Smilow Dentistry

CSE 3241 | Team 9

Yusuf Basha, Tiernan Farrell, Hajraan Hussain, Andrew Wang

Table of contents

Section 1 - Database Description

Introduction and project summary

As the scale of dentistry operations grows, in comes more customers, employees, forms, and then there's the respective meta data about each of these figures that must be kept in mind. The vast amount of data one must track soon grows to an insurmountable possibility. The best solution is to implement an efficient database to upkeep tracking all this information so that the business can run smoothly.

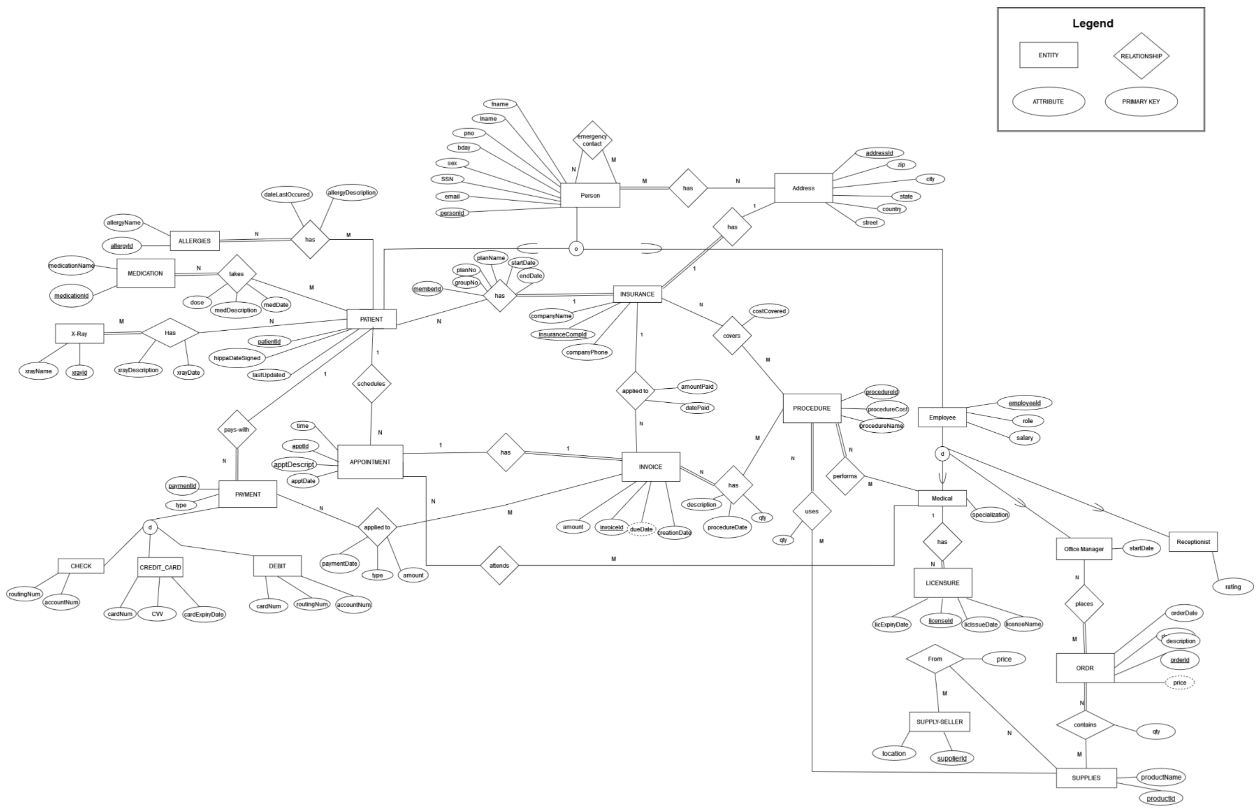
Dr. Hope Smilow of "Smilow Dentistry" requires a database to hold all of the necessary data for her dentistry. The database maintains a list of accepted insurance policy types, a list of standard dental procedures, and standard per unit charges for the procedures corresponding to the patient's insurance plan. The database accounts for all the employees' information, dentist, hygienists, dental assistants, receptionists, etc., as well as their licensure details if present. Also, the dentistry's patient data needs to be tracked, so their demographics, insurance information, medical history, and appointments are stored. In addition, the patients' signature of the HIPAA form, date of last x-ray, medications, and allergy information are also included in the database; this information needs to be allowed for updates if necessary. Furthermore, team 9 thought it was appropriate to include employee ratings, and supplies. Thus, making it easy to monitor if the employee is performing to a satisfactory level, and tracking the cost, and keeping track of supply costs, and seller information should an issue arise.

The database is designed so that any time a patient has an appointment, a billing record is generated with the date, doctor or medical professional, and the procedure. The patient's payment information is tracked by its cost amount, date, the type of payment (cash, credit, etc.), and the invoice. The payment information is stored and retrieved for future reference. Of course, a patient can be involved in numerous treatments or operations, so per appointment, there will be one bill assigned. Patients also might not have insurance, so if they do, the insurance is deducted from the bottom line and the patient is responsible for paying the remainder.

To tackle this project, team 9 has split the database creation to two sections: design scaffolding, and the database implementation. To make sure all the relationships and entities are connected with each other, an entity relation diagram is made to map out the various connections that are shared between the important components. A more logical model of the database is also created in the form of a relational schema which serves to explicitly state the parameters that are associated to entities and to provide a basis for the relational algebra queries. Normalizations of the database are applied to allow for unique identifiers being the key to data retrieval instead of the raw data itself. The secondary section of database creation is to write the necessary SQL to implement all of the previously discussed requirements in the format of the design documentation.

Included in this document are two parts corresponding to how the team responded to creating the database. The first part being a report on the database design, an user manual explaining it, and an overview of the team's contributions and the iterative process that the database underwent. The second part is the SQL database with a binary version of the database so that it is compatible with online SQLite, and five text files containing the SQL query scripts.

EERD Diagram



Relational Schema

Primary keys are underlined.
Foreign keys are in red.

Step 1: Regular Entities. Below are the regular entity types from the EERD. A relation for each is created with all of the attributes of the entity and an identifying primary key.

ALLERGIES(allergyId, allergyName)
MEDICATION(medicationId, medicationName)
X-RAY(xrayId, xrayName)
PAYMENT(paymentId, type)
PATIENT(patientId, hippaDateSigned, lastUpdated)
APPOINTMENT(apptId, time, apptDescript, apptDate)
CHEQUE(routingNum, accountNum)
CREDIT_CARD(cardNum, cvv, cardExpiryDate)
ORDR(orderId, description, price, orderDate)
DEBIT(cardNum, routingNum, accountNum)
INVOICE(invoiceId, creationDate, amount, dueDate)
PROCEDURE(procedureId, procedureCost, procedureName)
LICENSURE(licenseId, licExpiryDate, licIssueDate, licenseName)
EMPLOYEE(employeeId, role, salary)
SUPPLIES(productId, productName)
SUPPLY_SELLER(supplierId, location)
INSURANCE(insuranceCompId, companyName, companyPhone)
PERSON(personId, SSN, fname, lname, pno, bday, sex, email)
ADDRESS(addressId, street, country, state, city, zip)

Step 2: Mapping of weak entity types. There are no weak entities in the EERD, moving on to step 3.

Step 3: Mapping of Binary 1:N relationship type. Here we add any attributes of the relation to the many side along with the primary key from the 1 side as a foreign key. No new relations are introduced, however since there were some attributes to the relationship between patient and insurance, to store those attributes into the database would require creating a new relation. So, the relation PAT_INSUR(patientId, insuranceCompId, memberId, groupNo, planNo, startDate, endDate) was created in this step

ALLERGIES(allergyId, allergyName)
MEDICATION(medicationId, medicationName)
X-RAY(xrayId, xrayName)
PAYMENT(paymentId, type, patientId) FK patientId References Patient(patientId)
PATIENT(patientId, hippaDateSigned, lastUpdated, insuranceCompId) Reference Insurance
APPOINTMENT(apptId, time, apptDescript, apptDate, patientId) References Patient(patientId)
CHEQUE(routingNum, accountNum)

CREDIT_CARD(cardNum, cvv, cardExpiryDate)
DEBIT(cardNum, routingNum, accountNum)
ORDR(orderId, description, price, orderDate)
INVOICE(invoiceId, creationDate, amount, dueDate)
PROCEDURE(procedureId, procedureCost, procedureName)
LICENSURE(licenseId, licExpiryDate, licIssueDate, licenseName, employeeId) References Employee(employeeId)
EMPLOYEE(employeeId, role, salary)
SUPPLIES(productId, productName)
SUPPLY_SELLER(supplierId, location)
INSURANCE(insuranceCompId, companyName, companyPhone)
PERSON(personId, SSN, fname, lname, pno, bday, sex, email)
ADDRESS(addressId, street, country, state, city, zip)
PAT_INSUR(patientId, insuranceCompId, memberId, groupNo, planNo, planName, startDate, endDate) References Patient(patientId) and Insurance(insuranceCompId)

Step 4: Mapping 1:1 relationships. Foreign key approach - add primary key from partial to full participation side. Taken on relations between Insurance and Address, Appointment and Invoice. Added invoiceId to appointment.

ALLERGIES(allergyId, allergyName)
MEDICATION(medicationId, medicationName)
X-RAY(xrayId, xrayName)
PAYMENT(paymentId, type, patientId) References patient
PATIENT(patientId, hippaDateSigned, lastUpdated, insuranceCompId) Reference Insurance
APPOINTMENT(apptId, time, apptDescript, apptDate, patientId, invoiceId) References Patient(patientId)
CHEQUE(routingNum, accountNum)
CREDIT_CARD(cardNum, cvv, cardExpiryDate)
ORDR(orderId, description, price, orderDate)
DEBIT(cardNum, routingNum, accountNum)
INVOICE(invoiceId, creationDate, amount, dueDate, apptId) References Appointment
PROCEDURE(procedureId, procedureCost, procedureName)
LICENSURE(licenseId, licExpiryDate, licIssueDate, licenseName, employeeId) References employee
EMPLOYEE(employeeId, role, salary)
SUPPLIES(productId, price)
SUPPLY_SELLER(supplierId, location)
INSURANCE(insuranceCompId, companyName, companyPhone, addressId) References Address
PERSON(personId, SSN, fname, lname, pno, bday, sex, email)
ADDRESS(addressId, street, country, state, city, zip)
PAT_INSUR(patientId, insuranceCompId, memberId, groupNo, planNo, planName, startDate, endDate) References Patient(patientId) and Insurance(insuranceCompId)

Step 5: Mapping of binary m:n relationship types. Here we create a new relation with foreign keys of the primary keys of the two participating entities as well as any attributes of the relation. Here we run into an issue with relations on disjoint subclasses that have yet to be mapped; we deal with that in this step as well. Option 8a is used, new relations are created with foreign key pointing to superclass primary key as the new primary key and attributes of the subclass are also added.

ALLERGIES(allergyId, allergyName)
MEDICATION(medicationId, medicationName)
X-RAY(xrayId, xrayName)
PAYMENT(paymentId, type, patientId) References patient
PATIENT(patientId, hippaDateSigned, lastUpdated, insuranceCompId) Reference Insurance
APPOINTMENT(apptId, time, apptDescript, apptDate, patientId) Reference patient
CHEQUE(paymentId, routingNum, accountNum)Reference Payment
CREDIT_CARD(paymentId, cardNum, cvv, cardExpiryDate)Reference Payment
DEBIT(paymentId, cardNum, routingNum, accountNum)Reference Payment
INVOICE(invoiceId, creationDate, amount, dueDate, apptId) References Appointment
PROCEDURE(procedureId, procedureCost, procedureName)
LICENSURE(licenseId, licExpiryDate, licIssueDate, licenseName, employeeId) References employee
MEDICAL(employeeId, specialization)References Employee
ORDR(orderId, description, price, orderDate)
EMPLOYEE(employeeId, salary, role)
OFFICE_MANAGER(employeeId, startDate) References Employee
RECEPTIONIST(employeeId, rating) References Employee
SUPPLIES(productId, productName)
SUPPLY_SELLER(supplierId, location)
INSURANCE(insuranceCompId, companyName, companyPhone, addressId) References Address
PERSON(personId, SSN, fname, lname, pno, bday, sex, email)
ADDRESS(addressId, street, country, state, city, zip)
PERSON_ADDRESS(addressId, personId) References Person and Address
PAT_ALLERGIES(patientId, allergyId, dateLastOccured, allergyDescription) References Patient and Allergy
PAT_MEDS(patientId, medicationId, dose, medDescription, medDate) References Patient and Medication
PAT_XRAY(patientId, xrayId, xrayDescription, xrayDate) References Patient and Xray
PAY_INVOICE(paymentId, invoiceId, paymentDate, type, amount) References Payment and Invoice
INVOICE_PROC(invoiceId, procedureId, description, procedureDate, qty) References Invoice and Procedure
INSURANCE_COVERS(insuranceCompId, procedureId, costCovered) References Insurance and Procedure
PROCEDURE_SUPPLIES(procedureId, productId, qty) References procedure and Supplies
ATTENDS_APPT(employeeId, apptId) References Medical Employee and Appointment
PERFORMS_PROC(procedureId, employeeId) References Procedure and Medical Employee
SUPPLY_ORDER(orderId, productId, qty) References Ordr and Supplies
OFFICE_MAN_ORDER(employeeId, orderId) References Medical Employee and Ordr

SUPPLY_SELLER_PROVIDES(supplierId, productId, price) References SupplySeller and Supplies
EMERGENCY_CONTACT(personId, emergencyContactId) References Person and Person
PAT_INSUR(patientId, insuranceCompId, memberId, groupNo, planNo, planName, startDate, endDate) References Patient(patientId) and Insurance(insuranceCompId)

Step 6: Mapping of multivalued Attributes. None.
Step 7: Mapping of n-ary relationships. Also none.
Step 8: Mapping of specialization/generalizations. This step was completed with step 5 because there was a subclass involved in a m:n relationship. The option the group took was 8a, creating multiple relations for all super/sub classes. The primary key of the subclass is a foreign key that references the super class.
Step 9: Mapping of Union types. None.

With all of the schema mapping steps completed, the final schema now looks like this.

ALLERGIES(allergyId, allergyName)
MEDICATION(medicationId, medicationName)
X-RAY(xrayId, xrayName)
PAYMENT(paymentId, type, patientId) References patient
PATIENT(patientId, hippaDateSigned, lastUpdated, insuranceCompId) Reference Insurance
APPOINTMENT(apptId, time, apptDescript, apptDate, patientId) Reference patient
CHEQUE(paymentId, routingNum, accountNum)Reference Payment
CREDIT_CARD(paymentId, cardNum, cvv, cardExpiryDate)Reference Payment
DEBIT(paymentId, cardNum, routingNum, accountNum)Reference Payment
INVOICE(invoiceId, creationDate, amount, dueDate, apptId) References Appointment
PROCEDURE(procedureId, procedureCost, procedureName)
LICENSURE(licenseId, licExpiryDate, licIssueDate, licenseName, employeeId) References employee
MEDICAL(employeeId, specialization)References Employee
ORDR(orderId, description, price, orderDate)
EMPLOYEE(employeeId, salary, role)
OFFICE_MANAGER(employeeId, startDate) References Employee
RECEPTIONIST(employeeId, rating) References Employee
SUPPLIES(productId, productName)
SUPPLY_SELLER(supplierId, location)
INSURANCE(insuranceCompId, companyName, companyPhone, addressId) References Address
PERSON(personId, SSN, fname, lname, pno, bday, sex, email)
ADDRESS(addressId, street, country, state, city, zip)
PERSON_ADDRESS(addressId, personId) References Person and Address
PAT_ALLERGIES(patientId, allergyId, dateLastOccured, allergyDescription) References Patient and Allergy
PAT_MEDS(patientId, medicationId, dose, medDescription, medDate) References Patient and Medication
PAT_XRAY(patientId, xrayId, xrayDescription, xrayDate) References Patient and Xray
PAY_INVOICE(patmentId, invoiceId, paymentDate, type, amount) References Payment and Invoice

INVOICE_PROC(invoiceId, procedureId, description, procedureDate, qty) References Invoice and Procedure
INSURANCE_COVERS(insuranceCompId, procedureId, costCovered) References Insurance and Procedure
PROCEDURE_SUPPLIES(procedureId, productId, qty) References procedure and Supplies
ATTENDS_APPT(employeeId, apptId) References Medical Employee and Appointment
PERFORMS_PROC(procedureId, employeeId) References Procedure and Medical Employee
SUPPLY_ORDER(orderId, productId, qty) References Ordr and Supplies
OFFICE_MAN_ORDER(employeeId, orderId) References Medical Employee and Ordr
SUPPLY_SELLER_PROVIDES(supplierId, productId, price) References SupplySeller and Supplies
EMERGENCY_CONTACT(personId, emergencyContactId) References Person and Person
PAT_INSUR(patientId, insuranceCompId, memberId, groupNo, planNo, planName, startDate, endDate) References Patient(patientId) and Insurance(insuranceCompId)

Relational Algebra:

a. Create a list of patients and the medications they currently take
PERSON * (PATIENT ⋈ (patientId=patientId) MEDICATION)

b. Display patient information for patients who currently have Delta Dental insurance policy.
PERSON * (σ (companyName=Delta Dental) PATIENT)

c. Generate a list of procedures and dates of service performed by doctor Smilow.
SMILOW ← σ (lname=Smilow) PERSON
SMILOW_APPTS ← (SMILOW ⋈ (employeeId=employeeId) ATTENDS_APPT) * APPOINTMENT
π (procedureName, apptDate) (((SMILOW_APPTS * INVOICE) * INVOICE_PROC) * PROCEDURE)

d. 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.
OVERDUE ← σ ((Today - dueDate) > 30 AND (amountDue) > 10) INVOICE
INVOICE_PEOPLE ← (PATIENT ⋈ (patientId=patientId) (OVERDUE))
π (invoiceId, amountDue, dateDue, fname, lname, pno, email) INVOICE_PEOPLE

e. Find the patients who brought the most revenue in the past year.
YEAR_PAYMENTS ← σ ((Today - paymentDate) ≤ 365) PAY_INVOICE
PATIENT_PAYED ← σ (NOT (patientId = NULL)) (patientId F (SUM amountPayed) YEAR_PAYMENTS)
AVG ← F AVERAGE Sum_amountPayed PATIENT_PAYED
MOST_PAYED ← σ (Sum_amountPayedt > Average_Sum_amountPayedt) (PATIENT_PAYED X AVG)
PERSON * (PATIENT ⋈ (patientId=patientId) MOST_PAYED)

f. Create a list of doctors who performed less than 5 procedures this year.
PROCEDURES ← PREFORMS_PROC * LICENSURE
COUNTED ← employeeId F (COUNT procedureId) PROCEDURES

PERSON ⋈ (employeeId=employeeId AND Count_procedureId < 5) COUNTED

g. Find the highest paying procedures, procedure price, and the total number of those procedures performed.
AVG ← F (AVERAGE procedureCost) PROCEDURE
PRICES ← σ (procedureCost > Average_procedureCost) (PROCEDURE X AVG)
PROCEDURES ← ρ (procedureId, qty) (procedureId F (COUNT invoiceId) (INVOICE_PROC * PRICES))
π (procedureName, procedureCost, qty) (PRICES * PROCEDURES)

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.
COUNT_TYPE ← ρ (Count_paymentId, Times_Used) (type F (COUNT paymentId) PAYMENT)
CHARGED ← ρ (Sum_amountPayed, Total_Charged) (type F (SUM amountPayed) PAYMENT * PAY_INVOICE)
COUNT_TYPE * CHARGED

i. Find the name of the most popular insurance plan currently used by the patients.
COUNT_PLANS ← companyName F (COUNT patientId) PATIENT
MAX_PLAN ← F (MAXIMUM Count_patientId) COUNT_PLAN
π(companyName) (COUNT_PLAN ⋈ (Count_patientId=Maximum_Count_patientId) MAX_PLAN)

Three custom statements

1. Retrieve the patients, and insurance company name. This query includes all patients even without insurance.
(PERSON * PATIENT) ⟕ (patientId=patientId) INSURANCE

2. Find the average salary of the employees
F (Average salary) EMPLOYEE

3. Find the amount of payments that were paid late or weren't paid yet
F (count Today - dueDate > 0) INVOICE

Normalization:

Initially, our database was already in 1NF form since there were no multivalued attributes, so no changes were made.

