Observational Medical Outcomes Partnership Common Data Model(OMOP-CDM) for EHR Data

Yina Hou
Department of Computer Science
Tennessee State University
Nashville, TN, United States
yhou@tnstate.edu

Ibna Kowsar
Department of Computer Science
Tennessee State University
Nashville, TN, United States
ikowsar@tnstate.edu

I. Introduction

In the realm of modern healthcare, personalized medicine, and data-driven insights have become pivotal in advancing patient care and treatment strategies. The "All of Us" Research Program is a landmark initiative spearheaded by the National Institutes of Health (NIH) [1]. It revolutionizes medical research by gathering data from over one million people with various backgrounds. The program seeks to improve the understanding of how individual differences in lifestyle, environment, and genetics can influence health and disease.

A diverse and extensive database like that can be useful for knowing the risk factors for certain diseases. Researchers can investigate treatments that work best for people of different backgrounds and professionals can retrieve specific subsets of data from the database. All of Us uses the Observational Medical Outcomes Partnership (OMOP) Common Data Model (CDM) [2], which standardizes data into a consistent format, facilitating easier data analysis and sharing across different systems. This program maintains strict security protocols, including data encryption, secure access controls, and regular audits to ensure data protection. Moreover, using data from All of Us and reporting data outside the workbench is not allowed.

To explore relational database management systems (RDBMS) and demonstrate meaningful relationships in synthetic electronic health record (EHR) data, we developed a database following the OMOP-CDM guidelines [3], inspired by the All of Us workbench. It is worth mentioning that we have generated all the relational tables shown in this report inside the All of Us workbench as well. In this report, we use synthetic EHR data, which mimics real patient data while ensuring privacy and confidentiality, to demonstrate our database's capabilities in managing and analyzing health information [4], [5].

While our database effectively models complex data relationships and supports diverse analytical tasks, it is important to note that synthetic data might not capture all the nuances of real-world data. Additionally, our work strictly adheres to ethical standards and regulatory requirements, ensuring that data use and reporting are conducted responsibly.

II. Methodology

This section illustrates the theoretical concepts like schema, entity relationships, data definition language, and data manipulation language that are used to implement our RDBMS.

A. Database schema

A database schema is a blueprint that defines the structure, organization and relationships. It also outlines the tables, columns, data types and relationships between tables. It provides a framework for organizing data and an overview of the database structure.

B. Entity Relationship (E-R) diagram

An ER diagram is a representation that illustrates the entities, attributes and relationships within a database schema. It helps visualizing the database structure and relationships, making it more easier to understand the design of the database.

C. Data Definition Language(DDL) in MySql

DDL is a set of commands used to define the structure of a database schema. It is essential for implementing the database schema within database management system. With DDL, we can create tables, specify data types, define constraints and establish integrity rules for maintaining data consistency. In MySQL, commands like 'CREATE TABLE' are used to modify the structure of tables in database, including column names, data types and constraints. 'PRIMARY KEY','FOREIGN KEY' and 'NOT NULL' ate used to enforce integrity constraints, ensuring that data remains accurate and consistent.

D. Data Manipulation Language(DML) in MySql

DML is used to manipulate data within the database. It consists of commands for querying, inserting, updating, and deleting data stored in tables. We can perform various manipulation with it. In MySQL, DML allows us to manipulate data within tables. 'SELECT' help us to retrieve data from one or more tables. We use 'INSERT' to add new rows into tables. It allows users to specify the columns to retrieve, apply filtering conditions using 'WHERE' clause, sort the result set, and perform various aggregate functions such as 'SUM', 'COUNT', 'AVG', etc.

III. Results

A. Schema

The schema diagram as shown in Figure 1 illustrates the structure of the medical database we created. Each table represents an aspect of the healthcare domain. Within each table, columns provide more detailed information on various domains.

B. Entity-Relationship(E-R) diagram

ER diagram as shown in Figure 2 visualizes the relationship between tables that are shown in the schema. First, we have person table which provides the demographic information of patients. Then combined with the person table, we have other seven tables that show information of various medical aspect.
Lastly, an additional concept table is implemented to translate concept_id into names that are more readily comprehensible in real-world contexts. So the relationship between the person table and the concept table is many-to-many. For person with other seven medical tables and visit_occurrence with the other six tables, their relationships are one-to-many.

C. Data Definition Language(DDL)

In this section, we implement our database tables following our schema and entity relationships. Figure 3, 4 shows the DDL for all the tables that are defined in our database schema.

D. Key constraints

Once we have the DDL ready for all the tables we update each table's key constraints using SQL command ALTER TABLE. Here, our main focus is to initialize the primary keys for each table. Other constraints like NOT NULL, datatype, etc, are defined during the DDL implementation. Listing 1 shows the SQL commands that are used to update the key constraints.

Listing 1: SQL Constraints for Database Schema

```sql
ALTER TABLE PERSON ADD CONSTRAINT xpk_PERSON PRIMARY KEY (person_id);
ALTER TABLE VISIT_OCCURRENCE ADD CONSTRAINT xpk_VISIT_OCCURRENCE PRIMARY KEY (
    visit_occurrence_id);
ALTER TABLE CONDITION_OCCURRENCE ADD CONSTRAINT xpk_CONDITION_OCCURRENCE PRIMARY
     KEY (condition_occurrence_id);
ALTER TABLE DRUG_EXPOSURE ADD CONSTRAINT xpk_DRUG_EXPOSURE PRIMARY KEY (
    drug_exposure_id);
ALTER TABLE PROCEDURE_OCCURRENCE ADD CONSTRAINT xpk_PROCEDURE_OCCURRENCE PRIMARY
     KEY (procedure_occurrence_id);
ALTER TABLE DEVICE_EXPOSURE ADD CONSTRAINT xpk_DEVICE_EXPOSURE PRIMARY KEY (
    device_exposure_id);
ALTER TABLE MEASUREMENT ADD CONSTRAINT xpk_MEASUREMENT PRIMARY KEY (
    measurement_id);
ALTER TABLE OBSERVATION ADD CONSTRAINT xpk_OBSERVATION PRIMARY KEY (
    observation_id);
ALTER TABLE CONCEPT ADD CONSTRAINT xpk_CONCEPT PRIMARY KEY (concept_id);
```

measurement						
measurement_id	person_id	measurement_concept_id	measurement_date	measurement_datetime	measurement_time	measurement_type_concept_id
provider_id	unit_concept_id	measurement_source_value	range_high	operator_concept_id	visit_occurrence_id	visit_detail_id
range_low	value_source_value	measurement_source_concept_id	unit_source_value	value_as_concept_id	value_as_number	

visit_occurrence					
visit_occurrence_id	person_id	visit_concept_id	visit_start_date	preceding_visit_occurrence_id	visit_end_date
care_site_id	provider_id	visit_source_concept_id	admitted_from_concept_id	admitted_from_source_value	discharged_to_concept_id
visit_source_value	visit_start_datetime	visit_type_concept_id	discharged_to_source_value	visit_end_datetime	

observation						
observation_id	person_id	observation_concept_id	observation_date	observation_datetime	observation_type_concept_id	value_as_number
unit_concept_id	provider_id	visit_occurrence_id	visit_detail_id	observation_source_value	observation_source_concept_id	unit_source_value
value_as_concept_id	qualifier_concept_id	value_as_string	qualifier_source_value	value_source_value		

condition_occurrence					
condition_occurrence_id	person_id	condition_concept_id	condition_start_date	condition_start_datetime	condition_end_date
condition_status_concept_id	stop_reason	provider_id	visit_occurrence_id	visit_detail_id	condition_source_value
condition_source_concept_id	condition_status_source_value	condition_end_datetime	condition_type_concept_id		

drug_exposure					
drug_exposure_id	drug_exposure_start_date	drug_concept_id	drug_source_concept_id	sig	drug_exposure_end_date
provider_id	visit_occurrence_id	visit_detail_id	drug_source_value	refills	route_source_value
verbatim_end_date	drug_type_concept_id	stop_reason	dose_unit_source_value	quantity	days_supply
route_concept_id	drug_exposure_start_datetime	lot_number	drug_exposure_end_datetime	person_id	

procedure_occurrence						
procedure_occurrence_id	person_id	procedure_concept_id	procedure_date	procedure_datetime	quantity	visit_occurrence_id
procedure_source_value	procedure_source_concept_id	modifier_source_value	procedure_type_concept_id	modifier_concept_id	provider_id	visit_detail_id

person						
race_source_value	gender_concept_id	race_source_concept_id	month_of_birth	day_of_birth	ethnicity_concept_id	race_concept_id
person_source_value	gender_source_value	gender_source_concept_id	birth_datetime	year_of_birth	ethnicity_source_value	ethnicity_source_concept_id
person_id	location_id	provider_id	care_site_id			

device_exposure					
device_exposure_id	person_id	device_concept_id	device_exposure_start_date	device_exposure_start_datetime	device_exposure_end_datetime
unique_device_id	quantity	visit_detail_id	device_type_concept_id	device_exposure_end_date	device_source_concept_id
device_source_value	provider_id	visit_occurrence_id			

concept									
concept_id	concept_name	domain_id	vocabulary_id	concept_class_id	standard_concept	concept_code	valid_start_date	valid_end_date	invalid_reason

Figure 1: Schema of the Relational Database

E. Populate the database tables using Data Manipulation Language(DML)

We load data from CSV documents for all nine tables we have into MySQL using a Python script in SQLAlchemy.

