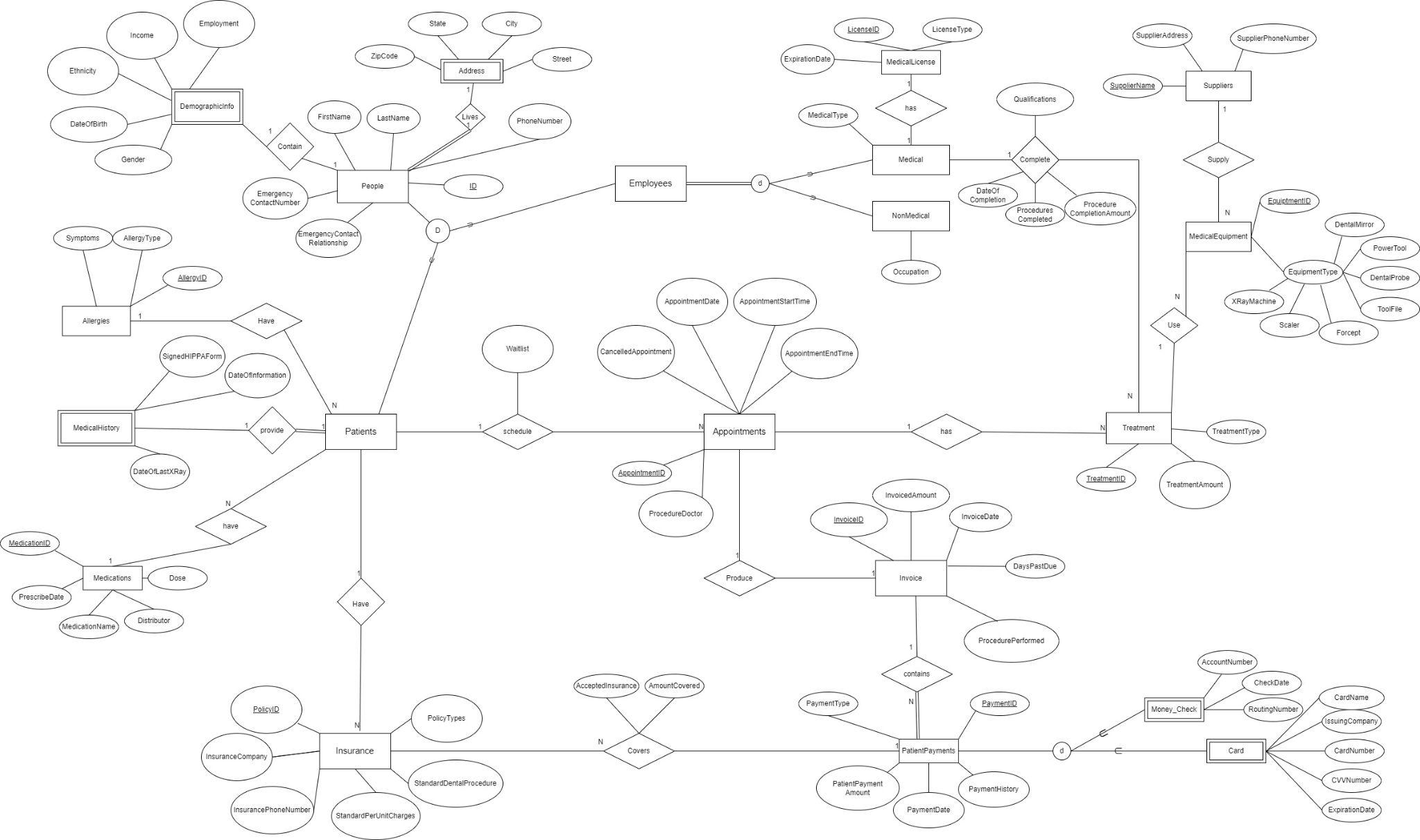
SUM ←InsuranceCompany FCOUNT PolicyID, COUNT InsuranceCompany, COUNT PatientID(INSURANCE)

Other Changes:

Because we changed the schema and ERD, everything else had to change slightly. For example out Create/Inserts changed slightly.

