Project PART 3 Regina Powers, Ben Borszcz, Bryan Vales, Lohith Maralla

\*For this part of the project we used SQLiteOnline.

1. Review feedback provided for PART 2 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. If your ERD was updated, describe all updates, and include the new ERD and the original ERD from PART 2.

**FEEDBACK**:

1. **ERD:** “amount should be shown as a derived attribute, patient should have an emergency contact”

Done, added emergency contact name and phone number. Changed amount to derived attribute.

2. **Schema:** “M:N relationships should have 2 foreign keys not primary keys”

Changed PKs to be FK as follows. Note green indicates a foreign key.

**Mapping of M:N relation**

(A new relation is made for each M:N relation)

PERSON\_HAS\_INSURANCE(SSN, Insurance\_ID)

PERSON\_HAS\_ADDRESS(SSN, Address\_ID)

PAYMENT\_COVERS\_BILLING(Payment\_ID, Billing\_ID, Amount)

PROCEDURE\_REQUIRES\_EQUIPMENT(Procedure\_ID, Equipment\_ID, Number\_Of)

PROCEDURE\_REQUIRES\_LICENSURE(Procedure\_ID, Licensure\_ID)

PRACTITIONER\_CAN\_PERFORM\_PROCEDURE(SSN, Procedure\_ID, ProcedureCount)

PRACTITIONER\_USES\_EQUIPMENT(SSN, Equipment\_ID)

PATIENT\_CREATES\_REVIEW(SSN, Review\_ID)

3. **Algebra:** “A few problems should have use joins instead of product”, Updated Algebra Below

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

Π First, Last, Medication\_Name((PATIENT \* PERSON) \* MEDICATION)

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

Π First, Middle, Last, SSN, Email, DOB, Gender, Patient\_ID, Signed\_HIPAA, Last\_Xray, Last\_Information\_Update (σCompany= ‘Delta Dental’(((PATIENT \* PERSON) \* PERSON\_HAS\_INSURANCE) \*INSURANCE)

* 1. Generate a list of procedures and dates of service performed by doctor Smilow. ΠProcedure\_Name, Datetime\_Performed (σlast=Smilow((PRACTITIONER\*PRACTITIONER\_CAN\_PERFORM\_PROCEDURE) \*PROCEDURE))
  2. 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.

ΠInvoice\_Name,Amount, Date, First, Middle, Last, Phone, Email (σ(Amount > 10) AND (GETDATE() - Billing\_date > 30)((((BILLING\* PAYMENT\_COVERS\_BILLING) \* PAYMENT) \*PATIENT) \*PERSON)

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

TOTAL\_AMOUNTS ← SSN FSUM Amount(((PAYMENT\_COVERS\_BILLING\*PAYMENT)\*PATIENT)\*PERSON)

MAX\_AMOUNTS ← SSN FMAX Sum\_Amount(TOTAL\_AMOUNTS)

ΠFirst,Middle,Last, Max\_Amount(MAX\_AMOUNTS)

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

ΠFirst, Middle, Last(σProcedureCount<5(((PROCEDURE \*PRACTITIONER\_CAN\_PERFORM\_PROCEDURE) \* PRACTITIONER) \* EMPLOYEE) \* PERSON))

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

HIGHEST\_PROCEDURES ← Procedure\_Name FMAX Standard\_Per\_Unit\_Charges (PROCEDURE)

NUM\_PROCEDURES ← Procedure\_Name FCOUNT Procedure\_Name (PROCEDURE)

ΠProcedure\_Name, Standard\_Per\_Unit\_Charge, Count\_Procedure(HIGHEST\_PROCEDURES \* NUM\_PROCEDURES)

* 1. 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.