ALLERGIES(<u>allergyId</u>, allergyName)
Each cell only contains one unique value.
Candidate key is allergyId.

FD: {allergyId} -> {allergyName}

1NF: Since all attributes are atomic and there are no nested relations, the relation is in 1NF.

2NF: There is no partial dependence on the primary key so relation is in 2NF

3NF: There is no transitive dependencies on the primary key so relation is in 3NF

BCNF: Since every determinant in the entity is a candidate key, relation is in BCNF.
ALLERGIES(allergyId, allergyName)

MEDICATION(medicationId, medicationName)
Candidate key is allergyId.
Each cell only contains one unique value.
Candidate key is allergyId.
FD: {medicationId} -> {medicationName}

1NF: Since all attributes are atomic and there are no nested relations, the relation is in 1NF.

2NF: There is no partial dependence on the primary key so relation is in 2NF

3NF: There is no transitive dependencies on the primary key so relation is in 3NF

BCNF: Since every determinant in the entity is a candidate key, relation is in BCNF.
MEDICATIONS(medicationId, medicationName)

X-RAY(xrayId, xrayName)
Candidate key is allergyId.
Each cell only contains one unique value.
Candidate key is xrayId.
FD: {xrayId} -> {xrayName}

1NF: Since all attributes are atomic and there are no nested relations, the relation is in 1NF.

2NF: There is no partial dependence on the primary key so relation is in 2NF

3NF: There is no transitive dependencies on the primary key so relation is in 3NF

BCNF: Since every determinant in the entity is a candidate key, relation is in BCNF.
X-RAY(xrayId, xrayName)

PAYMENT(paymentId, type, patientId) References patient

Candidate key is paymentId.

FD: {paymentId} -> {type, patientId}

1NF: Since all attributes are atomic and there are no nested relations, the relation is in 1NF.

2NF: No partial dependency on paymentId, in 2NF.

3NF: No transitive dependencies, relation in 3NF.

BCNF: Since every determinant in the entity is a candidate key, relation is in BCNF.
PAYMENT(paymentId, type, patientId)

PATIENT(patientId, hippaDateSigned, lastUpdated, insuranceCompId) Reference Insurance
Candidate key is patientId.

FD: {patientId} -> {hippaDateSigned, lastUpdated, insuranceCompId}

1NF: Since all attributes are atomic and there are no nested relations, the relation is in 1NF.

2NF: In 2NF because no partial dependency on patientId.

3NF: In 3NF because no transitive dependency on patientId.

BCNF: In BCNF because every determinant in the entity is the candidate key.
PATIENT(patientId, hippaDateSigned, lastUpdated, insuranceCompId)

APPOINTMENT(apptId, time, apptDescript, apptDate, patientId, invoiceID) Reference patient, Invoice

Candidate key is apptId.
FD: {apptId} -> {time, apptDescript, apptDate, patientId, invoiceId}

1NF: Since all attributes are atomic and there are no nested relations, the relation is in 1NF.

2NF: In 2NF because no partial dependency on apptId.

3NF: In 3NF because no transitive dependency on apptId.

BCNF: In BCNF because every determinant in the entity is the candidate key.
APPOINTMENT(apptId, time, apptDescript, apptDate, patientId, invoiceId)

CHEQUE(routingNum, accountNum, paymentId) Reference payment

Candidate keys is accountNum, routingNum.
FD: {accountNum, routingNum} -> {paymentId}

1NF: Since all attributes are atomic and there are no nested relations, the relation is in 1NF.

2NF: In 2NF because no partial dependency on accountNum, routingNum.

3NF: In 3NF because no transitive dependency on accountNum, routingNum.

In BCNF because every determinant in the entity is a candidate key.
CHEQUE(routingNum, accountNum, paymentId)

CREDIT_CARD(cardNum, cvv, cardExpiryDate, paymentId) References payment
Candidate key are cardNum, cvv.
FD: {cardNum, cvv} -> {cardExpiryDate, paymentId}

1NF: Since all attributes are atomic and there are no nested relations, the relation is in 1NF.

2NF: In 2NF because no partial dependency on accountNum, routingNum.

3NF: In 3NF because no transitive dependency on accountNum, routingNum.

In BCNF because every determinant in the entity is a candidate key.
CREDIT_CARD(cardNum, cvv, cardExpiryDate, paymentId)

DEBIT(cardNum, routingNum, accountNum, paymentId) References payment
Candidate key are cardNum, routingNum, accountNum
FD: {cardNum, routingNum, accountNum} -> {paymentId}

1NF: Since all attributes are atomic and there are no nested relations, the relation is in 1NF.

2NF: In 2NF because no partial dependency on cardNum, accountNum, routingNum.

3NF: In 3NF because no transitive dependency on cardNum, accountNum, routingNum.

In BCNF because every determinant in the entity is a candidate key.
DEBIT(cardNum, routingNum, accountNum, paymentId)

INVOICE(invoiceId, creationDate, amount, dueDate, apptId) References Appointment
Candidate key is invoiceId.
FD: {invoiceId} ->{creationDate, amount, dueDate, apptId}

1NF: Since all attributes are atomic and there are no nested relations, the relation is in 1NF.

2NF: In 2NF because no partial dependency on invoiceId

3NF: In 3NF because no transitive dependency on invoiceId

In BCNF because every determinant in the entity is a candidate key.
INVOICE(<u>invoiceId</u>, creationDate, amount, dueDate, apptId)

PROCEDURE(<u>procedureId</u>, procedureCost, procedureName)
Candidate key is procedureId.
FD: {procedureId} ->{procedureName, procedureCost}

1NF: Since all attributes are atomic and there are no nested relations, the relation is in 1NF.

2NF: In 2NF because no partial dependency on procedureId

3NF: In 3NF because no transitive dependency on procedureId

In BCNF because every determinant in the entity is a candidate key.
PROCEDURE(<u>procedureId</u>, procedureCost, procedureName)

LICENSURE(<u>licenseId</u>, licExpiryDate, licIssueDate, licenseName, employeeId) References employee
Candidate key is licenseId.
FD: {licenseId} ->{licExpiryDate, licIssueDate, licenseName, employeeId}

1NF: Since all attributes are atomic and there are no nested relations, the relation is in 1NF.

2NF: In 2NF because no partial dependency on licenseId

3NF: In 3NF because no transitive dependency on licenseId

In BCNF because every determinant in the entity is a candidate key.
LICENSURE(<u>licenseId</u>, licExpiryDate, licIssueDate, licenseName, employeeId)

MEDICAL(employeeId, specialization) References Employee
Candidate key is employeeId.

1NF: Since all attributes are atomic and there are no nested relations, the relation is in 1NF.

FD: {employeeId} -> {specialization}

2NF: In 2NF because no partial dependency on employeeId

3NF: In 3NF because no transitive dependency on employeeId

In BCNF because every determinant in the entity is a candidate key.
MEDICAL(employeeId, specialization) References Employee

EMPLOYEE(employeeId, salary, role)
Candidate key is employeeId.
FD: {employeeId} ->{salary, role}

1NF: Since all attributes are atomic and there are no nested relations, the relation is in 1NF.

2NF: In 2NF because no partial dependency on employeeId

3NF: In 3NF because no transitive dependency on employeeId

In BCNF because every determinant in the entity is a candidate key.
EMPLOYEE(employeeId, salary, role)

OFFICE_MANAGER(employeeId, startDate) References Employee
Candidate key is employeeId.
FD: {employeeId} ->{startDate}

1NF: Since all attributes are atomic and there are no nested relations, the relation is in 1NF.

2NF: In 2NF because no partial dependency on employeeId

3NF: In 3NF because no transitive dependency on employeeId

In BCNF because every determinant in the entity is a candidate key.
OFFICE_MANAGER(employeeId, startDate)

RECEPTIONIST(employeeId, rating) References Employee
Candidate key is employeeId.
FD: {employeeId} ->{rating}

1NF: Since all attributes are atomic and there are no nested relations, the relation is in 1NF.

2NF: In 2NF because no partial dependency on employeeId

3NF: In 3NF because no transitive dependency on employeeId

In BCNF because every determinant in the entity is a candidate key.
RECEPTIONIST(employeeId, rating)

SUPPLIES(productId, productName)

Candidate key is productId.

FD: {productId} -> {productName}

1NF: Since all attributes are atomic and there are no nested relations, the relation is in 1NF.

2NF:In 2NF because no partial dependency on employeeId

3NF: In 3NF because no transitive dependency on employeeId

In BCNF because every determinant in the entity is a candidate key.

SUPPLIES(productId, productName)

SUPPLY_SELLER(supplierId, location)

Candidate key is supplierId

FD: {supplierId} -> {location}

1NF: Since all attributes are atomic and there are no nested relations, the relation is in 1NF.

2NF: No partial dependency

3NF: No transitive dependency

BCNF: Every determinant in the entity is the candidate key.

SUPPLY_SELLER(supplierId, location)

INSURANCE(insuranceCompId, companyName, companyPhone, addressId) References Address

Candidate key is insuranceCompId

FD: {InsuranceCompId} ->{companyName, companyPhone, addressId}

1NF: Since all attributes are atomic and there are no nested relations, the relation is in 1NF.

2NF: No partial dependency

3NF: No transitive dependency

BCNF: Every determinant in the entity is the candidate key.

INSURANCE(insuranceCompId, companyName, companyPhone, addressId) References Address

PERSON(personId, SSN, fname, lname, pno, bday, sex, email)

Candidate key is personId

FD: {personId} -> {SSN, fname, lname, pno, bday, sex, email}

1NF: Since all attributes are atomic and there are no nested relations, the relation is in 1NF.

2NF: No partial dependency

3NF: No transitive dependency

BCNF: Every determinant in the entity is the candidate key.
PERSON(<u>personId</u>, SSN, fname, lname, pno, bday, sex, email)

ADDRESS(<u>addressId</u>, street, country, state, city, zip)
Candidate key is addressId

FD: {addressId} -> {street, country, state, city, zip}

1NF: Since all attributes are atomic and there are no nested relations, the relation is in 1NF.

2NF: No partial dependency

3NF: No transitive dependency

BCNF: Every determinant in the entity is the candidate key.
ADDRESS(<u>addressId</u>, street, country, state, city, zip)

PERSON_ADDRESS(<u>addressId, personId</u>) References Person and Address
Candidate key is addressId, personId

FD: No keys not part of candidate key

1NF: Since all attributes are atomic and there are no nested relations, the relation is in 1NF.

2NF: No partial dependency

3NF: No transitive dependency

BCNF: Every determinant in the entity is the candidate key.
PERSON_ADDRESS(<u>addressId, personId</u>)

PAT_ALLERGIES(patientId, allergyId, dateLastOccured, allergyDescription) References Patient and Allergy
Candidate key is patientId, allergyId

FD: {patientId, allergyId} -> {dateLastOccured, allergyDescription}

1NF: Since all attributes are atomic and there are no nested relations, the relation is in 1NF.

2NF: No partial dependency, in 2NF.

3NF: No transitive dependency, in 3NF.

BCNF: Every determinant in the entity is the candidate key, in BCNF.
PAT_ALLERGIES(patientId, allergyId, dateLastOccured, allergyDescription)

PAT_MEDS(patientId, medicationId, dose, medDescription, medDate) References Patient and Medication
Candidate key is patientId, medicationId

FD: {patientId, medicationId} ->{dose, medDescription, medDate}

1NF: Since all attributes are atomic and there are no nested relations, the relation is in 1NF.

2NF: No partial dependency, in 2NF.

3NF: No transitive dependency, in 3NF.

BCNF: Every determinant in the entity is the candidate key, in BCNF.
PAT_MEDS(patientId, medicationId, dose, medDescription, medDate)

PAT_XRAY(patientId, xrayId, xrayDescription, xrayDate) References Patient and Xray
Candidate key is patientId, xrayId

FD: {patientId, xrayId} -> {xrayDescription, xrayDate}

1NF: Since all attributes are atomic and there are no nested relations, the relation is in 1NF.

2NF: No partial dependency, in 2NF.

3NF: No transitive dependency, in 3NF.

BCNF: Every determinant in the entity is the candidate key, in BCNF.
PAT_XRAY(patientId, xrayId, xrayDescription, xrayDate)

PAY_INVOICE(<u>paymentId, invoiceId</u>, paymentDate, type, amount) References Payment and Invoice
Candidate key is paymentId, invoiceId,

FD: {paymentId, invoiceId} -> {paymentDate, amount}
{paymentId} -> {type}

1NF: Since all attributes are atomic and there are no nested relations, the relation is in 1NF.

2NF: Payment type is dependent of paymentId only, not the whole candidate key. Split up this relation to get into 2NF.

PAY_INVOICE(<u>paymentId, invoiceId</u>, paymentDate, amount)
PAY_TYPE(<u>paymentId,</u> type)

3NF: No transitive dependency on either of the new relations, in 2NF.

BCNF: Every determinate in both entities is the candidate key. Both relations are now in BCNF.
PAY_INVOICE(<u>paymentId, invoiceId</u>, paymentDate, amount)
PAY_TYPE(<u>paymentId,</u> type)

INVOICE_PROC(<u>invoiceId, procedureId</u>, description, procedureDate, qty) References Invoice and Procedure
Candidate key is (invoiceId, procedureId)

FD: {invoiceId, procedureId} -> {description, procedureDate, qty}

1NF: Since all attributes are atomic and there are no nested relations, the relation is in 1NF.

2NF: No partial dependency, in 2NF

3NF: No transitive dependency either, in 3NF.

BCNF: Every determinate in the entity is the candidate key
INVOICE_PROC(<u>invoiceId, procedureId</u>, description, procedureDate, qty)

INSURANCE_COVERS<u>(insuranceCompId, procedureId</u>, costCovered) References Insurance and Procedure
Candidate key is (insuranceCompId, procedureId)

FD: {insuranceCompId, procedureId} -> {costCovered}

1NF: Since all attributes are atomic and there are no nested relations, the relation is in 1NF.

2NF: No partial dependency, in 2NF

3NF: No transitive dependency, in 3NF

BCNF: Every determinate in the entity is the candidate key
INSURANCE_COVERS(insuranceCompId, procedureId, costCovered)

PROCEDURE_SUPPLIES(procedureId, productId, qty) References procedure and Supplies
Candidate key is (procedureId, productId)
FD: {procedureId, productId} -> {qty}

1NF: Since all attributes are atomic and there are no nested relations, the relation is in 1NF.

2NF: No partial dependency, in 2NF

3NF: No transitive dependency, in 3NF

BCNF: Every determinate in the entity is the candidate key
PROCEDURE_SUPPLIES(procedureId, productId, qty)

ATTENDS_APPT(employeeId, apptId) References Medical Employee and Appointment
Candidate key is (employeeId, apptId)
FD: No keys that are not part of candidate key

1NF: Since all attributes are atomic and there are no nested relations, the relation is in 1NF.

2NF: No partial dependency

3NF: No transitive dependency

BCNF: Every determinant in the entity is the candidate key.
ATTENDS_APPT(employeeId, apptId)

PERFORMS_PROC(procedureId, employeeId) References Procedure and Medical Employee
Candidate key is (procedureId, employeeId)
FD: No keys that are not part of candidate key

1NF: Since all attributes are atomic and there are no nested relations, the relation is in 1NF.

2NF: No partial dependency

3NF: No transitive dependency

BCNF: Every determinant in the entity is the candidate key.
PERFORMS_PROC(procedureId, employeeId)

ORDR(orderId, description, orderDate, price) References Procedure and Medical Employee
Candidate key is orderId
FD: {orderId} -> {orderDate, description, price)

1NF: Since all attributes are atomic and there are no nested relations, the relation is in 1NF.

2NF: No partial dependency

3NF: No transitive dependency

BCNF: Every determinant in the entity is the candidate key.
ORDR(orderId, description, orderDate, price)

SUPPLY_ORDER(orderId, productId, qty) References Ordr and Supplies
Candidate key is orderId, productId
FD: {orderId, productId} ->{qty}

1NF: Since all attributes are atomic and there are no nested relations, the relation is in 1NF.

2NF: No partial dependency

3NF: No transitive dependency

BCNF: Every determinant in the entity is the candidate key.
SUPPLY_ORDER(orderId, productId, qty)

OFFICE_MAN_ORDER(employeeId, orderId) References Medical Employee and Ordr
Candidate key is employeeId, orderId
FD: No keys that are not part of candidate key

1NF: Since all attributes are atomic and there are no nested relations, the relation is in 1NF.

2NF: No partial dependency

3NF: No transitive dependency

BCNF: Every determinant in the entity is the candidate key.
OFFICE_MAN_ORDER(employeeId, orderId) References Medical Employee and Ordr

SUPPLY_SELLER_PROVIDES(supplierId, productId, price) References SupplySeller and Supplies
Candidate key is (supplierId, productId)
FD: {supplierId, productId} -> {price}

1NF: Since all attributes are atomic and there are no nested relations, the relation is in 1NF.

2NF: No partial dependency

3NF: No transitive dependency

BCNF: Every determinant in the entity is the candidate key.
SUPPLY_SELLER_PROVIDES(supplierId, productId, price)

EMERGENCY_CONTACT(personId, emergencyContactId) References Person and Person
Candidate key is (personId, emergencyContactId)
FD: No keys that are not part of candidate key

1NF: Since all attributes are atomic and there are no nested relations, the relation is in 1NF.

2NF: No partial dependency

3NF: No transitive dependency

BCNF: Every determinant in the entity is the candidate key.

EMERGENCY_CONTACT(personId, emergencyContactId)

PAT_INSUR(patientId, insuranceCompId, memberId, groupNo, planNo, planName, startDate, endDate)
References Patient(patientId) and Insurance(insuranceCompId)
Candidate key is (patientId, insuranceCompId)
FD: {patientId, insuranceCompId} -> {memberId, groupNo, planNo, planName, startDate, endDate}

1NF: Since all attributes are atomic and there are no nested relations, the relation is in 1NF.

