CSDS 341 Final Report

Camaryn Petersen & Mauli Patel

31 July 2024

Integrated Medicare Data Repository (IMDR)

Table of Contents

1. Application Background

Medicare is the United States federal health insurance for anyone over the age of 65 or for people with qualifying conditions[1]. As of 2023, nearly 66 million Americans were enrolled in Medicare[2]. Despite its importance and widespread usage, Medicare data is not easily accessible or comprehensible for the average user.

Currently, there is a database similar to this concept called the Medicare & Medicaid Statistical Supplement[3]. This database is managed by the Centers for Medicare & Medicaid Services (CMS) and includes extensive data on Medicare beneficiaries, claims, and various healthcare services. The database is divided into several sections, each focusing on different aspects of Medicare data, such as Beneficiary Summary, Inpatient Claims, Outpatient Claims, Carrier Claims, and Prescription Drug Events. We wanted to create a database similar to this that mimicked the functionality of the larger database and also added some of our own unique features to it. Specifically, our data merges all aspects of Medicare data, creating a more comprehensive and accessible way of storing the information.

The goal of this project is to restructure Medicare Beneficiary, Inpatient, and Outpatient data into a more efficient database and create a management system that consolidates Medicare data into an accessible and cohesive format. This has the potential to be a publicly accessible database that can be used by healthcare analysts, policymakers, and researchers to better understand healthcare usage, patient demographics, and cost patterns. It would help analyze service patterns, assess Medicare policy effectiveness, and find opportunities for cost savings for the federal government. For example, a patient could query information about all claims that are associated with their beneficiary ID number. Insurance providers can easily identify patients who meet the criteria for specific coverage plans. Financial advisers could also use the information to determine if there is a high demand for costly prescriptions in a specific area. Overall, the primary objective is to develop an accessible platform that simplifies the extraction, analysis, and interpretation of Medicare data, aiding in better healthcare decision-making and policy formulation.

2. Data Description & Retrieval

Medicare data is managed by the Centers for Medicare and Medicaid (CMS) services and is currently stored in 5 sections: Beneficiary Summary, Inpatient Claims, Outpatient Claims, Carrier Clairvices, outpatient surgery, lab tests, X-rays, or any other hospital services and were not admitted by a doctor as an inpatient. A beneficiary's hospital status affects their insurance

[1] *Medicare*, United States Government, www.medicare.gov/what-medicare-covers/your-medicare-coverage-choices/whats-medicare.
[2] *Medicare Enrollment Numbers,* Centers for Medicare and Medicaid Services, https://medicareadvocacy.org/medicare-enrollment-numbers/
[3] *Medicare*, United States Government, https://www.medicare.gov/what-medicare-covers/what-part-a-covers/inpatient-or-outpatient-hospital-status

coverage and how much they pay for services[4]. Provider Claims (claims submitted by a physician or other clinician in a given health institute) and Prescription Drug Events (claims regarding medications prescribed to a beneficiary) are also included in Medicare data, but for simplicity purposes, will not be included in this project. ms, and Prescription Drug Events, or Parts A-E respectively. The CMS Beneficiary Summary contains 32 variables that describe information about the beneficiary (patient/insurance holder) such as race, sex, and date of birth. The CMS Inpatient Claims dataset contains 81 variables that describe inpatient encounters, while the CMS Outpatient Claims dataset contains 76 variables that describe outpatient encounters[5]. We will also be using The CMS Prescription Drug Events which outlines 8 variables related to prescription drug events. By definition, inpatient refers to those who are formally admitted to the hospital with a doctor's order. Outpatient refers to those who receive emergency department services and observation. The drug event captures all drug-related info such as product details, quantities dispensed, days of supply, and financial information such as patient payments and total drug costs. We will adjust and utilize these variables according to need.

We used the CMS 2008-2010 Data Entrepreneurs' Synthetic Public Use File (DE-SynPUF) which can be accessed at the following link:

CMS 2008-2010 Data Entrepreneurs' Synthetic Public Use File (DE-SynPUF)

We merged and combined different columns from each sample to make our own unique dataset. This dataset provided a realistic but synthetic set of claims data to allow data entrepreneurs to design software or applications that could be applied to actual claims data in the future. Due to HIPAA compliance issues, we utilized synthetic data for this project, as real Medicare data was not publicly available. Additionally, the sheer volume and dispersed nature of the insurance data that covered millions of U.S. citizens presented significant challenges in accessing it. This complexity emphasized the necessity of developing an efficient database system that had open access. Our current synthetic dataset captured a portion of this complexity, containing a vast 42 columns and 50,000 rows, it was designed to reflect the intricacies of actual Medicare data while still being small enough to maintain on both partner's computers.