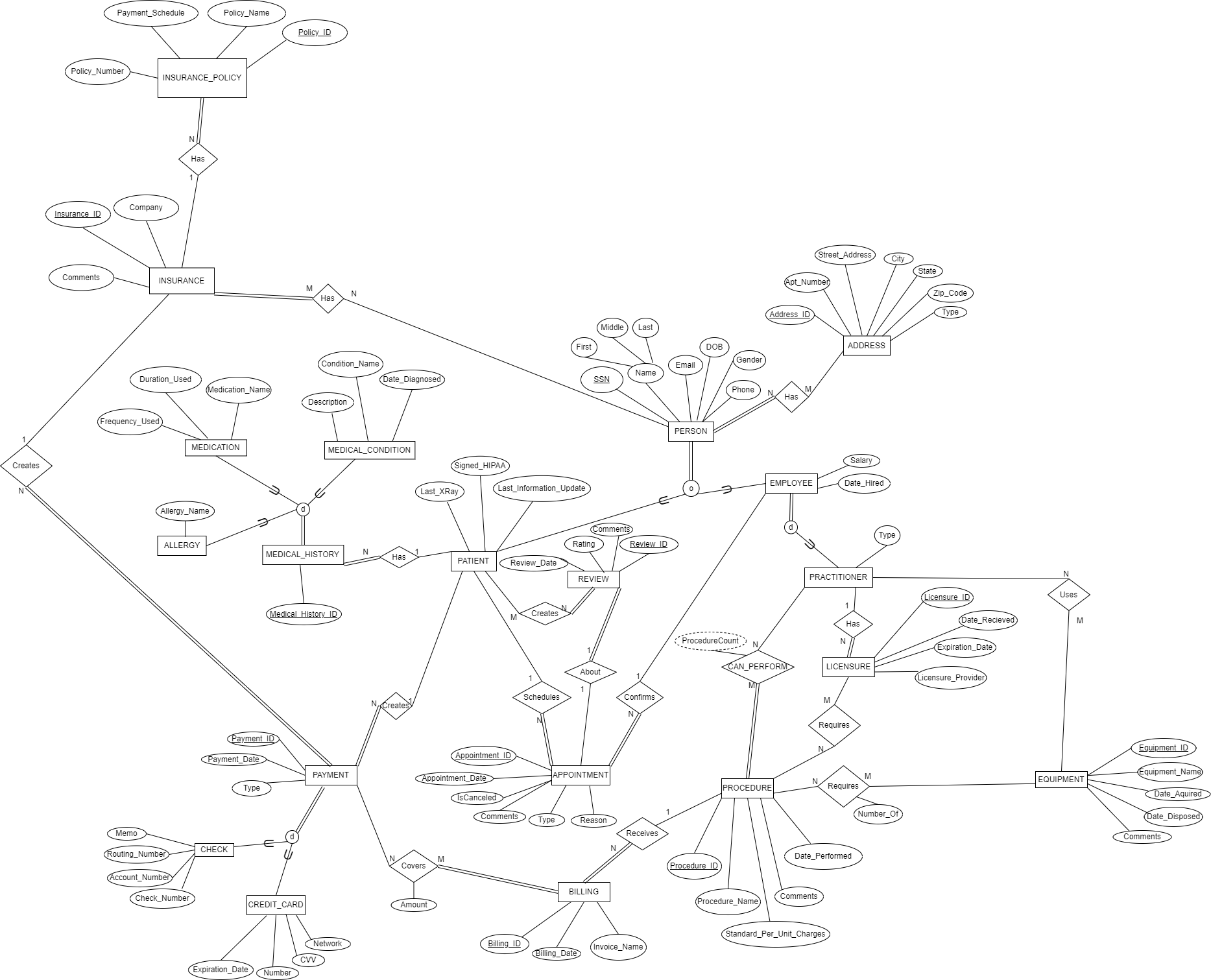
AMOUNT\_CHARGED ← Type FSUM Amount (PAYMENT\*PAYMENT\_COVERS\_BILLING)

TIMES\_USED ← Type FCOUNT Amount(PAYMENT\*PAYMENT\_COVERS\_BILLING)

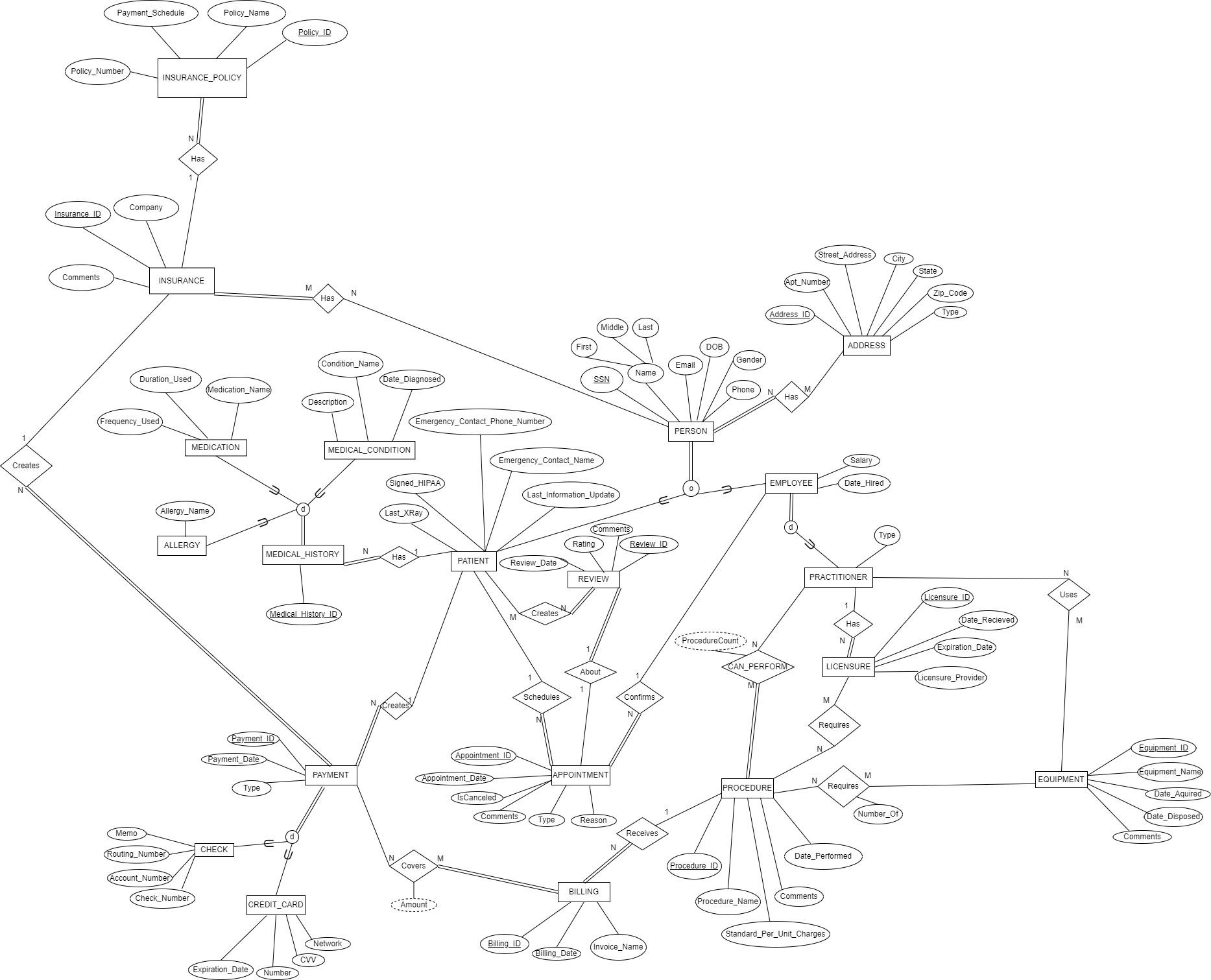
ΠType, Sum\_Amount, Count\_Amount (AMOUNT\_CHARGED \* TIMES\_USED )