2)

**New ERD**

****

**New Schema**

People(ID(PK), FirstName, LastName, PhoneNumber, EmergencyContanctNumber, EmergencyContactRelationship)

Address(ID (FK - People), ZipCode, State, City, Street)

DemographicInfo(ID (FK - People), LastName, Employment,

Income, Ethnicity, DateOfBirth, Gender)

(FK) ID reference to People

Patients(PatientID (FK - People))

Employees(EmployeeID (FK - People))

MedicalHistory(PatientID (FK - Patients), SignedHIPPAForm, DateOfInformation,

DateOfLastXRay)

Allergies(AllergyID (PK)PatientID (FK - Patients), AllergyType, Symptoms)

Medications(MedicationID (PK), PatientID (FK - Patients), Dose, MedicationName, PrescribeDate, Distributor)

Insurance(PolicyID (PK), PatientID (FK - Patient), InsuranceCompany

InsurancePhoneNumber, StandardPerUnitCharges, StanardDentalProcedure,

PolicyTypes)

Appointments(AppointmentID (PK), PatientID (FK - Patients), CancelledAppointment,

AppointmentDate, AppointmentStartTime, AppointmentEndTime, ProcedureDoctor)

Invoice(InvoiceID (PK), InvoicedAmount, InvoiceDate, ProcedurePerformed, DaysPastDue)

PatientPayments(PaymentID (PK), InvoiceID (FK - Invoice) PolicyID(FK - Insurance)

PatientPaymentAmount, PaymentDate, PaymentHistory, PaymentType)

Treatment(“TreatmentID” (PK), AppointmentID (FK - Appointment), EquipmentID (FK - MedicalEquipment), ProcedureAmount)

Medical(EmployeeID (FK - Employees), LicenseID (FK - MedicalLicense), MedicalType)

CompletedProcedures(EmployeeID (FK - Employees),

Qualifications, DateOfCompletion, ProceduresCompleted,

ProcedureCompletionAmount)

NonMedical(EmployeeID (FK - Employees), Occupation)

MedicalLicense(LicenseID (PK), LicenseType, ExpirationDate)

MedicalEquipment(EquipmentID (PK), SupplierName (FK - Suppliers), XRayMachine, Scaler, Frocept, ToolFile, DentalProbe, PowerTool, DentalMirror)

Suppliers(SupplierName (PK), SupplierAddress, SupplierPhoneNumber)

Money\_Check(AccountNumber, RoutingNumber, CheckDate, PaymentID (FK - PatientPayments))

Card(IssuingCompany, CardNumber, CVVNumber, ExpirationDate, CardName, PaymentID (FK - PatientPayments))

CoversInsurance(PolicyID (FK- Insurance), AcceptedInsurance, AmountCovered)

**INSERT and DELETE SQL CODE**

3 Queries will be implemented, and each will affect 2 or more tables. Using SQLiteOnline does not update/delete the value using referential integrity. The first example was supposed to have values inserted along with 2 tables being referenced to each other. Supposedly deleting values from “People” will also delete from “Address”. All examples below should work on update/delete/insert cascade.

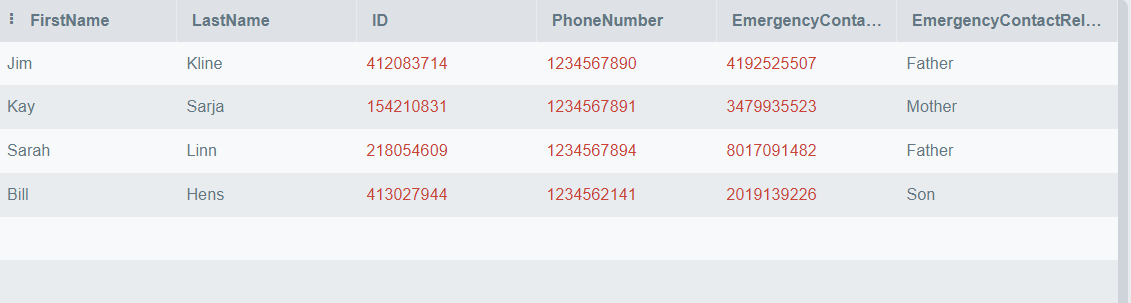
**1)People and Address**

Before SQL Inserts

(Table: People)