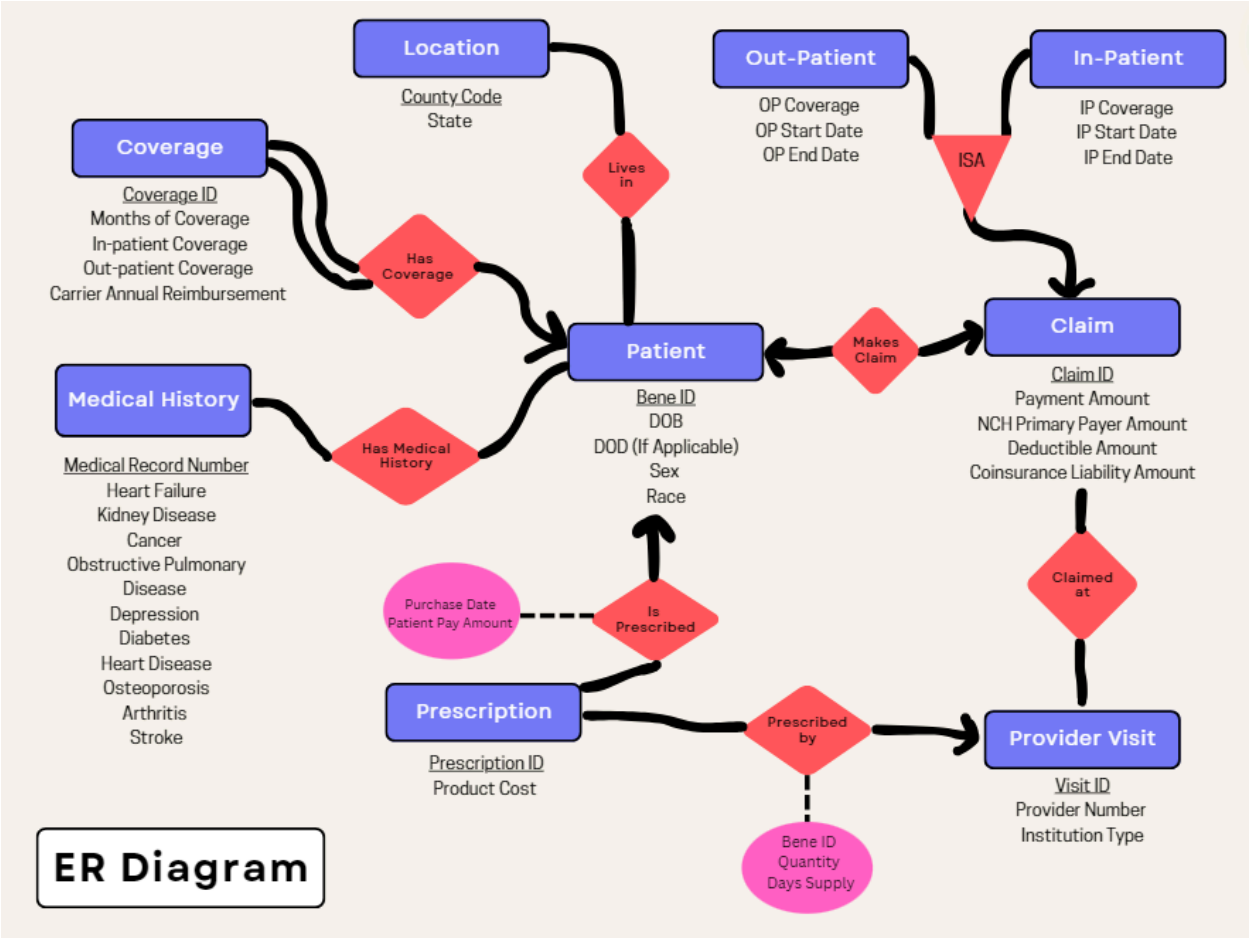
As part of our data preprocessing tasks, we decoded the dataset, which initially used numerical representations for categorical variables (e.g., 'Male' was coded as '1' and 'Female' as '2'). In addition to the initial dataset, we synthetically created data for three new columns: 'INST_TYPE,' 'CLM_ID,' and 'COVERAGE_ID.' These columns were necessary as the original

[4] *Medicare*, United States Government, https://www.medicare.gov/what-medicare-covers/what-part-a-covers/inpatient-or-outpatient-hospital-status

[5] Centers for Medicare & Medicaid Services, United States Government, https://www.cms.gov/data-research/statistics-trends-and-reports/medicare-claims-synthetic-public-use-files/cms-2008-2010-data-entrepreneurs-synthetic-public-use-file-de-synpuf.

dataset did not include them in the required format for our project, and they needed to contain unique values since they were designated as primary keys. We also had to manually fill in some data to ensure there were no gaps or missing values, adhering to constraints. To compile our final dataset, we randomly sampled relevant columns from five different categories: Beneficiary Summary, Beneficiary Info, Inpatient Claims, Outpatient Claims, Carrier Claims, and Prescription Drug Events (PDE). We used Python to perform the random sampling.

3. ER Diagram



All of the relations in the E/R diagram are mapped to the database relations with the same name and attributes. However, some constraints are not captured by the ER diagram. First, the “End Date” for the claim should have passed at the time of data entry, as we are not keeping track of active records. Additionally, the start date of a claim must always be earlier than or equal to its end date. Finally, the “Patient Pay Amount” for a prescription should be less than or equal to the “Product Cost” of the prescription (some patients will apply their insurance coverage and pay a lower out-of-pocket cost, others will pay the full cost of the drug).