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

Policy\_ID, Policy\_Name FCOUNT SSN(((INSURANCE\_POLICY\*INSURANCE)\*PERSON\_HAS\_INSURANCE)\*PERSON)

**Original ERD From pt 2**

**New/Updated ERD**

****

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. Normalization rules are applied to one relation at a time. **Make sure that your documentation shows 1NF-BCNF rules applied step by-step to each relation and listed by relation and not by NF. For each relation on your final schema:**

• Check that a relation is in 1NF and if it is not, bring it to 1NF. Explain the process and changes made.

• List functional dependencies. Make sure to consider all the possible dependencies in each relation and not just the ones from your primary keys.

• Determine the highest current normal form of that relation. Apply rules of 2NF, 3NF, and BCNF 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.

• Show your final relational schema after normalization.

**\*Note that primary keys are underlined and foreign keys are in green.**

PROCEDURE(Procedure\_ID, Procedure\_Name, Standard\_Per\_Unit\_Charges, Comments, Date\_Performed)

**1NF:** **(all attributes atomic)**

PROCEDURE: {Procedure\_ID, Procedure\_Name, Standard\_Per\_Unit\_Charges, Date\_Performed, Comments}

**2NF: (every attribute is fully dependent on any candidate key)**

-no changes needed to be made as all attributes are fully dependent on the key)

**3NF: (satisfies 2NF and no non prime attribute is transitively dependent on the primary key)**

-There are no transitive relations so nothing needs to be changed.

**BNCF: (every determinant in the relation is a candidate key)**

-no changes are necessary as the only determinant, Procedure\_Id, is a candidate key

**Functional Dependencies:**

Procedure\_ID → Procedure\_Name, Standard\_Per\_Unit\_Charges, Date\_Performed, Comments

EQUIPMENT(Equipment\_ID, Equipment\_Name, Date\_Aquired, Date\_Disposed, Comments)

**Functional Dependencies:**

Equipment\_ID -> Equipment\_Name

Equipment\_ID -> Date\_Aquired

Equipment\_ID -> Date\_Disposed

Equipment\_ID -> Comments

**Candidate Keys:** Equipment\_ID

**1NF:** The table is already in 1NF as there are no multi-valued or composite attributes.

EQUIPMENT: {Equipment\_ID, Equipment\_Name, Date\_Aquired, Date\_Disposed, Comments}

**2NF:** The table is already in 2NF as there are no partial dependencies.

**3NF:** The table is already in 3NF as there are no transitive dependencies.

**BCNF:** The table is already in BCNF as there are no violations of BCNF rules.

ADDRESS(Address\_ID, Apt\_Number, Street\_Address, City, State, Zip\_Code, Type)

**Functional Dependencies:**

Address\_ID -> Apt\_Number, Street\_Address, City, State, Zip\_Code, Type

**Candidate Keys:** Address\_ID

**1NF:** The table is already in 1NF as there are no multi-valued or composite attributes.

ADDRESS{Address\_ID, Apt\_Number, Street\_Address, City, State, Zip\_Code, Type}

**2NF:** The table is already in 2NF as there are no partial dependencies.

**3NF:** The table is already in 3NF as there are no transitive dependencies.

**BCNF:** The table is already in BCNF as there are no violations of BCNF rules.

INSURANCE(Insurance\_ID, Company, Comments)