(Table: Address)



INSERT INTO People (ID, LastName, FirstName, PhoneNumber, EmergencyContactNumber, EmergencyContactRelationship)

VALUES (140782201, "Marsh", "Timland", 4213251455, 5075774006, "Sister");

INSERT INTO Address(ID, ZipCode, State, City, Street)

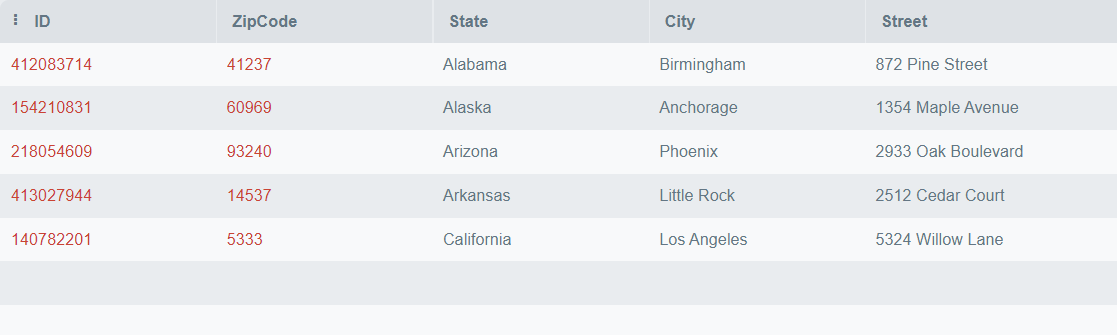
VALUES (140782201, 05333, 'California', 'Los Angeles', '5324 Willow Lane');

After SQL Insert

(Table: People)



(Table: Address)

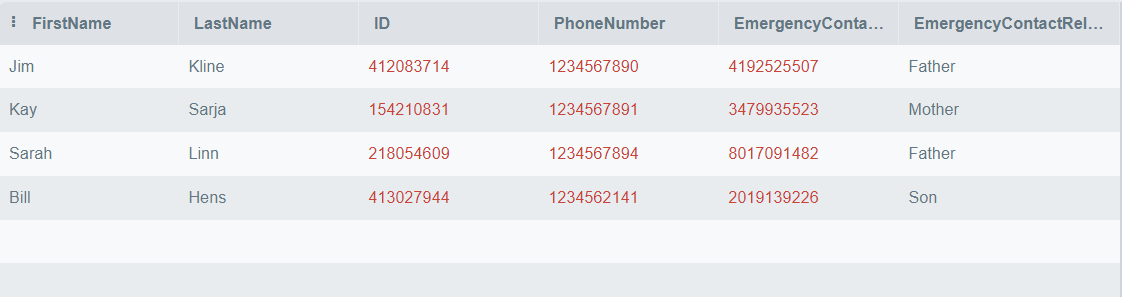


DELETE FROM People WHERE ID = 140782201

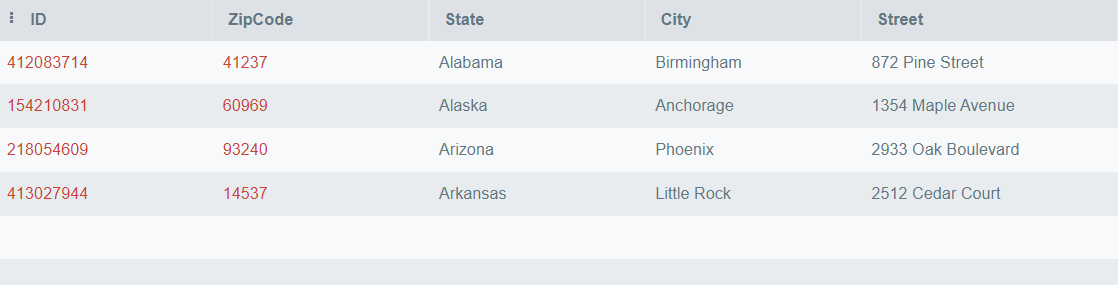
DELETE FROM Address WHERE ID = 140782201

