

Final Project Report

Submitted to:

Zina Pichkar

Created by:

Group Number: 3

Jacob Covert

Zhibin Mo

Yifan Dou

Dima Mandryka

CSE 3241 (8575)

The Ohio State University

Columbus, OH

July 29, 2022

Section 1	3
1.a Introduction and Project Summary	3
1.b (E)ERD – Model	4
1.c Relational Schema	6
1.d Relational Algebra for CP02 SELECT queries	12
1.e Normalize schemas:	17
Section 2	29
2.a Table Description	29
2.b Catalog of supplied SQL Queries	35
2.c INSERT and DELETE SQL code samples	42
2.d Two indexes explained, including SQL code	44
2.e Two views explained, including SQL code and execution	45
2.f Two transactions explained, including SQL code	46
Section 3	47
3.a Team contributions	47
3.b Reflection	48
3.c Description of feedback and revisions made	48

Section 1

1.a Introduction and Project Summary

Greetings Dr. Hope Smillow,

Our team has finally completed the database for your dental practice “Smillow Dentistry.” We all hope that our work has met your expectations and hopefully will make your life easier!

The team consists of Jacob Covert, Zhibin Mo, Yifan Dou, and Dima Mandryka. We began our work by doing an immense amount of research as it pertains to finding the latest solutions in information and telecommunications technology as the basis of creating an effective and efficient health system. Afterward, we began constructing the Database that includes every critical component of your business. You will be using this Database to store all the data that you will need and update data that will dynamically optimize itself across the entire schema. Updates can be made by either modifying current data, populating the database with more data as needed, or deleting data that is no longer needed or was wrongly inserted incorrectly. Additionally, we have added extra features that will not only improve the functionality of the Database but will also provide you with an easy-to-use interface to input the right pieces of information into their correct place in the corresponding format.

1.b (E)ERD – Model

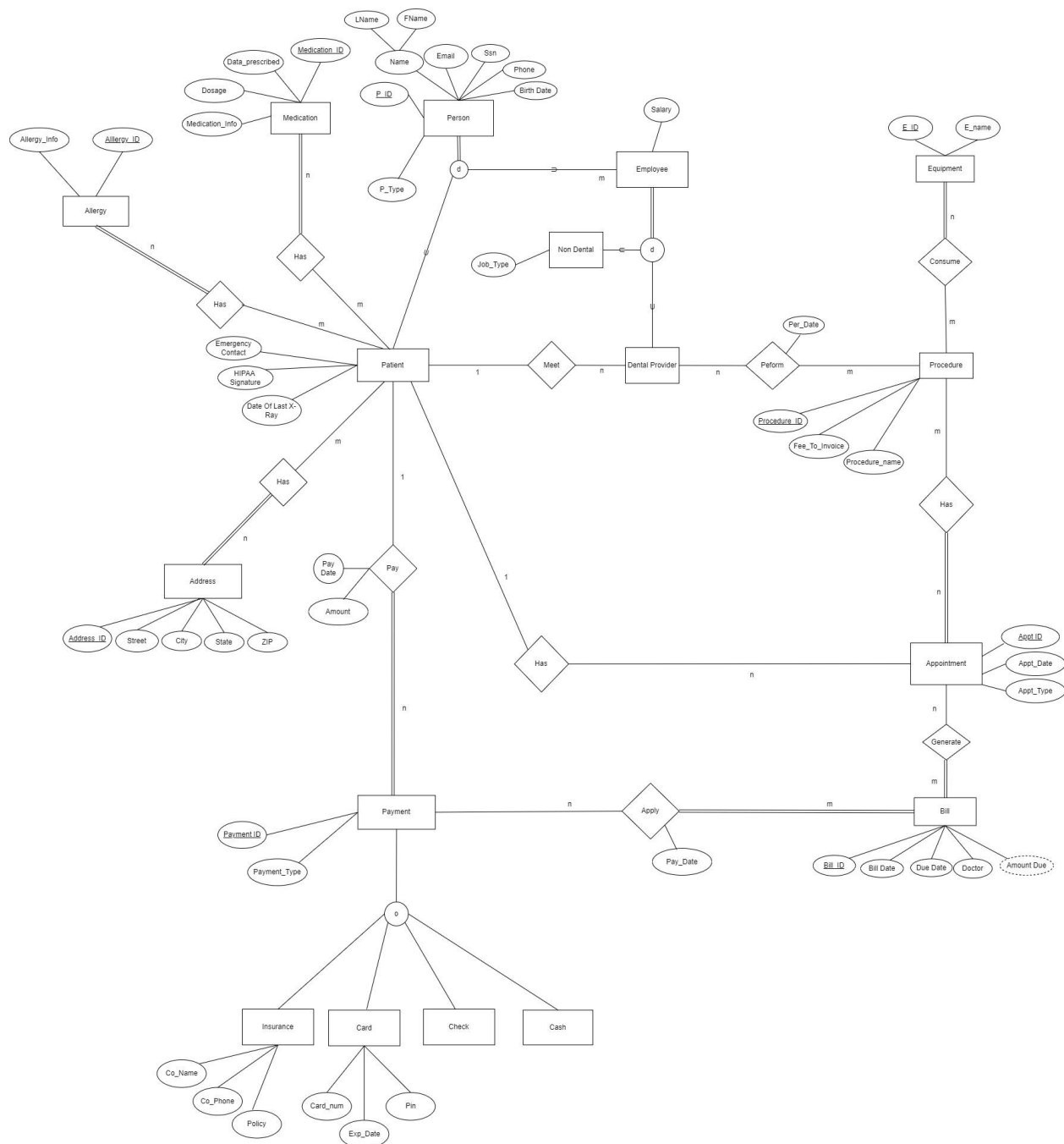


Figure 1: (E)ERD Model

The overall EERD model considers the relations among people, allergy, medication, procedure,

appointment, equipment, bill, payment, and address. Specifically, a person includes patient and employee, and employee includes dental provider and non-dental provider. Payments also have divisions, such as insurance, card, check, and cash.

Patients are the No.1 center as it interacts with most of the entities. Allergy, medication, and address are patient affiliated information and thus they all have many to many relations with patients. In our diagram, one patient may meet multiple dental providers(one to many between patient and dental provider), schedule multiple appointments(1 to many between patient and appointment), many appointments may generate many bills (many to many between appointment and bill), and each patient is supposed to choose one or more payments (1 to many between patient and payment) which are applied to the bills(many to many between patient and bill).

Dental provider is also an important center as it has relation with the rest of the entities. Several dental providers may perform several procedures, a couple of procedures will consume a couple of equipment, and many appointments may have many procedures. (They all appear as many to many relations in the diagram)

Beyond the entities and relations, we also attached appropriate attributes. For all the entities (except the divisions), we assign ID numbers to specify its uniqueness. And we also assign other relevant attributes based on the potentially useful information.

1.c Relational Schema

Person (P_ID, LName, FName, Birth Date, Phone, Email, Ssn, P_Type, Date Of Last X-ray, HIPAA Signature, Emergency_Contact(FK), E_Flag, Salary, Job_Type)

Emergency_Contact is a FK to P_ID in Person relation

Address (Address_ID, Street, City, State, ZIP)

This is a mapping of regular entity types. The relational schema just simply lists all the attributes.

Patient_Has_Address (P_ID(FK), Address_ID(FK))

P_ID is a FK to P_ID in Person relation

Address_ID is a FK to Address_ID in Address relation

Since patient and address have many-to-many relationship, we created another relational schema to demonstrate the relation between patients and their corresponding address. We use the primary key for Patient (P_ID) and that for address(Address_ID) to identify each one.

Medication (Medication_ID, Medication_Info, Date_prescribed, Dosage)

This is a mapping of regular entity types. The relational schema just simply lists all the attributes.

Patient_Has_Medication (P_ID(FK), Medication_ID(FK))

P_ID is a FK to P_ID in Person relation

Medication_ID is a FK to Medication_ID in Medication relation

Since patient and medication have many-to-many relationship, we created another relational schema to demonstrate the relation between patients and the medications. P-ID is the primary key of Patient and Medication_ID is the primary key of Medication. We use these two foreign attributes to identify the pairs.

Allergy (Allergy_ID, Allergy_Info)

This is a mapping of regular entity types. The relational schema just simply lists all the attributes.

Patient_Has_Allergy (P_ID(FK), Allergy_ID(FK))

P_ID is a FK to P_ID in Person relation

Allergy_ID is a FK to Allergy_ID in Allergy relation

Since patient and allergy have many-to-many relationship, we created another relational schema to demonstrate the relation between patients and the allergies. P-ID is the primary key of Patient and Allergy_ID is the primary key of Medication. We use these two foreign attributes to identify the pairs.