**Functional Dependencies:**

Insurance\_ID -> Company, Comments

**Candidate Keys:** Insurance\_ID

**1NF:** The table is already in 1NF as there are no multi-valued or composite attributes.

INSURANCE: {Insurance\_ID, Company, Comments}

**2NF:** The table is already in 2NF as there are no partial dependencies.

**3NF:** The table is already in 3NF as there are no transitive dependencies.

**BCNF:** The table is already in BCNF as there are no violations of BCNF rules.

PAYMENT(Payment\_ID, Payment\_date, *Insurance\_ID, SSN*)

**Functional Dependencies:**

Payment\_ID -> Payment\_date, Insurance\_ID, SSN

**Candidate Keys:** Payment\_ID and {Insurance\_ID, SSN}.

**1NF:** The table is already in 1NF as there are no multi-valued or composite attributes.

PAYMENT{Payment\_ID, Payment\_date, *Insurance\_ID, SSN*}

**2NF:** The table is already in 2NF as there are no partial dependencies.

**3NF:** The table is already in 3NF as there are no transitive dependencies.

**BCNF:** The table is already in BCNF as there are no violations of BCNF rules.

BILLING(Billing\_ID, Billing\_date, *Procedure\_ID*)

**1NF:** **(all attributes atomic)**

-already in 1NF as all attributes are atomic

BILLING{Billing\_ID, Billing\_date, *Procedure\_ID*}

**2NF: (every attribute is fully dependent on any candidate key)**

-no changes needed to be made as all attributes are fully dependent on the key)

**3NF: (satisfies 2NF and no non prime attribute is transitively dependent on the primary key)**

-There are no transitive relations so nothing needs to be changed.

**BNCF: (every determinant in the relation is a candidate key)**

-no changes are necessary as the determinants are candidate keys

**Functional Dependencies:**

Billing\_ID → Billing\_date, Procedure\_ID

LICENSURE(Licensure\_ID, Date\_Recieved, Expiration\_Date, Licensure\_Provider, *SSN*)

**1NF:** **(all attributes atomic)**

-already in 1NF as all attributes are atomic

LICENSURE{Licensure\_ID, Date\_Recieved, Expiration\_Date, Licensure\_Provider, *SSN*}

**2NF: (every attribute is fully dependent on any candidate key)**

-no changes needed to be made as all attributes are fully dependent on the key)

**3NF: (satisfies 2NF and no non prime attribute is transitively dependent on the primary key)**

-There are no transitive relations so nothing needs to be changed.

**BNCF: (every determinant in the relation is a candidate key)**

-no changes are necessary as the determinants are candidate keys

**Functional Dependencies:**

Licensure\_ID→Date\_Recieved, Expiration\_Date, Licensure\_Provider, *SSN*

APPOINTMENT(Appointment\_ID, Appointment\_date, IsCanceled, Comments, Type, Reason, *EmployeeSSN, PatientSSN*)

**1NF:** **(all attributes atomic)**

-already in 1NF as all attributes are atomic

APPOINTMENT{Appointment\_ID, Appointment\_date, IsCanceled, Comments, Type, Reason, *EmployeeSSN, PatientSSN*}

**2NF: (every attribute is fully dependent on any candidate key)**

-no changes needed to be made as all attributes are fully dependent on the key)

**3NF: (satisfies 2NF and no non prime attribute is transitively dependent on the primary key)**

-There are no transitive relations so nothing needs to be changed.

**BNCF: (every determinant in the relation is a candidate key)**

-no changes are necessary as the determinants are candidate keys

**Functional Dependencies:**

Appointment\_ID **→** Appointment\_date, IsCanceled, Comments, Type, Reason, *EmployeeSSN, PatientSSN*

MEDICAL\_HISTORY(Medical\_History\_ID, *SSN*)

**1NF:** **(all attributes atomic)**

-already in 1NF as all attributes are atomic

MEDICAL\_HISTORY{Medical\_History\_ID, *SSN*}

**2NF: (every attribute is fully dependent on any candidate key)**

-no changes needed to be made as all attributes are fully dependent on the key)

**3NF: (satisfies 2NF and no non prime attribute is transitively dependent on the primary key)**

-There are no transitive relations so nothing needs to be changed.

**BNCF: (every determinant in the relation is a candidate key)**

-no changes are necessary as the determinants are candidate keys

**Functional Dependencies:**

Medical\_History\_ID → *SSN*

INSURANCE\_POLICY(Policy\_ID, Payment\_Schedule, Policy\_Number, Policy\_Name, *Insurance\_ID*)

**1NF:** **(all attributes atomic)**

-already in 1NF as all attributes are atomic

INSURANCE\_POLICY{Policy\_ID, Payment\_Schedule, Policy\_Number, Policy\_Name, *Insurance\_ID*}

**2NF: (every attribute is fully dependent on any candidate key)**

-no changes needed to be made as all attributes are fully dependent on the key)

**3NF: (satisfies 2NF and no non prime attribute is transitively dependent on the primary key)**

-There is a transitive dependency between Policy\_ID and Policy\_number, and policy\_number and policy\_name