After Delete:

(Table: People)



(Table: Address)



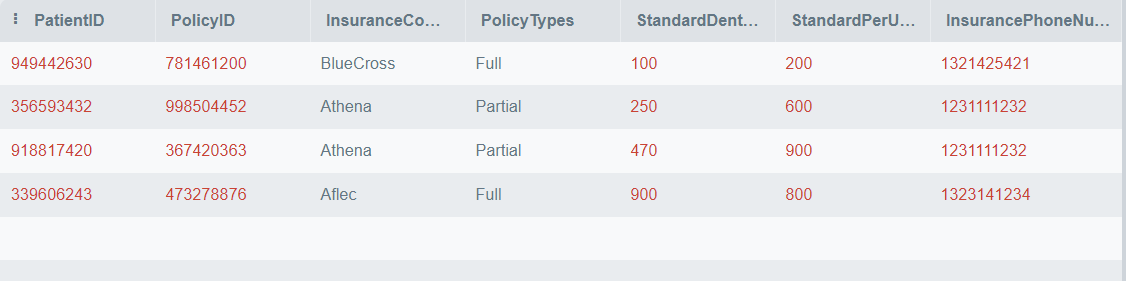
**2)Patients and Insurance**

Before Insert:

(Table: Patients)



(Table: Insurance)



INSERT INTO Patients (ID)

VALUES (381445177);

INSERT INTO Insurance(PolicyID, InsuranceCompany, PatientID, InsurancePhoneNumber, StandardPerUnitCharges, StandardDentalProcedure, PolicyTypes)

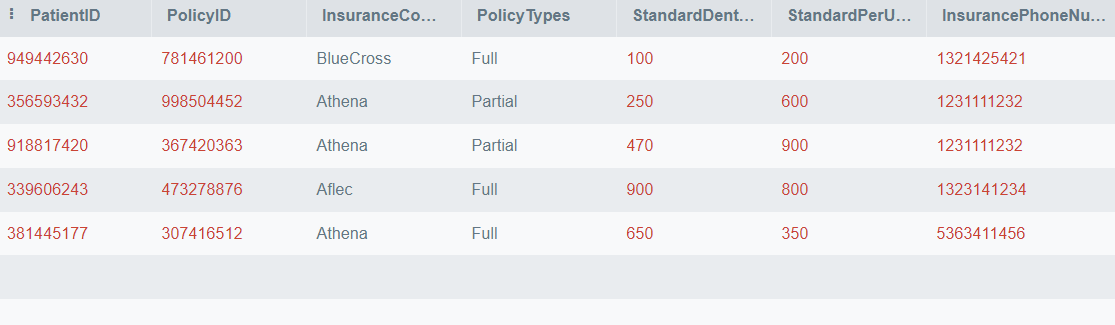
VALUES (307416512, "Athena", 381445177, 5363411456, 350, 650, "Full");

(After Insert)

(Table: Patients)



(Table: Insurance)



DELETE FROM Patients WHERE ID = 381445177;