2NF: No partial dependency

3NF: No transitive dependency

BCNF: Every determinant in the entity is the candidate key.
PAT_INSUR(patientId, insuranceCompId, memberId, groupNo, planNo, planName, startDate, endDate)

Normalized Schema in BCNF Form:

ALLERGIES(allergyId, allergyName)
MEDICATION(medicationId, medicationName)
XRAY(xrayId, xrayName)
PAYMENT(paymentId, type, patientId) References patient
PATIENT(patientId, hippaDateSigned, lastUpdated, insuranceCompId) Reference Insurance
APPOINTMENT(apptId, apptTime, apptDescript, apptDate, patientId, invoiceId) Reference patient
CHEQUE(paymentId, routingNum, accountNum)Reference Payment
CREDIT_CARD(paymentId, cardNum, cvv, cardExpiryDate)Reference Payment
DEBIT(paymentId, cardNum, routingNum, accountNum)Reference Payment
INVOICE(invoiceId, creationDate, amount, dueDate, apptId) References Appointment
PROCEDURE(procedureId, procedureCost, procedureName)
LICENSURE(licenseId, licExpiryDate, licIssueDate, licenseName, employeeId) References employee
MEDICAL(employeeId, specialization)References Employee
ORDR(orderId, description, price, orderDate)
EMPLOYEE(employeeId, salary, role)
OFFICE_MANAGER(employeeId, startDate) References Employee
RECEPTIONIST(employeeId, rating) References Employee
SUPPLIES(productId, productName)
SUPPLY_SELLER(supplierId, location)
INSURANCE(insuranceCompId, companyName, companyPhone, addressId) References Address
PERSON(personId, email, SSN, fname, lname, pno, bday, sex)
ADDRESS(addressId, street, country, state, city, zip)
PERSON_ADDRESS(addressId, personId) References Person and Address
PAT_ALLERGIES(patientId, allergyId, dateLastOccured, allergyDescription) References Patient and Allergy
PAT_MEDS(patientId, medicationId, dose, medDescription, medDate) References Patient and Medication
PAT_XRAY(patientId, xrayId, xrayDescription, xrayDate) References Patient and Xray
PAY_INVOICE(paymentId, invoiceId, paymentDate, amount) References Payment and Invoice
PAY_TYPE(paymentId, type) References Payment
INVOICE_PROC(invoiceId, procedureId, description, procedureDate, qty) References Invoice and Procedure
INSURANCE_COVERS(insuranceCompId, procedureId, costCovered) References Insurance and Procedure

PROCEDURE_SUPPLIES(procedureId, productId, qty) References procedure and Supplies
ATTENDS_APPT(employeeId, apptId) References Medical Employee and Appointment
PERFORMS_PROC(procedureId, employeeId) References Procedure and Medical Employee
SUPPLY_ORDER(orderId, productId, qty) References Ordr and Supplies
OFFICE_MAN_ORDER(employeeId, orderId) References Medical Employee and Ordr
SUPPLY_SELLER_PROVIDES(supplierId, productId, price) References SupplySeller and Supplies
EMERGENCY_CONTACT(personId, emergencyContactId) References Person and Person
PAT_INSUR(patientId, insuranceCompId, memberId, groupNo, planNo, planName, startDate, endDate) References Patient(patientId) and Insurance(insuranceCompId)

Section 2: User Manual

For all tables, includes a description with the table functions, keys, constraints, and data types

Table ALLERGIES:
There exists a table called allergies for holding allergies names and their respective ids. This table serves to return the appropriate allergy for a particular id. We ensure the primary key, allergyId, is an integer that is auto incremented and can be null to create a distinct value as its primary key, as well as the allergy name is a variable length string with a max length of 20 which cannot be null.

Table MEDICATION:
There exists a table called medications for holding medication names and their respective ids. This table serves to return the appropriate medication for a particular id. We ensure the primary key, medicationId, is an integer that is auto incremented and can be null to create a distinct value as its primary key, as well as the medication name is a variable length string with a max length of 20 which cannot be null.

Table XRAY:
There exists a table called xray for holding xrays names and their respective ids. This table serves to return the appropriate xray for a particular id. We ensure the primary key, xrayId, is an integer that is auto incremented and can be null to create a distinct value as its primary key, as well as the xray name is a variable length string with a max length of 20 which cannot be null.

Table PAYMENTS:
There exists a table called payments for relating a specific patient to a payment made by them as well as the payment's type. We ensure the primary key, paymentId, is an integer that is auto incremented and can be null to create a distinct value as its primary key, the payment type is a variable length string with a max length of 20 which cannot be null, and the patientId is an integer that cannot be null is a foreign key which references Patient(patientId).

TABLE PATIENT:
There exists a table called patients for storing useful information for the patient. We ensure the primary key, patientId, is an integer that is auto incremented and can be null to create a distinct value as its primary key, the hippaDateSigned stores the most recent date when the hippa was signed, lastUpdated stores the most recent date and time where the patient's information was

stored, and insuranceCompId is an integer that can be null (patient may not have insurance) which is a foreign key which references INSURANCE(insuranceCompId).

Table APPOINTMENT:
There exists a table called appointment for relating a patient to a specific appointment with general information like time, date, and a description/summary for reasons to schedule the appointment. We ensure the primary key, apptId, is an integer that is auto incremented and can be null to create a distinct value as its primary key, the time had to have been scheduled so cannot be null, text data type for description for why patient scheduled which cannot be null, the date data type which cannot be null for appointment date, and the patientId is an integer that cannot be null is a foreign key which references Patient(patientId).

Table CHEQUE:
There exists a table called cheque which serves as one of the disjoint subclasses of payment. The table will have basic information for any check - such as the routingNum being an integer which cannot be null as it must refer a financial institution, an accountNum being an integer which cannot be null as it refers to the payer’s bank account number, and paymentId is an integer which is a foreign key which references PAYMENT(paymentId).

Table CREDIT_CARD:
There exists a table called credit card which serves as one of the disjoint subclasses of payment. The table will have basic information for any credit card- such as the cardNum being an integer which cannot be null as it must refer to a valid card number as well as the cvv being an integer that cannot be null as it must refer to a valid security code. The credit card expiry date is a date data type that cannot be null so it must refer to a valid future date, and paymentId is an integer which is a foreign key which references PAYMENT(paymentId).

Table DEBIT:
There exists a table called debit which serves as one of the disjoint subclasses of payment. The table will have basic information for any debit card- such as the cardNum being an integer which cannot be null as it must refer a financial institution, the routingNum being an integer which cannot be null as it must refer a financial institution, an accountNum being an integer which cannot be null as it refers to the payer’s bank account number, and paymentId is an integer which is a foreign key which references PAYMENT(paymentId).

Table INVOICE:
There exists a table called invoice for maintaining invoice information for a particular appointment. We ensure the primary key, invoiceId, is an integer that is auto incremented and can be null to create a distinct value as its primary key, creationDate is a date data type which

cannot be null and holds the creation date for the invoice, the amount has the decimal data type with precision of 2 decimal values and the maximum total number of digits of 10 that cannot be null as it specifies the required amount needed or still needed to be paid by the patient or their respective insurance, the dueDate is a date data type which cannot be null as it is calculated by adding a set timeframe of a week to the creation date, and apptId is an integer which is a foreign key which references APPOINTMENT(apptId).

Table PROCEDURE:
There exists a table called procedure for maintaining procedure information. We ensure the primary key, procedureId, is an integer that is auto incremented and can be null to create a distinct value as its primary key, procedureName is a variable length string with a max length of 20 which cannot be null to store the procedure name, and procedureCost as a decimal data type with precision of 2 decimal values and the maximum total number of digits of 10 and cannot be null to store the procedure's cost.

Table LICENSURE:
There exists a table called license for maintaining the license information for a specific medical employee. We ensure the primary key, licenseId, is an integer that is auto incremented and can be null to create a distinct value as its primary key, the licExpiryDate as a date data type which cannot be null to store the license's expiry date, the licIssueDate as a date data type which cannot be null to store the license's issue date, licenseName is a variable length string with a max length of 25 which cannot be null to store the name of the license, and employeeId is an integer which is a foreign key which references EMPLOYEE(employeeId).

TABLE MEDICAL:
There exists a table medical which is one of the overlapping specializations of person. There will exist specialization which is a variable length string with a max length of 20 and cannot be null for storing the job specialization of the medical employee (i.e. a dentist that specializes in prosthodonticsl or periodontics), and employeeId is an integer which is a foreign key which references EMPLOYEE(employeeId).

Table ORDR:
There exists a table called ordr for maintaining order information for the many orders performed by the office manager. We ensure the primary key, orderId, is an integer that is auto incremented and can be null to create a distinct value as its primary key, description as a text data type that cannot be null for specifying the order details, the price as a decimal data type with precision of 2 decimal values and the maximum total number of digits of 10 and cannot be null to store the order's cost, and the orderDate being a date data type which cannot be null for storing the date the order was sent.

Table EMPLOYEE:
There exists a table called employee for maintaining our employee information. We ensure the primary key, employeeId, is an integer that is auto incremented and can be null to create a distinct value as its primary key, salary is a integer that cannot be null for storing the employee's yearly salary, and their role as a variable length string with a max length of 20 and cannot be null.

Table OFFICE_MANAGER:
There exists a table called office manager for storing one of the disjoint specializations of an employee. This table will hold the startDate attribute, which is a date data type that cannot be null which stores the office manager's first day on the job, and employeeId is an integer which is a foreign key and cannot be null which references EMPLOYEE(employeeId).

Table RECEPTIONIST:
There exists a table called office manager for storing one of the disjoint specializations of an employee. This table will hold the rating attribute, which is an integer data type that cannot be null which stores a patient's rating of the receptionist, and employeeId is an integer which is a foreign key and cannot be null which references EMPLOYEE(employeeId).

Table SUPPLIES:
There exists a table called supplies for holding the current inventory made available by the supply seller. We ensure the primary key, productId, is an integer that is auto incremented and can be null to create a distinct value as its primary key, and productName being an varchar data type that cannot be null for storing the product name.

Table SUPPLY_SELLER:
There exists a table called supply seller for holding the suppliers for the office. We ensure the primary key, supplierId, is an integer that is auto incremented and can be null to create a distinct value as its primary key, and location as a variable length string with a max length of 20 and cannot be null.

Table INSURANCE:
There exists a table called insurance for storing the general information for a dental insurance company. We ensure the primary key, insuranceCompId, is an integer that is auto incremented and can be null to create a distinct value as its primary key, companyName as a variable length string with a max length of 20 and cannot be null for storing the name of the insurance company, companyPhone as an integer data type that cannot be null for storing the phone number of the insurance company, and addressId is an integer which is a foreign key and cannot be null which references EMPLOYEE(employeeId).

Table PERSON:
There exists a table called person which serves a superclass for patient and employee. We ensure the primary key, personId, is an integer that is auto incremented and can be null to create a distinct value as its primary key, email as a variable length string with a max length of 25 and cannot be null for storing the person’s email, fname as a variable length string with a max length of 20 and cannot be null for storing the person’s first name, lname as a variable length string with a max length of 20 and cannot be null for storing the person’s last name, ssn as a integer data type that cannot be null for storing the person’s SSN, pno as a integer data type that cannot be null for storing the person’s phone number, bday as a date data type that cannot be null for storing the person’s date of birth, sex as a char data type that cannot be null for storing the person’s gender, and type as a tinyint that cannot be null for determining if that person is either a patient (type == 1) or an employee (type == 2).

Table ADDRESS:
There exists a table called address which stores the various addresses for individuals and insurance companies. We ensure the primary key, addressId, is an integer that is auto incremented and can be null to create a distinct value as its primary key, street as a variable length string with a max length of 35 and cannot be null, country as a variable length string with a max length of 20 and cannot be null, state as a variable length string with a max length of 2 and cannot be null, city as a variable length string with a max length of 20 and cannot be null, and zip as a integer data type and cannot be null for the zip code of the address.

Table PERSON_ADDRESS:
There exists a table called person_address for relating a specific person to an address. addressId is an integer which is a foreign key and a primary key and cannot be null which references ADDRESS(addressId). personId is an integer which is a foreign key and a primary key and cannot be null which references PERSON(personId).

Table PAT_ALLERGIES:
There exists a table called pat_allergies for relating a specific patient to an allergy. patientId is an integer which is a foreign key and a primary key and cannot be null which references PATIENT(patientId). allergyId is an integer which is a foreign key and a primary key and cannot be null which references ALLERGIES(allergyId). Additionally, there are attributes dateLastOcccured as a date data type for referring to the last time a patient was under the effects of the particular allergy, allergyDescription as a text data type which stores a description of the allergy and any side notes pertaining to the patient.

Table PAT_MEDS:
There exists a table called pat_allergies for relating a specific patient to an allergy. patientId is an integer which is a foreign key and a primary key and cannot be null which references

PATIENT(patientId). medicationId is an integer which is a foreign key and a primary key and cannot be null which references MEDICATION(medicationId). Additionally, there are attributes medDate as a date data type for referring to the last time a patient was under the effects of the particular medication, medDescription as a text data type which stores a description of the medication and any side notes pertaining to the patient, and dose as a variable length string with a max length of 20 and cannot be null for storing the regular dose taken by the patient.

Table PAT_XRAY:
There exists a table called pat_xray for relating a specific patient to an xray. patientId is an integer which is a foreign key and a primary key and cannot be null which references PATIENT(patientId). xrayId is an integer which is a foreign key and a primary key and cannot be null which references XRAY(xrayId). Additionally, there are attributes xraydate as a date data type for referring to the date where a patient had the xray, xrayDescription as a text data type which stores a description of the xrayand any side notes pertaining to the patient.

Table PAY_INVOICE:
There exists a table called pay_invoice for relating a specific payment to an invoice. paymentId is an integer which is a foreign key and a primary key and cannot be null which references PAYMENT(paymentId). invoiceId is an integer which is a foreign key and a primary key and cannot be null which references INVOICE(invoiceId). Additionally, there are attributes paymentDate as a date data type for referring to the date where a patient had made the payment, and amount as a decimal data type with precision of 2 decimal values and the maximum total number of digits of 10 for storing the amount paid by the patient.

Table PAY_TYPE:
There exists a table called pay_type for determining the payment type of a specific payment made. paymentId is an integer which is a foreign key and a primary key and cannot be null which references PAYMENT(paymentId). Type is a variable length string with a max length of 20 of storing the specific form of the payment (check, credit card, and debit are allowed types of payments)

Table INVOICE_PROC:
There exists a table called invoice_proc for relating a specific invoice to an invoice. procedureId is an integer which is a foreign key and a primary key and cannot be null which references PROCEDURE(procedureId). invoiceId is an integer which is a foreign key and a primary key and cannot be null which references INVOICE(invoiceId). Additionally, there are attributes procedureDate as a date data type for referring to the date where a patient had been performed the procedure on, description as a text data type for any side information for the procedure performed, and the qty as a integer data type indicating the number of times the specific procedure had been performed.

Table INSURANCE_COVERS:
There exists a table called insurance_covers for insurance detailing the specific procedures that are covered by an insurance plan for a patient. procedureId is an integer which is a foreign key and a primary key and cannot be null which references PROCEDURE(procedureId). insuranceCompId is an integer which is a foreign key and a primary key and cannot be null which references INSURANCE(insuranceCompId). Additionally, there is another attribute called costCovered as a decimal data type with precision of 2 decimal values and the maximum total number of digits of 10 for storing the amount that is covered by the insurance for a particular procedure.

Table PROCEDURE_SUPPLIES:
There exists a table called procedure_supplies to indicate the products that may be used for a particular procedure. procedureId is an integer which is a foreign key and a primary key and cannot be null which references PROCEDURE(procedureId). productId is an integer which is a foreign key and a primary key and cannot be null which references SUPPLIES(productId). Additionally, there is an attribute called qty as an integer data type for storing the quantity that particular product was used for the duration of the procedure.

Table ATTENDS_APPT:
There exists a table called attends_appt to indicate the medical employee that had attended a dentist appointment for a certain patient. employeeId is an integer which is a foreign key and a primary key and cannot be null which references MEDICAL(employeeId). apptId is an integer which is a foreign key and a primary key and cannot be null which references APPOINTMENT(apptId).

Table PERFORMS_PROC:
There exists a table called performs_proc to indicate the medical employee that had performed a certain procedure for a certain patient. employeeId is an integer which is a foreign key and a primary key and cannot be null which references MEDICAL(employeeId). procedureId is an integer which is a foreign key and a primary key and cannot be null which references PROCEDURE(procedureId).

Table SUPPLY_ORDER:
There exists a table called supply_order for indicating the certain products that were purchased by an order. procedureId is an integer which is a foreign key and a primary key and cannot be null which references PROCEDURE(procedureId). orderId is an integer which is a foreign key and a primary key and cannot be null which references ORDR(orderId)). Additionally, there is another attribute called qty as an integer data type for storing the quantity of a product that was purchased in the order.

Table OFFICE_MAN_ORDER:
There exists a table called office_man_order for specifying the orders pertaining to a certain office manager. employeeId is an integer which is a foreign key and a primary key and cannot be null which references MEDICAL(employeeId). orderId is an integer which is a foreign key and a primary key and cannot be null which references ORDR(orderId)).

Table SUPPLY_SELLER_PROVIDES:
There exists a table called supply_seller_provides for specifying the products that are purchasable by a certain supplier. supplierId is an integer which is a foreign key and a primary key and cannot be null which references SUPPLY_SELLER(supplierId). productId is an integer which is a foreign key and a primary key and cannot be null which references SUPPLIES(productId). Additionally, there is an attribute price as an integer data type that stores the cost for a particular product.

Table EMERGENCY_CONTACT:
There exists a table called emergency_contact for detailing the available emergency contacts for a specific person. personId is an integer which is a foreign key and a primary key and cannot be null which references PERSON(personId). emergencyContactId is an integer which is a foreign key and a primary key and cannot be null which references PERSON(personId)).