INSURANCE\_POLICY{Policy\_ID, Payment\_Schedule, Policy\_Number, *Insurance\_ID*}

INSURANCE\_POLICY\_NAME {Policy\_Number, Policy\_Name}

**BNCF: (every determinant in the relation is a candidate key)**

-added policy\_naming\_mapping to satisfy BCNF

INSURANCE\_POLICY{Policy\_ID, Payment\_Schedule, Policy\_Number, *Insurance\_ID*}

INSURANCE\_POLICY\_NAME {Policy\_Number, Policy\_Name}

POLICY\_NAMING\_MAPPING {Policy\_ID, Policy\_Number}

**Functional Dependencies:**

Policy\_ID → Payment\_Schedule, Policy\_Number, *Insurance\_ID*

Policy\_Number → Policy\_Name

REVIEW(Review\_ID, Review\_date, Rating, Comments, Appointment\_ID)

**1NF:** **(all attributes atomic)**

-already in 1NF as all attributes are atomic

REVIEW{Review\_ID, Review\_date, Rating, Comments, Appointment\_ID}

**2NF: (every attribute is fully dependent on any candidate key)**

-no changes needed to be made as all attributes are fully dependent on the key)

**3NF: (satisfies 2NF and no non prime attribute is transitively dependent on the primary key)**

-There is are no transitive dependencies

**BNCF: (every determinant in the relation is a candidate key)**

-already in BNCF

**Functional Dependencies:**

Review\_ID → Review\_date, Rating, Comments, Appointment\_ID

PERSON\_HAS\_INSURANCE(SSN, Insurance\_ID)

**1NF:** The query is already in 1NF since all attributes are atomic

PERSON\_HAS\_INSURANCE: {SSN, Insurance\_ID}

**2NF:** The query is already in 2NF since there is a candidate key and no partial dependencies

**3NF:** The query is already in 3NF since there are no transitive dependencies

**BCNF:** The query is already in BCNF since there are no violations of the BCNF rules

**Functional Dependencies:**

SSN-> Insurance\_ID

PERSON\_HAS\_ADDRESS(SSN, Address\_ID)

**1NF:** The query is already in 1NF since all attributes are atomic

PERSON\_HAS\_ADDRESS: {SSN, Address\_ID}

**2NF:** The query is already in 2NF since there is a candidate key and no partial dependencies

**3NF:** The query is already in 3NF since there are no transitive dependencies

**BCNF:** The query is already in BCNF since there are no violations of the BCNF rules

**Functional Dependencies:**

SSN-> Address\_ID

PAYMENT\_COVERS\_BILLING(Payment\_ID, Billing\_ID, Amount)

**1NF**: The table is already in 1NF as there are no multi-valued or composite attributes.

PAYMENT\_COVERS\_BILLING: {Payment\_ID, Billing\_ID, Amount}

**Functional Dependencies:**

- Payment\_ID, Billing\_ID -> Amount

Candidate Keys: (Payment\_ID, Billing\_ID)

**2NF:** The table is already in 2NF as there are no partial dependencies.

**3NF:** The table is already in 3NF as there are no transitive dependencies.

**BCNF**: The table is already in BCNF as there are no violations of BCNF rules.

PROCEDURE\_REQUIRES\_EQUIPMENT(Procedure\_ID, Equipment\_ID, Number\_Of)

**1NF**: The table is already in 1NF as there are no multi-valued or composite attributes.

PROCEDURE\_REQUIRES\_EQUIPMENT: {Procedure\_ID, Equipment\_ID, Number\_Of}

**Functional Dependencies**:

- Procedure\_ID, Equipment\_ID -> Number\_Of

Candidate Keys: (Procedure\_ID, Equipment\_ID)

**2NF:** The table is already in 2NF as there are no partial dependencies.

**3NF**: The table is already in 3NF as there are no transitive dependencies.

**BCNF:** The table is already in BCNF as there are no violations of BCNF rules.

PROCEDURE\_REQUIRES\_LICENSURE(Procedure\_ID, Licensure\_ID)

**1NF:** The table is already in 1NF as there are no multi-valued or composite attributes.

PROCEDURE\_REQUIRES\_LICENSURE{Procedure\_ID, Licensure\_ID}

**Functional Dependencies:**

- Procedure\_ID, Licensure\_ID -> (no other attributes)

Candidate Keys: (Procedure\_ID, Licensure\_ID)

**2NF:** The table is already in 2NF as there are no partial dependencies.

**3NF**: The table is already in 3NF as there are no transitive dependencies.

**BCNF**: The table is already in BCNF as there are no violations of BCNF rules.

PRACTITIONER\_CAN\_PERFORM\_PROCEDURE(SSN, Procedure\_ID, ProcedureCount)

**1NF:** The table is already in 1NF as there are no multi-valued or composite attributes.

PRACTITIONER\_CAN\_PERFORM\_PROCEDURE: {SSN, Procedure\_ID, ProcedureCount}

**Functional Dependencies:**

-SSN, Procedure\_ID -> ProcedureCount

Candidate Keys: (Practitioner\_SSN, Procedure\_ID)