4. Schemas for ER Diagram

Patient (Bene ID, DOB, DOD, Sex, Race)

Claim (Claim ID, Payment Amount, NCH Primary Payer Amount, Deductible Amount, Coinsurance Liability Amount)

InpatientClaim (Claim ID, Inpatient Coverage, Inpatient Start Date, Inpatient End Date)
Foreign Key (Claim ID) references Claim (Claim ID)
Foreign Key(Inpatient Coverage) references Coverage(Inpatient Coverage)

OutpatientClaim (Claim ID, Outpatient Coverage, Outpatient Start Date, Outpatient End Date)
Foreign Key (Claim ID) references Claim(Claim ID)
Foreign Key(Outpatient Coverage) references Coverage(Inpatient Coverage)

Visits (Visit ID, Provider Number, Institution Type,)

Prescription (Prescription ID, Product Cost)

MedicalHistory (Medical Record Number, Heart Failure, Kidney Disease, Cancer, Obstructive Pulmonary Disease, Depression, Diabetes, Heart Disease, Osteoporosis, Arthritis, Stroke)

Coverage (Coverage ID, Coverage Months, Inpatient Coverage, Outpatient Coverage, Carrier Reimbursement)

Location (County Code, State)

LivesIn (Bene ID, County Code)
Foreign Key (Bene ID) references Patient(Bene ID)
Foreign Key (County Code) references Location(County Code)

MakesClaim (Bene ID, Claim ID)
Foreign Key (Bene ID) references Patient(Bene ID)
Foreign Key (Claim ID) references Claim(Claim ID)

ClaimedAt (Claim ID, Visit ID)
Foreign Key (Claim ID) references Claim(Claim ID)
Foreign Key (Visit ID) references Provider(Visit ID)

PrescribedBy (Prescription ID, Visit ID, Quantity, Days Supply)
Foreign Key (Visit ID) references Provider(Visit ID)
Foreign Key (Prescription ID) references Prescription(Prescription ID)

IsPrescribed (Bene ID, Prescription ID, Purchase Date, Payment Amount)
Foreign Key (Bene ID) references Patient(Bene ID)
Foreign Key (Prescription ID) references Prescription(Prescription ID)

HasMedicalHistory (Bene ID, Medical Record Number)
Foreign Key (Bene ID) references Patient(Bene ID)
Foreign Key (Medical Record Number) references MedicalHistory(Medical Record Number)

hasCoverage(Bene ID, Coverage ID)
Foreign Key (Bene ID) references Patient(Bene ID)
Foreign Key (Coverage ID) references Coverage(Coverage ID)

5. Normalization Edits

Originally, we had "County" and "State" data stored in the Patient table. "State" functionally depends on "County" but "County Code" is not a superkey and the relationship is not trivial, so this violates BCNF. Therefore, we normalized this table by decomposition the original "Patient" table into "Patient", "Location", and "LivesIn".

Original Patient Schema

R = {Bene ID, DOB, DOD, Sex, Race, County Code, State}
F = {{Bene ID → DOB, DOD, Sex, Race, County Code, State},
{County Code → State}} violates BCNF

Decomposition

R_1 = {Bene ID, DOB, DOD, Sex, Race}
F_1 = {Bene ID → DOB, DOD, Sex, Race}
R_2 = {Bene ID, County Code}
F_2 = {Bene ID → County Code}
R_3 = {County Code, State}
F_3 = {County Code → State}

This decomposition is lossless because $R_1 \cap R_2$ = {Bene ID}, which is a key for R_1, and $R_2 \cap R_3$ = {County Code}, which is a key for R_2. This decomposition is dependency preserving because $F_1 \cup F_2 \cup F_3 = F$.

In our original schema, we stored all of the "prescription" information in one table. Each prescribed product is associated with a unique price and the quantity/days supply can be determined from the visit ID number. Patients may be prescribed multiple prescriptions, and the same prescription drug is likely prescribed to multiple patients. Since patients can refill a prescription, they can receive the same prescription at different visits, different quantities and days supply, and different purchase dates and payment amounts. Patients may receive multiple prescriptions at one visit. By decomposing the original Prescription table into Prescriptions, PrescribedBy, and isPrescribed, we take care of the function dependencies that violate BCNF.