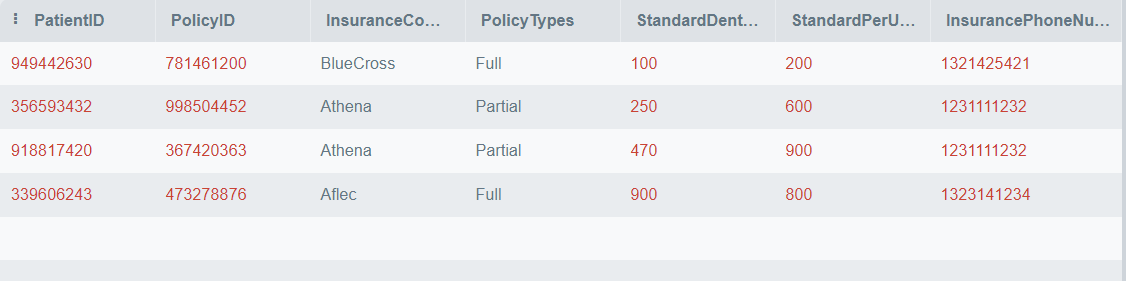
DELETE FROM Insurance WHERE PatientID = 381445177;

(After Delete)

(Table: Patients)

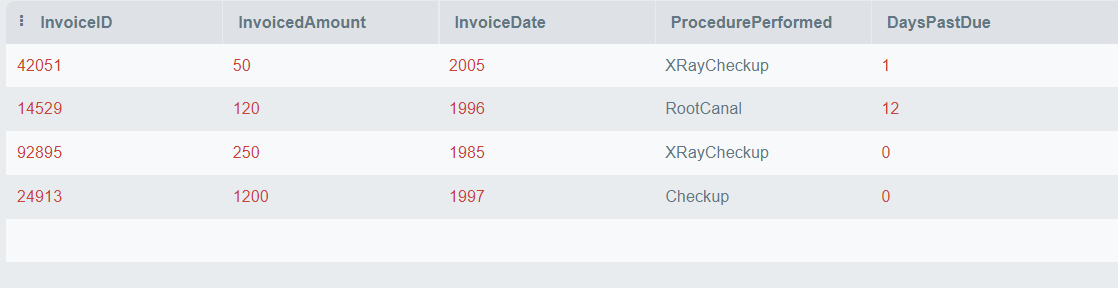


(Table: Insurance)



**3) Insurance and PatientPayments**

(Before Insert)

(Table: Insurance)  
  
(Table: PatientPayments)

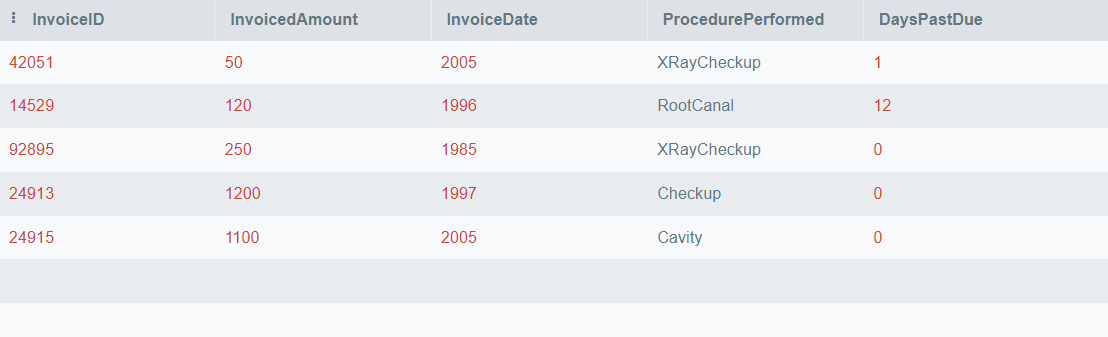
INSERT INTO Invoice(InvoiceID, InvoicedAmount, InvoiceDate, ProcedurePerformed, DaysPastDue)

VALUES (24915, 1100, 2022-09-08, "Cavity", 0.00);

INSERT INTO PatientPayments(PaymentID, InvoiceID, PolicyID, PaymentType, PatientPaymentAmount, PaymentDate, PaymentHistory)

VALUES (845321321, 24915, 168968600, "Check", 1100, 2022-09-08, "9/8/2022");

(After Insert)