**2NF:** The table is already in 2NF as there are no partial dependencies.

**3NF**: The table is already in 3NF as there are no transitive dependencies.

**BCNF**: The table is already in BCNF as there are no violations of BCNF rules.

PRACTITIONER\_USES\_EQUIPMENT(SSN, Equipment\_ID)

**1NF**: The table is already in 1NF as there are no multi-valued or composite attributes.

PRACTITIONER\_USES\_EQUIPMENT{SSN, Equipment\_ID}

**Functional Dependencies:**

- SSN, Equipment\_ID → (no other attributes)

Candidate Keys: (Practitioner\_SSN, Equipment\_ID)

**2NF**: The table is already in 2NF as there are no partial dependencies.

**3NF**: The table is already in 3NF as there are no transitive dependencies.

**BCNF**: The table is already in BCNF as there are no violations of BCNF rules.

PATIENT\_CREATES\_REVIEW(SSN, Review\_ID)

**1NF**: The table is already in 1NF as there are no multi-valued or composite attributes.

PATIENT\_CREATES\_REVIEW{SSN, Review\_ID}

**Functional Dependencies**:

- SSN, Review\_ID → (no other attributes)

Candidate Keys: (Patient\_SSN, Review\_ID)

**2NF**: The table is already in 2NF as there are no partial dependencies.

**3NF**: The table is already in 3NF as there are no transitive dependencies.

**BCNF**: The table is already in BCNF as there are no violations of BCNF rules.

MEDICATION(*Medical\_History\_ID*, Date\_Prescribed, Medication\_Name, Frequency\_Used, Duration\_Used)

**1NF**: The table is already in 1NF as there are no multi-valued or composite attributes.

MEDICATION{*Medical\_History\_ID*, Date\_Prescribed, Medication\_Name, Frequency\_Used, Duration\_Used}

**Functional Dependencies:**

- Medical\_History\_ID -> Date\_Prescribed, Medication\_Name, Frequency\_Used, Duration\_Used

Candidate Keys: Medication\_ID

**2NF**: The table is already in 2NF as there are no partial dependencies.

**3NF**: The table is already in 3NF as there are no transitive dependencies.

**BCNF**: The table is already in BCNF as there are no violations of BCNF rules.

MEDICAL\_CONDITION(*Medical\_History\_ID*, Condition\_Name, Description, Date\_Diagnosed)

**1NF**: The table is already in 1NF as there are no multi-valued or composite attributes.

MEDICAL\_CONDITION{*Medical\_History\_ID*, Condition\_Name, Description, Date\_Diagnosed}

**Functional Dependencies:**

- Medical\_History\_ID -> Condition\_Name, Description, Date\_Diagnosed

Candidate Keys: Condition\_ID

**2NF**: The table is already in 2NF as there are no partial dependencies.

**3NF**: The table is already in 3NF as there are no transitive dependencies.

**BCNF**: The table is already in BCNF as there are no violations of BCNF rules.

ALLERGY(*Medical\_History\_ID*, Allergy\_Name)

**1NF**: The table is already in 1NF as there are no multi-valued or composite attributes.

ALLERGY{*Medical\_History\_ID*, Allergy\_Name}

**Functional Dependencies:**

- *Medical\_History\_ID* -> Allergy\_Name

Candidate Keys: Allergy\_ID

**2NF**: The table is already in 2NF as there are no partial dependencies.

**3NF**: The table is already in 3NF as there are no transitive dependencies.

**BCNF**: The table is already in BCNF as there are no violations of BCNF rules.

PERSON(SSN, First, Middle, Last, Email, DOB, Gender, Phone)

**1NF**: The table is already in 1NF as there are no multi-valued or composite attributes.

PERSON{SSN, First, Middle, Last, Email, DOB, Gender, Phone}

**Functional Dependencies:**

- SSN -> First, Middle, Last, Email, DOB, Gender, Phone

Candidate Keys: SSN

**2NF**: The table is already in 2NF as there are no partial dependencies.

**3NF**: The table is already in 3NF as there are no transitive dependencies.

**BCNF**: The table is already in BCNF as there are no violations of BCNF rules.

EMPLOYEE(*SSN*, Salary, Date\_Hired)

**1NF**: The table is already in 1NF as there are no multi-valued or composite attributes.

EMPLOYEE{*SSN*, Salary, Date\_Hired}

**Functional Dependencies:**

- SSN -> Salary, Date\_Hired

Candidate Keys: SSN

**2NF**: The table is already in 2NF as there are no partial dependencies.

**3NF**: The table is already in 3NF as there are no transitive dependencies.

**BCNF**: The table is already in BCNF as there are no violations of BCNF rules.

PRACTITIONER(*SSN*, Type)