Original Prescription Schema

R = {Bene ID, Prescription ID, Product Cost, Payment Amount,
Purchase Date, VisitID, Quantity, Days Supply}
F = {Prescription ID → Product Cost},
{Prescription ID, Visit ID → Quantity, Days Supply},
{Bene ID, Prescription ID, Purchase Date} → Payment Amount} } FDs violate BCNF

Decomposition

R_1 = {Prescription ID, Product Cost}
F_1 = {Prescription ID → Product Cost}
R_2 = {Prescription ID, Visit ID, Quantity, Days Supply}
F_2 = {Prescription ID, Visit ID → Quantity, Days Supply}
R_3 = {Bene ID, Prescription ID, Payment Amount, Purchase Date}
F_3 = {Bene ID, Prescription ID, Purchase Date} → Payment Amount

This decomposition is lossless because $R_1 \cap R_2$ = {Prescription ID}, which is a key for R_1, and $R_1 \cap R_3$ = {Prescription ID}, which is a key for R_1. This decomposition is dependency preserving because $F_1 \cup F_2 \cup F_3 = F$.

6. Functional Dependencies

Patient

The "Patient" table consists of personal information about a patient/beneficiary, including their date of birth, sex, and race. These data are independent of each other, thus the only functional dependency for this table is the primary key implying all of the other attributes:

Bene ID → {DOB, DOD, Sex, Race)

This is in BCNF since Bene ID is a super key.

Claims

The "Claim" table lists details about each claim made by a beneficiary. The Payment Amount, Primary Payer Amount, Deductible Amount, and Coinsurance Liability Amount are uniquely calculated for each claim, thus, there is no redundant information in this table. Again, the only functional dependency is the primary key implying all of the other attributes:

Claim ID → {Payment Amount, NCH Primary Payer Amount, Deductible Amount, Coinsurance Liability Amount}

This is in BCNF since Claim ID is a super key.

Inpatient & Outpatient Claims

The Inpatient and Outpatient tables contain additional information about claims depending on whether the patient was seen at an inpatient or outpatient facility. The Coverage, Start Date, and End Date variables are all independent of each other, and can only be implied by the primary key:

Claim ID → {Inpatient Coverage, Inpatient Start Date, Inpatient End Date}

Claim ID → {Outpatient Coverage, Outpatient Start Date, Outpatient End Date}

This is in BCNF since Claim ID is a super key for both the inpatient and outpatient claims tables.

Visits

The Visits table contains information about the institution that the patient was seen at. Since providers can work at multiple institution types, and each institution type has multiple providers, the only functional dependency in this table depends on the primary key, which is the unique identifier for each visit in the system:

Visit ID → {Provider Number, Institution Type}

This is in BCNF since Visit ID is a super key.

Prescription

The Prescription table lists information about prescription products and their associated costs. Again, the only functional dependency in this table depends on the primary key, which is the unique identifier for each visit in the system:

Prescription ID → {Total Cost}

This is in BCNF since the Prescription ID is a super key.

Medical History

The Medical History table provides information about each patient's chronic condition. Storing this information separately from the Patient table accounts for patients who may not have a medical history record in the database (thus avoiding null values) and increases efficiency in updating medical history for a patient without updating their basic information. Since the diseases included in this table are independent of each other, we only have one functional dependency on the primary key:

Medical Record Number → {Heart Failure, Kidney Disease, Cancer, Obstructive Pulmonary Disease, Depression, Diabetes, Heart Disease, Osteoporosis, Arthritis, Stroke}

This is in BCNF since Medical Record Number is a super key.

Coverage

The Coverage table contains information about the unique coverage plan that each patient has. Since patients may have multiple coverage plans, we stored this data in a separate table from Patients to avoid redundant information. Again, the attributes in this table are independent of each other, so the only functional dependency is on the primary key:

Coverage ID → {Coverage Months, Inpatient Coverage, Outpatient Coverage, Carrier Reimbursement}

This is in BCNF since Coverage ID a super key.

Location

The Location table stores where each patient lives. Since the state is functionally dependent on the county code, we stored these attributes in a separate table from Patient. The only functional dependency in the Location table is on the primary key:

County Code → {State}

This is in BCNF since County Code is a super key.

LivesIn

The LivesIn table stores information about which county each patient lives in. Each patient can only live in one county, thus, the only functional dependency is on the primary key:

Bene ID → {County_Code}

This is in BCNF since Bene ID is a super key.

MakesClaim

The MakesClaim table connects patients to each claim they have made. Since each claim belongs to only one patient, but one patient can make multiple claims, the only function dependency is:

Claim ID → {Bene_ID}

This is in BCNF since Claim ID is a super key.

ClaimedAt

The ClaimedAt table connects claims to visits to institutions. Since the Claim ID and Visit ID are independent of each other, and the only two attributes in this table, there are no functional dependencies for this table.

{Claim ID, Visit ID} → {}

Prescribed By

The PrescribedBy table stores information about prescriptions written for patients during their visit. Physicians may prescribe the same drug (with the same prescription ID) with different quantities/day supply for the same patient at different visits. Additionally, a patient can receive multiple prescriptions at the same visit, and the same quantity can equate to different day supplies for different patients. Thus, the only functional dependency is:

{Prescription ID, Visit ID} → {Quantity, Days Supply}

This is in BCNF since Prescription ID and Visit ID are super keys.