Payment (Payment_ID, Amount, Pay_date, Payment_Type, P_ID(FK))

P_ID is a FK to P_ID in Person relation

We first map the regular entity type and list the attributes : Payment_ID and Payment_Type.

Then because patient and payment have 1-to-many relationship, we add the primary key of patient (P_ID) to identify the pair as a foreign key, and add the attributes of the 'pay' relationship (Amount and Pay_date) to payment.

Insurance (Payment_ID(FK), Co_name, Co_phone, Policy)

Payment_ID is a FK to Payment_ID in Payment relation

Insurance is one of many types of Payment that the patient could use. We list the direct attributes of the Insurance entity: the Co_name, Co_phone, and the Policy and we add a foreign key Payment_ID that is a primary key for the Payment entity that will help us track the payment method used per patient.

Card (Payment_ID(FK), Card_num, Exp_date, Pin)

Payment_ID is a FK to Payment_ID in Payment relation

Card is one of many Payment types that the patient could use. We list the direct attributes of the Card entity: the Card_num, Exp_date, and the Pin and we add a foreign key Payment_ID which is a primary key for the Payment entity that will help us track the payment method used per patient.

Check (Payment_ID(FK))

Payment_ID is a FK to Payment_ID in Payment relation

Check is one of many Payment types that the patient could use. We list the only attribute of the Check entity Payment_ID which is a foreign key in this instance since it's a primary key of the Payment entity.

Cash (Payment_ID(FK))

Payment_ID is a FK to Payment_ID in Payment relation

Cash is one of many Payment types that the patient could use. We list the only attribute of the Cash entity Payment_ID which is a foreign key in this instance since it's a primary key of the Payment entity.

Bill (Bill_ID, Bill_Date, Due_Date, Doctor(FK), Amount Due)

Doctor is a FK to P_ID in Person relation

The Bill entity consists of attributes Bill_ID, Bill_Date, Due_Date, Doctor, and Amount Due. Bill_ID is a primary key of the relation because it's the primary identifier of the Bill while Doctor is a foreign key of the doctor who performed the procedure on the patient that we got from using P_ID which is a primary key of the Person relation. It's our systematic approach to giving each person a unique key.

Payment_Apply_Bill (Payment_ID(FK), Bill_ID(FK), Pay_Date)

Payment_ID is a FK to Payment_ID in Payment relation

Bill_ID is a FK to Bill_ID in Bill relation

Created a separate entity of Payment_Apply_Bill since Payment and Bill have a many-to-many relation with Payment_ID labeled as a foreign key from Payment entity and Bill_ID also a foreign key coming from Bill entity and adding a Pay_Date as an attribute of the relation.

Appointment (Appt_ID, Appt_Date, Appt_Type, P_ID(FK), Bill_ID(FK))

P_ID is a FK to P_ID in Person relation

Bill_ID is a FK to Bill_ID in Bill relation

The Appointment entity is generated from foreign keys coming from the primary key of Person relation and primary key of Bill relation. We also added a direct primary key in Appt_ID as the main unique identifier with Appt_Date and Appt_Type as other attributes of the Appointment entity.

Procedure (Procedure_ID, Procedure_name, Fee_To_Invoice, Bill_ID(FK))

Bill_ID is a FK to Bill_ID in Bill relation

The primary key of Procedure is Procedure_ID, and the foreign key is Bill_ID we derived from the Bill entity in which Bill_ID functions as a primary key in. We used the Bill_ID as a foreign key because Procedure has a relation with the Appointment entity which ultimately generates the Bill. Other attributes are Procedure_name and Fee_To_Invoice.

Appt_Has_Procedure (Appt_ID(FK), Procedure_ID(FK))

Appt_ID is a FK to Appt_ID in Appointment relation

Procedure_ID is a FK to Procedure_ID in Procedure relation

Since we have a many-to-many relation between Appointment and Procedure entities, we used their primary keys to create foreign keys for the relation in Appt_ID coming from Appointment and Procedure_ID which came from Procedure.

Dental_provider_Perform_Procedure (Provider_ID(FK), Procedure_ID(FK), Per_date)

Provider_ID is a FK to P_ID in Person relation

Procedure_ID is a FK to Procedure_ID in Procedure relation

We derived a relation of the Dental Provider and Procedure with Procedure_ID as a foreign key from the Procedure entity. We also added the Provider_ID as the foreign key which is a primary key and a unique identifier of P_ID in the Person entity. This labels every person in the database with a unique number that can be used for easier retrieval if needed.

Equipment (E_ID, E_name)

A simple entity with direct attributes E_name and E_ID acting as a primary key.

Procedure_Consume_Equipment (Procedure_ID(FK), E_ID(FK))

Procedure_ID is a FK to Procedure_ID in Procedure relation

E_ID is a FK to E_ID in Equipment relation

Since Procedure and Equipment have a many-to-many relationship, we created a separate entity with Procedure_ID as a foreign key that we got from Procedure and E_ID as another foreign key from the Equipment relation.

1.d Relational Algebra for CP02 SELECT queries

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

$$\sigma_{\text{GETDATE}() \leq \text{Date_Prescribed}}(\text{Patient_Has_Medication} \bowtie_{\text{P_ID}=\text{P_ID}} \text{Person})$$

We use the Equi-Join to combine the tuples of Patient_Has_Medication and those of Person to get a list of all the patients and medications. Then we use a select operation to make sure the medications are taken currently.

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

$$\pi_{(\text{LName}, \text{FName}, \text{Phone}, \text{Email}, \text{ssn})}(\sigma_{\text{policy} = \text{'Delta Dental'}}(\text{Insurance} \bowtie_{\text{Payment_ID}=\text{Payment_ID}} \text{Person}))$$

We first use Equi-join to get a list of the patients and their insurance. Then we select the 'Delta Dental' insurance and list the last name, first name, phone, email and ssn of the patients.

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

Smilow.

$$\pi_{(\text{Procedure_Name}, \text{Per_Date})}(\sigma_{\text{LName} = \text{"Smilow"}}(\text{Dental_provider_Perform_Procedure} \bowtie_{\text{P_ID} = \text{P_ID}} \text{Person}) \bowtie_{\text{Procedure_ID} = \text{Procedure_ID}} \text{Procedure})$$

We first used Equi-Join to combine Dental_provider_Perform_Procedure and Person, and select the last name ‘Smilow’ and get all the Procedure_IDs related to Dr. Smilow. Then we used the procedure_ID to extract tuples from Procedure. Finally, we list the procedures’ names and dates.

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.

$$\text{Past_Due} \leftarrow \sigma_{(\text{DATEDIFF}(\text{day}, \text{GETDAY}(), \text{Due_Date}) > 30)}(\sigma_{\text{Sum_Fee_To_Invoice} - \text{Sum_Amount} > 10}(\text{Payment}))$$

$$\text{Person_Past_Due} \leftarrow \pi_{(\text{LName}, \text{FName}, \text{Email}, \text{Phone})}(\text{Past_Due} \bowtie_{\text{Appt_ID} = \text{Appt_ID}} \text{Appointment} \bowtie_{\text{P_ID} = \text{P_ID}} \text{Patient})$$

We first select all the due over 30 days with a balance over 10 dollars from Payment.

Then we use all the Appt_ID from the table to match the P_ID in Appointment.

And we use Equi-Join between the result table and Patient table.

Finally we list the last name, first name, email, and phone of the patients.

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

$$\text{Amount_From_Patient} \leftarrow \text{P_ID} \text{SUM}_{\text{Amount}} (\sigma_{\text{DATEDIFF}(\text{day}, \text{GETDAY}(), \text{Pay_date}) > 365}(\text{Payment}))$$

$$\text{Most_Amount} \leftarrow \text{P_ID} \text{MAX}_{\text{Sum_Amount}} (\text{Amount_From_Patient} \bowtie_{\text{P_ID} = \text{P_ID}} \text{Person})$$

We first select all the payments paid in the past year. Then we calculate the summation of the payment by each patient. The generated table is called 'Amount_From_Patient'.

Then from the table Amount_From_Patient we find the maximum amount.

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

$$\text{Procedure_Count} \leftarrow \text{P_ID} \text{COUNT}_{\text{P_ID}} (\text{Procedure} \bowtie \sigma_{\text{YEAR}(\text{GETDATE}) = \text{YEAR}(\text{Per_date})} \text{Dental_provide_Perform_Procedure})$$

$$\text{Doctor} \leftarrow \pi_{(\text{LName}, \text{FName})} (\sigma_{(\text{Count_P_ID} < 5)} (\text{Procedure_Count} \bowtie_{\text{P_ID} = \text{P_ID}} \text{Person}))$$

We first use Equi-Join to select all the procedures conducted in this year.