**1NF**: The table is already in 1NF as there are no multi-valued or composite attributes.

PRACTITIONER{*SSN*, Type}

**Functional Dependencies:**

- SSN -> Type

Candidate Keys: SSN

**2NF**: The table is already in 2NF as there are no partial dependencies.

**3NF**: The table is already in 3NF as there are no transitive dependencies.

**BCNF**: The table is already in BCNF as there are no violations of BCNF rules.

PATIENT(*SSN*, Last\_XRay, Signed\_HIPAA, Last\_Information\_Update, Emergency\_Contact\_Name, Emergency\_Contact\_Phone\_Number)

**1NF**: The table is already in 1NF as there are no multi-valued or composite attributes.

PATIENT{*SSN*, Last\_XRay, Signed\_HIPAA, Last\_Information\_Update, Emergency\_Contact\_Name, Emergency\_Contact\_Phone\_Number}

**Functional Dependencies:**

- SSN -> Last\_XRay, Signed\_HIPAA, Last\_Information\_Update, Emergency\_Contact\_Name, Emergency\_Contact\_Phone

Candidate Keys: SSN

**2NF**: The table is already in 2NF as there are no partial dependencies.

**3NF:** The table is already in 3NF as there are no transitive dependencies.

**BCNF**: The table is already in BCNF as there are no violations of BCNF rules.

PAYMENT(Payment\_ID, Payment\_date, Type *Insurance\_ID, SSN*)

**1NF**: The query is in 1NF so all values are atomic with no repeating groups

PAYMENT{Payment\_ID, Payment\_date, Type *Insurance\_ID, SSN*}

**2NF:** The query is already in 2NF since each attribute is dependent on Pyament\_ID and it is the

primary key so there are no partial dependencies

**3NF:** There are no transitive dependencies since none of the non-key attributes are dependent on any other non-key attribute. The query is in 3NF.

**BCNF:** The query is in BCNF since there are no violations of BCNF rules.

**Functional Dependencies:** Payment\_ID-> Payment\_date, Type, Insurance\_ID, SSN

CREDIT\_CARD(*Payment\_ID*, Expiration\_Date, Number, CVV, Network)

**1NF**: All domain values are atomic and there are no repeating groups

CREDIT\_CARD{*Payment\_ID*, Expiration\_Date, Number, CVV, Network}

**2NF**: The query is already in 2NF since there are no partial dependencies on the primary key, and each non-key attribute is fully dependent on the entire primary key

**3NF**: The query is in 3NF since there are no transitive dependencies between non-key attributes

**BCNF**: The query is in BCNF since there are no non-trivial functional dependencies of non-key attributes on any of the candidate keys

**Functional Dependencies:**

Payment\_ID-> Expiration\_Date, Number, CVV, Network

CHECK(*Payment\_ID*, Memo, Routing\_Number, Account\_Number, Check\_Number)

**1NF**: The query is already in 1NF since it has a primary key and no repeating groups

CHECK: { *Payment\_ID*, Memo, Routing\_Number, Account\_Number, Check\_Number}

**2NF**: The query is in 2NF since there are not any partial dependencies on the primary key, and each non-key attribute is fully dependent on the entire primary key

**3NF**: The query is also in 3NF since there are no transitive dependencies between non-key attributes

**BCNF**: The query is in BCNF since non-trivial functional dependencies of non key attributes exist on any candidate key

**Functional dependencies**:

Payment\_ID-> Memo, Routhing\_Number, Account\_Number, Check\_Number

**Final Relational Schema After Normalization:**

PROCEDURE: {Procedure\_ID, Procedure\_Name, Standard\_Per\_Unit\_Charges, Date\_Performed, Comments}

EQUIPMENT(Equipment\_ID, Equipment\_Name, Date\_Aquired, Date\_Disposed, Comments)

ADDRESS(Address\_ID, Apt\_Number, Street\_Address, City, State, Zip\_Code, Type)

INSURANCE(Insurance\_ID, Company, Comments)

INSURANCE\_POLICY{Policy\_ID, Payment\_Schedule, Policy\_Number, *Insurance\_ID*}

INSURANCE\_POLICY\_NAME {Policy\_Number, Policy\_Name}

POLICY\_NAMING\_MAPPING {Policy\_ID, Policy\_Number}

PAYMENT(Payment\_ID, Payment\_date, *Insurance\_ID, SSN*)

BILLING(Billing\_ID, Billing\_date, *Procedure\_ID*)

LICENSURE(Licensure\_ID, Date\_Recieved, Expiration\_Date, Licensure\_Provider, *SSN*)

APPOINTMENT(Appointment\_ID, Appointment\_date, IsCanceled, Comments, Type, Reason, *EmployeeSSN, PatientSSN*)

REVIEW(Review\_ID, Review\_date, Rating, Comments, Appointment\_ID)

PERSON\_HAS\_INSURANCE(SSN, Insurance\_ID)

PERSON\_HAS\_ADDRESS(SSN, Address\_ID)

PAYMENT\_COVERS\_BILLING(Payment\_ID, Billing\_ID, Amount)

PROCEDURE\_REQUIRES\_EQUIPMENT(Procedure\_ID, Equipment\_ID, Number\_Of)