Listing 2: SQLAlchemy code for populating observation table

```
import pandas as pd
csv_file_path = 'cohort/observation.csv'
data = pd.read_csv(csv_file_path)

table_name = 'observation'

# Load the DataFrame into MySQL
data.to_sql(name=table_name, con=engine, if_exists='append', index=False)

print("Data has been successfully loaded into the database.")

query = text(f"SELECT * FROM {table_name}")
result = pd.read_sql(query, con=engine)
print("Data retrieved from the database:")
result
```

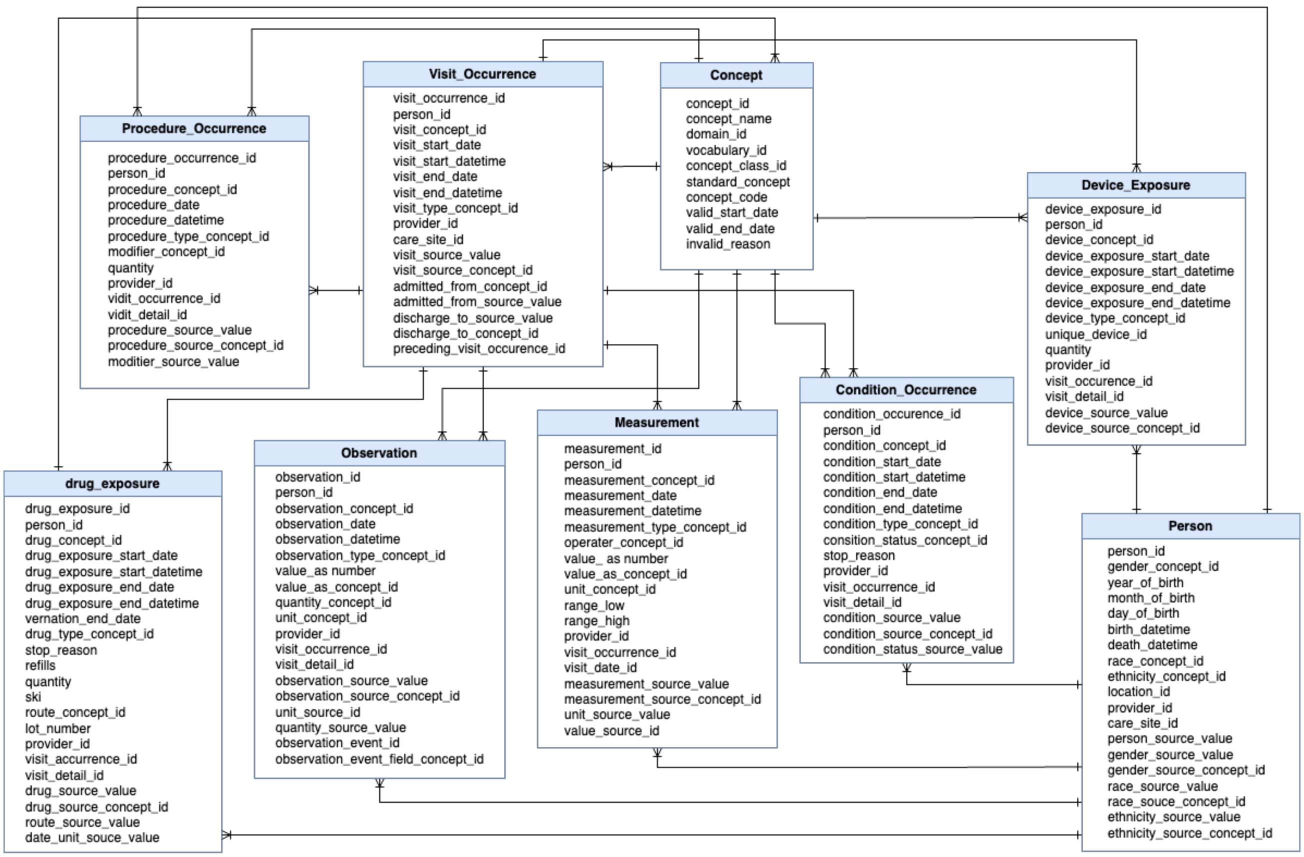

Figure 2: Entity-relationship diagram

Listing 3: SQLAlchemy code for populating Person

```
import pandas as pd
csv_file_path = 'cohort/person.csv'
data = pd.read_csv(csv_file_path)

table_name = 'person'
# Load the DataFrame into MySQL
data.to_sql(name=table_name, con=engine, if_exists='append', index=False)

print("Data has been successfully loaded into the database.")

query = text(f"SELECT * FROM {table_name}")
result = pd.read_sql(query, con=engine)
print("Data retrieved from the database:")
result
```

Listing 4: SQLAlchemy code for populating Condition Occurrence

```
import pandas as pd
csv_file_path = 'cohort/condition_occurrence.csv'  # Update this to the path of
    your CSV file
data = pd.read_csv(csv_file_path)

table_name = 'condition_occurrence'
# schema = 'cdmDatabaseSchema'

# Load the DataFrame into MySQL
data.to_sql(name=table_name, con=engine, if_exists='append', index=False)

print("Data has been successfully loaded into the database.")
```

```sql
CREATE TABLE @cdmDatabaseSchema.DRUG_EXPOSURE (
            drug_exposure_id integer NOT NULL,
            person_id integer NOT NULL,
            drug_concept_id integer NOT NULL,
            drug_exposure_start_date date NOT NULL,
            drug_exposure_start_datetime datetime NULL,
            drug_exposure_end_date date NOT NULL,
            drug_exposure_end_datetime datetime NULL,
            verbatim_end_date date NULL,
            drug_type_concept_id integer NOT NULL,
            stop_reason varchar(20) NULL,
            refills integer NULL,
            quantity float NULL,
            days_supply integer NULL,
            sig varchar(255) NULL,
            route_concept_id integer NULL,
            lot_number varchar(50) NULL,
            provider_id integer NULL,
            visit_occurrence_id integer NULL,
            visit_detail_id integer NULL,
            drug_source_value varchar(50) NULL,
            drug_source_concept_id integer NULL,
            route_source_value varchar(50) NULL,
            dose_unit_source_value varchar(50) NULL );
```

((a)) Drug Exposure Details

```sql
CREATE TABLE @cdmDatabaseSchema.MEASUREMENT (
            measurement_id integer NOT NULL,
            person_id integer NOT NULL,
            measurement_concept_id integer NOT NULL,
            measurement_date date NOT NULL,
            measurement_datetime datetime NULL,
            measurement_time varchar(10) NULL,
            measurement_type_concept_id integer NOT NULL,
            operator_concept_id integer NULL,
            value_as_number float NULL,
            value_as_concept_id integer NULL,
            unit_concept_id integer NULL,
            range_low float NULL,
            range_high float NULL,
            provider_id integer NULL,
            visit_occurrence_id integer NULL,
            visit_detail_id integer NULL,
            measurement_source_value varchar(50) NULL,
            measurement_source_concept_id integer NULL,
            unit_source_value varchar(50) NULL,
            unit_source_concept_id integer NULL,
            value_source_value varchar(50) NULL,
            measurement_event_id bigint NULL,
            meas_event_field_concept_id integer NULL );
```