And then we count the number of procedures of each doctor, and store the results as 'Procedure_Count'.

We then use Equi-Join of Procedure_Count and Person to find more info about the doctor, and filter the results to produce the list of doctors who performed less than 5 procedures, and finally list the last and first names.

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

Procedure_Count \leftarrow $\text{Procedure_Name} \text{ COUNT}_{\text{Procedure_Name}}$ (**Procedure**)

Highest_Pay \leftarrow $\text{Procedure_Name} \text{ MAX}_{\text{Fee_To_Invoice}}$ (**Procedure**)

Result \leftarrow $\pi_{(\text{Procedure_Name}, \text{Fee_To_Invoice}, \text{Count_Procedure_Name})}$ (**Procedure** \bowtie

$\text{Procedure_Name} = \text{Procedure_Name}$ **Procedure_Count** \bowtie $\text{Procedure_Name} = \text{Procedure_Name}$

Highest_Pay)

We first created a new table called 'Procedure_Count' to store the count of procedure_name by per procedure name.

Then we created another new table called 'Highest_Pay' to store a list of fee_to_Invoice by per procedure_name sorted from high to low.

Finally we use Equi-Join to combine the tuples of Procedure, Procedure_Count, and Highest_Pay by the Procedure_Name, and list the procedures' name, fee to invoice and count procedure name.

- 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_Payment \leftarrow $\text{Payment_Type} \text{ COUNT}_{\text{Payment Type}}$ (**Payment**)

$$\begin{aligned} \text{Total} &\leftarrow \text{Payment_Type} \text{SUM}_{\text{Amount}}(\text{Payment}) \\ \text{Result} &\leftarrow \pi_{\text{Payment_Type}, \text{Count_Payment_Type}, \text{Sum_Payment_Type}}(\text{Count_Payment} \\ &\bowtie_{\text{Payment_Type} = \text{Payment_Type}} \text{Total}) \end{aligned}$$

We first created a table Count_Payment to store the count of each payment type.

Then we created another table Total to store the summation of the count of each payment type.

Finally we use Equi-Join to combine these two tables and list the payment type, count payment type, sum payment type.

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

$$\begin{aligned} \text{Count_Policy} &\leftarrow \text{Policy} \text{COUNT}_{\text{Policy}}(\text{Insurance}) \\ \text{Max_Count} &\leftarrow \text{Policy} \text{MAX}_{\text{Count_Policy}}(\text{Count_Policy}) \\ \text{Result} &\leftarrow \pi_{(\text{Policy})}(\text{Insurance} \bowtie_{\text{Policy} = \text{Policy}} \text{Max_Count}) \end{aligned}$$

We first list the count of each policy in Insurance.

Then we find the most frequent policy from the first step.

Finally, we combine the Insurance and the second new table, and list the policy name.

1.e Normalize schemas:

Person

Person (P_ID, LName, FName, Birth Date, Phone, Email, Ssn, P_Type, Date Of Last X-ray, HIPAA Signature, Emergency_Contact(FK), E_Flag, Salary, Job_Type)

FD:

(P_ID, Emergency_Contact(FK)) → (LName, FName, Birth Date, Phone, Email, Ssn, P_Type, Date Of Last X-ray, HIPAA Signature, E_Flag, Salary, Job_Type)

(Ssn) → (LName, FName, Birth Date, Phone, Email)

Person is in 2NF, since the relation doesn't have either nested relations or non-atomic attributes (satisfies 1NF) and every non-key attribute is fully dependent on the key (satisfies 2NF).

And there exists attributes which are transitive dependency (not satisfies 3NF).

To convert to BCNF:

Person (P_ID, Ssn(FK), Phone, Email, P_Type, Date Of Last X-ray, HIPAA Signature, Emergency_Contact(FK), E_Flag, Salary, Job_Type)

Person_Info (Ssn, LName, FName, Birth Date, Ssn(FK))

Patient (P_ID(FK), Date Of Last X-ray, HIPAA Signature, Emergency_Contact(FK))

Employee (P_ID(FK), Salary, Job_Type, Ssn(FK))

Now Person is in BCNF as we create a separate table Person_Info acting as a superkey to identify the person attributes so now it is in 3rd Normal Form and satisfies BCNF.

Address

Address (Address_ID, Street, City, State, ZIP)

FD:

$(\text{Address_ID}) \rightarrow (\text{Street, City, State, ZIP})$

Address is in BCNF, because this relation doesn't have nested relations or non-atomic attributes (satisfies 1NF), every non-key attribute is fully functional dependency (satisfies 2NF) and has no transitive dependency (satisfies 3NF), and every determinant is a candidate key (satisfies BCNF)

Patient_Has_Address

Patient_Has_Address (P_ID(FK), Address_ID(FK))

FD:

None

Patient_Has_Address is in BCNF, because this relation doesn't have nested relations or non-atomic attributes (satisfies 1NF), every non-key attribute is fully functional dependency (satisfies 2NF) and has no transitive dependency (satisfies 3NF), and every determinant is a candidate key (satisfies BCNF)

Medication

Medication (Medication_ID, Medication_Info, Date_prescribed, Dosage)

FD:

(Medication_ID) \rightarrow (Medication_Info, Date_prescribed, Dosage)

Medication is in BCNF, because this relation doesn't have nested relations or non-atomic attributes (satisfies 1NF), every non-key attribute is fully functional dependency (satisfies 2NF) and has no transitive dependency (satisfies 3NF), and every determinant is a candidate key (satisfies BCNF)

Patient_Has_Medication

Patient_Has_Medication (P_ID(FK), Medication_ID(FK))

FD:

None

Patient_Has_Medication is in BCNF, because this relation doesn't have nested relations or non-atomic attributes (satisfies 1NF), every non-key attribute is fully functional dependency (satisfies 2NF) and has no transitive dependency (satisfies 3NF), and every determinant is a candidate key (satisfies BCNF)

Allergy

Allergy (Allergy_ID, Allergy_Info)

FD:

$(\text{Allergy}) \rightarrow (\text{Allergy_Info})$

Allergy is in BCNF because this relation doesn't have nested relations or non-atomic attributes (satisfies 1NF), every non-key attribute is fully functional dependency (satisfies 2NF) and has no transitive dependency (satisfies 3NF), and every determinant is a candidate key (satisfies BCNF)

Patient_Has_Allergy

Patient_Has_Allergy (P_ID(FK), Allergy_ID(FK))

FD:

None

Patient_Has_Allergy is in BCNF, because this relation doesn't have nested relations or non-atomic attributes (satisfies 1NF), every non-key attribute is fully functional dependency (satisfies 2NF) and has no transitive dependency (satisfies 3NF), and every determinant is a candidate key (satisfies BCNF)

Payment

Payment (Payment_ID, Amount, Pay_date, Payment_Type, P_ID(FK))

FD:

(Payment_ID, P_ID(FK)) → (Amount, Pay_date, Payment_Type)

Payment is in BCNF, because this relation doesn't have nested relations or non-atomic attributes (satisfies 1NF), every non-key attribute is fully functional dependency (satisfies 2NF) and has no transitive dependency (satisfies 3NF), and every determinant is a candidate key (satisfies BCNF)

Insurance

Insurance (Payment_ID(FK), Co_name, Co_phone, policy)

FD:

(Payment_ID(FK)) → (Co_name, Co_phone, policy)

Insurance is in BCNF, because this relation doesn't have nested relations or non-atomic attributes (satisfies 1NF), every non-key attribute is fully functional dependency (satisfies 2NF) and has no transitive dependency (satisfies 3NF), and every determinant is a candidate key (satisfies BCNF)

Card

Card (Payment_ID(FK), Card_num, Exp_date, pin)

FD:

(Payment_ID(FK)) \rightarrow (Card_num, Exp_date, pin)

(Card_num) \rightarrow (Exp_date, pin)

Card is in 2NF, because this relation doesn't have nested relations or non-atomic attributes (satisfies 1NF), every non-key attribute is fully functional dependency (satisfies 2NF), but there exists transitive dependency (not satisfies 3NF)

In order to convert to BCNF:

Card (Payment_ID(FK), Card_num(FK))

Card_Info (Card_num, Exp_date, pin)

Now there's no transitive dependency (satisfies 3NF), and every determinant is a candidate key (satisfies BCNF)

Check

Check (Payment_ID(FK))

FD:

None

Check is in BCNF, because this relation doesn't have nested relations or non-atomic attributes (satisfies 1NF), every non-key attribute is fully functional dependency (satisfies 2NF) and has no transitive dependency (satisfies 3NF), and every determinant is a candidate key (satisfies BCNF)