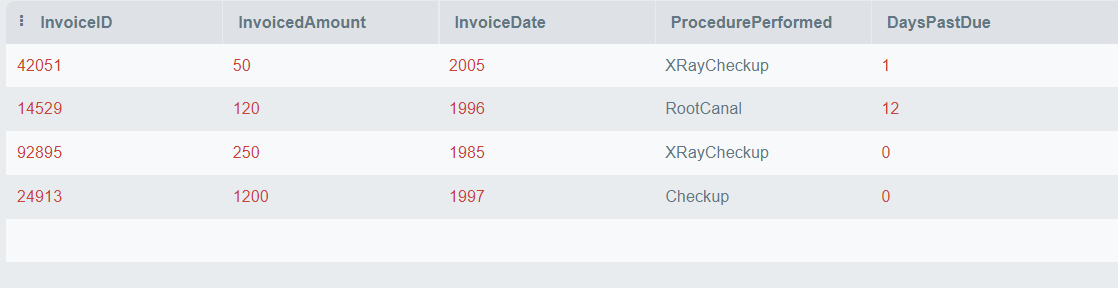
(Table: Insurance)  
  
(Table: PatientPayments)



DELETE FROM Invoice WHERE InvoiceID = 24915;

DELETE FROM PatientPayments WHERE InvoiceID = 24915;

(After Delete)

(Table: Insurance)  
  
(Table: PatientPayments)



**IMPLEMENT Indexing (Vivian)**

Idex 1: Index that alphabetically orders People data by the LastName followed by FirstName

* The purpose of this index is to order the names of customers alphabetically. This would allow for Queries and searches regarding employee or patient names to be completed quicker and more easily. It would also allow for the query results and views to be sorted in alphabetized lists of the names.
* This would be a B-Tree index because it works better for multiple columns and results in our data being in sorted order so all name results can be ordered.

CREATE INDEX index\_name\_sort

ON People (LastName, FirstName);

Index 2: Index that groups Medical Employee information by MedicalType

* The purpose of this index is to rearrange and group the Medical employees by their type. This would allow us to easily distinguish and return which employees do which type of role and could be useful in indicating which procedure they assist in and scheduling. For example: all Dentists would be grouped together and all dental technicians would be grouped together.
* This would be a clustering index because it rearranges the data in the tables to be grouped by like indices. There is also only one column and would be useful when looking up anything regarding the MedicalType of the employees.

CREATE INDEX index\_medicalType

ON Medical(MedicalType);

Index 3: Index Insurance policies based on whether they are accepted and the Amounts they cover

* The purpose of this index is to order the Insurance policies based on whether they are accepted and how much they cover. This would make it quicker to determine which insurances are accepted and to query/determine which policies cover the most costs.
* This would be a B-Tree index because it utilized two columns and would have the advantage of making range searches about the AmountCovered.

CREATE INDEX index\_insurance\_coverage

ON CoversInsurance (AcceptedInsurance, AmountCovered);