((b)) Measurement Overview

```sql
CREATE TABLE @cdmDatabaseSchema.PROCEDURE_OCCURRENCE (
            procedure_occurrence_id integer NOT NULL,
            person_id integer NOT NULL,
            procedure_concept_id integer NOT NULL,
            procedure_date date NOT NULL,
            procedure_datetime datetime NULL,
            procedure_end_date date NULL,
            procedure_end_datetime datetime NULL,
            procedure_type_concept_id integer NOT NULL,
            modifier_concept_id integer NULL,
            quantity integer NULL,
            provider_id integer NULL,
            visit_occurrence_id integer NULL,
            visit_detail_id integer NULL,
            procedure_source_value varchar(50) NULL,
            procedure_source_concept_id integer NULL,
            modifier_source_value varchar(50) NULL );
```

((c)) Procedure Occurrences

```sql
CREATE TABLE @cdmDatabaseSchema.CONDITION_OCCURRENCE (
            condition_occurrence_id integer NOT NULL,
            person_id integer NOT NULL,
            condition_concept_id integer NOT NULL,
            condition_start_date date NOT NULL,
            condition_start_datetime datetime NULL,
            condition_end_date date NULL,
            condition_end_datetime datetime NULL,
            condition_type_concept_id integer NOT NULL,
            condition_status_concept_id integer NULL,
            stop_reason varchar(20) NULL,
            provider_id integer NULL,
            visit_occurrence_id integer NULL,
            visit_detail_id integer NULL,
            condition_source_value varchar(50) NULL,
            condition_source_concept_id integer NULL,
            condition_status_source_value varchar(50) NULL );
```

((d)) Condition Occurrences

```sql
--HINT DISTRIBUTE ON KEY (person_id)
CREATE TABLE @cdmDatabaseSchema.VISIT_OCCURRENCE (
            visit_occurrence_id integer NOT NULL,
            person_id integer NOT NULL,
            visit_concept_id integer NOT NULL,
            visit_start_date date NOT NULL,
            visit_start_datetime datetime NULL,
            visit_end_date date NOT NULL,
            visit_end_datetime datetime NULL,
            visit_type_concept_id Integer NOT NULL,
            provider_id integer NULL,
            care_site_id integer NULL,
            visit_source_value varchar(50) NULL,
            visit_source_concept_id integer NULL,
            admitted_from_concept_id integer NULL,
            admitted_from_source_value varchar(50) NULL,
            discharged_to_concept_id integer NULL,
            discharged_to_source_value varchar(50) NULL,
            preceding_visit_occurrence_id integer NULL );
```

((e)) Visit Occurrences

```sql
CREATE TABLE @cdmDatabaseSchema.DEVICE_EXPOSURE (
            device_exposure_id integer NOT NULL,
            person_id integer NOT NULL,
            device_concept_id integer NOT NULL,
            device_exposure_start_date date NOT NULL,
            device_exposure_start_datetime datetime NULL,
            device_exposure_end_date date NULL,
            device_exposure_end_datetime datetime NULL,
            device_type_concept_id integer NOT NULL,
            unique_device_id varchar(255) NULL,
            production_id varchar(255) NULL,
            quantity integer NULL,
            provider_id integer NULL,
            visit_occurrence_id integer NULL,
            visit_detail_id integer NULL,
            device_source_value varchar(50) NULL,
            device_source_concept_id integer NULL,
            unit_concept_id integer NULL,
            unit_source_value varchar(50) NULL,
            unit_source_concept_id integer NULL );
```

((f)) Device Exposure

Figure 3: Overview of Medical Data Visualization

```sql
CREATE TABLE @cdmDatabaseSchema.PERSON (
			person_id integer NOT NULL,
			gender_concept_id integer NOT NULL,
			year_of_birth integer NOT NULL,
			month_of_birth integer NULL,
			day_of_birth integer NULL,
			birth_datetime datetime NULL,
			race_concept_id integer NOT NULL,
			ethnicity_concept_id integer NOT NULL,
			location_id integer NULL,
			provider_id integer NULL,
			care_site_id integer NULL,
			person_source_value varchar(50) NULL,
			gender_source_value varchar(50) NULL,
			gender_source_concept_id integer NULL,
			race_source_value varchar(50) NULL,
			race_source_concept_id integer NULL,
			ethnicity_source_value varchar(50) NULL,
			ethnicity_source_concept_id integer NULL );
```

((a)) Person

```sql
CREATE TABLE @cdmDatabaseSchema.OBSERVATION (
			observation_id integer NOT NULL,
			person_id integer NOT NULL,
			observation_concept_id integer NOT NULL,
			observation_date date NOT NULL,
			observation_datetime datetime NULL,
			observation_type_concept_id integer NOT NULL,
			value_as_number float NULL,
			value_as_string varchar(60) NULL,
			value_as_concept_id Integer NULL,
			qualifier_concept_id integer NULL,
			unit_concept_id integer NULL,
			provider_id integer NULL,
			visit_occurrence_id integer NULL,
			visit_detail_id integer NULL,
			observation_source_value varchar(50) NULL,
			observation_source_concept_id integer NULL,
			unit_source_value varchar(50) NULL,
			qualifier_source_value varchar(50) NULL,
			value_source_value varchar(50) NULL,
			observation_event_id bigint NULL,
			obs_event_field_concept_id integer NULL );
```

((b)) Observation

```sql
CREATE TABLE @cdmDatabaseSchema.CONCEPT (
			concept_id integer NOT NULL,
			concept_name varchar(255) NOT NULL,
			domain_id varchar(20) NOT NULL,
			vocabulary_id varchar(20) NOT NULL,
			concept_class_id varchar(20) NOT NULL,
			standard_concept varchar(1) NULL,
			concept_code varchar(50) NOT NULL,
			valid_start_date date NOT NULL,
			valid_end_date date NOT NULL,
			invalid_reason varchar(1) NULL );
```

((c)) Concept

Figure 4: Overview of Medical Data Visualization

```python
query = text(f"SELECT * FROM {table_name}")
result = pd.read_sql(query, con=engine)
print("Data retrieved from the database:")
result
```

Listing 5: SQLAlchemy code for populating Visit Occurrence

```python
import pandas as pd
csv_file_path = 'cohort/visit_occurrence.csv'
data = pd.read_csv(csv_file_path)

table_name = 'visit_occurrence'
# schema = 'cdmDatabaseSchema'
```

	observation_id	person_id	observation_concept_id	observation_date	observation_datetime	observation_type_concept_id	value_as_number
0	1002510856	1372782	294002	2020-03-13	2020-03-13 21:32:00	7460767	None
1	1002704901	1372782	2770454	2020-03-13	2020-03-13 21:32:00	7460767	None
2	1022704905	1372782	2759786	2020-03-13	2020-03-13 21:32:00	7460767	None
3	1042511804	1372782	31002266	2020-03-13	2020-03-13 21:32:00	7460767	None
4	1042693544	1372782	2841127	2020-03-13	2020-03-13 21:32:00	7460767	None
5	1062511800	1372782	31003260	2020-03-13	2020-03-13 21:32:00	7460767	None
6	1202664983	1372782	2693100	2020-03-13	2020-03-13 21:32:00	7460767	None
7	1562510832	1372782	2791757	2020-03-13	2020-03-13 21:32:00	7460767	None
8	1603093411	1372782	2823743	2020-03-13	2020-03-13 21:32:00	7460767	None
9	1672635332	1372782	2719982	2020-03-13	2020-03-13 21:32:00	7460767	None

10 rows × 21 columns

Figure 5: Observation Details

```
# Load the DataFrame into MySQL
data.to_sql(name=table_name, con=engine, if_exists='append', index=False)

print("Data has been successfully loaded into the database.")

query = text(f"SELECT * FROM {table_name}")
result = pd.read_sql(query, con=engine)
print("Data retrieved from the database:")
result
```

Listing 6: SQLAlchemy code for populating Drug Exposure

```
import pandas as pd
csv_file_path = 'cohort/drug_exposure.csv'
data = pd.read_csv(csv_file_path)

table_name = 'drug_exposure'
# schema = 'cdmDatabaseSchema'

# Load the DataFrame into MySQL
data.to_sql(name=table_name, con=engine, if_exists='append', index=False)

print("Data has been successfully loaded into the database.")

query = text(f"SELECT * FROM {table_name}")
result = pd.read_sql(query, con=engine)
print("Data retrieved from the database:")
result
```

