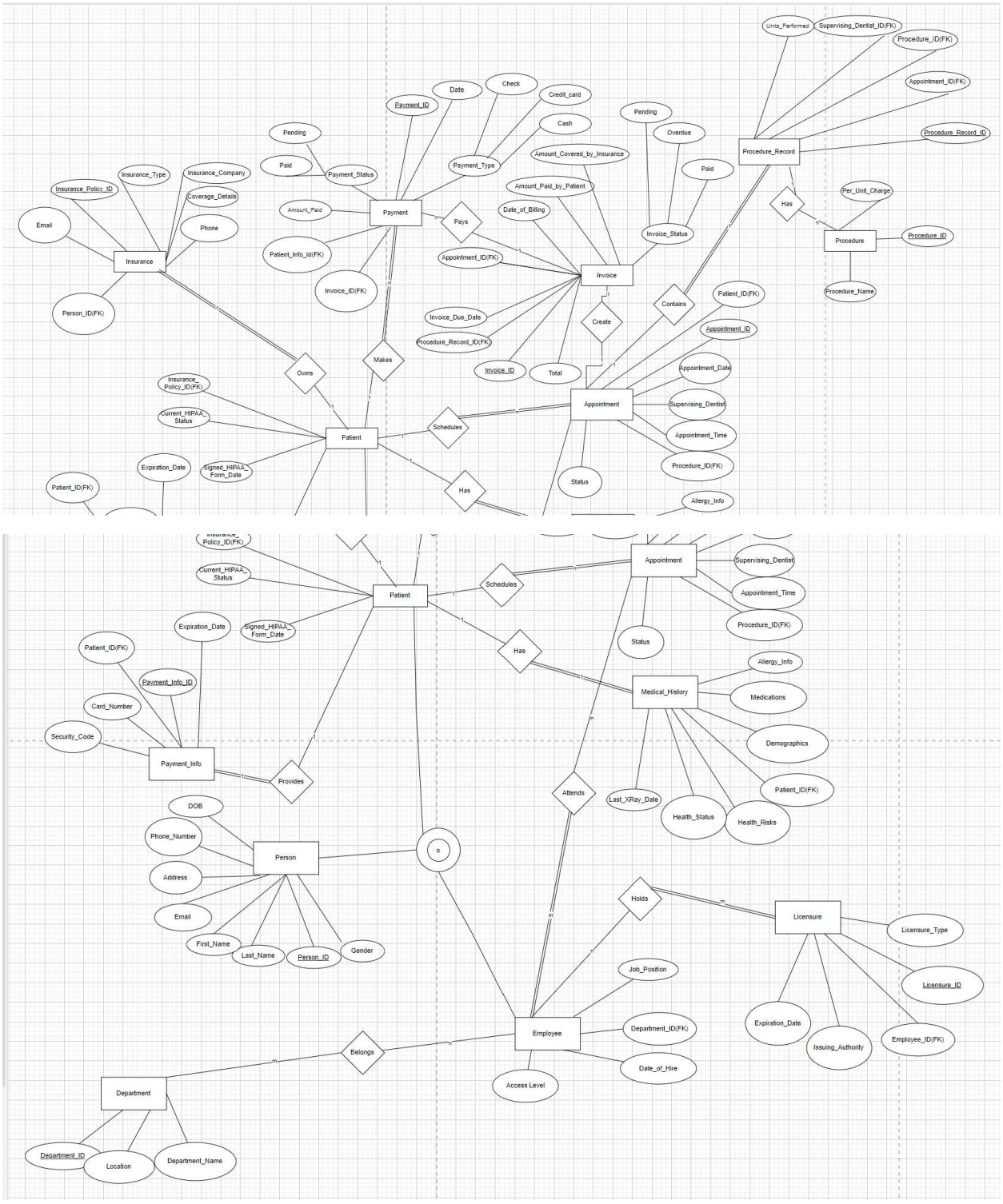
Cody Yang, Maverick Walker, Andy Vong, To Minh

There were no updates to our ERD. It remains the same.



We updated our relational schema to what’s below.