IsPrescribed

The IsPrescribed table stores information about prescriptions that the patient has received. Patients can purchase the same prescription product on multiple dates and pay different amounts. Patients may have multiple prescriptions and products are prescribed to multiple patients. Overall, Prescription ID, Purchase date, Payment amount, and Bene ID are independent of each other, so there are no functional dependencies in this table.

HasCoverage

This table connects patients to the various coverage plans they may have. Since each unique coverage plan can only belong to one patient, the only functional dependency is:

{Coverage ID → Bene ID}

This is in BCNF since Coverage ID is a super key.

HasMedicalHistory

This table connects patients to their medical history. Each patient can only have one medical history, and each medical history record is connected to only one patient. Thus, the functional dependencies for this table are:

{Bene ID} → {Medical Record Number}

{Medical Record Number} → {Bene ID}

This is in BCNF since Bene ID and Medical Record Number are super keys.

7. Relation Schemas

Entities

```
create table Patient (
  beneID char(16) primary key,
  DOB int(8) null,
  DOD int(8) null,
  sex char(10) null,
  race char(10) null,
  countyCode int(3) null,
  medicalRecordNumber int(7) null,
  constraint Patient_countyCode_FK
          foreign key (countyCode) references Location (countyCode)
                  on update cascade
   constraint Patient_medicalRecordNumber_FK
          foreign key (medicalRecordNumber) references MedicalHistory(medicalRecordNumber)
                  on update cascade
 );

create table Visits (
   visitID char(14) primary key,
   providerNumber char(14) null,
   institutionType char(30) null,
   countyCode int(3) null,
   constraint Patient_countyCode_FK
          foreign key (countyCode) references Location (countyCode)
                  on update cascade
 );

 create table Coverage (
   beneID char(16),
   coverageID char(16),
   coverageMonths int(2) null,
   inpatientCoverage int(8) null,
   outpatientCoverage int(8) null,
   carrierReimbursement int(8) null,
   primary key (beneID, coverageID),
   constraint Coverage_beneID_FK
          foreign key (beneID) references Patient(beneID)
                  on update cascade on delete cascade
);
create table Claims(
  claimID int(14) primary key,
  paymentAmount int(8) null,
  payerAmount int(8) null,
  deductibleAmount int(8) null,
  coInsuranceAmount int(8) null,
  beneID char(16) not null,
  visitID char(14) NOT null,
  constraint Claims_beneID_FK
          foreign key (beneID) references Patient(beneID)
                  on update cascade
```

```
  constraint Claims_visitID_FK
        foreign key (visitID) references Visits(visitID)
            on update cascade
);
create table InpatientClaims (
  claimID int(14) primary key,
  inpatientStartDate int(8),
  inpatientEndDate int(8),
  inpatientCoverage int(8) null references Coverage(inpatientCoverage),
  constraint inpatientClaims_claimID_FK
        foreign key (claimID) references Claims(claimID)
            on update cascade on delete cascade
);
create table OutpatientClaims (
  claimID int(14) primary key,
  outpatientStartDate int(8),
  outpatientEndDate int(8),
  outpatientCoverage int(8) null references Coverage(outpatientCoverage),
  constraint outpatientClaims_claimID_FK
        foreign key (claimID) references Claims(claimID)
            on update cascade on delete cascade
);
create table Prescription (
  prescriptionID int(11) primary key,
  productCost int(4)
);
create table Location (
  countyCode int(3) primary key,
  state char(6) NULL
);
create table MedicalHistory (
  medicalRecordNumber int(7) primary key,
  'Alzheimers?' char(3) null,
  'Heart Failure?' char(3) null,
  'Kidney Disease?' char(3) null,
  'Cancer?' char(3) null,
  'OPD?' char(3) null,
  'Depression?' char(3) null,
  'Diabetes?' char(3) null,
  'Heart Disease?' char(3) null,
  'Osteoporosis?' char(3) null,
  'Arthritis?' char(3) null,
  'Stroke?' char(3) null
);
```