Listing 7: SQLAlchemy code for populating Device Exposure

```
import pandas as pd
csv_file_path = 'cohort/device_exposure.csv'
data = pd.read_csv(csv_file_path)

table_name = 'device_exposure'
# schema = 'cdmDatabaseSchema'

# Load the DataFrame into MySQL
data.to_sql(name=table_name, con=engine, if_exists='append', index=False)

print("Data has been successfully loaded into the database.")

query = text(f"SELECT * FROM {table_name}")
result = pd.read_sql(query, con=engine)
print("Data retrieved from the database:")
result
```

	person_id	gender_concept_id	year_of_birth	month_of_birth	day_of_birth	birth_datetime	race_concept_id	e
0	680331	41310599	1946	2	27	1946-02-27	66754	
1	711599	1995432138	1978	4	6	1978-04-06	817420	
2	872883	41312805	1989	11	20	1989-11-20	66754	
3	1021059	41310599	1969	1	20	1969-01-20	66754	
4	1372782	1995432138	1976	6	1	1976-06-01	1099923437	
5	1543037	1995432138	1968	9	3	1968-09-03	817420	
6	1851447	41310599	1964	12	10	1964-12-10	1099923437	
7	1955206	41312805	1959	8	5	1959-08-05	1099923437	
8	2146445	41312805	1951	5	15	1951-05-15	66754	
9	2334201	1995432138	1965	7	16	1965-07-16	817420	
10	5675534	41312805	1977	3	19	1977-03-19	66754	

Figure 6: Person

Listing 8: SQLAlchemy code for populating Concept

```
import pandas as pd
csv_file_path = 'cohort/concept.csv'
data = pd.read_csv(csv_file_path)

table_name = 'concept'
# schema = 'cdmDatabaseSchema'

# Load the DataFrame into MySQL
data.to_sql(name=table_name, con=engine, if_exists='append', index=False)

print("Data has been successfully loaded into the database.")

query = text(f"SELECT * FROM {table_name}")
result = pd.read_sql(query, con=engine)
print("Data retrieved from the database:")
result
```

Listing 9: SQLAlchemy code for populating Measurement

```
import pandas as pd
csv_file_path = 'cohort/measurement.csv'
data = pd.read_csv(csv_file_path)

table_name = 'measurement'
# schema = 'cdmDatabaseSchema'

# Load the DataFrame into MySQL
data.to_sql(name=table_name, con=engine, if_exists='append', index=False)

print("Data has been successfully loaded into the database.")

query = text(f"SELECT * FROM {table_name}")
result = pd.read_sql(query, con=engine)
print("Data retrieved from the database:")
result
```

Listing 10: SQLAlchemy code for populating Procedure Occurrence

```
import pandas as pd
csv_file_path = 'cohort/procedure_occurrence.csv'
data = pd.read_csv(csv_file_path)

table_name = 'procedure_occurrence'
# schema = 'cdmDatabaseSchema'

# Load the DataFrame into MySQL
```

	condition_occurrence_id	person_id	condition_concept_id	condition_start_date	condition_start_datetime	condition_end_date
0	66677	5675534	319835	2018-08-04	2018-08-04 10:43:00	None
1	238140	1372782	4043371	2020-03-13	2020-03-13 21:32:00	None
2	248088	1372782	196152	2020-03-13	2020-03-13 21:32:00	None
3	276646	1372782	37110250	2020-03-13	2020-03-13 21:32:00	None
4	276647	1372782	317577	2020-03-13	2020-03-13 21:32:00	None
5	289289	1372782	197596	2020-03-13	2020-03-13 21:32:00	None
6	289385	1372782	319835	2020-03-13	2020-03-13 21:32:00	None
7	289387	1372782	193782	2020-03-13	2020-03-13 21:32:00	None
8	301436	1372782	4000609	2020-03-13	2020-03-13 21:32:00	None
9	306327	1372782	4281826	2020-03-13	2020-03-13 21:32:00	None
10	318699	1372782	434004	2020-03-13	2020-03-13 21:32:00	None
11	645454	5675534	319837	2018-08-21	2018-08-21 09:52:00	None
12	675654	5675534	319836	2018-09-05	2018-09-05 16:08:00	None

Figure 7: Condition Occurrence

```
data.to_sql(name=table_name, con=engine, if_exists='append', index=False)

print("Data has been successfully loaded into the database.")

query = text(f"SELECT * FROM {table_name}")
result = pd.read_sql(query, con=engine)
print("Data retrieved from the database:")
result
```

F. Export data tables

After loading all the tales, some scenarios were provided to show more data manipulation using DML operations within database. When a person goes to a medical place, medical staff will provide various examinations. These activities or processes are stored in procedure_occurrence table and we have visit_occurrence table to store basic information about visit. With the help of visit_occurrence_id, combing these tables together gives more records.

Listing 11: Combined Procedure Visit

```
create_table_command = """CREATE TABLE combined_procedure_visit AS
SELECT vo.visit_occurrence_id, c1.concept_name AS visit_concept_name, vo.
    visit_start_date,
    vo.visit_end_date, vo.care_site_id, p.person_id, p.procedure_occurrence_id,
    c2.concept_name AS pocedure_name, p.procedure_date, p.procedure_source_value
FROM
    VISIT_OCCURRENCE vo
JOIN
    PROCEDURE_OCCURRENCE p ON vo.visit_occurrence_id = p.visit_occurrence_id
JOIN
    CONCEPT c1 ON vo.visit_concept_id = c1.concept_id
JOIN
    CONCEPT c2 ON p.procedure_concept_id = c2.concept_id;
"""
with engine.begin() as conn:
    conn.execute(text(create_table_command))

table_name = 'combined_procedure_visit'
query = text(f"SELECT * FROM {table_name}")
result = pd.read_sql(query, con=engine)
print("Data retrieved from the database:")
result
```

	visit_occurrence_id	person_id	visit_concept_id	visit_start_date	visit_start_datetime	visit_end_date	visit_end_datetime
0	8790700	3651691	9202	2016-07-20	2016-07-19 16:48:00	2017-09-19	2017-09-18 16:48:00
1	9552000	2333210	9202	2015-11-06	2015-11-05 16:48:00	2017-03-06	2017-03-05 16:48:00
2	39627400	2680855	9202	2014-11-26	2014-11-25 16:48:00	2016-05-26	2016-05-25 16:48:00
3	40848600	3924012	9202	2015-08-08	2015-08-07 16:48:00	2016-09-08	2016-09-07 16:48:00
4	42780100	1503632	9202	2015-08-13	2015-08-12 16:48:00	2016-10-13	2016-10-12 16:48:00
5	43037600	3590868	9202	2016-03-01	2016-02-29 16:48:00	2017-04-01	2017-03-31 16:48:00
6	43140500	3485274	9202	2016-03-22	2016-03-21 19:12:00	2017-04-21	2017-04-20 16:48:00
7	43246200	1667878	9202	2016-03-29	2016-03-28 19:12:00	2017-06-29	2017-06-28 16:48:00
8	43552600	3743446	9202	2016-03-16	2016-03-15 16:48:00	2016-11-22	2016-11-21 16:48:00
9	66217400	2746928	262	2018-06-27	2018-06-26 16:48:00	2019-08-17	2019-08-16 16:48:00

Figure 8: Visit Occurrence

When considering medical data, various chemical elements become relevant. The measurement table serves to store results from diverse measurements for patients. Combining personal information with measurement details enhances the usability and significance of the data for users.

Listing 12: Combined Person Measurement

```
create_table_command = """CREATE TABLE combined_person_measurement AS
SELECT p.person_id, c1.concept_name AS gender, p.birth_datetime, c2.concept_name
     AS race, m.measurement_id,
    m.person_id AS measurement_person_id, mc.concept_name AS
        measurement_concept_name, m.measurement_datetime,
    m.value_as_number, m.range_low, m.range_high, m.unit_source_value
FROM
    PERSON p
JOIN
    MEASUREMENT m ON p.person_id = m.person_id
LEFT JOIN
    CONCEPT c1 ON p.gender_concept_id = c1.concept_id
LEFT JOIN
    CONCEPT c2 ON p.race_concept_id = c2.concept_id
LEFT JOIN
    CONCEPT mc ON m.measurement_concept_id = mc.concept_id;"""
with engine.begin() as conn:
    conn.execute(text(create_table_command))

table_name = 'combined_person_measurement'
query = text(f"SELECT * FROM {table_name}")
result = pd.read_sql(query, con=engine)
print("Data retrieved from the database:")
result
```

Following a diagnosis, doctors frequently prescribe medications to patients. By integrating information from the drug_exposure, condition_occurrence, person, and concept tables, users can gain insights into diseases and the specific medications being administered to patients.