Cash

Cash (Payment_ID(FK))

FD:

None

Cash is in BCNF, because this relation doesn't have nested relations or non-atomic attributes (satisfies 1NF), every non-key attribute is fully functional dependency (satisfies 2NF) and has no transitive dependency (satisfies 3NF), and every determinant is a candidate key (satisfies BCNF)

Bill

Bill (Bill_ID, Bill_date, Due_date, Doctor(FK))

FD:

(Bill_ID) \rightarrow (Bill_date, Due_date, Doctor)

Bill is in BCNF, because this relation doesn't have nested relations or non-atomic attributes (satisfies 1NF), every non-key attribute is fully functional dependency (satisfies 2NF) and has no transitive dependency (satisfies 3NF), and every determinant is a candidate key (satisfies BCNF)

Payment_Apply_Bill

Payment_Apply_Bill (Payment_ID(FK), Bill_ID(FK), Pay_date)

FD:

(Payment_ID(FK), Bill_ID(FK)) \rightarrow (Pay_date)

Payment_Apply_Bill is in BCNF, because this relation doesn't have nested relations or non-atomic attributes (satisfies 1NF), every non-key attribute is fully functional dependency (satisfies 2NF) and has no transitive dependency (satisfies 3NF), and every determinant is a candidate key (satisfies BCNF)

Appointment

Appointment (Appt_ID, Appt_Date, Appt_Type, P_ID(FK), Bill_ID(FK))

FD:

$(\text{Appt_ID}) \rightarrow (\text{Appt_Date}, \text{Appt_Type}, \text{P_ID}(\text{FK}), \text{Bill_ID}(\text{FK}))$

Appointment is in BCNF, because this relation doesn't have nested relations or non-atomic attributes (satisfies 1NF), every non-key attribute is fully functional dependency (satisfies 2NF) and has no transitive dependency (satisfies 3NF), and every determinant is a candidate key (satisfies BCNF)

Procedure

Procedure (Procedure_ID, Procedure_name, Fee_To_Invoice, Bill_ID(FK))

FD:

$(\text{Procedure_ID}) \rightarrow (\text{Procedure_name}, \text{Fee_To_Invoice}, \text{Bill_ID}(\text{FK}))$

$(\text{Bill_ID}(\text{FK})) \rightarrow (\text{Fee_To_Invoice})$

Procedure is in 2NF, because this relation doesn't have nested relations or non-atomic attributes (satisfies 1NF), every non-key attribute is fully functional dependency (satisfies 2NF), but there exists transitive dependency (not satisfies 3NF)

In order to convert to BCNF:

Procedure (Procedure_ID(FK), Procedure_name, Bill_ID(FK))

Bill_Info (Bill_ID(FK), Fee_To_Invoice)

Now there's no transitive dependency (satisfies 3NF) and every determinant is a candidate key.

(satisfies BCNF)

Appt_Has_Procedure

Appt_Has_Procedure (Appt_ID(FK), Procedure_ID(FK))

FD:

None

Appt_Has_Procedure is in BCNF, because this relation doesn't have nested relations or non-atomic attributes (satisfies 1NF), every non-key attribute is fully functional dependency (satisfies 2NF) and has no transitive dependency (satisfies 3NF), and every determinant is a candidate key (satisfies BCNF)

Dental_provider_Perform_Procedure

Dental_provider_Perform_Procedure (Provider_ID(FK), Procedure_ID(FK), Per_Date)

FD:

$(\text{Provider_ID(FK)}, \text{Procedure_ID(FK)}) \rightarrow (\text{Per_Date})$

Dental_provider_Perform_Procedure is in fact in BCNF, because this relation doesn't have nested relations or non-atomic attributes (satisfies 1NF), every non-key attribute is fully

functional dependency (satisfies 2NF) and has no transitive dependency (satisfies 3NF), and every determinant is a candidate key (satisfies BCNF)

Equipment

Equipment (E_ID, E_name)

FD:

$(E_ID) \rightarrow (E_name)$

Equipment is in BCNF, because this relation doesn't have nested relations or non-atomic attributes (satisfies 1NF), every non-key attribute is fully functional dependency (satisfies 2NF) and has no transitive dependency (satisfies 3NF), and every determinant is a candidate key (satisfies BCNF)

Procedure_Consume_Equipment

Procedure_Consume_Equipment (Procedure_ID(FK), E_ID(FK))

FD:

None

Procedure_Consume_Equipment is in BCNF because this relation doesn't have nested relations or non-atomic attributes (satisfies 1NF), every non-key attribute is fully functional dependency

(satisfies 2NF) and has no transitive dependency (satisfies 3NF), and every determinant is a candidate key (satisfies BCNF)

Section 2

2.a Table Description

The PERSON_INFO table is to store the personal information of each person (including both employees and patients).

The personal information includes Ssn, first name, last name, and birth date. Ssn is the primary key. Null value is not allowed in every field in this table.

Ssn is of integer type, first name and last name are of char type (within 15 length), and birth date is of date type.

The PERSON table is an overview of each person.

It includes P_ID, Ssn, Email, Phone, and P_type.. P_Id is the primary key. Ssn references Ssn in PERSON_INFO table.

No Null value is allowed in P_ID, Ssn, Email, Phone, and P_type.

P_ID and Ssn are of integer type. Email, Phone and P_Type are char types with limits length 30,12, and 10 respectively.

The PATIENT table stores information specific to patients.

It includes P_ID, Ssn, Date_Of_Last_X_ray, HIPAA_Signature, and Emergency_Contact. P_ID, Ssn, and Emergency_Contact are of integer type, Date_Of_Last_X_ray is of date type, and HIPAA_Signature is of varchar type with length limit of 30. No Null values are allowed for P_ID and Ssn. P_ID is the primary key for this table. P_ID is a foreign key which references P_ID in PERSON table. Ssn is a foreign key which references Ssn in PERSON_INFO table.

Emergency_Contact is a foreign key which references P_ID in PERSON table.

The EMPLOYEE table stores information specific to employees in the office.

It includes P_ID, Ssn, Salary, and Job_Type. P_ID, Ssn, and Salary are of integer type. Job_Type is of varchar type with a length limit of 10. No Null values are allowed for any attributes of this table. P_ID is the primary key for this table. P_ID is a foreign key which references P_ID in PERSON table. Ssn is a foreign key which references Ssn in PERSON_INFO table.

The ADDRESS table contains all necessary address information for each patient.

It includes Address_ID, Street, City, State, and ZIP. Address_ID is the primary key. There are no foreign keys for this table, all values are stored natively. No Null value is allowed for any attributes of this table, as they are all necessary values for a full address.

Address_ID is of integer type, Street and City are of varchar types with limits on length of 30 and 20 respectively. State and ZIP are of type char, with limits on length of 2 and 5 respectively.

The PATIENT_HAS_ADDRESS table is to match the patient and address pairs.

It includes Address_ID and P_ID. Both of them are of Integer type. No Null value is allowed in the table. Address_ID references Address_ID in Address table, and P_ID references P_ID in PERSON table.

The Medication table is to store the information related to all the medications.

It includes Medication_ID, Medication_Info, Date_prescribed, and Dosage. Medication_ID is the primary key. No Null value is allowed. Medication_ID is of integer type, Date_presecribed is of date type, Medication_Info and Dosage are of char type with length limit 20.

The PATIENT_HAS_MEDICATION table stores the relationship between medication types and the patients who use said medications.

It includes Medication_ID and P_ID, both of which are of integer types. No Null values are allowed for either attribute. Medication_ID is a foreign key which references Medication_ID in MEDICATION table, and P_ID is a foreign key which references P_ID in PERSON table.

The ALLERGY table stores the information of allergy.

It includes Allergy_ID and Allergy_info. No Null value is allowed. Allergy_ID is the primary key. Allergy_ID is of type integer and Allergy_info is of type char with 20 length limit.

The PATIENT_HAS_ALLERGY table stores the relationship between allergies and the patients who have them.

It includes Allergy_ID and P_ID, both of which are of integer types. No Null values are allowed for either attribute. Allergy_ID is a foreign key which references Allergy_ID in ALLERGY table, and P_ID is a foreign key which references P_ID in PERSON table.

The PAYMENT table stores the information of payments, including the amount, date, and type of each payment.

It includes Payment_ID, Amount, Pay_Date, Payment_Type and P_ID. No Null value is allowed.

Primary key is Payment_ID, and P_ID is a foreign key referencing P_ID in PERSON.

Payment_ID, Amount, and P_ID are of type integer, Pay_Date is of type date, and

Payment_Type is of type char with length limit 10.