Relationships

```
create table MakesClaim (
  beneID char(16),
  claimID char(14),
```

```
  primary key (beneID, claimID),
  constraint makesCLaim_beneID_FK
        foreign key (beneID) references Patient(beneID)
              on update cascade on delete cascade,
  constraint makesClaim_claimID_FK
        foreign key (claimID) references Claims(claimID)
              on update cascade on delete cascade
 );

 create table ClaimedAt (
   claimID char(14),
   visitid char(14),
   primary key (visitid, claimID),
   constraint ClaimedAt_claimID_FK
        foreign key (claimID) references Claims(claimID)
              on update cascade on delete cascade
   constraint ClaimedAt_visitID_FK
        foreign key (visitID) references Visit(visitid)
              on update cascade on delete cascade
);
create table IsPrescribed (
  beneID char(14),
  prescriptionID int(11),
  purchaseDate date null,
  paymentAmount numeric(6,2) null,
  primary key (beneID, prescriptionID),
  constraint IsPrescribed_beneID_FK
        foreign key (beneID) references Patient(beneID)
              on update cascade on delete cascade,
  constraint IsPrescribed_prescriptionID_FK
        foreign key (prescriptionID) references Prescription(prescriptionID)
              on update cascade on delete cascade
);
create table PrescribedBy (
  beneID char(14),
  prescriptionID int(11),
  visitid char(14),
  quantity int(3) null,
  daySupply int(3) null,
  primary key (beneID, prescriptionid, visitid),
  constraint IsPrescribed_visitid_FK
        foreign key (visitid) references Visits(visitid)
              on update cascade on delete cascade,
  constraint IsPrescribed_prescriptionid_FK
        foreign key (prescriptionid) references Prescription(prescriptionid)
              on update cascade on delete cascade,
  constraint IsPrescribed_beneID_FK
        foreign key (beneID) references Patient(beneID)
              on update cascade on delete cascade
```

```
);
 create table hasCoverage (
  beneID char(16),
  coverageID char(6),
  primary key (beneID, coverageID),
  constraint hasCoverage_beneID_FK
          foreign key (beneID) references Patient(beneID)
                  on update cascade on delete cascade,
  constraint hasCoverage_coverageID_FK
          foreign key (coverageID) references Coverage(coverageID)
                  on update cascade on delete cascade
  );
  create table livesIn (
    beneID char(16),
    countyCode int(3),
    primary key (beneID, countyCode)
    constraint locatedAt_beneID_FK
    foreign key (beneID) references Patient(beneID)
                  on update cascade on delete cascade,
    constraint locatedAt_countyCode_FK
    foreign key (countyCode) references Location(countyCode)
                  on update cascade on delete cascade
    );
 create table hasMedicalHistory (
  beneID char(16),
  medicalRecordNumber int(7),
  primary key (beneID, medicalRecordNumber)
    constraint hasCondition_beneID_FK
    foreign key (beneID) references Patient(beneID)
                  on update cascade on delete cascade,
  constraint hasCondition_MRN_FK
    foreign key (medicalRecordNumber) references HealthRecord(medicalRecordNumber)
                  on update cascade on delete cascade
  );
```

8. Example Queries

Example 1

From a public health perspective, an epidemiologist is accessing the Medicare database and is interested in understanding the number of female Hispanic patients there are in the system. This is because the group is often identified as being at higher risk for certain health conditions, such as diabetes and hypertension.

SQL Query:

```
SELECT COUNT(*)
FROM Patient
WHERE Sex = 'Female' AND Race = 'Hispanic';
```

Tuple Relational Calculus:

{ t | t.count = COUNT(P) AND ∃ P (P ∈ Patient ∧ P.Sex = 'Female' ∧ P.Race = 'Hispanic') }

**Please be advised that TRC is unable to demonstrate aggregate functions due to a known limitation. The following is a sample query illustrating how it would function if this capability were available.

Relational Algebra:

$G_{count(*)}(\sigma_{Sex = Female \wedge Race = "Hispanic"}(Patient))$

Result: 622

Example 2

From a business perspective, a pharmacy is accessing the Medicare database to evaluate the potential profitability of opening a new branch in California. By querying the number of patients who have purchased prescriptions with a product cost over $100, the pharmacy can gauge the demand for higher-cost medications in the area. This information helps in assessing whether there is sufficient market potential to justify the investment in a new location.

SQL Query:

```
SELECT COUNT(DISTINCT Patient.BeneID) AS NumberOfPatients
FROM Patient, Location, IsPrescribed, Prescription
WHERE Patient.BeneID = IsPrescribed.BeneID
AND IsPrescribed.PrescriptionID = Prescription.PrescriptionID AND Patient.BeneID IN
        (SELECT BeneID FROM LivesIn WHERE LivesIn.CountyCode =
Location.CountyCode AND Location.State = 'CA')
                AND Prescription.productCost > 100;
```

Tuple Relational Calculus:

{t | (∃ P, L, IP, PR, LI) (P ∈ Patient ∧ L ∈ Location ∧ IP ∈ IsPrescribed ∧ PR ∈ Prescription ∧ LI ∈ LivesIn ∧ P.BeneID = IP.BeneID ∧ IP.PrescriptionID = PR.PrescriptionID ∧ P.BeneID = LI.BeneID ∧ LI.CountyCode = L.CountyCode ∧ L.State = 'CA' ∧ PR.productCost > 100 ∧ t[BeneID] = P.BeneID) ∧ t.count = COUNT(DISTINCT P.BeneID) }

**Please be advised that TRC is unable to demonstrate aggregate functions due to a known limitation. The following is a sample query illustrating how it would function if this capability were available.

Relational Algebra:

$G_{count(Patient.BeneID)}(\sigma_{productCost > 100}(Prescription) \bowtie (IsPrescribed) \ldots$
$\bowtie (LivesIn \bowtie (\sigma_{State = "CA"}(Location)) \bowtie (Patient))$

Result: 187

Example 3

From an insurance provider's perspective, the goal is to identify patients who have no recorded medical conditions and determine the average number of months of coverage they possess. By analyzing the coverage data for patients without conditions such as heart failure, diabetes, or cancer, the provider can better understand the risk profile and potential financial exposure associated with insuring individuals who are currently in good health. This information can assist in developing targeted insurance plans and setting appropriate premiums.