- INSURANCE(**Insurance_Policy_ID,** Person_ID, Insurance_Type, Insurance_Company, Coverage_Details, Phone, Email)
    - Person_ID is a foreign key that maps to PERSON
- PROCEDURE(**ProcedureID,** Per_Unit_Charge, Procedure_Name, Procedure_Record_ID(FK))
    - Procedure_Record_ID is a foreign key that maps to PROCEDURE_RECORD.
- PAYMENT_INFO(**Payment_Info_ID,** Card_Number, Security_Code, Person_ID(FK))
    - Person_ID is a foreign key that maps to PERSON
- PERSON(**PersonID**, First_Name, Last_Name, Email, Address, Phone, Date_of_Birth, Gender, Pflag, Signed_HIPAA_Form_Date, Current_HIPAA_Status, Eflag, Job_Position, Date_of_Hire, Access_level)
- LICENSURE (**License_ID,** Licensure_Type, Employee_ID (FK), Issuing_Authoriy, Expiration_Date)
    - Employee_ID is a foreign key that maps to EMPLOYEE
- PROCEDURE(**Procedure_ID**, Procedure_Name, Per_Unit_Charge)
- INSURANCE(**Insurance_Policy_ID**, Insurance_Type, Insurance_Company, Coverage_Details, Phone, Email, Person_ID(FK))
    - Person_ID related to Person
- INVOICE(**Invoice_ID**, Total, Amount_Covered_by_Insurance, Insurance_Type, Amount_Paid_by_Patient, Date_of_Billing, Amount_Due, Invoice_Status, Appointment_ID, Invoice_Due_Date, Procedure_Record_ID)
- Appointment_ID is a foreign key that maps to APPOINTMENT
- Procedure_Record_ID is a foreign key that maps to PROCEDURE_RECORD
- PAYMENT(**Payment_ID**, Amount_Paid, Date, Payment_Type, Payment_Status)
- PROCEDURE_RECORD(**Procedure_Record_ID,** Appointment_ID, Procedure_ID, Supervising_Dentist_ID(FK), Units_Performed)
- Appointment_ID is a foreign key that maps to APPOINTMENT
- Procedure_ID is a foreign key that maps to PROCEDURE
- Supervising_Dentist_ID is related to Person

APPOINTMENT(**Appointment_ID**, Appointment_Date, Supervising_Dentist(FK), Appointment_Time, Procedure_Record_ID(FK), Billing_Record(FK), Status)

Procedure_Record_ID is related to PROCEDURE_RECORD

Supervising_Dentist is a foreign key that maps to EMPLOYEE

MEDICAL_HISTORY(Allergy_Info, Medications, Demographics, Patient_ID(FK), Health_Risks, Health_Status, Last x_ray_date)

Patient_ID is related to Patient

DEPARTMENT(**Department_ID**, Location, Department_Name)

Our relational algebra is below also.

Create a list of patients and the medications they currently take

- Result ← $\pi_{\text{Patient\_ID, Medications}}$(MEDICAL_HISTORY)

Display patient information for patients who currently have Delta Dental insurance policy.

- $\pi_{\text{First\_Name,Last\_Name,Email,Address,Phone\_Number,DOB,Gender,Current\_HIPAA\_Status,Signed\_HIPAA\_Form\_Date}}(\sigma_{\text{Insurance\_Company='Delta Dental'}}(\text{INSURANCE}) \bowtie_{\text{Person\_ID=Person\_ID}} \text{PERSON})$

Generate a list of procedures and dates of service performed by doctor Smilow. (Andy)

- Smillow_Records <- $\sigma_{\text{Supervising\_Dentist = 1}}$(Procedure Record)
- Smillow_Procedures <- $\sigma_{\text{Procedure\_Record.Procedure\_Record\_ID = Procedure.Procedure\_Record\_ID}}$(Procedure Record X Procedure)
    - Procedures_with_Dates <- Smillow_Procedures $\bowtie_{\text{Smillow\_Procedures.Appointment\_ID=Appointment.Appointment\_ID}}$(Appointment)

Print out a list of past due invoices with patient contact information. Past due is defined as over 30 days old with a balance over $10.

- PastDueInvoices <- $\sigma_{\text{(CURRENT\_DATE−Date\_of\_Billing>30)} \cap \text{ Amount\_Due>10}}$(INVOICE) X APPOINTMENT
- InvoiceWithPatientInfo <- (PastDueInvoices) $X_{\text{ APPOINTMENT.Patient\_ID=PATIENT.Patient\_ID}}$ PATIENT
- GetContact <- (InvoiceWithPatientInfo) $X_{\text{ PATIENT.Person\_ID=PERSON.Person\_ID}}$ PERSON
- ListPastDue <- $\pi_{\text{PERSON.Email,PERSON.Phone,INVOICE.Invoice\_ID,INVOICE.Amount\_Due,INVOICE.Date\_of\_Billing}}$(GetContact)

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

- Result←$\pi_{\text{First\_Name,Last\_Name}}$(MaxRevenue ⋈ Patient_IDPATIENT)

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

$\pi_{\text{Supervising\_Dentist\_ID, First\_Name, Last\_Name, Total\_Procedures}}(\sigma_{\text{Total\_Procedures < 5}}(\gamma_{\text{Supervising\_Dentist\_ID, First\_Name, Last\_Name, COUNT(Procedure\_ID)} \rightarrow \text{ Total\_Procedures}}(\text{PROCEDURE\_RECORD} \bowtie_{\text{ Supervising\_Dentist\_ID = Person\_ID}} \text{PERSON})))$

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

- Highest Paying <- $_{Procedure\_Name}F_{COUNT\ Procedure\_Name,\ MAXPer\_Unit\_Charge}$(PROCEDURE)

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.