Listing 13: Condition Drug Analysis

```
create_table_command = """
                    CREATE TABLE con_drug AS
                    SELECT p.person_id,
                        p.birth_datetime,
                        c2.concept_name AS gender,
                        c1.concept_name AS race,
```

	drug_exposure_id	person_id	drug_concept_id	drug_exposure_start_date	drug_exposure_start_datetime
0	3456	5675534	36249642	2018-08-04	2018-08-04 09:05:00
1	86043	5675534	36249642	2018-08-22	2018-08-22 01:46:00
2	94533	5675534	36249642	2018-08-04	2018-08-04 10:08:00
3	156714	1372782	19072176	2020-01-23	2020-01-23 00:00:00
4	156731	1372782	19079524	2020-01-24	2020-01-24 00:00:00
5	156742	1372782	19049106	2020-04-26	2020-04-26 00:00:00
6	156757	1372782	19127952	2020-05-27	2020-05-27 00:00:00
7	156766	1372782	19130823	2020-01-29	2020-01-29 00:00:00
8	156786	1372782	1127433	2020-01-30	2020-01-30 00:00:00
9	156828	1372782	40167259	2020-02-02	2020-02-02 00:00:00
10	156881	1372782	19073526	2020-05-06	2020-05-06 00:00:00
11	156940	1372782	42708201	2020-04-11	2020-04-11 00:00:00
12	157016	1372782	1707348	2020-02-13	2020-02-13 00:00:00

13 rows × 23 columns

Figure 9: Drug Exposure

```
                cc.concept_name AS condition_name,
                c.condition_start_date,
                c.condition_end_date,
                dc.concept_name AS drug_name,
                d.drug_exposure_start_date,
                d.drug_exposure_end_date
            FROM DRUG_EXPOSURE d
            INNER JOIN PERSON p ON d.person_id = p.person_id
            LEFT JOIN CONDITION_OCCURRENCE c ON d.person_id = c.
                person_id
            LEFT JOIN CONCEPT c1 ON p.race_concept_id = c1.concept_id
            LEFT JOIN CONCEPT c2 ON p.gender_concept_id = c2.concept_id
            LEFT JOIN CONCEPT cc ON c.condition_concept_id = cc.
                concept_id
            LEFT JOIN CONCEPT dc ON d.drug_concept_id = dc.concept_id;
            """
# Execute the create table command
with engine.begin() as conn:
    conn.execute(text(create_table_command))
```

	device_exposure_id	person_id	device_concept_id	device_exposure_start_date	device_exposure_start_datetime
0	417	493797	4097216	2021-01-08	2021-01-08 14:18:00
1	5675	2915590	44844102	2021-11-25	2021-11-24 23:00:00
2	12375	2915590	44908056	2017-04-24	2017-04-24 00:34:20
3	28284	493797	4224038	2021-09-09	2021-09-09 12:25:00
4	432678	493797	4224038	2021-09-09	2021-09-09 13:20:00
5	467858	2915590	44908056	2017-04-24	2017-04-24 14:15:24
6	843098	2915590	44844102	2020-09-14	2020-09-13 23:00:00
7	973871	493797	4224038	2021-01-09	2021-01-09 03:00:00

Figure 10: Device Exposure

	concept_id	concept_name	domain_id	vocabulary_id	concept_class_id	standard_concept	concept_code
0	262	Emergency Room and Inpatient Visit	Visit	Visit	Visit	S	ERIP
1	8516	White	race	R	x	s	1801294
2	8524	Outpatient Visit	Visit	Visit	Visit	S	OP
3	8717	Inpatient Hospital	Visit	CMS Place of Service	Visit	S	21
4	9202	Outpatient Visit	Visit	Visit	Visit	S	OP
5	32817	EHR	Type Concept	Type Concept	Type Concept	S	OMOP4976890
6	66754	Black Or African American	race	s	s	s	1801294
7	193782	End-stage renal disease	Condition	LOINC	Lab Test	S	39791-17
8	196152	Peritonitis	Condition	LOINC	Lab Test	S	39791-10
9	197596	Toxic gastroenteritis	Condition	LOINC	Lab Test	S	39791-16
10	317577	Arteriosclerotic gangrene	Condition	LOINC	Lab Test	S	39791-14
11	319835	Congestive heart failure	Condition	SNOMED	Clinical Finding	S	42343007
12	434004	Hypervolemia	Condition	LOINC	Lab Test	S	39791-15
13	678546	PMI: Skip	Observation	PPI	Answer	S	PMI_Skip
14	1127433	acetaminophen 325 MG Oral Tablet	Drug	LOINC	Lab Test	S	39791-26

Figure 11: Concept

	measurement_id	person_id	measurement_concept_id	measurement_date	measurement_datetime
0	58776	5675534	3023103	2017-01-01	2016-12-31 20:40:00
1	74577	5675534	3023103	2018-08-07	2018-08-07 02:38:00
2	435677	5675534	3023103	2018-08-22	2018-08-22 02:27:00
3	459800	2217111	3027172	2022-04-07	2022-04-07 13:32:00
4	612700	11549211	3027172	2020-12-15	2020-12-15 13:55:00
5	3475780	5675534	3023103	2018-08-20	2018-08-20 03:36:00
6	5688646	5675534	3023103	2018-08-04	2018-08-04 07:06:00
7	6151470	1749910	3027172	2023-09-17	2023-09-17 13:22:00
8	6435436	5675534	3023103	2018-08-21	2018-08-21 05:59:00
9	7298130	1874170	3027172	2019-12-21	2019-12-22 14:09:05
10	7765980	1996097	3027172	2023-02-05	2023-03-06 18:54:00
11	9072310	4155869	3027172	2023-11-20	2023-11-20 12:51:00
12	11178500	5609338	3027172	2022-10-22	2022-10-22 15:05:00
13	14547850	2725534	3027172	2022-06-29	2023-07-01 21:58:00
14	15946070	4560103	3027172	2022-03-02	2022-03-02 11:50:00
15	16767460	3226358	3027172	2021-12-21	2022-01-21 16:33:00

16 rows × 23 columns

Figure 12: Measurement

```
table_name = 'con_drug'
query = text(f"SELECT * FROM {table_name}")
result = pd.read_sql(query, con=engine)
print("Data retrieved from the database:")
result
```

In the next scenario, we use condition_concept_id for a specific disease (i.e., heart failure), the medicine used to treat heart failure may affect the level of potassium in the blood. For this point, we can add more constraints about the date and monitor the changes in potassium levels. With the help of the concept table, we will know the name of the measurement.

	procedure_occurrence_id	person_id	procedure_concept_id	procedure_date	procedure_datetime	procedure_end_date
0	11289	1861428	2746520	2020-02-27	2020-02-27	None
1	11443	1861428	2746773	2020-02-27	2020-02-27	None
2	26955	1861428	2747274	2020-02-27	2020-02-27	None
3	30589	1861428	2732505	2020-03-09	2020-03-09	None
4	34474	1861428	2746803	2020-02-27	2020-02-27	None
5	38182	1861428	2788126	2020-03-09	2020-03-09	None
6	43885	1861428	2788717	2020-03-11	2020-03-11	None
7	58054	1861428	2749338	2020-03-14	2020-03-14	None
8	84354	1861428	2747017	2020-02-27	2020-02-27	None