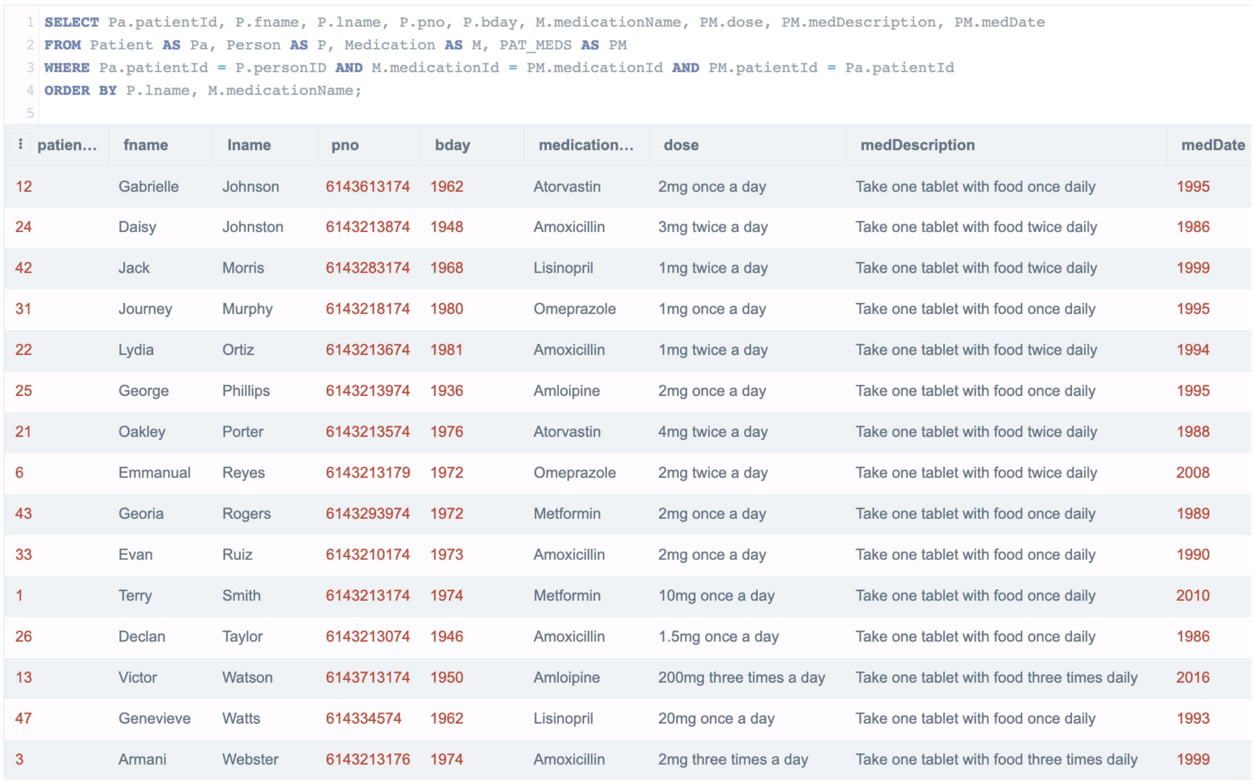
Table PAT_INSUR:
There exists a table called pat_insur for maintaining the specific insurance information for a patient. patientid is an integer which is a foreign key and a primary key and cannot be null which references Patient(patientId). insuranceCompId is an integer which is a foreign key and a primary key and cannot be null which references Insurance(insuranceCompId). Additionally, there are some attributes like memberId which is a variable length string with a max length of 20 that cannot be null for storing the patient's insured id given by their insurance company, groupNo is a variable length string with a max length of 20 that cannot be null for for storing the patient's group number given by their insurance company, planName is a variable length string with a max length of 50 that cannot be null for storing the patient's plan name given by their insurance company, planNo is a variable length string with a max length of 20 that cannot be null for storing the patient's plan number given by their insurance company, startDate as a date data type that cannot be null for storing the start time the patient have had their insurance, and endDate is a date data type that can be null for storing the endtime the patient have had their insurance, if applicable.

A catalog of SELECT SQL Queries with explanations and sample output for each

1. This query can be used to retrieve patient and medication information including dose description and date.

   SELECT Pa.patientId, P.fname, P.lname, P.pno, P.bday, M.medicationName, PM.dose, PM.medDescription, PM.medDate
   FROM Patient AS Pa, Person as P, Medication as M, PAT_MEDS as PM
   WHERE Pa.patientId = P.personID AND M.medicationId = PM.medicationId AND PM.patientId = Pa.patientId
   ORDER BY P.lname, M.medicationName;

   The results this query gives looks like this.



```
SELECT Pa.patientId, P.fname, P.lname, P.pno, P.bday, M.medicationName, PM.dose, PM.medDescription, PM.medDate
FROM Patient AS Pa, Person AS P, Medication AS M, PAT_MEDS AS PM
WHERE Pa.patientId = P.personID AND M.medicationId = PM.medicationId AND PM.patientId = Pa.patientId
ORDER BY P.lname, M.medicationName;
```

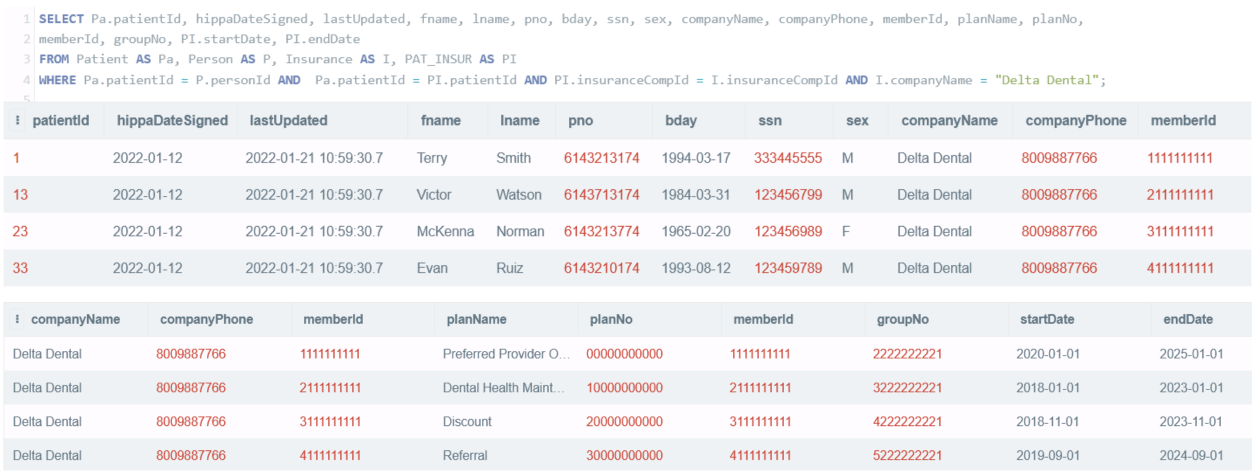
patien...	fname	lname	pno	bday	medication...	dose	medDescription	medDate
12	Gabrielle	Johnson	6143613174	1962	Atorvastin	2mg once a day	Take one tablet with food once daily	1995
24	Daisy	Johnston	6143213874	1948	Amoxicillin	3mg twice a day	Take one tablet with food twice daily	1986
42	Jack	Morris	6143283174	1968	Lisinopril	1mg twice a day	Take one tablet with food twice daily	1999
31	Journey	Murphy	6143218174	1980	Omeprazole	1mg once a day	Take one tablet with food once daily	1995
22	Lydia	Ortiz	6143213674	1981	Amoxicillin	1mg twice a day	Take one tablet with food twice daily	1994
25	George	Phillips	6143213974	1936	Amloipine	2mg once a day	Take one tablet with food once daily	1995
21	Oakley	Porter	6143213574	1976	Atorvastin	4mg twice a day	Take one tablet with food twice daily	1988
6	Emmanual	Reyes	6143213179	1972	Omeprazole	2mg twice a day	Take one tablet with food twice daily	2008
43	Georia	Rogers	6143293974	1972	Metformin	2mg once a day	Take one tablet with food once daily	1989
33	Evan	Ruiz	6143210174	1973	Amoxicillin	2mg once a day	Take one tablet with food once daily	1990
1	Terry	Smith	6143213174	1974	Metformin	10mg once a day	Take one tablet with food once daily	2010
26	Declan	Taylor	6143213074	1946	Amoxicillin	1.5mg once a day	Take one tablet with food once daily	1986
13	Victor	Watson	6143713174	1950	Amloipine	200mg three times a day	Take one tablet with food three times daily	2016
47	Genevieve	Watts	614334574	1962	Lisinopril	20mg once a day	Take one tablet with food once daily	1993
3	Armani	Webster	6143213176	1974	Amoxicillin	2mg three times a day	Take one tablet with food three times daily	1999

2. This query returns patient and insurance data for patients insured by Delta Dental. This can also be slightly changed if the goal was to get patient and insurance information from a different insurance company.

   SELECT Pa.patientId, hippaDateSigned, lastUpdated, fname, lname, pno, bday, ssn, sex, companyName, companyPhone, memberId, planName, planNo, memberId, groupNo, PI.startDate, PI.endDate

FROM Patient as Pa, Person as P, Insurance as I, PAT_INSUR as PI
WHERE Pa.patientId = P.personId AND Pa.patientId = PI.patientId and PI.insuranceCompId = I.insuranceCompId AND I.companyName = "Delta Dental";

The result of executing this query is as shown below.



```
SELECT Pa.patientId, hippaDateSigned, lastUpdated, fname, lname, pno, bday, ssn, sex, companyName, companyPhone, memberId, planName, planNo,
memberId, groupNo, PI.startDate, PI.endDate
FROM Patient AS Pa, Person AS P, Insurance AS I, PAT_INSUR AS PI
WHERE Pa.patientId = P.personId AND  Pa.patientId = PI.patientId AND PI.insuranceCompId = I.insuranceCompId AND I.companyName = "Delta Dental";
```

patientId	hippaDateSigned	lastUpdated	fname	lname	pno	bday	ssn	sex	companyName	companyPhone	memberId
1	2022-01-12	2022-01-21 10:59:30.7	Terry	Smith	6143213174	1994-03-17	333445555	M	Delta Dental	8009887766	1111111111
13	2022-01-12	2022-01-21 10:59:30.7	Victor	Watson	6143713174	1984-03-31	123456799	M	Delta Dental	8009887766	2111111111
23	2022-01-12	2022-01-21 10:59:30.7	McKenna	Norman	6143213774	1965-02-20	123456989	F	Delta Dental	8009887766	3111111111
33	2022-01-12	2022-01-21 10:59:30.7	Evan	Ruiz	6143210174	1993-08-12	123459789	M	Delta Dental	8009887766	4111111111

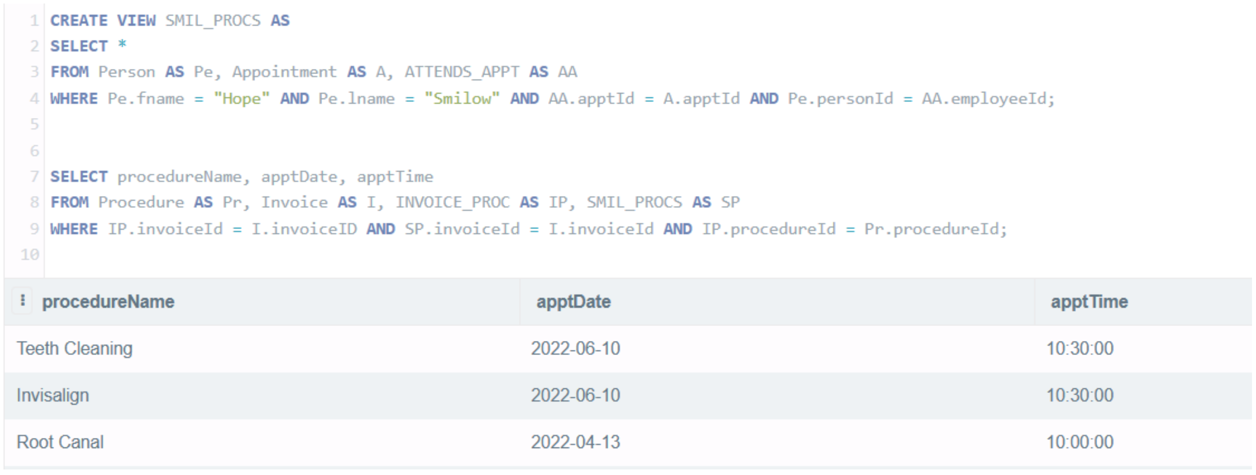
companyName	companyPhone	memberId	planName	planNo	memberId	groupNo	startDate	endDate
Delta Dental	8009887766	1111111111	Preferred Provider O…	00000000000	1111111111	2222222221	2020-01-01	2025-01-01
Delta Dental	8009887766	2111111111	Dental Health Maint…	10000000000	2111111111	3222222221	2018-01-01	2023-01-01
Delta Dental	8009887766	3111111111	Discount	20000000000	3111111111	4222222221	2018-11-01	2023-11-01
Delta Dental	8009887766	4111111111	Referral	30000000000	4111111111	5222222221	2019-09-01	2024-09-01

3. This query generates a list of procedures and dates of service performed by doctor Smilow.) This can be important for determining work activity by a certain medical professional.

   CREATE VIEW SMIL_PROCS AS
   SELECT *
   FROM Person AS Pe, Appointment AS A, ATTENDS_APPT as AA
   WHERE Pe.fname = "Hope" AND Pe.lname = "Smilow" AND AA.apptId = A.apptId AND Pe.personId = AA.employeeId;

   SELECT procedureName, apptDate, apptTime
   FROM Procedure AS Pr, Invoice as I, INVOICE_PROC as IP, SMIL_PROCS as SP
   WHERE IP.invoiceId = I.invoiceID AND SP.invoiceId = I.invoiceId AND IP.procedureId = Pr.procedureId;

   The results of this query are shown here.



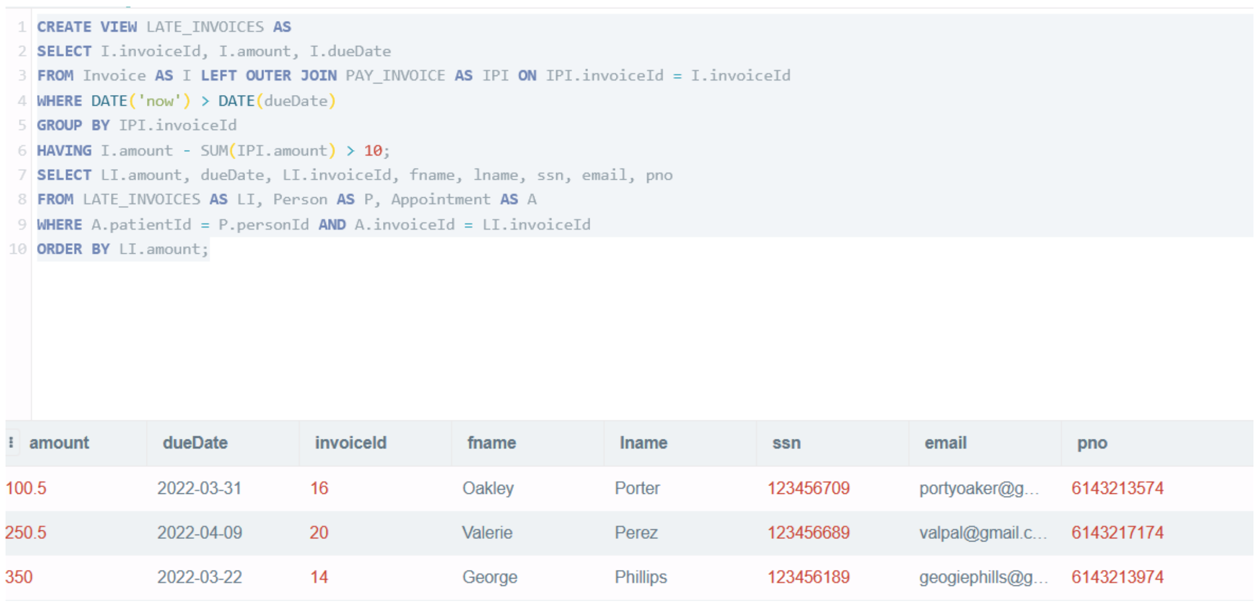
```
CREATE VIEW SMIL_PROCS AS
SELECT *
FROM Person AS Pe, Appointment AS A, ATTENDS_APPT AS AA
WHERE Pe.fname = "Hope" AND Pe.lname = "Smilow" AND AA.apptId = A.apptId AND Pe.personId = AA.employeeId;


SELECT procedureName, apptDate, apptTime
FROM Procedure AS Pr, Invoice AS I, INVOICE_PROC AS IP, SMIL_PROCS AS SP
WHERE IP.invoiceId = I.invoiceID AND SP.invoiceId = I.invoiceId AND IP.procedureId = Pr.procedureId;
```

procedureName	apptDate	apptTime
Teeth Cleaning	2022-06-10	10:30:00
Invisalign	2022-06-10	10:30:00
Root Canal	2022-04-13	10:00:00

4. This query returns a list of past due invoices with patient contact information. Here, past due is defined as over 30 days old with a balance over $10.