- GroupByPaymentType <- F $_{Payment\_Type,COUNT(Payment\_ID),SUM(Amount\_Paid)}$(PAYMENT)
- Result ← $\pi_{Payment\_Type,COUNT(Payment\_ID),SUM(Amount\_Paid)}$ (GroupByPaymentType)

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

- ProjectedData ← $\pi_{Insurance\_Policy\_ID,\ Insurance\_Type,\ Patient\_ID,\ First\_Name,\ Last\_Name}$(PatientInsurance)

We also created 3 separate new and unique interesting queries in both plain English and relational algebra.

For finding patients with the highest total cost of procedures over a year:

- Appointments_Past_Year <- $\sigma_{Appointment\_Date > Today - 1\ year}$(Appointment $\bowtie_{Person\_ID = Person.PersonID}$(Person))
- Appointments_With_Invoice <- Appointments_Past_Year $\bowtie_{Appointments\_Past\_Year.Appointment\_ID = Invoice.Appointment\_ID}$(Invoice)

For listing all patients along with their insurance company names:

- PatientWithPerson <- PATIENT X $_{PATIENT.Person_ID=PERSON.PersonID}$PERSON
- PatientInsurance <-PatientWithPerson X $_{PERSON.Insurance_Policy_ID=INSURANCE.Insurance_Policy_ID}$INSURANCE
- PatientInsuranceCompany <-$\pi_{PERSON.First\_Name,PERSON.Last\_Name,INSURANCE.Insurance\_Company}$(PatientInsurance)

Finally, total amount of money covered by insurance on certain procedures

- PROCEDURE_RECORD$\bowtie_{Procedure_Record_ID=INVOICE.Procedure_Record_ID}$INVOICE
- Total_Covered((PROCEDURE$\bowtie_{Procedure_ID=PROCEDURE_RECORD.Procedure_ID}$PROCEDURE_RECORD
- $\pi_{Procedure\_ID,Procedure\_Name,Total\_Covered}(\gamma_{Procedure\_ID,Procedure\_Name,sum(Amount\_Covered\_by\_Insurance)} \rightarrow Total\_Covered$ ((PROCEDURE$\bowtie_{Procedure_ID=PROCEDURE_RECORD.Procedure_ID}$PROCEDURE_RECORD) LEFT OUTER JOIN (PROCEDURE_RECORD$\bowtie_{Procedure_Record_ID=INVOICE.Procedure_Record_ID}$INVOICE)))

We also created two separate indexes, explained below.

these indexes are meant for SQLite, not SSMS

Index on patient flag:

The first index that we would like to implement is an index on patients. Given the setting in which the database will be utilized, they'll always be querying about patients. Because we are checking for equality (pflag = true), it would be best to use a B-tree index since a B-tree allows for multiple multiple children under a node, keeping it's height as small as possible. This will improve performance for queries for patient lookup.

SQL Code:

```
CREATE INDEX pflag_index

ON PERSON(pflag);
```

Index on Appointment_Date:

The second index that our team would like to add to the database is an index for appointment dates. Scheduling appointments at the office is an important aspect of a dental office. In doing so, people who are responsible for scheduling appointments will need to look up what appointments are scheduled on a certain date and time. A receptionist might need to select all appointment rows where the appointment date is 1 month from today. A B-tree index will also be beneficial for the same reasons as the index on the patient flag in the PATIENTS table.

SQL Code:

```
CREATE INDEX appointment_date_index

ON APPOINTMENT(appointment_date, appointment_time);
```

Below are the views we created to work within the database for important issues.

This view combines PERSON and INVOICE tables to identify patients with outstanding invoices. It includes patient contact information and the total outstanding amount.

```
CREATE VIEW Outstanding_Invoices

AS SELECT

    P.Person_ID,

    CONCAT(P.First_Name, ' ', P.Last_Name) AS Full_Name,

    P.Phone,

    P.Email,

    SUM(I.Amount_Due) AS Total_Amount_Outstanding
```

```
FROM
    PERSON P
JOIN
    INVOICE I ON P.Person_ID = I.Appointment_ID
WHERE
    I.Invoice_Status = 'Unpaid'
GROUP BY
    P.Person_ID, P.First_Name, P.Last_Name, P.Phone, P.Email
ORDER BY
    Total_Amount_Outstanding DESC;
```

Person_ID	Full_Name	Phone	Email	Total_Amount_Outsta...
2	Jane Smith	5552345678	jsmith@example.com	50