Figure 13: Procedure Occurrence

	visit_occurrence_id	visit_concept_name	visit_start_date	visit_end_date	care_site_id	person_id	procedure_occurrence_id	pocedure_name	procedure_date	procedure_source_value
0	8790700	Outpatient Visit	2016-07-20	2017-09-19	None	1861428	11289	Excision of Upper Esophagus, Via Natural or Ar...	2020-02-27	0DB18ZX
1	66217400	Emergency Room and Inpatient Visit	2018-06-27	2019-08-17	None	1861428	11443	Excision of Stomach, Via Natural or Artificial...	2020-02-27	0DB68ZX
2	66217400	Emergency Room and Inpatient Visit	2018-06-27	2019-08-17	None	1861428	26955	Excision of Ascending Colon, Via Natural or Ar...	2020-02-27	0DBK8ZX
3	66217400	Emergency Room and Inpatient Visit	2018-06-27	2019-08-17	None	1861428	30589	Dilation of Left Anterior Tibial Artery, Percu...	2020-03-09	047Q3ZZ
4	66217400	Emergency Room and Inpatient Visit	2018-06-27	2019-08-17	None	1861428	34474	Excision of Duodenum, Via Natural or Artificia...	2020-02-27	0DB98ZX
5	8790700	Outpatient Visit	2016-07-20	2017-09-19	None	1861428	38182	Fluoroscopy of Right Lower Extremity Arteries ...	2020-03-09	B41FYZZ
6	66217400	Emergency Room and Inpatient Visit	2018-06-27	2019-08-17	None	1861428	43885	Transfusion of Nonautologous Red Blood Cells i...	2020-03-11	30233N1
7	8790700	Outpatient Visit	2016-07-20	2017-09-19	None	1861428	58054	Inspection of Peritoneum, Percutaneous Endosco...	2020-03-14	0DJW4ZZ
8	66217400	Emergency Room and Inpatient Visit	2018-06-27	2019-08-17	None	1861428	84354	Excision of Ileum, Via Natural or Artificial O...	2020-02-27	0DBB8ZX

Figure 14: Combined Procedure Visit

Listing 14: Measurement comparison before and after drug

```
table_name = 'PERSON'
query = text(f"""SELECT
    p.person_id, c1.concept_name as measurement_before_drug_name, m1.
        measurement_datetime AS measurement_before_drug_datetime,
    m1.value_as_number AS measurement_before_drug_value, c2.concept_name as
        measurement_after_drug_name,
    m2.measurement_datetime AS measurement_after_drug_datetime, m2.
        value_as_number AS measurement_after_drug_value
FROM
    {table_name} p
JOIN
    CONDITION_OCCURRENCE co ON p.person_id = co.person_id
JOIN
    MEASUREMENT m1 ON p.person_id = m1.person_id AND m1.measurement_datetime >
        co.condition_start_date
JOIN
    DRUG_EXPOSURE de ON p.person_id = de.person_id AND de.
        drug_exposure_start_date > co.condition_start_date
JOIN
    MEASUREMENT m2 ON p.person_id = m2.person_id AND m2.measurement_datetime >
        de.drug_exposure_start_date

LEFT JOIN CONCEPT c1 ON m1.measurement_concept_id = c1.concept_id
LEFT JOIN CONCEPT c2 ON m2.measurement_concept_id = c2.concept_id

WHERE
    co.condition_concept_id = 319835;""")
result = pd.read_sql(query, con=engine)
print("Data retrieved from the database:")
result
```

G. Final SQL script

Listing 15: Person and drug

```
table_name = 'PERSON'
from sqlalchemy import create_engine, Column, Integer, String, Date, DateTime,
    Float, BigInteger, Text
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker
engine = create_engine("sqlite://", echo=True)
Session = sessionmaker(bind=engine)
```

	person_id	gender	birth_datetime	race	measurement_id	measurement_person_id	measurement_concept_name	measurement_datetime	value_as_number	range_low	range_high	unit_source_value
0	5675534	male	1977-03-19	Black Or African American	58776	5675534	Potassium [Moles/volume] in Serum or Plasma	2016-12-31 20:40:00	4.0	3.5	5.1	mmol/L
1	5675534	male	1977-03-19	Black Or African American	74577	5675534	Potassium [Moles/volume] in Serum or Plasma	2018-08-07 02:38:00	4.2	3.5	5.1	mmol/L
2	5675534	male	1977-03-19	Black Or African American	435677	5675534	Potassium [Moles/volume] in Serum or Plasma	2018-08-22 02:27:00	4.4	3.5	5.1	mmol/L
3	5675534	male	1977-03-19	Black Or African American	3475780	5675534	Potassium [Moles/volume] in Serum or Plasma	2018-08-20 03:36:00	3.8	3.5	5.1	mmol/L
4	5675534	male	1977-03-19	Black Or African American	5688646	5675534	Potassium [Moles/volume] in Serum or Plasma	2018-08-04 07:06:00	3.1	3.5	5.1	mmol/L
5	5675534	male	1977-03-19	Black Or African American	6435436	5675534	Potassium [Moles/volume] in Serum or Plasma	2018-08-21 05:59:00	3.8	3.5	5.1	mmol/L

Figure 15: Combined Person Measurement

	person_id	birth_datetime	gender	race	condition_name	condition_start_date	condition_end_date	drug_name	drug_exposure_start_date	drug_exposure_end_date
0	5675534	1977-03-19	male	Black Or African American	None	2018-08-21	None	Microencapsulated potassium chloride 20 MEQ Ex...	2018-08-22	2018-08-22
1	5675534	1977-03-19	male	Black Or African American	Congestive heart failure	2018-08-04	None	Microencapsulated potassium chloride 20 MEQ Ex...	2018-08-22	2018-08-22
2	1372782	1976-06-01	Female	Asian	Hypervolemia	2020-03-13	None	potassium chloride 20 MEQ Extended Release Ora...	2020-04-26	2020-03-23
3	1372782	1976-06-01	Female	Asian	Lymphocytosis	2020-03-13	None	potassium chloride 20 MEQ Extended Release Ora...	2020-04-26	2020-03-23
4	1372782	1976-06-01	Female	Asian	None	2020-03-13	None	potassium chloride 20 MEQ Extended Release Ora...	2020-04-26	2020-03-23
5	1372782	1976-06-01	Female	Asian	End-stage renal disease	2020-03-13	None	potassium chloride 20 MEQ Extended Release Ora...	2020-04-26	2020-03-23
6	1372782	1976-06-01	Female	Asian	Congestive heart failure	2020-03-13	None	potassium chloride 20 MEQ Extended Release Ora...	2020-04-26	2020-03-23
7	1372782	1976-06-01	Female	Asian	Toxic gastroenteritis	2020-03-13	None	potassium chloride 20 MEQ Extended Release Ora...	2020-04-26	2020-03-23
8	1372782	1976-06-01	Female	Asian	Arteriosclerotic gangrene	2020-03-13	None	potassium chloride 20 MEQ Extended Release Ora...	2020-04-26	2020-03-23
9	1372782	1976-06-01	Female	Asian	Atherosclerosis of artery of lower limb	2020-03-13	None	potassium chloride 20 MEQ Extended Release Ora...	2020-04-26	2020-03-23
10	1372782	1976-06-01	Female	Asian	Peritonitis	2020-03-13	None	potassium chloride 20 MEQ Extended Release Ora...	2020-04-26	2020-03-23
11	1372782	1976-06-01	Female	Asian	None	2020-03-13	None	potassium chloride 20 MEQ Extended Release Ora...	2020-04-26	2020-03-23
12	1372782	1976-06-01	Female	Asian	Hypervolemia	2020-03-13	None	sevelamer carbonate 800 MG Oral Tablet	2020-05-27	2020-03-23
13	1372782	1976-06-01	Female	Asian	Lymphocytosis	2020-03-13	None	sevelamer carbonate 800 MG Oral Tablet	2020-05-27	2020-03-23
14	1372782	1976-06-01	Female	Asian	None	2020-03-13	None	sevelamer carbonate 800 MG Oral Tablet	2020-05-27	2020-03-23
15	1372782	1976-06-01	Female	Asian	End-stage renal disease	2020-03-13	None	sevelamer carbonate 800 MG Oral Tablet	2020-05-27	2020-03-23
16	1372782	1976-06-01	Female	Asian	Congestive heart failure	2020-03-13	None	sevelamer carbonate 800 MG Oral Tablet	2020-05-27	2020-03-23
17	1372782	1976-06-01	Female	Asian	Toxic gastroenteritis	2020-03-13	None	sevelamer carbonate 800 MG Oral Tablet	2020-05-27	2020-03-23