The INSURANCE table stores all information specific to the insurance companies and any payments made by them.

It includes Payment_ID, Co_Name, Co_Phone, and Policy. Payment_ID is of integer type, Co_Name, Co_Phone, and Policy are of varchar type, with length limits of 15, 12, and 15 respectively. No Null values are allowed for any attributes. Payment_ID is a foreign key which references Payment_ID from the PAYMENT table.

The CARD_INFO table stores all necessary information for each credit/debit card used for payments.

It includes Card_Num, Exp_Date, and PIN. Card_Num and PIN are of char type with length limits of 16 and 3 respectively. Exp_Date is of date type. No Null values are allowed for any of these attributes, as they are all necessary to store a full card. Card_Num is the primary key of this table.

The CARD table stores the relationship between Payment and Card info.

It includes Payment_ID and Card_Num. No null value is allowed. Payment_ID and Card_Num are foreign keys referencing Payment_ID in PAYMENT and Card_number in CARD_INFO.

Payment_ID is of integer type and Card_number is of char type with length limit 16.

The PAPER_CHECK table stores all payments made by check.

It includes Payment_ID which is of integer type. This attribute must not be null. It is a foreign key of Payment_ID from the PAYMENT table.

The CASH table stores all payments made by CASH.

It includes Payment_ID which is of integer type. It must not be null. It is a foreign key of Payment_ID from the PAYMENT table.

The BILL table stores all information necessary for employees and patients to view the bill for each appointment.

It includes Bill_ID, Bill_date, Due_date, and Doctor. Bill_ID and Doctor are of integer type, while Bill_date and Due_date are of date type. No Null values are allowed for any attributes. Bill_ID is the primary key for this table. Doctor is a foreign key which references P_ID from the PERSON table.

The Payment_Apply bill stores the relationship between payments and bills.

It includes Payment_ID, Bill_ID, and Pay_date. No null value is allowed. Pay_date is of date type, and payment_ID and Bill_ID are of integer type. Payment_ID and Bill_ID are foreign keys referencing Payment_ID and Bill_ID in PAYMENT and BILL.

The APPOINTMENT table stores all necessary information about a specific appointment.

It includes Appt_ID, Appt_Date, Appt_Type, P_ID, and Bill_ID. Appt_ID, P_ID and Bill_ID are all of integer type, Appt_Date is of date type, and Appt_Type is of varchar type with a limit on length of 10. No Null values are allowed for these attributes. Appt_ID is the primary key for this table. P_ID is a foreign key which references P_ID in PERSON table and Bill_ID is a foreign key which references Bill_ID in BILL table.

The DENTAL_PROCEDURE table stores related information about a specific procedure.

It includes Procedure_ID, Procedure_name, Fee_to_Invoice, and Bill_ID. No null value is allowed. Procedure_ID is a primary key and Bill_ID is a foreign key referencing Bill_ID from the BILL table. Procedure_ID, Fee_to_Invoice, and Bill_ID are of integer type, and Procedure_name is of char type with length limit 50.

The APPT_HAS_PROCEDURE table stores the relationship between appointments and the procedures done at those appointments.

It includes Appt_ID and Procedure_ID. Both attributes are of integer type. No Null values are allowed for these attributes. Appt_ID is a foreign key of Appt_ID from the APPOINTMENT table, and Procedure_ID is a foreign key of Procedure_ID from the DENTAL_PROCEDURE table.

The DENTAL_PROVIDER_PERFORM_PROCEDURE stores the relationship between dental providers and procedures.

It includes Procedure_ID, Provider_ID, and Per_date. No null value is allowed. Procedure_ID and Provider_ID are foreign keys referencing Procedure_ID and Provider_ID from the table DENTAL_PROCEDURE and PERSON. Per_date is of date type and Procedure_ID and Provider_ID are of integer type.

The EQUIPMENT table stores information on the types of equipment used in the office.

It includes E_ID and E_name. E_ID is of integer type and E_name is of varchar type. No Null values are allowed for these attributes. The primary key for this table is E_ID.

The PROCEDURE_CONSUME_EQUIPMENT stores equipment consumed by each procedure.

It includes Procedure_ID and E_ID. Both of them are of integer type and they must not be null.

Procedure_ID and E_ID are foreign keys referencing Procedure_ID from the table

DENTAL_PROCEDURE and E_ID from the table EQUIPMENT.

2.b Catalog of supplied SQL Queries

1. The first query lists all patients and the medications they take currently.

SQL:

```

SELECT
    PERSON_INFO.Fname,
    PERSON_INFO.Lname,
    MEDICATION.Medication_Info,
    MEDICATION.Dosage,
    MEDICATION.Date_prescribed
FROM
    PATIENT_HAS_MEDICATION,
    PERSON,
    MEDICATION,
    PERSON_INFO
WHERE
    PATIENT_HAS_MEDICATION.P_ID = PERSON.P_ID
    AND MEDICATION.Medication_ID =
PATIENT_HAS_MEDICATION.Medication_ID
    AND PERSON_INFO.Ssn = PERSON.Ssn
    AND MEDICATION.Date_prescribed >= GETDATE()
ORDER BY
    PERSON_INFO.Lname

```

Sample Output:

	Fname	Lname	Medication_Info	Dosage	Date_prescribed
1	Linda	Champion	Sildenafil	1 per day	2023-02-16
2	Jamie	Frank	amoxicillin	2 per day	2023-09-21
3	Jone	Hemann	amoxicillin	1 per day	2023-02-16
4	James	Roman	Atibiotics	3 per day	2023-09-21
5	Adam	Smith	ibuprofen	1 per day	2023-09-21

Figure 2.1: Sample Output for Simple Query 1

2. The next query displays patient information for any patients who use Delta Dental as their insurance provider.

SQL:

```

SELECT
    PERSON_INFO.Fname,
    PERSON_INFO.Lname,
    PERSON.Phone,
    PERSON.Email,
    PERSON.Ssn,
    INSURANCE.Policy
FROM
    PERSON,
    PERSON_INFO,
    PAYMENT,
    INSURANCE
WHERE
    PERSON_INFO.Ssn = PERSON.Ssn
    AND INSURANCE.Policy = 'Delta Dental'
    AND PAYMENT.P_ID = PERSON.P_ID
    AND PAYMENT.Payment_ID = INSURANCE.Payment_ID

```

Sample Output:

	Fname	Lname	Phone	Email	Ssn	Policy
1	Jamie	Frank	787-997-6612	Frank.11@osu.edu	852817001	Delta Dental
2	Adam	Smith	455-247-3452	Smith.22@osu.edu	852817002	Delta Dental
3	Jone	Hemann	275-175-7742	hemann656@gmail.edu	852817006	Delta Dental
4	James	Roman	816-22-8177	Smillow.12@osu.edu	852817003	Delta Dental

Figure 2.2: Sample Output for Simple Query 2

3. The next query shows all procedures as well as the dates performed by Dr. Smilow.

SQL:

```

SELECT
    PERSON_INFO.LName,
    DENTAL_PROCEDURE.Procedure_name,
    DENTAL_PROVIDER_PERFORM_PROCEDURE.Per_date
FROM
    PERSON,
    PERSON_INFO,
    DENTAL_PROCEDURE,
    DENTAL_PROVIDER_PERFORM_PROCEDURE
WHERE
    PERSON.Ssn = PERSON_INFO.Ssn
    AND DENTAL_PROCEDURE.Procedure_ID =
DENTAL_PROVIDER_PERFORM_PROCEDURE.Procedure_ID
    AND DENTAL_PROVIDER_PERFORM_PROCEDURE.Provider_ID =
PERSON.P_ID
    AND PERSON_INFO.LName = 'Smilow'

```

Sample Output:

	LName	Procedure_name	Per_date
1	Smilow	Cleaning	2022-07-03
2	Smilow	Cleaning	2021-01-01
3	Smilow	Cleaning	2021-12-18
4	Smilow	Cleaning	2022-01-03
5	Smilow	Cleaning	2018-04-28

Figure 2.3: Sample Output for Simple Query 3

4. The next query prints out past due invoices with patient contact information. It is assumed that an invoice is overdue if it is 30 days past the due date with at least \$10 due.