**IMPLEMENT Views (Macray)**

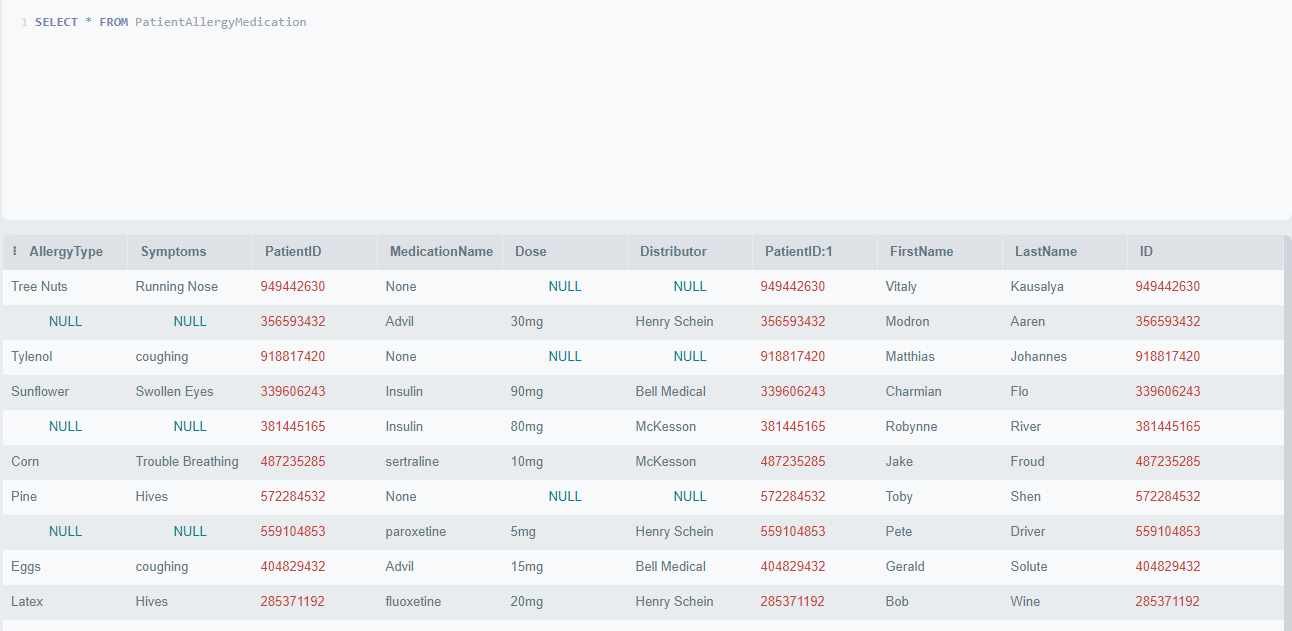
View 1: View where all patients allergies and medication are listed

CREATE VIEW [PatientAllergyMedication] AS

SELECT a.AllergyType, a.Symptoms, a.PatientID, m.MedicationName, m.Dose, m.Distributor, m.PatientID, p.FirstName, p.LastName, p.ID

FROM Allergies as a, Medications as m, People as p

WHERE p.ID = a.PatientID AND p.ID = m.PatientID AND m.PatientID = a.PatientID



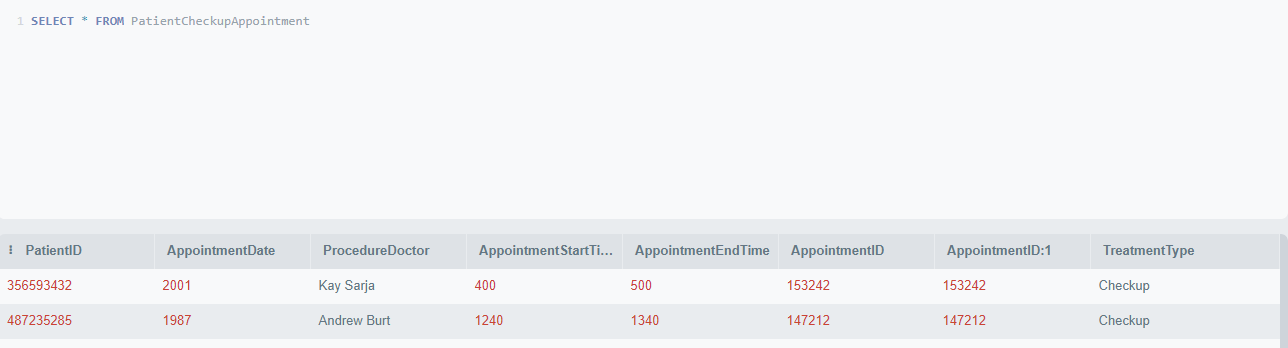
View 2: A View of all checkup appointments

CREATE VIEW [PatientCheckupAppointment] AS

SELECT p.patientID, p.AppointmentDate, p.ProcedureDoctor, p.AppointmentStartTime, p.AppointmentEndTime, p.AppointmentID, t.AppointmentID, t.TreatmentType

FROM Appointments as p, Treatments as t

WHERE t.AppointmentID = p.AppointmentID AND t.TreatmentType = "Checkup";



**IMPLEMENT Transactions (Cameron)**

List two sample transactions that you want to establish for your DB. Clearly document their purpose and function. Explain why it is crucial to execute those as one unit of processing. Each transaction should include read and at least 2 write operations on at least two tables, with appropriate error and constraint checks and responses. Provide valid SQL code for each transaction.