   CREATE VIEW LATE_INVOICES AS
   SELECT I.invoiceId, I.amount, I.dueDate
   FROM Invoice AS I LEFT OUTER JOIN PAY_INVOICE AS IPI ON IPI.invoiceId = I.invoiceId
   WHERE DATE('now') > DATE(dueDate)
   GROUP BY IPI.invoiceId
   HAVING I.amount - SUM(IPI.amount) > 10;
   SELECT LI.amount, dueDate, LI.invoiceId, fname, lname, ssn, email, pno
   FROM LATE_INVOICES AS LI, Person as P, Appointment as A
   WHERE A.patientId = P.personId AND A.invoiceId = LI.invoiceId
   ORDER BY LI.amount;

   The results of this query are shown here.



```
CREATE VIEW LATE_INVOICES AS
SELECT I.invoiceId, I.amount, I.dueDate
FROM Invoice AS I LEFT OUTER JOIN PAY_INVOICE AS IPI ON IPI.invoiceId = I.invoiceId
WHERE DATE('now') > DATE(dueDate)
GROUP BY IPI.invoiceId
HAVING I.amount - SUM(IPI.amount) > 10;
SELECT LI.amount, dueDate, LI.invoiceId, fname, lname, ssn, email, pno
FROM LATE_INVOICES AS LI, Person AS P, Appointment AS A
WHERE A.patientId = P.personId AND A.invoiceId = LI.invoiceId
ORDER BY LI.amount;
```

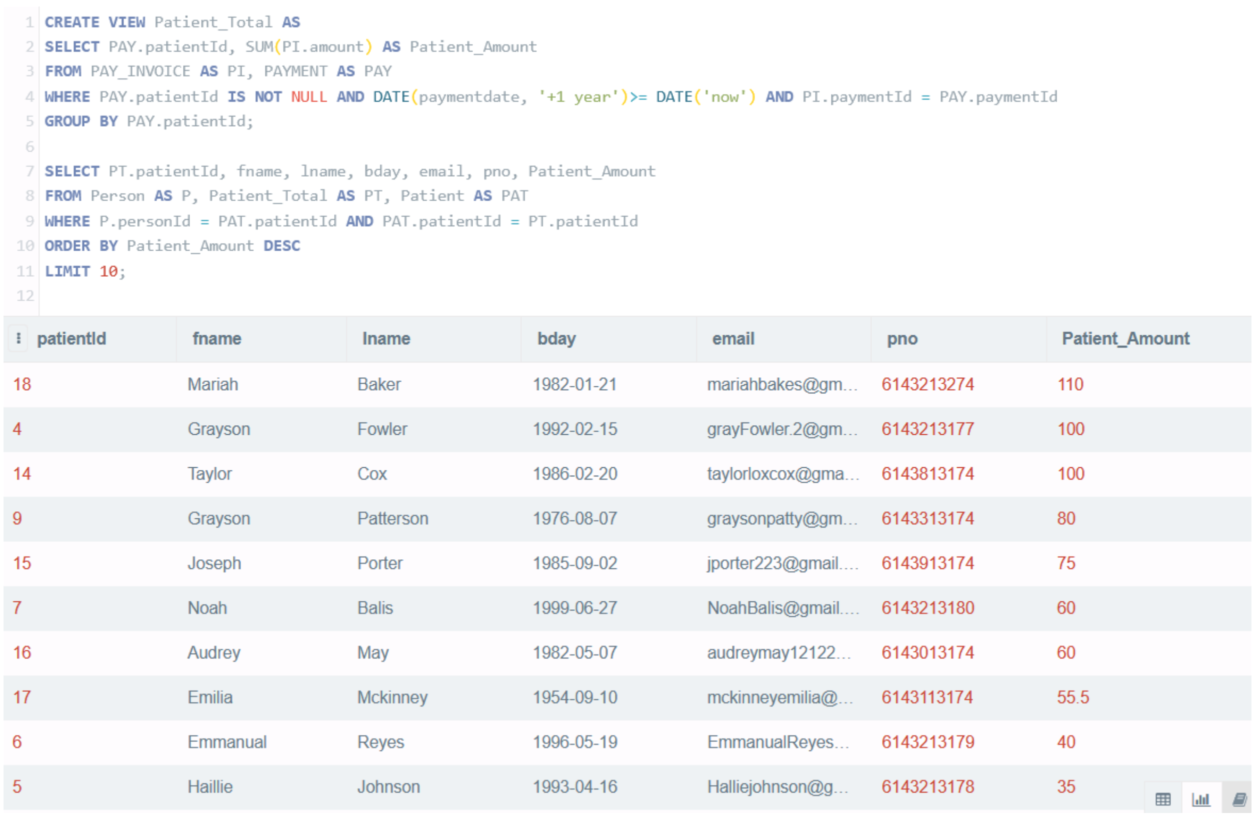
amount	dueDate	invoiceId	fname	lname	ssn	email	pno
100.5	2022-03-31	16	Oakley	Porter	123456709	portyoaker@g...	6143213574
250.5	2022-04-09	20	Valerie	Perez	123456689	valpal@gmail.c...	6143217174
350	2022-03-22	14	George	Phillips	123456189	geogiephills@g...	6143213974

5. This query returns the top 10 patients who brought the most revenue in the past year.

   CREATE VIEW Patient_Total AS
   SELECT PAY.patientId, SUM(PI.amount) AS Patient_Amount
   FROM PAY_INVOICE as PI, PAYMENT AS PAY
   WHERE PAY.patientId IS NOT NULL AND DATE(paymentdate, '+1 year')>= DATE('now')
   AND PI.paymentId = PAY.paymentId
   GROUP BY PAY.patientId;

   SELECT PT.patientId, fname, lname, bday, email, pno, Patient_Amount
   FROM Person as P, Patient_Total as PT, Patient as PAT
   WHERE P.personId = PAT.patientId AND PAT.patientId = PT.patientId
   ORDER BY Patient_Amount DESC
   Limit 10;

   The results of this query are shown here.



```
CREATE VIEW Patient_Total AS
SELECT PAY.patientId, SUM(PI.amount) AS Patient_Amount
FROM PAY_INVOICE AS PI, PAYMENT AS PAY
WHERE PAY.patientId IS NOT NULL AND DATE(paymentdate, '+1 year')>= DATE('now') AND PI.paymentId = PAY.paymentId
GROUP BY PAY.patientId;

SELECT PT.patientId, fname, lname, bday, email, pno, Patient_Amount
FROM Person AS P, Patient_Total AS PT, Patient AS PAT
WHERE P.personId = PAT.patientId AND PAT.patientId = PT.patientId
ORDER BY Patient_Amount DESC
LIMIT 10;
```

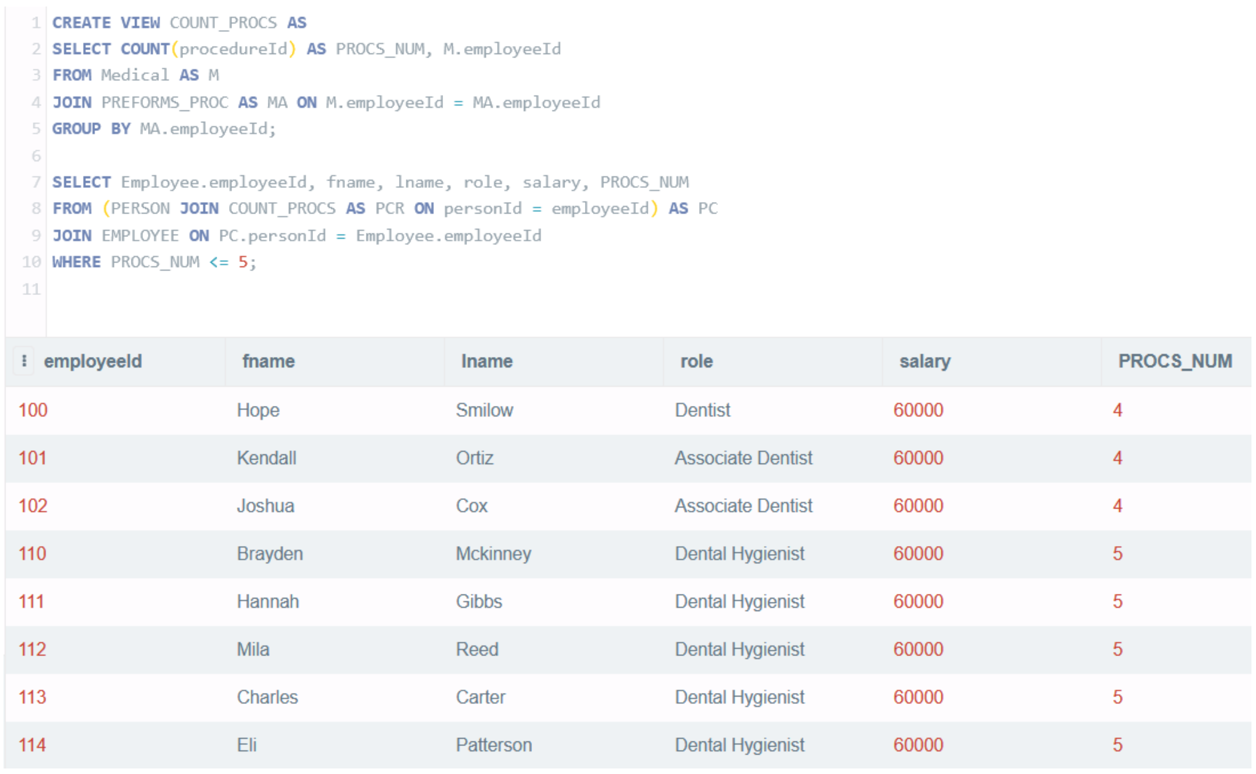
patientId	fname	lname	bday	email	pno	Patient_Amount
18	Mariah	Baker	1982-01-21	mariahbakes@gm...	6143213274	110
4	Grayson	Fowler	1992-02-15	grayFowler.2@gm...	6143213177	100
14	Taylor	Cox	1986-02-20	taylorloxcox@gma...	6143813174	100
9	Grayson	Patterson	1976-08-07	graysonpatty@gm...	6143313174	80
15	Joseph	Porter	1985-09-02	jporter223@gmail....	6143913174	75
7	Noah	Balis	1999-06-27	NoahBalis@gmail....	6143213180	60
16	Audrey	May	1982-05-07	audreymay12122...	6143013174	60
17	Emilia	Mckinney	1954-09-10	mckinneyemilia@...	6143113174	55.5
6	Emmanual	Reyes	1996-05-19	EmmanualReyes...	6143213179	40
5	Haillie	Johnson	1993-04-16	Halliejohnson@g...	6143213178	35

6. This query returns a list of doctors who performed less than 5 procedures this year.

   CREATE VIEW Proc_Count
   AS SELECT L.Emp_ID, COUNT(Pro_ID) AS Num_Procedures
   FROM Licensed AS L JOIN Performs_Procedure AS PP ON L.Emp_ID=PP.Emp_ID
   GROUP BY L.Emp_ID;

   SELECT Emp_ID, F_Name, L_Name, Num_Procedures, Salary, Start_Date, Position
   FROM (Person AS P JOIN Proc_Count AS PC ON P.ID=PC.Emp_ID) AS Per JOIN
   Employee AS E ON Per.ID=E.ID
   WHERE Num_Procedures < 5;

   The results of this query are shown below.



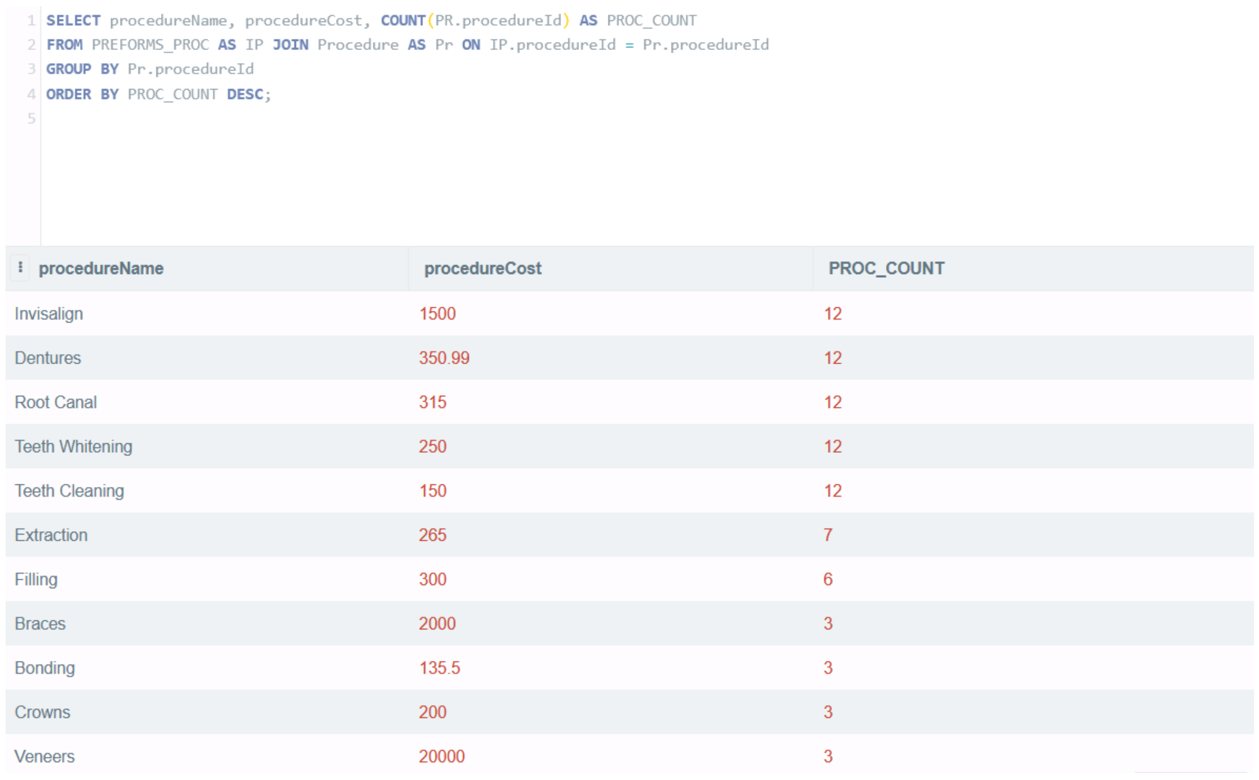
```sql
CREATE VIEW COUNT_PROCS AS
SELECT COUNT(procedureId) AS PROCS_NUM, M.employeeId
FROM Medical AS M
JOIN PREFORMS_PROC AS MA ON M.employeeId = MA.employeeId
GROUP BY MA.employeeId;

SELECT Employee.employeeId, fname, lname, role, salary, PROCS_NUM
FROM (PERSON JOIN COUNT_PROCS AS PCR ON personId = employeeId) AS PC
JOIN EMPLOYEE ON PC.personId = Employee.employeeId
WHERE PROCS_NUM <= 5;
```

employeeId	fname	lname	role	salary	PROCS_NUM
100	Hope	Smilow	Dentist	60000	4
101	Kendall	Ortiz	Associate Dentist	60000	4
102	Joshua	Cox	Associate Dentist	60000	4
110	Brayden	Mckinney	Dental Hygienist	60000	5
111	Hannah	Gibbs	Dental Hygienist	60000	5
112	Mila	Reed	Dental Hygienist	60000	5
113	Charles	Carter	Dental Hygienist	60000	5
114	Eli	Patterson	Dental Hygienist	60000	5

7. This query returns the highest paying procedures, procedure price, and the total number of those procedures performed with highest paying procedures showing at the top of the list.

   SELECT procedureName, procedureCost, COUNT(PR.procedureId) AS PROC_COUNT
   FROM PREFORMS_PROC as IP JOIN Procedure as Pr ON IP.procedureId =
   Pr.procedureId
   GROUP BY Pr.procedureId
   ORDER BY PROC_COUNT DESC;

   The results of this query are shown below.



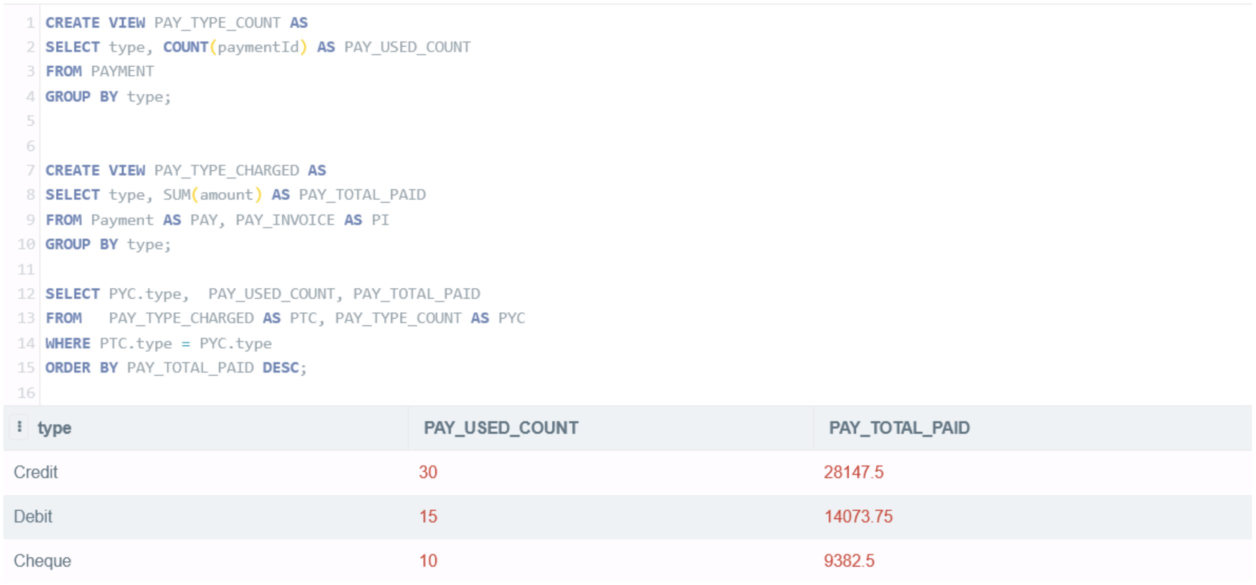
```
SELECT procedureName, procedureCost, COUNT(PR.procedureId) AS PROC_COUNT
FROM PREFORMS_PROC AS IP JOIN Procedure AS Pr ON IP.procedureId = Pr.procedureId
GROUP BY Pr.procedureId
ORDER BY PROC_COUNT DESC;
```

procedureName	procedureCost	PROC_COUNT
Invisalign	1500	12
Dentures	350.99	12
Root Canal	315	12
Teeth Whitening	250	12
Teeth Cleaning	150	12
Extraction	265	7
Filling	300	6
Braces	2000	3
Bonding	135.5	3
Crowns	200	3
Veneers	20000	3

8. This query returns a list of all payment types accepted, number of times each of them was used, and total amount charged to that type of payment.

```
CREATE VIEW PAY_TYPE_COUNT AS
SELECT type, COUNT(paymentId) as PAY_USED_COUNT
FROM PAYMENT
GROUP BY type;

CREATE VIEW PAY_TYPE_CHARGED AS
SELECT type, SUM(amount) as PAY_TOTAL_PAID
FROM Payment as PAY, PAY_INVOICE AS PI
GROUP BY type;

SELECT PYC.type,  PAY_USED_COUNT, PAY_TOTAL_PAID
FROM   PAY_TYPE_CHARGED AS PTC, PAY_TYPE_COUNT AS PYC
WHERE PTC.type = PYC.type
ORDER BY PAY_TOTAL_PAID DESC;
```

The results of this query are shown below.



```
CREATE VIEW PAY_TYPE_COUNT AS
SELECT type, COUNT(paymentId) AS PAY_USED_COUNT
FROM PAYMENT
GROUP BY type;


CREATE VIEW PAY_TYPE_CHARGED AS
SELECT type, SUM(amount) AS PAY_TOTAL_PAID
FROM Payment AS PAY, PAY_INVOICE AS PI
GROUP BY type;

SELECT PYC.type,  PAY_USED_COUNT, PAY_TOTAL_PAID
FROM   PAY_TYPE_CHARGED AS PTC, PAY_TYPE_COUNT AS PYC
WHERE PTC.type = PYC.type
ORDER BY PAY_TOTAL_PAID DESC;
```

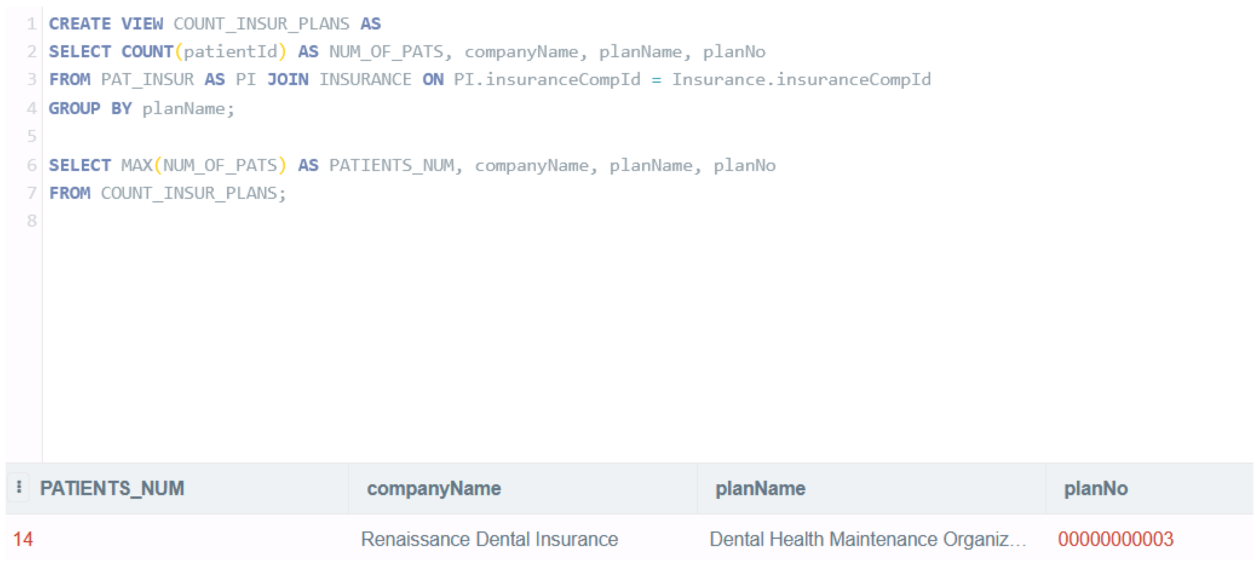
type	PAY_USED_COUNT	PAY_TOTAL_PAID
Credit	30	28147.5
Debit	15	14073.75
Cheque	10	9382.5

9. This query returns the name of the most popular insurance plan currently used by the patient

   CREATE VIEW COUNT_INSUR_PLANS AS
   SELECT COUNT(patientId) AS NUM_OF_PATS, companyName, planName, planNo
   FROM PAT_INSUR AS PI JOIN INSURANCE ON PI.insuranceCompId = Insurance.insuranceCompId
   GROUP BY planName;

   SELECT MAX(NUM_OF_PATS) AS PATIENTS_NUM, companyName, planName, planNo
   FROM COUNT_INSUR_PLANS;

   The results of this query are shown here.