Figure 16: Condition Drug Analysis

```
session = Session()
Base = declarative_base()

from sqlalchemy import create_engine, text
import re

# Connection details
USERNAME = 'root'
PASSWORD = 'TSUtigers123'
SERVER = 'localhost'
DATABASE = 'cdm'

# SQLAlchemy connection string for MySQL using mysqlconnector
connection_string = f'mysql+mysqlconnector://{USERNAME}:{PASSWORD}@{SERVER}/{
    DATABASE}'

# Creating the engine for MySQL
engine = create_engine(connection_string, echo=True)


sql_file_path = './OMOPCDM_sql_server_5.4_ddl.sql'

cleaned_sql_commands = []
with open(sql_file_path, 'r') as sql_file:
    for line in sql_file:
        if not line.strip().startswith(('HINT', '--')):
            line = re.sub(r'--.*$', '', line)
            line = line.replace('@cdmDatabaseSchema.', '')
            cleaned_sql_commands.append(line)

sql_commands = ''.join(cleaned_sql_commands)
commands = re.split(r';\s*(?=\n)', sql_commands)

with engine.begin() as conn:
    for command in commands:
        if command.strip():  # Avoid executing empty or whitespace-only commands
            conn.execute(text(command))


sql_file_path = './OMOPCDM_sql_server_5.4_primary_keys.sql'

with open(sql_file_path, 'r') as file:
    sql_commands = file.read().replace('@cdmDatabaseSchema.', '')

commands = sql_commands.split(';\n')

with engine.begin() as connection:
    for command in commands:
        command = command.strip()
        if command:
            connection.execute(text(command))

from sqlalchemy.engine.reflection import Inspector

inspector = Inspector.from_engine(engine)

# Retrieve and print all table names using the Inspector
table_names = inspector.get_table_names()
print("Tables in the database:")
```

	person_id	measurement_before_drug_name	measurement_before_drug_datetime	measurement_before_drug_value	measurement_after_drug_name	measurement_after_drug_datetime	measurement_after_drug_value
0	5675534	Potassium [Moles/volume] in Serum or Plasma	2018-08-21 05:59:00	3.8	Potassium [Moles/volume] in Serum or Plasma	2018-08-22 02:27:00	4.4
1	5675534	Potassium [Moles/volume] in Serum or Plasma	2018-08-04 07:06:00	3.1	Potassium [Moles/volume] in Serum or Plasma	2018-08-22 02:27:00	4.4
2	5675534	Potassium [Moles/volume] in Serum or Plasma	2018-08-20 03:36:00	3.8	Potassium [Moles/volume] in Serum or Plasma	2018-08-22 02:27:00	4.4
3	5675534	Potassium [Moles/volume] in Serum or Plasma	2018-08-22 02:27:00	4.4	Potassium [Moles/volume] in Serum or Plasma	2018-08-22 02:27:00	4.4
4	5675534	Potassium [Moles/volume] in Serum or Plasma	2018-08-07 02:38:00	4.2	Potassium [Moles/volume] in Serum or Plasma	2018-08-22 02:27:00	4.4

Figure 17: Measurement comparison before and after Drug

```
for table_name in table_names:
    print(table_name)


import re

# Path to your SQL file containing CREATE INDEX statements
sql_file_path = './OMOPCDM_sql_server_5.4_indices.sql'

# Read SQL commands from the file and replace the placeholder
with open(sql_file_path, 'r') as file:
    # Read the entire file content
    sql_commands = file.read()
# Remove multiline comments
sql_commands = re.sub(r'/\*.*?\*/', '', sql_commands, flags=re.DOTALL)

# Replace the placeholder
sql_commands = sql_commands.replace('@cdmDatabaseSchema.', '')

# Split the modified SQL commands into individual commands
commands = sql_commands.split(';')

# Filter out empty commands and commands that are commented out
filtered_commands = []
for command in commands:
    # Trim whitespace and remove single-line comments
    stripped_command = command.strip()
    if stripped_command and not stripped_command.startswith('--'):
        filtered_commands.append(stripped_command)

# Execute each filtered command
with engine.begin() as connection:
    for command in filtered_commands:
        try:
            connection.execute(text(command))
        except Exception as e:
            print(f"An error occurred: {e}")
            print(f"While executing: {command}\n")

print("Index creation commands executed successfully.")

import pandas as pd
csv_file_path = 'cohort/observation.csv'
data = pd.read_csv(csv_file_path)

table_name = 'observation'

# Load the DataFrame into MySQL
data.to_sql(name=table_name, con=engine, if_exists='append', index=False)

print("Data has been successfully loaded into the database.")

query = text(f"SELECT * FROM {table_name}")
result = pd.read_sql(query, con=engine)
print("Data retrieved from the database:")
result

...
...
...

import pandas as pd
csv_file_path = 'cohort/procedure_occurrence.csv'  # Update this to the path of
    your CSV file
data = pd.read_csv(csv_file_path)

table_name = 'procedure_occurrence'
```

```
# schema = 'cdmDatabaseSchema'

# Load the DataFrame into MySQL
data.to_sql(name=table_name, con=engine, if_exists='append', index=False)

print("Data has been successfully loaded into the database.")

query = text(f"SELECT * FROM {table_name}")
result = pd.read_sql(query, con=engine)
print("Data retrieved from the database:")
result

## Insert all the data

import pandas as pd
from sqlalchemy import create_engine, exc
import os
directory_path = './cohort/'
csv_files = [f for f in os.listdir(directory_path) if f.endswith('.csv')]
print(csv_files)
for file_name in csv_files:
    table_name = file_name[:-4]  # Remove the '.csv' part to get the table name
    csv_file_path = os.path.join(directory_path, file_name)

    try:
        print(f'Loading data from {table_name}')
        # Read the CSV file into a DataFrame
        data = pd.read_csv(csv_file_path)

        # Convert NaT in datetime columns to None (if applicable)
        datetime_columns = data.select_dtypes(include=['datetime']).columns
        data[datetime_columns] = data[datetime_columns].where(data[
            datetime_columns].notna(), None)

        # Load the DataFrame into MySQL
        data.to_sql(name=table_name, con=engine, if_exists='append', index=False
            )
        print(f"Data from {file_name} has been successfully loaded into the {
            table_name} table in the database.")

    except exc.IntegrityError as e:
        print(f"IntegrityError: {e}. Skipping duplicate entries for {table_name
            }.")
    except Exception as e:
        print(f"An error occurred with {table_name}: {e}")

print("All data has been loaded into the database.")

import pandas
query = text("""SELECT p.person_id,
       p.gender_concept_id,
       p.year_of_birth,
       p.month_of_birth,
       p.day_of_birth,
       p.race_concept_id,
       p.ethnicity_concept_id,
       d.drug_exposure_id,
       d.drug_concept_id,
       d.drug_exposure_start_date,
       d.drug_exposure_end_date,
       c.condition_occurrence_id,
       c.condition_concept_id,
       c.condition_start_date,
       c.condition_end_date,
       cc.concept_name AS condition_name,
       dc.concept_name AS drug_name
FROM DRUG_EXPOSURE d
```

```
INNER JOIN PERSON p ON d.person_id = p.person_id
LEFT JOIN CONDITION_OCCURRENCE c ON d.person_id = c.person_id
LEFT JOIN CONCEPT cc ON c.condition_concept_id = cc.concept_id
LEFT JOIN CONCEPT dc ON d.drug_concept_id = dc.concept_id;""")

result = pd.read_sql(query, con=engine)
print("Data retrieved from the database:")
result

create_table_command = """
                    CREATE TABLE con_drug AS
                    SELECT p.person_id,
                        p.birth_datetime,
                        c2.concept_name AS gender,
                        c1.concept_name AS race,
                        cc.concept_name AS condition_name,
                        c.condition_start_date,
                        c.condition_end_date,
                        dc.concept_name AS drug_name,
                        d.drug_exposure_start_date,
                        d.drug_exposure_end_date
                    FROM DRUG_EXPOSURE d
                    INNER JOIN PERSON p ON d.person_id = p.person_id
                    LEFT JOIN CONDITION_OCCURRENCE c ON d.person_id = c.
                        person_id
                    LEFT JOIN CONCEPT c1 ON p.race_concept_id = c1.concept_id
                    LEFT JOIN CONCEPT c2 ON p.gender_concept_id = c2.concept_id
                    LEFT JOIN CONCEPT cc ON c.condition_concept_id = cc.
                        concept_id
                    LEFT JOIN CONCEPT dc ON d.drug_concept_id = dc.concept_id;
                    """
# Execute the create table command
with engine.begin() as conn:
    conn.execute(text(create_table_command))

create_table_command = """CREATE TABLE combined_person_measurement AS
SELECT p.person_id, c1.concept_name AS gender, p.birth_datetime, c2.concept_name
     AS race, m.measurement_id,
    m.person_id AS measurement_person_id, mc.concept_name AS
        measurement_concept_name, m.measurement_datetime,
    m.value_as_number, m.range_low, m.range_high, m.unit_source_value
FROM
    PERSON p
JOIN
    MEASUREMENT m ON p.person_id = m.person_id
LEFT JOIN
    CONCEPT c1 ON p.gender_concept_id = c1.concept_id
LEFT JOIN
    CONCEPT c2 ON p.race_concept_id = c2.concept_id
LEFT JOIN
    CONCEPT mc ON m.measurement_concept_id = mc.concept_id;"""
with engine.begin() as conn:
    conn.execute(text(create_table_command))

create_table_command = """CREATE TABLE combined_procedure_visit AS
SELECT vo.visit_occurrence_id, c1.concept_name AS visit_concept_name, vo.
    visit_start_date,
    vo.visit_end_date, vo.care_site_id, p.person_id, p.procedure_occurrence_id,
    c2.concept_name AS pocedure_name, p.procedure_date, p.procedure_source_value
FROM
    VISIT_OCCURRENCE vo
JOIN
    PROCEDURE_OCCURRENCE p ON vo.visit_occurrence_id = p.visit_occurrence_id
JOIN
    CONCEPT c1 ON vo.visit_concept_id = c1.concept_id
JOIN
    CONCEPT c2 ON p.procedure_concept_id = c2.concept_id;
```

```
"""
with engine.begin() as conn:
    conn.execute(text(create_table_command))

table_name = 'con_drug'
query = text(f"SELECT * FROM {table_name}")
result = pd.read_sql(query, con=engine)
print("Data retrieved from the database:")
result

query = text("""SELECT p.person_id,
    p.birth_datetime,
    cc.concept_name AS condition_name,
    c.condition_start_date,
    dc.concept_name AS drug_name,
    d.drug_exposure_start_date
FROM DRUG_EXPOSURE d
INNER JOIN PERSON p ON d.person_id = p.person_id
LEFT JOIN CONDITION_OCCURRENCE c ON d.person_id = c.person_id
LEFT JOIN CONCEPT cc ON c.condition_concept_id = cc.concept_id
LEFT JOIN CONCEPT dc ON d.drug_concept_id = dc.concept_id;""")

con_drug_df = pd.read_sql(query, con=engine)
con_drug_df

import pandas
query = text("""SELECT p.person_id,
    p.birth_datetime,
    c2.concept_name as gender,
    c1.concept_name as race,
    cc.concept_name AS condition_name,
    c.condition_start_date,
    c.condition_end_date,
    dc.concept_name AS drug_name,
    d.drug_exposure_start_date,
    d.drug_exposure_end_date
FROM DRUG_EXPOSURE d
INNER JOIN PERSON p ON d.person_id = p.person_id
LEFT JOIN CONDITION_OCCURRENCE c ON d.person_id = c.person_id
LEFT JOIN CONCEPT c1 ON p.race_concept_id = c1.concept_id
LEFT JOIN CONCEPT c2 ON p.gender_concept_id = c2.concept_id
LEFT JOIN CONCEPT cc ON c.condition_concept_id = cc.concept_id
LEFT JOIN CONCEPT dc ON d.drug_concept_id = dc.concept_id;""")

con_drug_df = pd.read_sql(query, con=engine)
con_drug_df

table_name = 'CONDITION_OCCURRENCE'
query = text(f"SELECT * FROM {table_name}")
result = pd.read_sql(query, con=engine)
print("Data retrieved from the database:")
result

table_name = 'con_drug'
query = text(f"SELECT * FROM {table_name}")
result = pd.read_sql(query, con=engine)
print("Data retrieved from the database:")
result

table_name = 'combined_person_measurement'
query = text(f"SELECT * FROM {table_name}")
result = pd.read_sql(query, con=engine)
print("Data retrieved from the database:")
result

table_name = 'PROCEDURE_OCCURRENCE'
query = text(f"SELECT * FROM {table_name}")
```

```
result = pd.read_sql(query, con=engine)
print("Data retrieved from the database:")
result

table_name = 'combined_procedure_visit'
query = text(f"SELECT * FROM {table_name}")
result = pd.read_sql(query, con=engine)
print("Data retrieved from the database:")
result

table_name = 'con_drug'
query = text(f"SELECT * FROM {table_name} cd WHERE cd.drug_exposure_start_date >
     cd.condition_start_date;")
result = pd.read_sql(query, con=engine)
print("Data retrieved from the database:")
result

table_name = 'PERSON'
query = text(f"""SELECT
    p.person_id, c1.concept_name as measurement_before_drug_name, m1.
        measurement_datetime AS measurement_before_drug_datetime,
    m1.value_as_number AS measurement_before_drug_value, c2.concept_name as
        measurement_after_drug_name,
    m2.measurement_datetime AS measurement_after_drug_datetime, m2.
        value_as_number AS measurement_after_drug_value
FROM
    {table_name} p
JOIN
    CONDITION_OCCURRENCE co ON p.person_id = co.person_id
JOIN
    MEASUREMENT m1 ON p.person_id = m1.person_id AND m1.measurement_datetime >
        co.condition_start_date
JOIN
    DRUG_EXPOSURE de ON p.person_id = de.person_id AND de.
        drug_exposure_start_date > co.condition_start_date
JOIN
    MEASUREMENT m2 ON p.person_id = m2.person_id AND m2.measurement_datetime >
        de.drug_exposure_start_date

LEFT JOIN CONCEPT c1 ON m1.measurement_concept_id = c1.concept_id
LEFT JOIN CONCEPT c2 ON m2.measurement_concept_id = c2.concept_id

WHERE
    co.condition_concept_id = 319835;""")
result = pd.read_sql(query, con=engine)
print("Data retrieved from the database:")
result
```