PROCEDURE\_REQUIRES\_LICENSURE(Procedure\_ID, Licensure\_ID)

PRACTITIONER\_CAN\_PERFORM\_PROCEDURE(SSN, Procedure\_ID, ProcedureCount)

PRACTITIONER\_USES\_EQUIPMENT(SSN, Equipment\_ID)

PATIENT\_CREATES\_REVIEW(SSN, Review\_ID)

MEDICAL\_HISTORY(Medical\_History\_ID, *SSN*)

MEDICATION(*Medical\_History\_ID*, Date\_Prescribed, Medication\_Name, Frequency\_Used, Duration\_Used)

MEDICAL\_CONDITION(*Medical\_History\_ID*, Condition\_Name, Description,

ALLERGY(*Medical\_History\_ID*, Allergy\_Name)

PERSON(SSN, First, Middle, Last, Email, DOB, Gender, Phone)

EMPLOYEE(*SSN*, Salary, Date\_Hired)

PRACTITIONER(*SSN*, Type)

PATIENT(*SSN*, Last\_XRay, Signed\_HIPAA, Last\_Information\_Update, Emergency\_Contact\_Name, Emergency\_Contact\_Phone\_Number)

PAYMENT(Payment\_ID, Payment\_date, Type *Insurance\_ID, SSN*)

CREDIT\_CARD(*Payment\_ID*, Expiration\_Date, Number, CVV, Network)

CHECK(*Payment\_ID*, Memo, Routing\_Number, Account\_Number, Check\_Number)

3. Given your normalized 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 the table. However, that number can fluctuate depending on the 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, constraints, views, and data. Ensure that your code runs and produces correct results in either SSMS or SQLiteOnline (sqliteonline.com) as we will be using one of those platforms to test your code. Clearly indicate in your solution which one was used. Save all CREATE / ALTER TABLE STATEMENTS in a file called “CreateQueries.txt” and all applicable INSERT statements in a file called “InsertQueries.txt”.

-Done, see attached files . We used SQLiteOnline. Note, when using SQLiteOnline, do not include the first two commands in the createQueries.txt. They are commented out but shown for it this code is tested in other systems.

-- Create the database

--CREATE DATABASE DentalClinic;

-- Use the created database

--USE DentalClinic;

These are not needed in SQLiteOnline

**IMPORANT NOTE: For the following questions, if your relational schema cannot provide answers to these queries, revise your (E)ERD, 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.

-Done, see simpleQueries.txt

5. For Project PART 2 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 PART 1

See ExtraQueries.txt

1. Retrieving a person’s name and address who left a review below 5 stars:

SUBPAR\_REVIEW ← σRating<5 (REVIEW)

PERSON\_DETAILS ← ΠFirst, Middle, Last, ,Appointment\_Date, Street\_Address (((PATIENT \* PERSON)\*APPOINTMENT)\*ADDRESS)

PERSON\_BAD\_REVIEW ← PERSON\_DETAILS ⟕Appointment\_Date=Review\_Date SUBPAR\_REVIEW

2. Finds the highest paying position:

EMPLOYEE\_DETAILS ← ΠFirst, Middle, Last, Employee\_ID, Salary (EMPLOYEE \* PERSON)

PRACTIONER\_NAMES ← EMPLOYEE\_DETAILS⟗SSN= SSNPRACTITIONERS

MAX\_PROFESSION ← Type FMAX Salary(PRACTIONER\_NAMES)

3. Lists the employees and the amount of equipment they have used:

PRACTIONER\_DETAILS ← ΠFirst, Middle, Last, ((EMPLOYEE \*PRACTITIONER)\* PERSON)

Equipment\_Name FCOUNT SSN(PRACTIONER\_DETAILS\*EQUIPMENT)

0.

6. CROSS\_CHECK: Make sure that all your SQL code is properly formatted, easy to read, and label each query with SQL supported comments. DO not use any non-SQL supported contents in your scripts. They should execute as one unit by using copy / paste commands. All SELECT queries should produce meaningful results.

7. Document work being done for this portion of the project and team member contributions. Report any relevant team issues / praises / concerns.

Bryan Vales - Normalization and Query Writing

Lohith Maralla - Normalization, InsertQueries, and Simple Queries

Ben Borszcz - Normalization and CreateTable/SimpleQueries/ExtraQueries Writing

Regina Powers - Feedback, Normalization, InsertQueries, and Simple Queries.

Our team continues to work well together. We schedule times when all of us can meet and then work together on tasks, asking questions, and helping each other.

8. 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 PART 2 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 either SSMS or the SQLiteOnline application (\*sqlite, \*.db). Clearly specify in your report and code comments if it was created in SSMS or SQLiteonline.

• Text formatted SQL files for questions 3-5:

o **CreateQueries.txt**

o **InsertQueries.txt**

o **SimpleQueries.txt**

o **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 to identify each query, do not use any non-SQL compatible text or syntax in your code. Entire team is responsible to check for the presence and correctness of all submitted work. **Clearly indicate what RDBMS you used to create your code so we can use the same one to test it!**

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