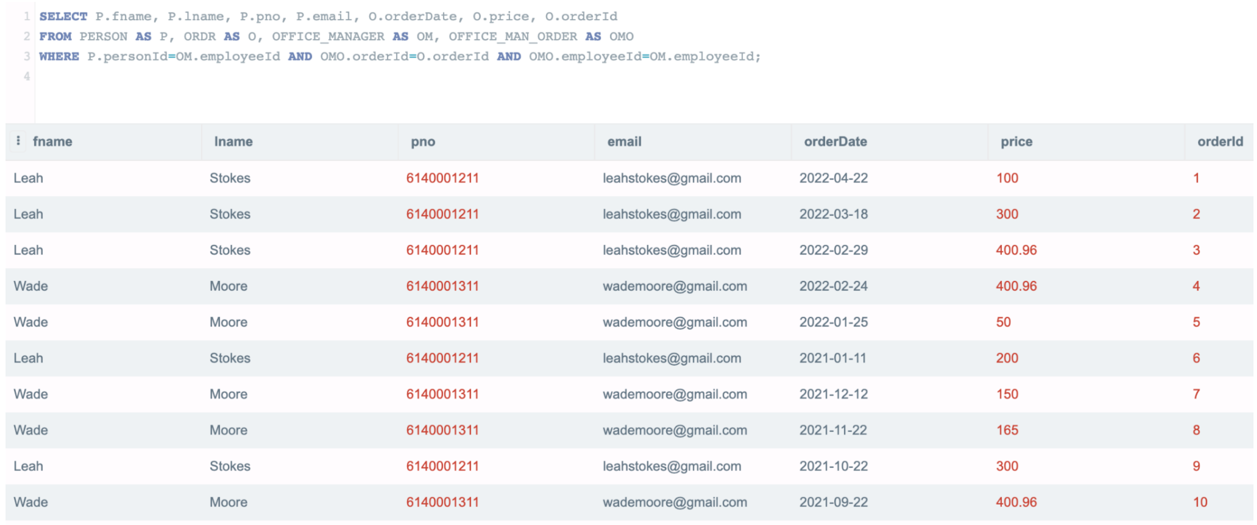
```sql
CREATE VIEW COUNT_INSUR_PLANS AS
SELECT COUNT(patientId) AS NUM_OF_PATS, companyName, planName, planNo
FROM PAT_INSUR AS PI JOIN INSURANCE ON PI.insuranceCompId = Insurance.insuranceCompId
GROUP BY planName;

SELECT MAX(NUM_OF_PATS) AS PATIENTS_NUM, companyName, planName, planNo
FROM COUNT_INSUR_PLANS;
```

PATIENTS_NUM	companyName	planName	planNo
14	Renaissance Dental Insurance	Dental Health Maintenance Organiz...	00000000003

10. This query returns the office manager contact information for the office manager along with orderId and price of each order they placed. This can be helpful for finding who placed what orders with quick access to their contact info.

    SELECT P.fname, P.lname, P.pno, P.email, O.orderDate, O.price, O.orderId
    FROM PERSON AS P, ORDR AS O, OFFICE_MANAGER AS OM,
    OFFICE_MAN_ORDER AS OMO
    WHERE P.personId=OM.employeeId AND OMO.orderId=O.orderId AND
    OMO.employeeId=OM.employeeId;

    The results of this query are shown here.



```sql
SELECT P.fname, P.lname, P.pno, P.email, O.orderDate, O.price, O.orderId
FROM PERSON AS P, ORDR AS O, OFFICE_MANAGER AS OM, OFFICE_MAN_ORDER AS OMO
WHERE P.personId=OM.employeeId AND OMO.orderId=O.orderId AND OMO.employeeId=OM.employeeId;
```

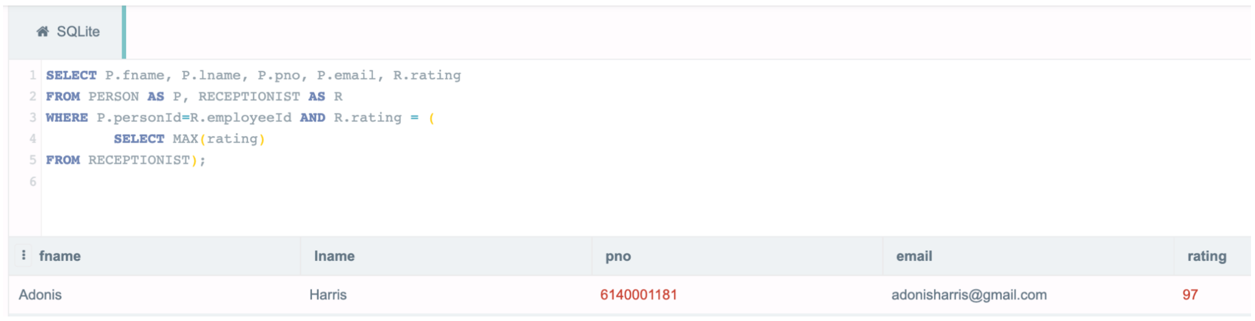
fname	lname	pno	email	orderDate	price	orderId
Leah	Stokes	6140001211	leahstokes@gmail.com	2022-04-22	100	1
Leah	Stokes	6140001211	leahstokes@gmail.com	2022-03-18	300	2
Leah	Stokes	6140001211	leahstokes@gmail.com	2022-02-29	400.96	3
Wade	Moore	6140001311	wademoore@gmail.com	2022-02-24	400.96	4
Wade	Moore	6140001311	wademoore@gmail.com	2022-01-25	50	5
Leah	Stokes	6140001211	leahstokes@gmail.com	2021-01-11	200	6
Wade	Moore	6140001311	wademoore@gmail.com	2021-12-12	150	7
Wade	Moore	6140001311	wademoore@gmail.com	2021-11-22	165	8
Leah	Stokes	6140001211	leahstokes@gmail.com	2021-10-22	300	9
Wade	Moore	6140001311	wademoore@gmail.com	2021-09-22	400.96	10

11. This is a query that finds the highest rated receptionist and displays their contact information along with the rating they have. Could be helpful to quickly find the highest

rated receptionist at the time.

```
SELECT P.fname, P.lname, P.pno, P.email
FROM PERSON AS P, RECEPTIONIST AS R
WHERE P.personId=R.employeeId AND R.rating = (
        SELECT MAX(rating)
        FROM RECEPTIONIST);
```

The results are shown here.



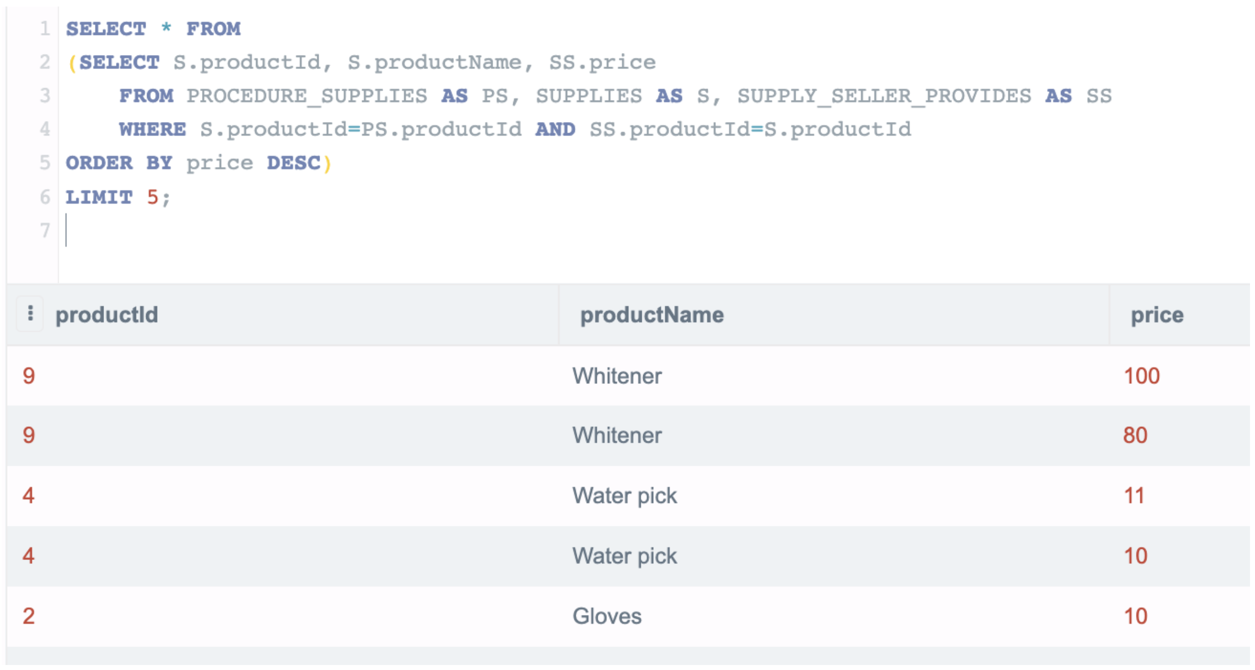
```
SELECT P.fname, P.lname, P.pno, P.email, R.rating
FROM PERSON AS P, RECEPTIONIST AS R
WHERE P.personId=R.employeeId AND R.rating = (
        SELECT MAX(rating)
FROM RECEPTIONIST);
```

fname	lname	pno	email	rating
Adonis	Harris	6140001181	adonisharris@gmail.com	97

12. This query finds the top 5 most expensive items used in procedures. Since there are more than one supplier of each item at different prices, this shows the highest overall prices for any item used in a procedure. Even though the top results are the same item, they are both included because they are supplied from different suppliers at different prices.

```
        SELECT * FROM
(SELECT S.productId, S.productName, SS.price
        FROM PROCEDURE_SUPPLIES AS PS, SUPPLIES AS S,
SUPPLY_SELLER_PROVIDES AS SS
        WHERE S.productId=PS.productId AND SS.productId=S.productId
ORDER BY price DESC)
LIMIT 5;
```

The results of this query are as shown.



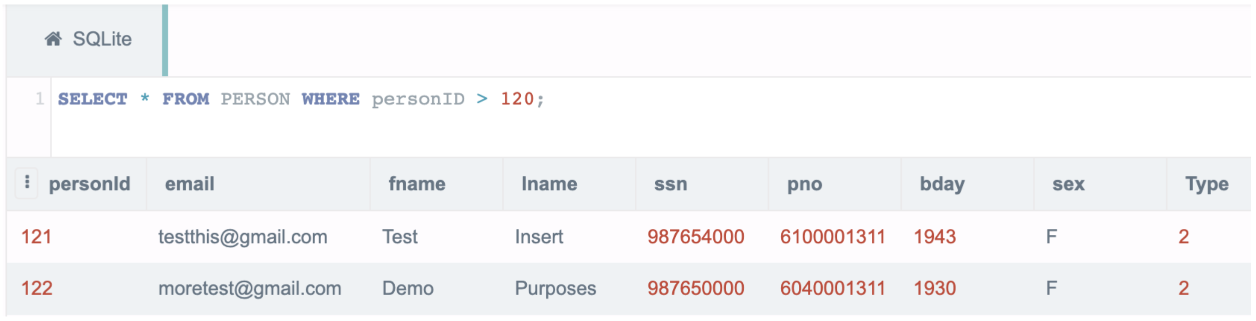
```
SELECT * FROM
(SELECT S.productId, S.productName, SS.price
    FROM PROCEDURE_SUPPLIES AS PS, SUPPLIES AS S, SUPPLY_SELLER_PROVIDES AS SS
    WHERE S.productId=PS.productId AND SS.productId=S.productId
ORDER BY price DESC)
LIMIT 5;
```

productId	productName	price
9	Whitener	100
9	Whitener	80
4	Water pick	11
4	Water pick	10
2	Gloves	10

INSERT and DELETE SQL code and output samples

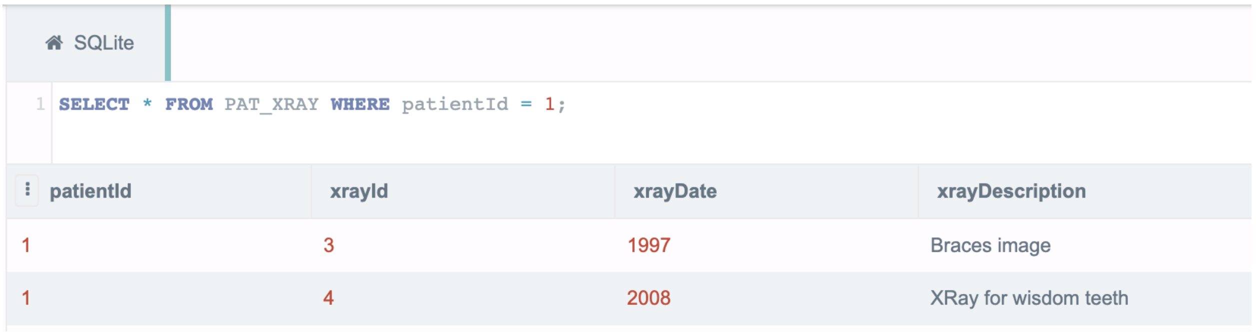
For sample inserts, two data items are added to PERSON and two to PAT_XRAY. These insert queries are shown below with some sample Select queries to show that they were inserted.

INSERT INTO PERSON VALUES(121, "testthis@gmail.com", "Test", "Insert", 987654000, 6100001311,’ 1971-02-26’, 'F', 2);
INSERT INTO PERSON VALUES(122, "moretest@gmail.com", "Demo", "Purposes", 987650000, 6040001311, ‘1948-09-09’, 'F', 2);
INSERT INTO PAT_XRAY VALUES(1, 3, 2020-10-13, "Braces image");
INSERT INTO PAT_XRAY VALUES(1, 4, 2022-11-03, "XRay for wisdom teeth");
INSERT INTO PAT_MEDS VALUES(14, 1009, "2mg once a day", "Take one tablet with food once daily", 2020-03-29);
INSERT INTO PAT_MEDS VALUES(14, 1007, "1mg once a day", "Take one tablet with food once daily", 2020-03-29);



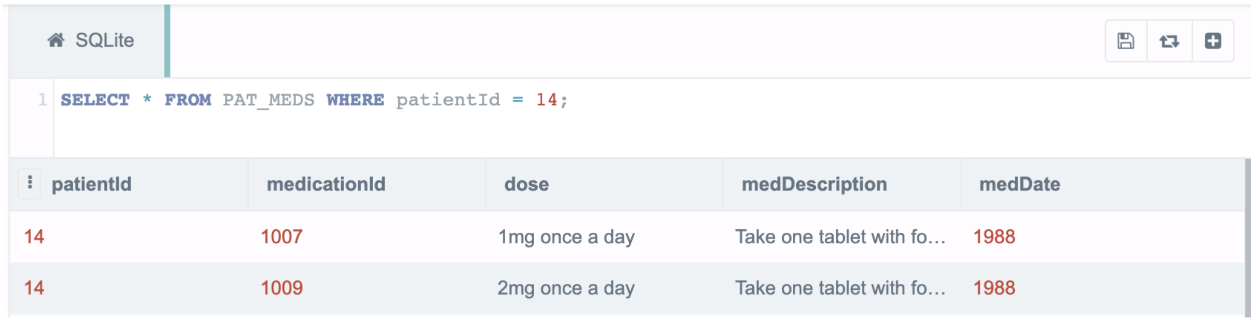
```
SELECT * FROM PERSON WHERE personID > 120;
```

personId	email	fname	lname	ssn	pno	bday	sex	Type
121	testthis@gmail.com	Test	Insert	987654000	6100001311	1943	F	2
122	moretest@gmail.com	Demo	Purposes	987650000	6040001311	1930	F	2



```
SELECT * FROM PAT_XRAY WHERE patientId = 1;
```

patientId	xrayId	xrayDate	xrayDescription
1	3	1997	Braces image
1	4	2008	XRay for wisdom teeth

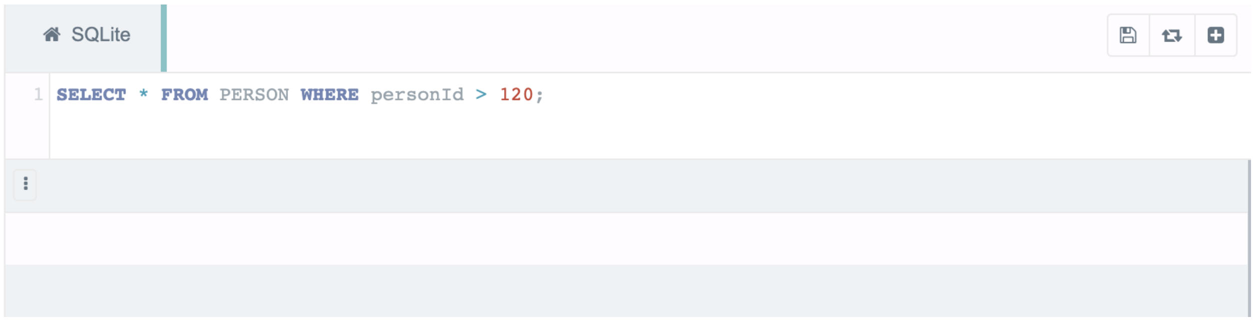


```
SELECT * FROM PAT_MEDS WHERE patientId = 14;
```