IV. Challenges

During the project, we faced several challenges:

- **Synthetic Data Generation:** Creating synthetic data that maintains privacy without compromising on utility.
- **Meaningful Synthetic Data:** Ensuring the synthetic data is not only statistically valid but also meaningful for health research.
- **Data Schema and Relationships:** Incorporating insights from the All of Us and Athena databases to design an effective schema.
- **Data Management Challenges:** Managing large volumes of data to optimize query performance and storage.
- **Simplification of Data Features:** Focusing on essential features to simplify the data model while retaining key information.

V. Recommendations for Future Development

Future work should aim to:

- **Expand Data Tables:** Add new tables to provide additional relevant data, enhancing the depth of research insights.
- **Enhance Medical Relevance:** Integrate deeper medical knowledge to increase the dataset's utility in health research.
- **Continuous Data Enhancement:** Regularly update and refine the dataset based on emerging medical research and data availability.

VI. Discussions and conclusions

The "All of Us" Research Program offers invaluable advantages in broadening the scope of medical research and deepening our understanding of intricate health-related phenomena. Through its expansive patient count and comprehensive data tables, the program significantly enhances research insights across multiple dimensions, providing researchers with rich and diverse datasets for analysis. The conceptual framework and data tables employed within the program uncover complex relationships within clinical data, offering critical insights into the interconnected nature of health factors and outcomes. By synthesizing data representative of the diverse patient population and information captured within the program, our project aims to mimic the characteristics and complexities present in real-world clinical datasets. As part of our future work, we aim to expand the dataset by identifying additional tables that can provide supplementary information, thereby enriching our understanding of the factors influencing health outcomes. Moreover, by incorporating medical knowledge and expertise, we seek to generate a more meaningful dataset that not only captures diverse dimensions of health but also facilitates deeper analysis and interpretation of the data. The implementation of this project can be found in https://github.com/kawseribn/COMP5400_Project.git.

References

[1] A. H. Ramirez *et al.*, "The all of us research program: Data quality, utility, and diversity," *Patterns*, vol. 3, no. 8, 2022.
[2] Observational Health Data Sciences and Informatics (OHDSI), "Ohdsi/commondatamodel," GitHub repository, n.d. [Online]. Available: https://github.com/OHDSI/CommonDataModel
[3] O. H. D. Sciences and I. (OHDSI), "Omop database," Web resource, n.d. [Online]. Available: https://www.ohdsi.org/omop-database/
[4] ——, "Athena," Web application, n.d. [Online]. Available: https://www.ohdsi.org/athena/
[5] ——, "Atlas – ohdsi," Web application, n.d. [Online]. Available: https://atlas-demo.ohdsi.org/#/home