1. This view shows how much of the total invoice amount for each patient has been covered by their insurance. It joins the PERSON and INVOICE tables, calculates the insurance contribution, and provides a summary.

```
CREATE VIEW Insurance_Coverage_Per_Patient
AS SELECT
    P.Person_ID,
    CONCAT(P.First_Name, ' ', P.Last_Name) AS Full_Name,
    COUNT(I.Invoice_ID) AS Total_Invoices,
    SUM(I.Amount_Covered_by_Insurance) AS Insurance_Covered,
    SUM(I.Total - I.Amount_Covered_by_Insurance) AS Patient_Out_Of_Pocket
FROM
    PERSON P
JOIN
    INVOICE I ON P.Person_ID = I.Appointment_ID
GROUP BY
```

```
    P.Person_ID, P.First_Name, P.Last_Name
ORDER BY
    Insurance_Covered DESC;
```

Person_ID	Full_Name	Total_Invoices	Insurance_Covered	Patient_Out_Of_Pocket
2	Jane Smith	1	150	150
1	John Doe	1	100	100

Finally, we created two transactions to assist with managing the database.

Transaction 1 - Payment Processing

Explanation:

- **Purpose & Function:** This transaction efficiently handles a patient's payment, updates the invoice status to indicate that payment has been made, and accurately records the payment details for future reference.

- **Operations:**
    - Obtain the invoice information to compute the outstanding balance.
    - Enter the payment information into the PAYMENT table.
    - Revise the invoice status and outstanding balance in the INVOICE table.

- **Importance:**
    - By executing this transaction as a single unit, we can guarantee that payments are accurately recorded and that the invoice reflects the revised payment status. Any partial updates could lead to financial issues.

Code: (In order to run, remove one statement and keep the other, whether it be BEGIN or COMMIT)

```
-- First Transaction (Payment Processing)
BEGIN TRANSACTION;
```

```
-- Insert the new payment into the PAYMENT table
INSERT INTO PAYMENT (Payment_ID, Amount_Paid, Date, Payment_Type, Payment_Status)
VALUES (3, 100.0, '2024-12-04', 'Credit Card', 'Completed');

-- Update the INVOICE table to mark the invoice as "Paid" once payment is made
UPDATE INVOICE
SET Invoice_Status = 'Paid', Amount_Paid_by_Patient = 100.0
WHERE Invoice_ID = 2;

-- Update the REVENUE table to record the payment received
UPDATE REVENUE
SET revenue_amount = revenue_amount + 100.0
WHERE patient_id = 102;

-- Commit the first transaction
COMMIT;
```

Transaction 2 - Patient Medical History Update

Explanation:

- **Purpose & Function:**
    - This transaction enhances a patient's medical history by updating their medication details and allergy information, all while recording the changes in the MEDICATION_HISTORY table. With this process, it ensures that the patient's records remain precise and up to date, which facilitates safe and effective treatment.

- **Operations:**
    - Obtain the current medical history for a particular patient
    - Update that patient's medical history

- Add a new entry to the PATIENT_MEDICATIONS table for the recently prescribed medication.

- **Importance:**
  - Executing these updates a single unit is essential to maintain consistency. Allowing partial updates risks resulting in inaccurate or incomplete medical records, which may jeopardize patient safety.

Code: (In order to run, remove one statement and keep the other, whether it be BEGIN or COMMIT)

```
-- Second Transaction (Patient Medical History Update)
BEGIN TRANSACTION;

-- Retrieve existing medical history for verification
SELECT * FROM MEDICAL_HISTORY WHERE Patient_ID = 1;

-- Update medical history
UPDATE MEDICAL_HISTORY
SET
   Allergy_Info = 'Peanuts, Latex',
   Health_Risks = 'High Blood Pressure, Asthma'
WHERE
   Patient_ID = 1;

-- Add new medication for the patient
INSERT INTO PATIENT_MEDICATIONS (patient_id, medication_id, date_prescribed)
VALUES (1, 204, '2024-12-03');

-- Ensure changes are valid and committed
COMMIT;
```

Our SQL code itself is also attached in the zip file too. All SQL work was created and verified in SQLiteonline.

For this part of the assignment, Cody did bullet point 4, Andy did bullet point 5, Minh did bullet point 6, and Maverick did bullet point 7.

Overall, the team worked pretty well together. Something that might've been set up better is arranging a meeting after each project part finished so there could be more uptime and more clarity for each member to do their part. However, every member completed their work on time and had good communication too. Our suggestion is that having regular meetings, while they might seem perfunctory, is something that greatly contribute to progress whether they be in-person or not in-person. Just having face-to-face conversations greatly improves understanding and the speed of work.