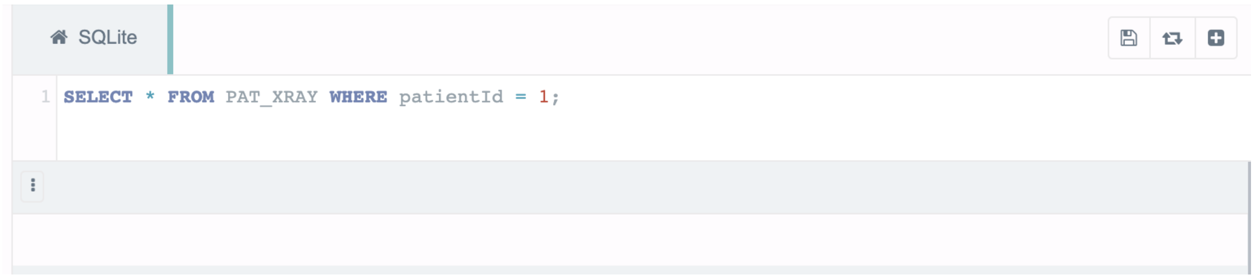
patientId	medicationId	dose	medDescription	medDate
14	1007	1mg once a day	Take one tablet with fo…	1988
14	1009	2mg once a day	Take one tablet with fo…	1988

To delete those inserts, the following delete SQL code segments can be executed.
DELETE FROM PERSON WHERE personId > 120;
DELETE FROM PAT_XRAY WHERE patientId =1;
DELETE FROM PAT_MEDS WHERE patientId = 14;

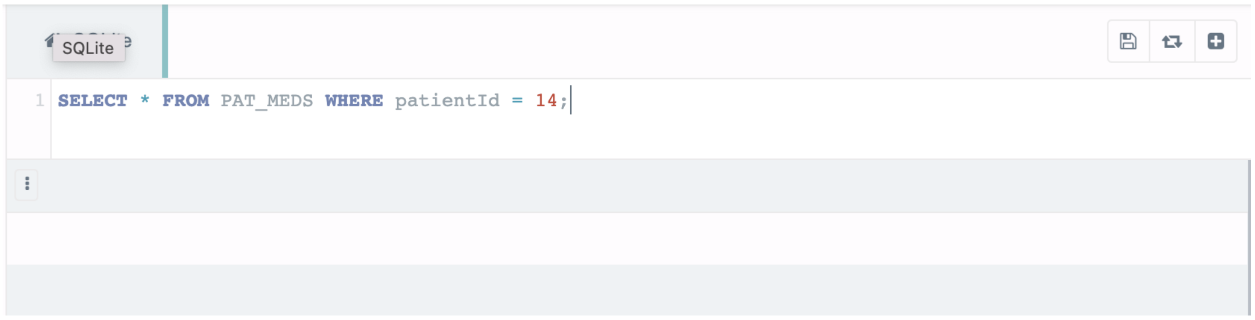
After those commands run, the same select queries from above are ran again, this time producing no results because those data entries have been successfully deleted.



```
SELECT * FROM PERSON WHERE personId > 120;
```



```
SELECT * FROM PAT_XRAY WHERE patientId = 1;
```



```
SELECT * FROM PAT_MEDS WHERE patientId = 14;
```

Two indexes properly explained, including SQL code

1. We can add a tree index to EMPLOYEE.salary. With a large workforce a tree-index would be needed to sift through EMPLOYEE. This way we can search through and find employees with salaries that meet the condition in a more efficient manner. For example, to find all employees with a salary < $100,000

   CREATE INDEX EMPLOYEE_SAL ON EMPLOYEE (salary);

2. A second index that we can implement is adding a hash-index on PROCEDURE for procedureName. Adding this index would speed up operations that include WHERE procedureName=”name_of_procedure”. In this case, using a hash-index is better because we are checking for equality

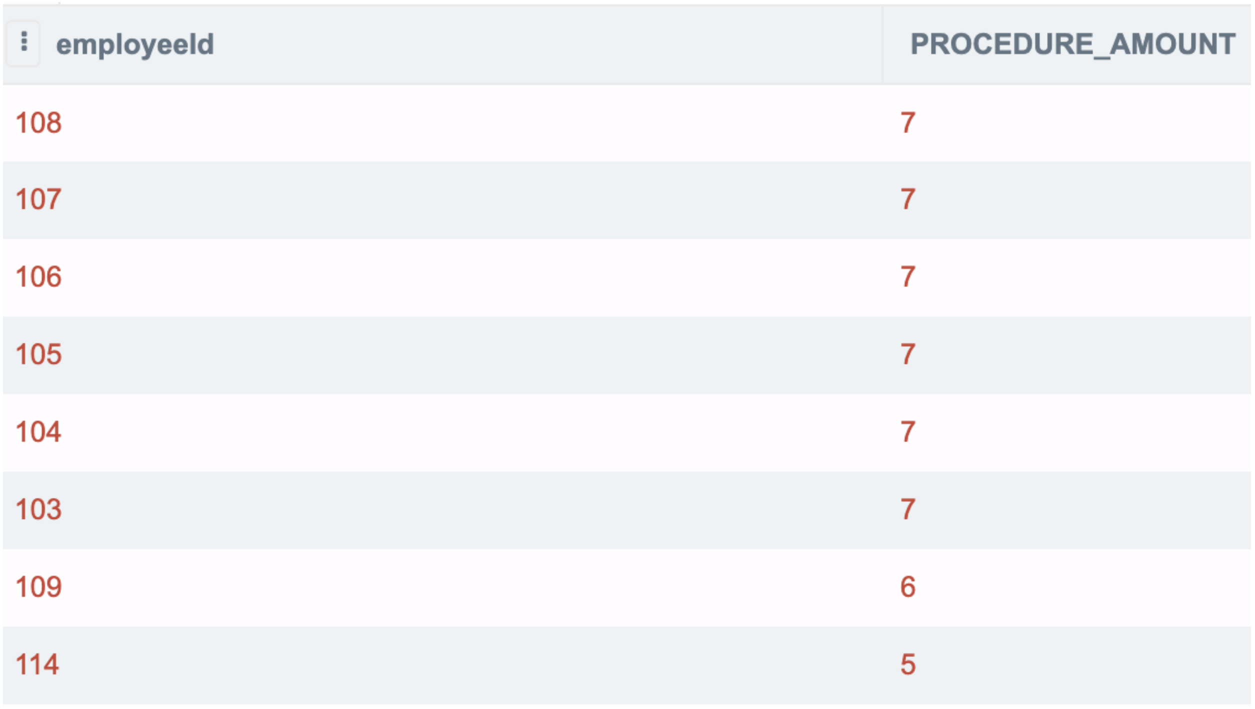
   CREATE INDEX PROCEDURE_NAME ON PROCEDURE (procedureName);

Two views explained, including SQL code and data resulting from the execution

1. Show licensured id’s with the number of procedures they have performed, sorted from max to minimum of amount of procedures done.

CREATE VIEW EMPLOYEE_PROCEDURES
AS SELECT LI.employeeId AS employeeId, COUNT(procedureId) AS PROCEDURE_AMOUNT
FROM LICENSURE AS LI JOIN PREFORMS_PROC AS PR ON
LI.employeeId=PR.employeeId

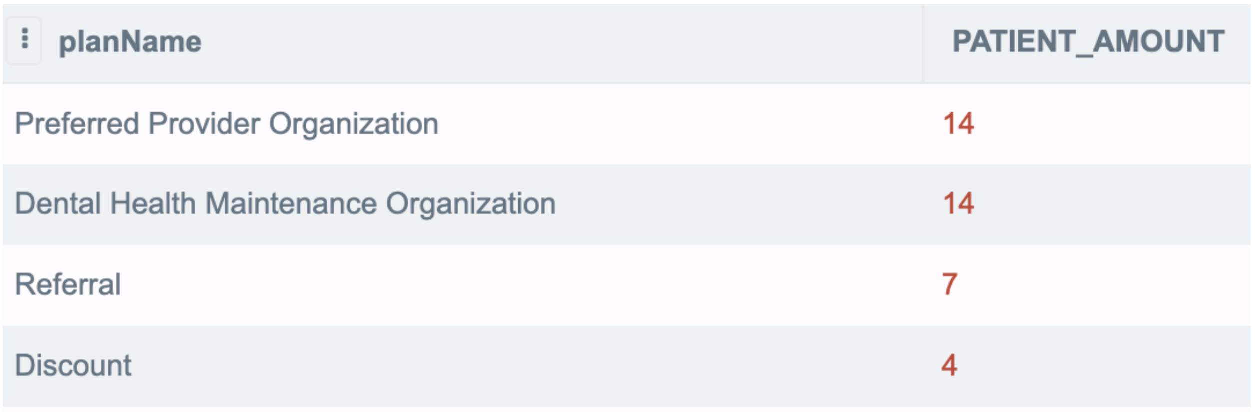
```
GROUP BY LI.employeeId ORDER BY PROCEDURE_AMOUNT DESC;
```



employeeId	PROCEDURE_AMOUNT
108	7
107	7
106	7
105	7
104	7
103	7
109	6
114	5

2. List insurance plans and show the amount of patients using that plan, sorted by maximum to minimum of the amount of patients using that plan

```
CREATE VIEW INSURANCE_PLANS
AS SELECT planName, COUNT(patientId) AS PATIENT_AMOUNT
FROM PAT_INSUR AS PI JOIN INSURANCE AS I ON PI.insurancecompid=I.insurancecompid
GROUP BY planName ORDER BY PATIENT_AMOUNT DESC;
```



planName	PATIENT_AMOUNT
Preferred Provider Organization	14
Dental Health Maintenance Organization	14
Referral	7
Discount	4

Two transactions explained, including SQL code.

1. If the employee gets verification from their license to be a dentist, the salary updates based on the update of the license

```
BEGIN TRANSACTION
        INSERT INTO LICENSURE
                VALUES (12343, '2024-10-24', '2022-03-15', ‘Dentist’, 112);
                IF error THEN GO TO UNDO; END IF;
        UPDATE EMPLOYEE
                SET salary = 224190;
                WHERE employeeId = 112;
                IF error THEN GO TO UNDO; END IF;
        COMMIT;
        GO TO FINISH;
UNDO:
        ROLLBACK;
FINISH:
END TRANSACTION;
```

2. If a new supply order comes in, the suppliers' basic naming info must be added, the procedure supplies must update its inventory, and the order information must be added as well.

```
BEGIN TRANSACTION
        INSERT INTO SUPPLY_ORDER
        VALUES (567, 8675, 300);
        IF error THEN GO TO UNDO; END IF;
        UPDATE PROCEDURE_SUPPLIES
        SET qty = qty + 300;
        WHERE productID = 8675;
        IF error THEN GO TO UNDO; END IF;
        INSERT INTO ORDR(orderId, description, price, orderDate)
        VALUES(567, 'order of needles', 100.00, '2022-04-25');
        IF error THEN GO TO UNDO; END IF;
INSERT INTO SUPPLIES
        VALUES(8675, ‘needles’);
        IF error THEN GO TO UNDO; END IF;
        COMMIT;
        GO TO FINISH;
UNDO:
        ROLLBACK;
FINISH:
END TRANSACTION;
```

Section 3: Team Reports and Graded Checkpoint Documents

Team members contributions

1. Tiernan Farrell - responsible for making corrections and helping in the design process of the EERD. Worked closely with Yusuf to create the schema from the EERD as well as the normalization section. Responsible for the insertQueries.txt and the extraQueries.txt files. Split work with Yusuf on the catalog of SQL Select queries. Completed the insertDeleteQueries.txt and added the results and explanation to the document. Wrote the reflection on the project completion process. Worked with Yusuf to describe revisions made after each checkpoint feedback. Wrote the readMe file that explains how to use the SQL code in the other txt files.

2. Yusuf Basha - Helped make corrections on the design of the EERD diagram. Worked with Tiernan on the relational schema and normalization sections. Responsible for the SimpleQueries.txt. Worked with Tiernan on developing the catalog for supplied SQL Queries with explanations and sample output for each.
3. Hajraan Hussain - Supported other team members on EERD diagram as well as initial schema. Worked and completed relational algebra statements. Responsible for CreateQueries.txt. Created the two view queries as well as the two index queries
4. Andrew Wang - responsible for formatting, table of contents, introduction and summary, and the two transactions.

Reflection on Project Completion Process

The group's main form of communication was through a discord server. This worked well for members in the group to be able to track progress as well as look through old messages. The group met a few times via zoom to complete the design phase. The design phase was the hardest part of this project, because without a fully correct EERD and schema, it was impossible to move forward. The group should have spent more time in the beginning of the semester on the research and design phase of the project. The checkpoints should have been helpful to keep the group on track throughout the semester, but the group failed to produce high quality checkpoint submissions at times. That being said, the group was able to recover from those and collaborate to fix the EERD and the schema. Once that was done, the rest of the work was much easier to complete. Work schedules for the group did not always line up, but through the communication on discord, the group was able to complete work at separate times. Suggestions for future groups include working together early and often on the design. A group of people working on the design together makes it much easier to spot errors and complete a correct EERD.

Description of Feedback received, and revisions completed

The feedback for checkpoint one:

Remove unnecessary specializations or add attributes to them as discussed in class today. Make ERD more uniform, it is very hard to read. Do not connect Procedure with Appointment. Some appointments may be canceled. Payment methods not fully implemented. Consider specializations. They should be disjointed as each payment logically belongs to one category. You can apply more than one payment to the same invoice. Incorrect use of nested multipart attributes for payment types, consider Specializations instead. Employee: Medical (have licensure) and NonMedical. Consider Person generalization and Address entity. Allergy, Medication, Licensure entities-? Insurance can be a brand/type. Patients will have insurance from a particular provider. Research what basic info you would need to store. Various plans may have various coverage for certain procedures.

After CP01, the group had made some revisions detailed in the checkpoint. However, the EERD had several errors relating to participation and cardinality. The group had improperly revised the design process and since a majority of the work had branched off the diagram, the group's progression was slow and full of errors. The group did not address uniforming the EERD diagram, but instead had made it much more chaotic. The group did remove the connection between Procedure and Appointment. Additionally, the group did not correctly address the relationship between procedure and payment, and did not make needed changes for having more than one payment to the same invoice. The group did not properly attend to the subclasses of faculty, which should have medical and nonmedical. However, the group created multiple, unneeded specializations of faculty like dentist, receptionist, and owner. The changes recommended to make person and address entity were not implemented by the group until much later. The group had made some improvements relating to the insurance entity - useful attributes like company and plan were included but improperly designed. The info needed for dentist insurance was not correctly implemented in the second checkpoint.

The feedback for checkpoint two:

ERD is very hard to follow. Use exact notation discussed in class. ERD does not reflect on changes discussed in class and listed in CP01 Feedback announcement. Your team will need to redraw the diagram to clearly show generalizations and specializations, show entities, relationships and attributes using straight lines, using tree structure with branching etc. This ERD is unacceptable, missing PKs. Large number of errors in showing participation and cardinality. Your schema will need to be updated accordingly. Mapping is incomplete, incorrect, steps not shown. PKs and FKs are not shown. RA would not produce proper results as ERD and schema are incorrect and incomplete. Most of your RA statements do not answer questions. Q4 needs more complex questions and appropriate RA to answer them. Your team will need to work together and put much more effort into this if you want to produce a proper project solution.

After CP02, the group redesigned the EERD once again and reorganized some items as well as changing some participation and cardinality. Primary keys were added, but there were still many errors in the diagram. The schema was updated to match the current state of the diagram, but no normalization was done. Primary and foreign keys were shown in the new schema.

The feedback from checkpoint three:

Updated EERD still does not take into account feedback from CP01 announcement, such as making Person superclass, Address entity. Improvements made, but still a very large number of errors. Your team should be able to apply previous feedback and material learned in class to identify these errors and produce a better quality diagram. Service entities are not well defined. Derived attributes are not shown correctly and carried over into the schema. Errors in cardinality and participation. Cardinality cannot be required on both sides, and will create deadlock. Owner and Licensure do not have a primary key. Insurance not connected to correct entities. Schema documentation is improved but still needs many changes after ERD is corrected. FKs should be clearly identified and FK mapping should be included in your schema. No Relational Algebra. No normalization documentation. SQL code is present but is based on a faulty design and will ot produce a working DB. Create table script executes, some issues with keys, such as join tables not having proper keys. Missing semicolon caused an issue in the first Insert but otherwise worked fine. Data is insufficient. Select SQL: Missing half of simple queries and all of extra queries. Only one simple query works from submitted work, one can be fixed, other three have syntax errors or don't populate results. I feel that your team did not put sufficient effort in completion of this assignment.

After CP03, the group worked hard to fully fix the EERD taking into account feedback from all of the checkpoints. The group did add Person as a superclass to Employee and Patient. Also, an Address entity was created with relations to person and address. The group went through and removed where cardinality was included on both sides of a relation to avoid deadlocks. A primary key was added to Licensure and the Owner entity was removed. Insurance was connected to Invoice as well as Address so it could have an Address and apply Payments to an invoice. The relational algebra was added as well as normalization. The schema was updated to clearly show the primary and foreign keys and the steps that were taken are now shown. The SQL code was all rewritten after this checkpoint to match up with the current schema and diagram. Lots of Data was added to the InsertQueries.txt in order to have sufficient database population to show the effects of the Select queries. The rest of the simple queries were completed and tested as well as all of the extra queries.

Marked Project Checkpoints and Worksheets

Project

Checkpoint1

02/07/2022

Yusuf Basha

Hajraan

Hussain Andrew

Wang

Tiernan Farrell

1. List participating team members. Describe how you plan to communicate and approach this project. Any issues related to time differences, technology constraints, or any other issues/concerns?

Yusuf Basha, Hajraan Hussain, Andrew Wang, Tiernan Farrell. We plan to communicate using Discord and work on the project through Google Docs/Excel. There is no technology or time constraints.

2. Conduct additional research and information gathering to identify the initial list of entities and attributes for your DB. Take this phase very seriously. You will need to research how a small business of this type operates, collects, stores, and uses the information. Get a full understanding of how registration, scheduling, and billing processes work. Understand what information needs to be stored for a patient, employee, licensure, insurance, payments, etc. You can use online resources, interview friends and family members, etc. Refer to the project discussion lecture as well. Document all the work completed and resources used.

Initial list of entities: Patients, Employees, Appointments, Payment, Treatments/Procedures, Insurance Plan/Coverage

Collects patient registration forms to store in the database.

https://www.vadhiohiofamilydental.com/files/2019/02/PatientRegistrationForms.pdf

Patients can schedule an appointment via online/in-person/phone. Database n eed to store appointment date & time, patient information (name, email, phone), patient's reason for visit, is
patient new or returning.

https://www.vadhiohiofamilydental.com/contact-us/

For each appointment, billing records contain date, doctor/ supervising doctor/ medical professional, and procedures performed.

Each payment consists of an amount, a date, and a type of payment (cash, check, credit card), and invoicee. There will be one invoice (bill) per appointment. For patients who have insurance coverage, insurance payments are applied first, and the patient is billed for the remainder when Applicable.

https://www.oralhealthgroup.com/blogs/10-ways-to-ensure-successful-and-ethical-dental-insur a nce-billing/

https://www.dentalclaimsupport.com/ultimate-guide-dental-billing

3. Provide at least two additional features to supplement the minimum requirements. List entities, attributes, and relationships you plan to use to accommodate those additional features. Give a brief rationale for why these extra features would be interesting/useful to the stakeholders.