SQL Query:

```
SELECT AVG(Coverage.CoverageMonths) AS AverageCoverageMonths
FROM Patient, HasMedicalHistory, MedicalHistory, HasCoverage, Coverage
WHERE Patient.BeneID = HasMedicalHistory.BeneID
AND HasMedicalHistory.MedicalRecordNumber = MedicalHistory.MedicalRecordNumber
AND Patient.BeneID = HasCoverage.BeneID
AND HasCoverage.CoverageID = Coverage.CoverageID
AND MedicalHistory.HeartFailure = 'No'
AND MedicalHistory.KidneyDisease = 'No'
AND MedicalHistory.Cancer = 'No'
AND MedicalHistory.OPD = 'No'
AND MedicalHistory.Depression = 'No'
AND MedicalHistory.Diabetes = 'No'
AND MedicalHistory.HeartDisease = 'No'
AND MedicalHistory.Osteoporosis = 'No'
AND MedicalHistory.Arthritis = 'No'
AND MedicalHistory.Stroke = 'No';
```

Tuple Relational Calculus:

$$\{ t \mid (\exists P, HM, MH, HC, C) (P \in Patient \wedge HM \in HasMedicalHistory \wedge MH \in MedicalHistory \wedge HC \in HasCoverage \wedge C \in Coverage \wedge P.BeneID = HM.BeneID \wedge HM.MedicalRecordNumber = MH.MedicalRecordNumber \wedge P.BeneID = HC.BeneID \wedge HC.CoverageID = C.CoverageID \wedge MH.HeartFailure = 'No' \wedge MH.KidneyDisease = 'No' \wedge MH.Cancer = 'No' \wedge MH.OPD = 'No' \wedge MH.Depression = 'No' \wedge MH.Diabetes = 'No' \wedge MH.HeartDisease = 'No' \wedge MH.Osteoporosis = 'No' \wedge MH.Arthritis = 'No' \wedge MH.Stroke = 'No' \wedge t[CoverageMonths] = C.CoverageMonths) \wedge t.avgCoverageMonths = AVG(C.CoverageMonths) \}$$

**Please be advised that TRC is unable to demonstrate aggregate functions due to a known limitation. The following is a sample query illustrating how it would function if this capability were available.

Relational Algebra:

$G_{AVG(Coverage.CoverageMonths)}(\sigma_{HeatFailure = "No" \wedge KidneyDisease = "No" \wedge Cancer = "No" \wedge OPD = "No" \wedge Depression = "No" \wedge ... Diabetes = "No" \wedge HeartDisease = "No" \wedge Osteoporosis = "No" \wedge Arthritis = "No" \wedge Stroke = "No"}(HasMedicalHistory) \bowtie HasMedicalHistory \bowtie Patient \bowtie HasCoverage \bowtie Coverage)$

Result: 2.5679

Example 4

In a specific county the Medicare insurers are starting a subsidized health care plan. We need a query to check who fits into specific policy criteria for a subsidized healthcare plan. These beneficiaries must have exactly 12 months of continuous coverage, no high-cost claims (over $500), and no high-cost prescriptions (over $150). Patients who meet these criteria and have a low-claim and low-cost prescription requirements receive the benefits of the subsidized plan.

```
SELECT p.BeneID
FROM Patient p, LivesIn l, HasCoverage hc, Coverage c
WHERE p.BeneID = l.BeneID
AND p.BeneID = hc.BeneID
AND hc.CoverageID = c.CoverageID
AND l.CountyCode = '4710'
AND c.CoverageMonths = 12
AND NOT EXISTS (
   SELECT 1
   FROM MakesClaim mc, Claims cl
   WHERE mc.BeneID = p.BeneID
   AND mc.ClaimID = cl.ClaimID
   AND cl.PaymentAmount > 500
)
AND NOT EXISTS (
   SELECT 1
   FROM IsPrescribed ip, Prescription pr
   WHERE ip.BeneID = p.BeneID
   AND ip.PrescriptionID = pr.PrescriptionID
   AND pr.productCost > 150
);
```

Tuple Relational Calculus:

{ t | (∃ P, L, HC, C) (P ∈ Patient ∧ L ∈ LivesIn ∧ HC ∈ HasCoverage ∧ C ∈ Coverage ∧ P.BeneID = L.BeneID ∧ P.BeneID = HC.BeneID ∧ HC.CoverageID = C.CoverageID ∧ L.CountyCode = '4710' ∧ C.CoverageMonths = 12

$\wedge \neg(\exists$MC, CL) (MC $\in$ MakesClaim $\wedge$ CL $\in$ Claim $\wedge$ MC.BeneID = P.BeneID $\wedge$ MC.ClaimID = CL.ClaimID $\wedge$ CL.PaymentAmount > 500)
$\wedge \neg(\exists$IP, PR) (IP $\in$ IsPrescribed $\wedge$ PR $\in$ Prescription $\wedge$ IP.BeneID = P.BeneID $\wedge$ IP.PrescriptionID = PR.PrescriptionID $\wedge$ PR.productCost > 150) $\wedge$ t[BeneID] = P.BeneID) }