SQL:

```

SELECT

```

```

PERSON_INFO.Fname,
PERSON_INFO.Lname,
PERSON.Phone,
PERSON.Email,
PERSON.Ssn,
Fee_to_Invoice AS Procedure_Charge,
Amount AS Paid
FROM
PERSON,
PERSON_INFO,
PAYMENT,
Payment_Apply_Bill,
DENTAL_PROCEDURE,
APPT_HAS_PROCEDURE,
APPOINTMENT,
BILL
WHERE
PERSON.Ssn = PERSON_INFO.Ssn
AND BILL.Bill_ID = Payment_Apply_Bill.Bill_ID
AND PAYMENT.Payment_ID = Payment_Apply_Bill.Payment_ID
AND PAYMENT.P_ID = PERSON.P_ID
AND DENTAL_PROCEDURE.Procedure_ID =
APPT_HAS_PROCEDURE.Procedure_ID
AND APPOINTMENT.Appt_ID = APPT_HAS_PROCEDURE.Appt_ID
AND APPOINTMENT.P_ID = PERSON.P_ID
GROUP BY
PERSON_INFO.FName,
PERSON_INFO.LName,
PERSON.Phone,
PERSON.Email,
PERSON.Ssn,
DENTAL_PROCEDURE.Fee_to_Invoice,
PAYMENT.Amount,
BILL.Due_date
HAVING
Fee_to_Invoice - Amount > 10
AND DATEDIFF(day,BILL.Due_date,GETDATE()) > 30

```

Sample Output:

	Fname	Lname	Phone	Email	Ssn	Procedure_Charge	Paid
1	Adam	Smith	455-247-3452	Smith.22@osu.edu	852817002	1065	265
2	Jamie	Frank	787-997-6612	Frank.11@osu.edu	852817001	999	199
3	James	Roman	816-22-8177	Smillow.12@osu.edu	852817003	265	222
4	James	Roman	816-22-8177	Smillow.12@osu.edu	852817003	365	222

Figure 2.4: Sample Output for Simple Query 4

5. This query prints out the patients who brought in the most revenue in the past year.

SQL:

```

WITH Payment_Amount_From_Patient AS (
SELECT
    PERSON_INFO.LName,
    SUM(Amount) as Total
FROM
    PERSON,
    PERSON_INFO,
    PAYMENT,
    Payment_Apply_Bill
WHERE
    PAYMENT.P_ID = PERSON.P_ID
    AND PAYMENT.Payment_ID = Payment_Apply_Bill.Payment_ID
    AND PERSON.Ssn = PERSON_INFO.Ssn
    AND DATEDIFF(day,PAYMENT.pay_date,GETDATE()) < 365
GROUP BY
    PERSON_INFO.LName
) SELECT
    LName,
    MAX(total) AS Total_Revenue
FROM
    Payment_Amount_From_Patient
GROUP BY
    LName;

```

Sample Output:

	LName	Total_Revenue
1	Champion	2600
2	Frank	199
3	Frost	862
4	Hemann	52
5	Smith	1330

Figure 2.5: Sample Output for Simple Query 5

6. This query creates a list of doctors who performed less than 5 procedures in the last year.

SQL:

```

WITH Procedure_Count AS (
    SELECT
        PERSON_INFO.LName,
        PERSON_INFO.FName,
        COUNT(*) AS Times
    FROM
        PERSON,
        PERSON_INFO,
        DENTAL_PROVIDER_PERFORM_PROCEDURE
    WHERE
        PERSON_INFO.Ssn = PERSON.Ssn
        AND
        DENTAL_PROVIDER_PERFORM_PROCEDURE.Provider_ID = PERSON.P_ID
        AND
        DATEDIFF(day,DENTAL_PROVIDER_PERFORM_PROCEDURE.Per_date,GETDATE()) < 365
    GROUP BY
        PERSON_INFO.LName,
        PERSON_INFO.FName
)SELECT
    LName,
    FName,
    Times
FROM
    Procedure_Count
WHERE
    Times < 5;

```

Sample Output:

	LName	FName	Times
1	Smilow	Alison	3
2	Johnson	Mary	2

Figure 2.6: Sample Output for Simple Query 6

7. This query lists out the highest paying procedures and how much income they have brought in.

SQL:

```
SELECT Procedure_Name, Fee_To_Invoice AS Paying,
COUNT(Procedure_Name)
FROM DENTAL_PROCEDURE
GROUP BY Procedure_name, Fee_to_Invoice
ORDER BY Paying DESC;
```

Sample Output:

	Procedure_Name	Paying
1	Extraction	2600
2	Extraction	1065
3	White	999
4	Extraction	862
5	Cleaning	365
6	Cleaning	265
7	Cleaning	222
8	Cleaning	200
9	Cleaning	199
10	Cleaning	52

Figure 2.7: Sample Output for Simple Query 7

8. This query lists all payment types accepted, as well as the total count of each type's use and how much income each type has earned the office.

SQL:

```
SELECT Payment_Type, COUNT(Payment_Type) AS Times, SUM(Amount) AS
Total
FROM PAYMENT
```

GROUP BY Payment_Type;

Sample Output:

	Payment_Type	Times	Total
1	Card	1	366
2	Cash	2	317
3	Check	1	199
4	Insurance	6	5948

Figure 2.8: Sample Output for Simple Query 8

9. This query finds the most popular insurance policies and lists the total number of patients who use each policy.

SQL:

```
SELECT policy, Count(*) as p_count
FROM INSURANCE
GROUP BY policy
ORDER BY p_count DESC;
```

Sample Output:

	policy	p_count
1	Delta Dental	4
2	OH Dental	2

Figure 2.9: Sample Output for Simple Query 9

2.c INSERT and DELETE SQL code samples

INSERT samples:

- Below are three insert samples to PERSON_INFO table. We add Ssn, Last name, First name, and birth date values respectively:

```

INSERT INTO PERSON_INFO VALUES ('852817001', 'Frank', 'Jamie', '1995-7-12');
INSERT INTO PERSON_INFO VALUES ('852817002', 'Smith', 'Adam', '2000-6-2');
INSERT INTO PERSON_INFO VALUES ('852817003', 'Roman', 'James', '1980-1-6');

```

Ssn	LName	FName	Birth_Date
852817001	Frank	Jamie	1995-7-12
852817002	Smith	Adam	2000-6-2
852817003	Roman	James	1980-1-6

Figure 2.10: The PERSON_INFO table

- 2) Below are two insert samples to the PATIENT table. We add P_ID, Ssn, Date_Of_Last_X_ray, HIPAA_Signature, Emergency_contact respectively.

```

INSERT INTO PATIENT VALUES('1', '852817001','2020-9-6', 'Yes', NULL);
INSERT INTO PATIENT VALUES('2', '852817002','2021-10-7', 'Yes', NULL);

```

P_ID	Ssn	Date_Of_Last_X_ray	HIPAA_Signature	Emergency_Contact
1	852817001	2020-9-6	Yes	NULL
2	852817002	2021-10-7	Yes	NULL

Figure 2.11: The PATIENT table

DELETE samples:

Below are three delete samples to remove Ssn ends with '002', '003', from the PERSON_INFO table, and remove P_ID = 2 from the patient table:

```

DELETE FROM PATIENT WHERE P_ID = 2;
DELETE FROM PERSON_INFO WHERE Ssn = 852817003;
DELETE FROM PERSON_INFO WHERE Ssn = 852817002;

```

P_ID	Ssn	Date_Of_Last_X_ray	HIPAA_Signature	Emergency_Contact
1	852817001	2020-9-6	Yes	NULL

Figure 2.12: The patient table

Ssn	LName	FName	Birth_Date
852817001	Frank	Jamie	1995-7-12

Figure 2.13: The PERSON_INFO table

2.d Two indexes explained, including SQL code

```

SQLQuery3.sql - L:\PKTSPV\dmand (63))
USE [CSE 3241 Project]
GO

/***** Object: Index [NonClusteredIndex-20220729-142616] Script Date: 7/29/2022 2:36:22 PM *****/
CREATE NONCLUSTERED INDEX [NonClusteredIndex-20220729-142616] ON [dbo].[Address]
(
    [Address_ID] DESC
)
INCLUDE([City],[ZIP]) WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
GO

```

Figure 2.14: Address Non-Clustered Index

The first Index is a non-clustered index with the primary key in Address clustered on Address_ID with added columns of City and ZIP from dbo.Address in descending order. By creating an index, we are creating an invisible copy of dbo.Address that can assist us in finding a particular data set much more quickly and efficiently than if we were to not create an index. The non-clustered index is different from a clustered index in that it does not touch the actual data pages except for pointers that point to the particular data pages. The only downside is that it takes extra space on the server to store another copy of the entity.

```

SQLQuery1.sql - L:\PKTSPV\dmand (73))
USE [CSE 3241 Project]
GO

/***** Object: Index [ClusteredIndex-20220729-145822] Script Date: 7/29/2022 2:59:41 PM *****/
CREATE UNIQUE CLUSTERED INDEX [ClusteredIndex-20220729-145822] ON [dbo].[Employee]
(
    [P_ID] ASC
)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, IGNORE_DUP_KEY = OFF, DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
GO

```

Figure 2.15: Employee Clustered Index

The second index created is a clustered index on `dbo.Employee` clustered on `P_ID` which is a unique variable that is a primary key in the `dbo.Person` and a foreign key in the `dbo.Employee` filtering data in ascending order. The clustered index works by rearranging the data in the table and ordering it according to the key field of the index which is not exactly an index but is used to speed up simple but most importantly complex queries. The index will organize the data in sorted ascending order which makes it much easier to retrieve the appropriate data from a large database.