Neither of these have error checking because SQLite doesn’t support GO TO or IF THEN that was talked about in class and so I don’t know how to do error checking otherwise. SQLite will auto rollback on a handful of common error codes though. Because there is no IF or GO TO I also couldn’t figure out intuitive branching so the transactions are not as cool as they could be.

Update invoice and waitlist post appointment:

* Read appointment
  + if today > appointment date then update invoice
* Read Treatment/Procedure by AppointmentID and grab Procedure amount and Procedure Performed
* Write Invoice with new ID, amount from T/P, today's date, procedure from T/P, and set days past due to 0
* Update the waitlist to confirm the appointment that occurred is not marked as being on the waitlist

It is very important to update the invoice information after an appointment to guarantee that the dentistry has it in their system that an invoice has been sent out. Similarly it is important to confirm that an appointment is no longer on the waitlist so as to not cause any confusion with scheduling. These should happen at the same time as they both happen after an appointment and so grouping them reduces the need for multiple transactions.

SQL CODE:

BEGIN TRANSACTION UPDATE\_INVOICE;

SELECT appointmentid as update\_id FROM Appointments WHERE appointmentdate >= DATE('now') LIMIT 1;

INSERT INTO Invoice VALUES((SELECT (abs(random()) %100000)),

(SELECT treatmentamount FROM Treatments WHERE appointmentid ==

(SELECT appointmentid as update\_id FROM Appointments WHERE appointmentdate >= DATE('now') LIMIT 1)

LIMIT 1),

DATE('now'),

(SELECT treatmenttype FROM Treatments WHERE appointmentid ==

(SELECT appointmentid as update\_id FROM Appointments WHERE appointmentdate >= DATE('now') LIMIT 1)

LIMIT 1),

0);

UPDATE Schedule SET waitlist = 0 WHERE appointmentid ==

(SELECT appointmentid as update\_id FROM Appointments WHERE appointmentdate >= DATE('now') LIMIT 1);

END TRANSACTION;

Populate Patient Medication and Allergy Information:

Filling in allergies and medications tables with empty values for patients that don’t currently have entries in those tables so that those tables become populated and will pop up as empty values next time that the employees check for their information, this will remind the employees to ask the patients for their medical information allowing them to update and fill in the table. It is beneficial for them to happen at the same time as they are doing roughly the same task and will limit the number of transactions that need to occur.

SQL CODE:

BEGIN TRANSACTION populate\_patient\_info;

SELECT \* FROM Patients WHERE id != (SELECT patientid FROM Medications);

INSERT INTO Medications VALUES('',

2,

(SELECT \* FROM Patients WHERE id != (SELECT patientid FROM Medications) LIMIT 1),

'',

'',

'');

INSERT INTO Allergies VALUES((SELECT \* FROM Patients WHERE id != (SELECT id FROM Allergies) LIMIT 1),

'',

'',

'');

END TRANSACTION;

**Contributions**

a)

Macray Curran - Reworking ERD, Schema, and SQL code

Jason Fong - Extra inserts and deletes, algebra rework, ERD changes

Vivian Comer - Index, Contributions

Cameron Erdman - Transactions

b)

For future teams, being organized and having open communication with your team is very important, especially if the team has difficulty finding times to meet. This allows you to help each other and troubleshoot more effectively, as well as ensure that everyone contributes equally. Laying out expectations on the timeline and planning meetings early can ensure the work is organized and completed on time when everyone has busy schedules. We also recommend that future students make the most of the project work time in class and to ask for help from the instructor if they are stuck. Those are time and resources that are provided with the class and they should take advantage of them. To make the process more efficient we could have established these meeting times and expectations for work completion earlier on at the very beginning of part 1.