1. Shifts, so we know which employee is working when
2. Patient rating for receptionist, it's receptionist job to be the face of the building so it would be good if patients could rate their experiences
3. Supplies - Dental practice will need supplies for it, entity supply seller, att - price of supplies, quant of supplies, receipts/dates of purchase

4. Create a list of additional requirements and assumptions that you are going to make for your design.

We will have all employees registered into this database

We will also have supplies registered into the database

We should be able to get the receipts of the purchase

We will add a shift attribute to the employees

Assume there is one and only one owner.

Make insurance plan with insurance company and id

Supply seller has company name and location, number id

Supplies are managed by faculty and inventory is kept of supplies

Payment method for patient can be credit or debit

5. Based on the requirements given in the project overview, list the entities to be modeled in this database. For each entity, provide a list of associated attributes. Make sure that your design allows for proper handling of scheduling and billing.

Patient - insurance info, medical history, appointment, HIPPA form - if signed, when signed, allergy info, info still current Insurance Plan - type, patient id,

faculty - faculty id, salary, fname, lname, shifts
Dentist - license

Receptionist - patient rating

Hygienist - license

Dental Assistants - certification

Owner - part of person

Supply seller - name, phone, location

Supplies

Insurance plan: insurance company, phone number, holder's first/last name, insured id

Appointments - date, medical professional, procedures performed, payment - amount, date, type, bill, patient ID,

6. Based on the requirements given in the project overview, what are the various relationships between entities? (For example, “A patient schedules an appointment”).

Dentists, Receptionists, Hygienists, .. are specializations of faculty..

Medical professional treats the patient in the appointments

Patients can schedule an appointment.

Patient has insurance plan

Insurance plan pays for appointments

Owner buys from supply seller, receipt of purchase which includes transaction should have date, price, quantity bought

Faculty members must manage/keep track of current inventory.

7. Give at least four examples of some informal queries/reports that it might be useful for this database to support. Informal means that you can state them in a plain English sentence format without use of any type of prototyping or coding. (For example, “A patient wants to see a list of charges for last year”).

A Dentist wants to see an appointment schedule for the day.

The user wants to see the list of purchase of supplies

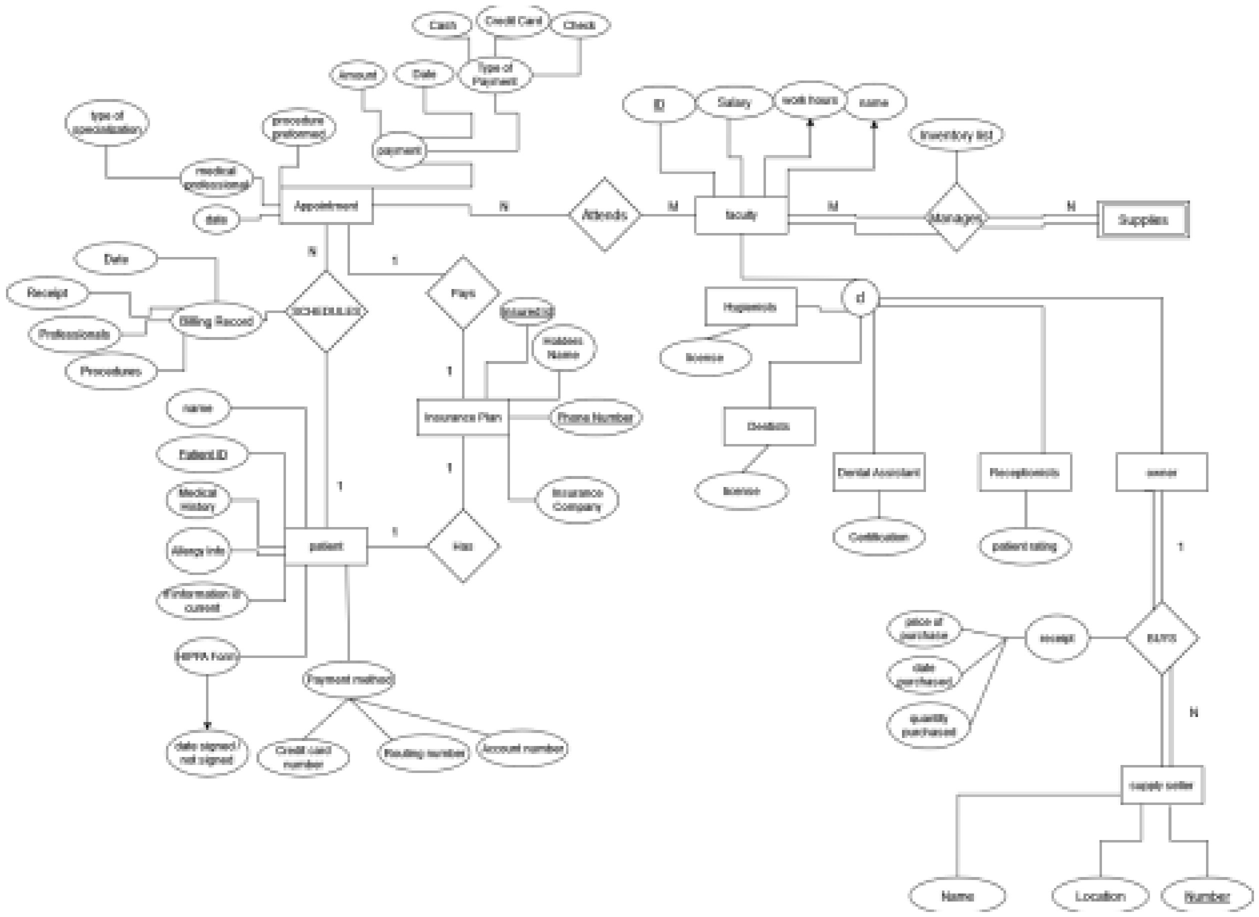
A patient wishes to see date & time of appointment

The receptionists wants to see the patient's insurance plan

8. Provide a full (E)ER diagram for your database. Use Chen’s notation as learned in class. Do not use any other types of notations. Make sure you include ALL the entities, attributes, and relationships to fully cover all the minimum requirements as well as the additional features. Remember that (E)ER model cannot have any standalone entities.

Ensure that you use a proper notation and include a legend. Each entity must have a proper key. All relationships must have cardinality and participation shown. Be mindful about using weak entities. Check for presence and correct identification of all attributes. Use draw.io for your diagram or another drawing tool if preferred. Hand drawn diagrams will not be accepted.

https://drive.google.com/file/d/1an2DNfOQIv7BeHPoYdSslSAMj_BqZD19/view?usp=sharin g



9. Construct a small sample MS Access DB or MS Excel Spreadsheets with sample data to serve as a first prototype for your DB design. It should match your (E)ERD. You should have approx. 5-7 records per entity to start with and will need to add more data when the actual DB is going to be constructed.

https://buckeyemailosu-my.sharepoint.com/:x:/g/personal/farrell_331_buckeyemail_osu_edu / EcSqZ9XVmK9Ggm0DZ_ziroMBU365Jo_GI4W-9X8A4ei2Cg?e=TO5BsU

10. CROSS-CHECK 1: Suppose we want to add a new appointment record to the database. How would we do that given the entities and relationships you have outlined above for a new patient vs for an existing patient? What other information do you need to correctly generate a new appointment? If a patient has multiple allergies and medications, how would your DB store them?

For a new patient, we'll need their names, dob, gender, medical history, known allergies, whether or not they signed the HIPPA form.

For an existing patient, most of the needed information is already known, and all that's required for creating an appointment is their name, phone, email, and reason for their visit. Patients will have separate attributes that list their medical history and known allergies.

11. CROSS-CHECK 2: Confirm that your model supports ALL listed minimum requirements to correctly process billing and payment handling. If it does not, make changes that allow your design to fully support all listed requirements.

CSE 3241 Spring 2022

Project – CP02

Dr. Hope Smilow is very pleased with the progress your team is making and with your initial proposal for her DB. Now it is the time to solidify the logical design of your database and to think how you are going to create queries to help her with information retrieval and making use of the data in her DB.

1. Review feedback provided for your initial design and make necessary changes. Your

(E)ERD should be fully correct and ready to be mapped into the schema. **Remember, if you start with an even partially incorrect ERD and create a model based on that, you will not have a working DB and all your work from this point forward may be faulty. It is extremely important to have a fully correct ERD before starting schema development.**

Link to ER diagram here

2. Map your (E)ER diagram to a relational schema. Indicate all primary and foreign keys and show how they relate to each other. Make sure that you properly follow the mapping algorithm and evaluate and map each element shown in your ERD. Your relational schema must be fully consistent with your ERD. Show and explain all the steps you take in the process. Carefully review primary keys and relations/attribute names and make changes if necessary. Use your best judgment when mapping specializations and try to avoid NULLs.

PATIENT(patient-id, lname, fname, address, phone, medical-history, HIPPA-forms, payment-id) PAYMENT(payment-id, patient-id, amount, date, name, p_type)

CREDIT CARD(cardnum, CVV, Expiration)

CHECK(Account no, Routing no)

DEBIT(cardnum, Account no, routing no)

APPOINTMENT(date, medical-professional, specialization-type, service_type, faculty, patient-id, receipt)

INSURANCE(insured-id, holder-name, phone-number, company, plan)
MEDICATION(date, dose, name, patient-id)

X-RAY(type, date)

FACULTY(faculty-id, role, salary, name, work-hours, f_type)

LICENSURE(faculty-id, expiry-date, type)

DENTAL_ASSISTANT(certification)

RECEPTIONIST(patient-rating)

SUPPLY-SELLER(location, number, name)

RECEIPT(date, quantity, purchase-price, purchaser-id)

ALLERGY(patient-id, name, date-last-occurred)

PROCEDURE(type, date, professional)

3. Given your relational schema, provide the relational algebra to perform the following queries. If your schema cannot provide answers to these queries, revise your ER Model and your relational schema to contain/supply the appropriate information for these queries:

a. Create a list of patients and the medications they currently take

$$\Pi_{\text{name, date, dose}}(\text{PATIENT} * \text{MEDICATION})$$

b. Display patient information for patients who currently have Delta Dental insurance policy. $\sigma_{\text{company = “Delta Dental Insurance”}}(\text{PATIENT} * \text{INSURANCE})$

c. Generate a list of procedures and dates of service performed by doctor Smilow.

$\Pi_{\text{date, type}}(\sigma_{\text{professional = “Dr. Smilow”}}(\text{PROCEDURE}))$

d. 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.

$\Pi_{\text{lname, fname, address, phone}}(\sigma_{\text{amount\_due > = 10} \wedge \text{days\_old >=30}}$

$(\text{PATIENT * SERVICE})$

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

${}_{\text{patient-id}}F_{\textbf{MAX}\text{ amount}}\ (F_{\textbf{Sum}\text{ amount}}\ (\text{PATIENT} \bowtie_{\text{patient-id=ptnt-id}}$

$\text{PAYMENT}))$ f. Create a list of doctors who performed less than 5 procedures this year. $\sigma_{\text{num\_procedures<=5}}(\text{DENTIST})$

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

$\Pi_{\text{cost, type, procedure\_count}}(F_{\textbf{COUNT}\text{ procedure\_count}}(\text{PROCEDURE}))$

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.

$\Pi_{\text{num-times-used, total-amount-charged}}(\text{CHECK})$ U

$\Pi_{\text{num-times-used,total-amount charged}}(\text{CREDIT})$ U

$\Pi_{\text{num-times-used,total-amount-charged}}(\text{CREDIT\_CARD})$ U

$\Pi_{\text{num-times-used,total-amount-charged}}(\text{DEBIT})$ U

$\Pi_{\text{num-times-used,total-amount charged}}(\text{CASH})$

i. Find the name of the most popular insurance plan currently used by the patients. F_max(F_count(insured-id))(INSURANCE)

4. Provide three additional interesting queries in plain English and relational algebra. Each of your queries should include at least one of these:

a. outer joins

b. aggregate function

c. “extra” entities from CP01

1. Retrieve the patient’s name, patient-id, medical history, and insurance company name. This query includes all patients even without a insurance.

$\text{R1} \leftarrow \text{EMPLOYEE} ⟕_{\text{patient-id =insured-id}}$

$\text{INSURANCE}\ \Pi(\text{fname, name, patient-id,}$

$\text{medical-history, company}(\text{R1}))$

2. Find the maximum salary of the faculty and their associated name.

MAX_SALARY ← nameF_max(salary)FACULTY

3. Find the average patient rating.

AVG_RATING ← F_average(patient-rating)RECEPTIONIST

5. Submit a professionally written and well formatted report showing **ALL** your work. **Include your original ERD and feedback from CP01.**

6. Save all your work as you will need to use it for next phase of the project.

CSE 3241 Spring 2022

Project – CP03

Your team is almost done with the logical design of the DB for Dr. Hope Smilow's business. Now you need verify your design, make few last improvements, and process to its implementation using SQL. After completing tasks described here, you should be able to present a working DB populated with sample data to your client. In addition to that, you will also present your client with a list of SQL queries that will allow her to retrieve specific data and create reports.

1. Review feedback provided for CP02 and make necessary changes. Your (E)ERD, relational schema, and relational algebra should be fully correct and consistent. Do not proceed until these tasks are complete. Your entire team needs to work on improving and verifying the design.

EERD link:

https://app.diagrams.net/#G1an2DNfOQlv7BeHPoYdSslSAMj_BqZD19

PATIENT(patientId, lname, fname, address, phone, HIPAALastSigned, xRayDate, insuranceId, payedLastYear)

PAT_PHONE(pId, phone)

PAT_MEDICAL_HISTORY(pId, procedure)

PAT_ALLERGIES(pId, allergen)

OFFICE(officeId, address)

FACULTY(facultyId, lname, fname, workHours, salary, role, bday, sex)

PAYMENT(paymentId, patientId, pType, amount) (last on added, check)

CREDIT_CARD(cardNum, CVV, expiration)

CHEQUE(accountNo, routingNo)

DEBIT(cardNum, accountNo, routingNo)

APPOINTMENT(appointmentId, dateOf, medicalProfessional, totalCost, faculty, patientId)

INSURANCE(memberId, groupNum, insuredId, holderName, phoneNumber, company, policy) //maybe can take out holderName and phoneNumber since Im connecting it to patientId through insuredId not so sure on that

MEDICATION(medicationId, patientId, dose, name)

APPOINTMENT_PAYMENT(paymentId, appointmentId, amount, dateOf)

SERVICE(serviceId, patientId amountDue, dateDue)

APPOINTMENT_SERVICE(serviceId, appointmentId)

SERVICE_PROCEDURES(serviceId, procedureId)

LICENSURE(facultyId, expiryDate, type)

SUPPLY_SELLER(location, number, name)

OWNER(Name, officeId)

SUPPLY_PURCHASE(ownerName, supplierNumber, purchasePrice, dateOf, quantity)

PROCEDURE(type, dateOf, professionalName, cost)

2. Apply process of normalization as learned in class to each table in your relational schema. AT the end of the process all relations in your schema must be in BCNF. • Check that each relation in your schema is in 1NF and if they are not, bring them to 1NF. Explain the process and changes made.

• For each relation schema (table) in your model, indicate the functional dependencies. Make sure to consider all the possible dependencies in each relation and not just the ones from your primary keys.

• For each relation schema in your model, determine the highest normal form of the relation. Apply rules of 2NF, 3NF, and BCNF to each relation, one at a time in the proper order. Explain the process and changes made if any.

• You do not need to update ERD at this point, but you need to update your relational schema to ensure that after this step all relations are in BCNF.

CSE 3241 Spring 2022

3. Given your relational schema, create a text file containing the SQL code to create your database and all the tables in your schema. Populate all tables in your DB with an appropriate number of records to test your queries and produce meaningful results. Recommended number of records per table is between 10-20 depending on table. However, that number can fluctuate

depending on table's role in your DB. Save all your SQL code including INSERT statements used to populate tables with data. If your DB is deleted, you should be able to execute your SQL code as a script in proper order to fully recreate your DB including all tables, constrains, views, and data. Ensure that your code runs and produces correct results in SQLiteOnline (sqliteonline.com) as we will be using that platform to test your code. Save all CREATE / ALTER TABLE STATEMENTS in a file called "CreateQueries.txt" and all applicable INSERT statements in a file called "InsertQueries.txt".

IMPORANT NOTE: For the following questions, if your relational schema cannot provide answers to these queries, revise your (E)ER Model, relational schema, and SQL code in question 3 above to contain the appropriate data for constructing and running all the queries outlined below. On the other hand, if your database contains needed source data but in non-aggregated form, you should NOT revise your model but instead figure out how to aggregate it for the queries!

4. Given your relational schema, provide the SQL to perform the following queries that were previously documented in RA. If your schema cannot provide answers to these queries, revise your ER Model, your relational schema, and your SQL code in question 3 to contain the appropriate information for these queries. These queries should be provided in a plain text file named “SimpleQueries.txt”. Clearly label each query using SQL comments.

   a. Create a list of patients and the medications they currently take. Sort your list by patient’s last name and medication name in alphabetical order. Include other applicable details such as date prescribed and dosage.

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

   c. Generate a list of procedures and dates of service performed by doctor Smilow.

   d. 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.

   e. Find the patients who brought the most revenue in the past year. You can define how many records you want to display in the result of this query.

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

g. Find the highest paying procedures, procedure price, and the total number of those procedures performed. Sort your list with highest paying procedures showing at the top of your list.

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. Find the name of the most popular insurance plan currently used by the patients.

5. For Project Checkpoint 02 question 4, you were asked to come up with three additional interesting queries that your database can provide. Provide the SQL to perform those queries. These queries should be provided in a plain text file named “ExtraQueries.txt”. Clearly label each query using SQL comments. Each of your queries should include at least one of these. Make

sure queries are sufficiently complex and utilize multiple tables and operations in addition to one of the required here:

a. outer joins

b. aggregate function (min, max, average, etc.)

c. “extra” entities from CP01

6. Document work being done for this CP and team member contributions. If your ERD was updates, describe all updates, and include the original ERD from CP02.

7. Once you have completed all your work, create a ZIP archive containing: • A

document showing your most current version of (E)ERD, relational schema, and relational algebra with CP02 feedback addressed. **Submit a professionally written and well formatted report showing ALL your work. Your ERD, schema, RA, and all the written work must be submitted in one document. Do not submit separate files or links.**

- **A binary version of your database**, suitable for opening with the SQLiteOnline application (*sqlite, *.db).
- Text formatted SQL files for questions 3-5:
  - **CreateQueries.txt**
  - **InsertQueries.txt**
  - **SimpleQueries.txt**
  - **ExtraQueries.txt**

  Before submitting your work: Make sure that the information presented in your (E)ERD, relational schema, and all your queries is fully consistent, and all your queries execute correctly and produce expected results! Remember that each of the SQL files should execute as a script, use SQL comments do identify each query, do not use any non-SQL compatible text or syntax in your code. Entire team is responsible to check for presence and correctness of all submitted work.

8. Save all your work as you will need to use it for next phase of the project.