2.e Two views explained, including SQL code and execution

```
CREATE VIEW Allergy_info AS
SELECT COUNT (Allergy_ID) AS Allergy_needed
FROM Allergy
WHERE Allergy_ID > 4
```

Figure 2.16: Allergy View Code

The new view created renames the original `Allergy_ID` to `Allergy_needed` and counts the number of allergies that are above the `Allergy_ID` listed as 4.

	Allergy need...
	6
»»	NULL

Figure 2.17: Allergy View

```

CREATE VIEW Scheduled_Appointments AS
SELECT
    APPOINTMENT.Appt_Type,
    COUNT (Appt_Type) AS Appt
FROM
    Appointment
GROUP BY
    APPOINTMENT.Appt_Type

```

Figure 2.18: Appointment Type View Code

The following is the count of appointment types (However, we only have one type).

Appt Type	Appt
Dental	10
NULL	NULL

Figure 2.19: Appointment Type View

2.f Two transactions explained, including SQL code

```

BEGIN TRANSACTION
INSERT INTO [dbo].[Medication] (
    [Medication_ID], [Medication_Info], [Data_prescribed], [Dosage])
VALUES (22, 'Tylenol', '2023-7-13', '2 per day')
COMMIT TRANSACTION

```

Figure 2.20: Medication Dataset Transaction

The first transaction has a basic function of adding another data set into the table `dbo.Medication`. Since we have 21 unique `Medication_ID` in the table, we can add `Medication_ID` 22 into the table with its corresponding values listed in the order as the listed corresponding values in the correct data types. It's very important to use the proper SQL language to begin the Transaction with a `BEGIN TRANSACTION` statement and end with a `COMMIT TRANSACTION`.

```

BEGIN TRANSACTION
INSERT INTO [dbo].[Allergy] (
    [Allergy_info], [Allergy_ID])
VALUES ('peanut butter', 10)

SELECT * FROM [dbo].[Allergy]

ROLLBACK TRANSACTION

SELECT * FROM [dbo].[Allergy]

```

Figure 2.21: Allergy Dataset Transaction

The second transaction is another basic transaction that inserts a new data set into `dbo.Allergy` table with correctly corresponded data with the matching data types. However, this transaction also allows us to revert the table to its original form if an error occurs using the `ROLLBACK` function and placing the `SELECT` statements inside and outside of the transaction.

Section 3

3.a Team contributions

Dima Mandryka - wrote Introduction and project summary in **Section 1, 1.a**. Wrote explanations and broke down the relational schemas in **Section 1, 1.c**. Helped Zhibin Mo write descriptions for the normalizations in **Section 1, 1.e**. Created two indexes with explanations in **Section 2, 2.d**. Created and described the transactions in **Section 2, 2.f**. Helped Zhibin finish views in **Section 2, 2.e**.

Zhibin Mo - Revise ERD, Revise Relational Scheme, Revise Relational Algebra, Revise Normalization, Revise SQL Code which contains Create, Insert, Select, Extra Queries.

Jacob Covert - Revised SQL Code to fix errors in database creation and in the insertion of data, revised SQL query code, helped Yifan Dou write **Section 2, 2.a**, wrote **Section 2, 2.b**, checked report for grammatical or spelling errors, worked on formatting of report.

Yifan Dou- wrote explanations for the (E)ERD Model in **Section 1, 1.b**. Wrote a couple of explanations of relational schemas in **Section 1, 1.c**. Wrote explanations of Relational algebra in **Section 1, 1.d**. Wrote table descriptions in **Section 2, 2.a** with Jacob Covert. Wrote Insert/Delete samples explanations in **Section2, 2.c**

3.b Reflection

We performed a great job in the initial submission of the report, CP01, but then we missed so many credits in CP02 and CP03. During the last part, we carefully examined the feedback from 01, 02, and 03 and decided it would be easier to start from scratch than it would be to find all of our mistakes. One thing that has made things easier during the project is that everyone has been collaborating with one another and doing the tasks properly, despite the fact that our group members have very conflicting schedules. There's no use in crying over spilled milk; we needed to stay calm and contemplate how to finish the job properly. The only thing that could bring our grade back up is to put in our best work and do extremely well on the project instead of complaining about the score.

3.c Description of feedback and revisions made

Feedback From CP01




CP01 Project Checkpoints	May 31 by 8pm	8.5	10	  
Comments				Close
Great job on research, data samples, and documentation. Great choice of extra features.				
ERD is well developed for the first draft but still needs more work. No need for so many specializations. Remove unnecessary ones for job types. Avoid weak entities as they are now in your ERD. Should not use specializations on both sides: laticies. Relationships cannot be mandatory on both sides.				
Consider using Person generalization and an Address entity. Allergy and Medication are most likely going to be entities not attributes. Make it visually clear that person is generalization. Procedures are associated with invoices. When a patient checks in an invoice can be started. Invoice will be associated with an appointment. Need to expand Payments to store all required payment info.				Zina Pichkar (pichkar.3), Jun 2 at 11:36am
Consider attributes for relationships to show how many units billed, how payment is applied, when it took place, etc. Insurance entity needs to be developed further: Company Vs Policy.				
Simplify ERD but at the same time make sure it also shows required data storage functionality and meets all the requirements.				
Assessment by Zina Pichkar (pichkar.3)				

Figure 3.1: CP01 Feedback

ERD From CP01

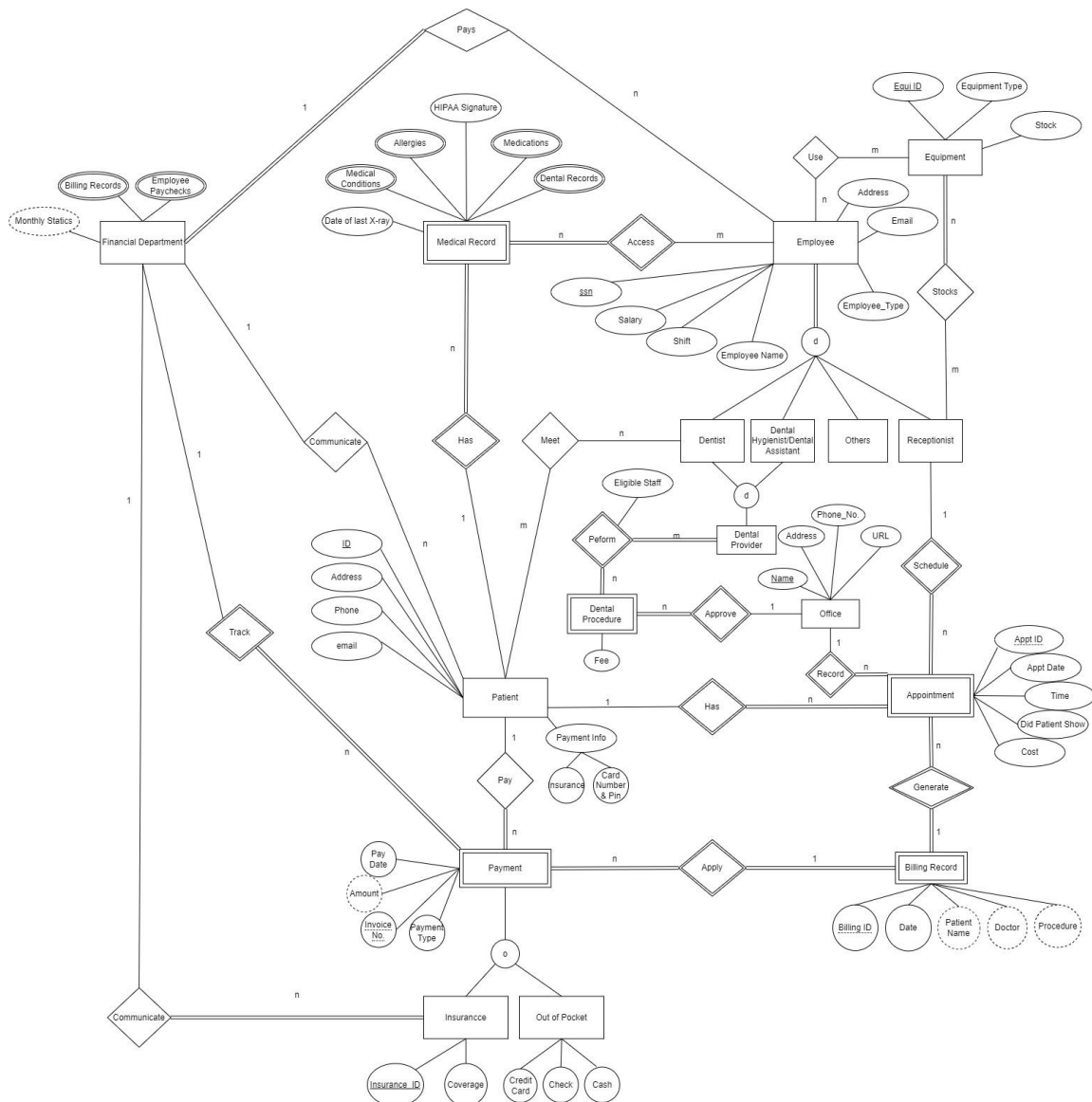


Figure 3.2: ERD from CP01

Change From CP01:

- We deleted many unnecessary weak entities and multi valued attributes
- Add allergy and medication as an entity
- Delete the Out Of Pocket Entity and change it more specify like cash and check
- Correct the cardinality on each relationship

Feedback From CP02

CP02 Project Checkpoints	Jun 21 by 8pm	2.5	10	
Comments				Close
ERD seems to be very incomplete and most of the previous feedback and guidelines discussed in lectures are not addressed.				
Cardinality errors. For example, Payment:Invoice should be M:N. Emerg contact not stored correctly. It is a Person. Missing attributes to properly describe some relationships.				
Teams were clearly instructed not to use any links and include all details in one report. Unable to assess your work for RA or Relational schema. RA uses correct symbols. Extra queries are way too simplistic. RA does not use join operations that would be obviously needed to answer the questions.				Zina Pichkar (pichkar.3), Jun 25 at 1:30pm
Your team should put more work in this project and follow instructions in order to get a good grade for the next part.				
I will not be regarding this part as it was the team's responsibility to check for presence and correctness of all submitted work.				

Figure 3.3: CP02 Feedback

ERD From CP02

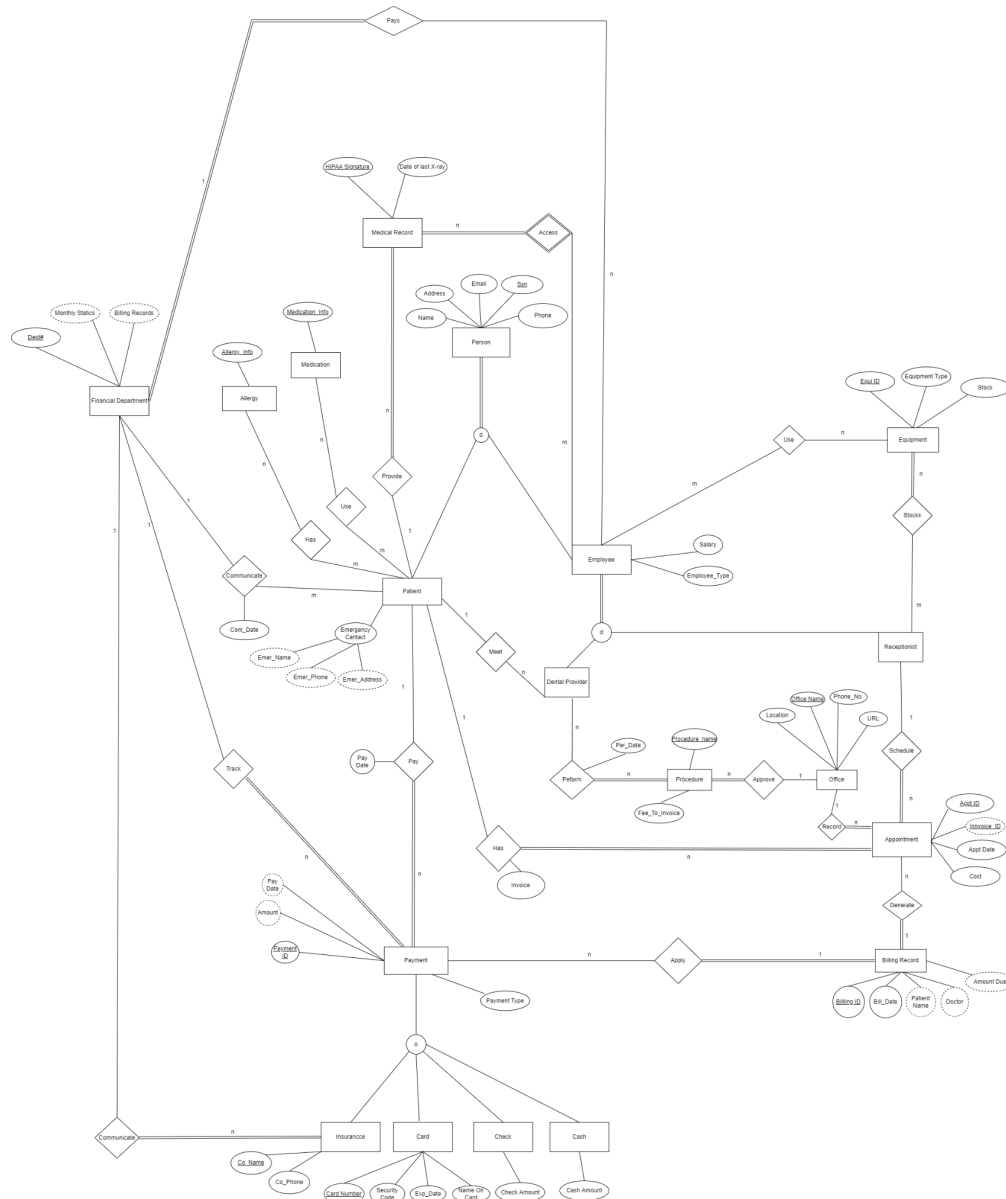


Figure 3.4: ERD from CP02

Change From CP02:

- Delete all link that arrowing outer file
- Delete not necessary entities such as financial department
- Delete some unnecessary derived attributes
- Put all relational schema into the report

Feedback From CP03

<ul style="list-style-type: none"> CP03 Project Checkpoints 	Jul 13 by 8pm	4.2	10	✉ 📅 🗨
Comments				Close
<p>Formatting and Organization:</p> <p>Good formatting for the report. Updated (E)ERD, Schema, RA:ERD needs work- some aspects are not needed or related properly (e.g. employee access medical record relationship), why is patient SSN a derived attribute for many entities? Is this supposed to be a key or foreign key instead? Entities like allergy should likely have their own ID – what if many patients have the same type of allergy? Address attribute incomplete- consider an address entity. SSN shouldn't be a PK. Schema and RA will need to be updated to match the ERD...Schema does not map or explain where all FKs come from, Normalization:Normalization is attempted for all tables. Again, this will need to be updated once ERD and everything else is changed. Create/Insert SQL:Create and insert SQL executes fully without errors in SSMS. I hope that there is an understanding among your team that event though SQL code executes, it is based on faulty logical design and will not produce a functional DB. Select queries refer to derived attributes and attributes stored in wrong tables and will not produce logically correct results at this point. SQL will need to be reworked once ERD and schema are fully complete and correct.</p> <p>Select SQL:Simple queries:A. Does not sort in alphabetical order by med name also, shows people who do not take medicine (NULLs), does not show dosage/anything relevant</p> <p>D. Seems like it is showing every invoice regardless of being paid because there does not appear to be a check for paid/unpaid</p> <p>F. Does not seem correct based off insert data</p> <p>I.Shows 2 separate insurances, not most popular</p> <p>Extra Queries: none used aggregate function... B. was very simplistic and similar to the simple queries</p> <p>Your team has a full potential to do much better with the project if you put in a sufficient amount of effort and time. I hope to see much better quality work and fully developed and functioning DB in your final project.</p>				
				Zina Pichkar (pichkar.3), Jul 18 at 12:43pm

Figure 3.5: CP03 Feedback

Change From CP03:

- **We add specific ID for each entity**
- **Redo the relational scheme**
- **Redo the relational algebra**
- **Redo the SQL queries such as Create, Insert, Select**
- **Redo the Extra Queries**

Part II

The SQL Database

*.sqlite, or *.db file: correctly formatted, and ready to open with either SSMS or SQLiteOnline *.txt files: All CREATE, INSERT, DELETE, and SELECT SQL scripts required to reproduce and test your DB, submitted in separate text files.