Relational Algebra:

$$PatientList \leftarrow (Patient \bowtie (\sigma_{CountyCode = "4170"} LivesIn) \bowtie HasCoverage \bowtie (\sigma_{CoverageMonths=12} Coverage)$$

$$HighCostClaims \leftarrow PatientList \bowtie MakesClaim \bowtie (\sigma_{PaymentAmount > 500} Claims)$$

$$HighCostRx \leftarrow PatientList \bowtie HasPrescription \bowtie (\sigma_{productCost > 150} Prescription)$$

$$\Pi_{PatientList.BeneID}(PatientList - HighCostClaims - HighCostRx)$$

Result:

beneID
000D6D88463D8A76
001E5211DED6A6A2
00247AD32AE68EFE
002A425E967ED186
0030C3E959BE6A67
0031E4B9F2F11B24
00364D31615716BE

9. Technologies Used

Task	**Technology Used**
OS	Windows 11 Pro 64-bit operating system, x64-based processor
Database Cleaning	Python 3.11.5
Database Creation	SQLite(3) & SQL Online IDE
Command Line Interface	Python 3.11.5, VS Code

Libraries used: Pandas, Sqlite3, NumPY

10. Issues Encountered

In order to complete the command line interface task, the code from the SQL Online IDE had to be revised to match the requirements for SQLite(3) much of the syntax did not match hence, we had to re-write the syntax according to the type of SQL we use. Additionally, the online compiler did not correctly identify the data types in the CSV file (i.e. categorizing SSN as a text variable instead of an int), which required additional revision to the code. Much of the data also had to be re-cleaned and adjusted according to the constraints.

11. Conclusions & Learnings

In conclusion, this project aimed to create an efficient, accessible database to store Medicare claims data. By restructuring and merging the Medicare Beneficiary, Claims, and Prescription data, we developed a relational database that simplifies the extraction and analysis of complex Medicare data. In the future, this database could be expanded to include additional attributes and information that is stored in Medicare data. Overall, our project applied database management techniques to Medicare data in order to make the information more accessible and comprehensible.

Camaryn Learning

I learned how to use SQL to store data efficiently and apply normalization theory to real-world scenarios. I took complex insurance data and successfully represented it using an ER diagram, a schema in SQL, and translated it into Relational Algebra (RA). This experience deepened my understanding of database design and enhanced my ability to handle intricate data structures.It also strengthened my understanding of database from end to end.

Mauli Learning

Through this project, I learned to identify logical applications of the International Medical Device Regulators Forum (IMDRF) for various vendors and developed sample SQL queries tailored to their various different uses. I also then converted those queries to Tuple Relational Calculus (TRC) and understood the limitations of TRC. I connected the database to the command line, gaining a deep understanding of its functionality and building various command-line functions that various different users would use to access the database. Additionally, I prepared data for database input, exploring different methods of importing data to ensure accuracy and also understanding the proper structure of the data. I also tested the functionality of constraints and triggers, applying them in practice to maintain data integrity and streamline database operations. Through this process, I improved my problem-solving skills and learned to optimize database performance and how to build a database from scratch.

11. Author Contribution

Mauli Patel

- Background research
- Cleaning, Merging data & creating Synthetic data
- Building and updating ER Diagram
- Building Initial Schema
- Developing Constraints
- Writing SQL Queries
- Writing Tuple Relational Calculus Queries
- Converting/Building DB file and python file for extract
- Writing command line interface

Camaryn Petersen

- Background research & data acquisition
- Constructing SQL tables
- Importing data into SQL tables
- Developing SQL Triggers
- Determining functional dependencies for each schema
- Normalization & decomposition
- Revising & updating schema
- Writing Relational Algebra Query

Works Cited

Centers for Medicare & Medicaid Services, United States Government, https://www.cms.gov/data-research/statistics-trends-and-reports/medicare-claims-synthetic-public-use-files/cms-2008-2010-data-entrepreneurs-synthetic-public-use-file-de-synpuf

CMS 2008-2010 Data Entrepreneurs' Synthetic Public Use File (DE-SynPUF), Centers Centers for Medicare & Medicaid Services. https://www.cms.gov/data-research/statistics-trends-and-reports/medicare-claims-synthetic-public-use-files/cms-2008-2010-data-entrepreneurs-synthetic-public-use-file-de-synpuf

Medicare, United States Government, www.medicare.gov/what-medicare-covers/your-medicare-coverage-choices/whats-medicare.

Medicare, United States Government, https://www.medicare.gov/what-medicare-covers/what-part-a-covers/inpatient-or-outpatient-hospital-status

Medicare Enrollment Numbers, Centers for Medicare and Medicaid Services, https://medicareadvocacy.org/medicare-enrollment-numbers/.