FINAL PROJECT DOCUMENTATION

TEAM MEMBERS:

1. ARADHANA PANCHAL
2. PRIYAK TAILOR
3. SHRUTHI KOLLURU

Business Problem Definition:

The business problem we aim to solve is tracking the spread of infectious diseases and analyzing the effectiveness of various treatments on a regional basis. By creating a comprehensive database, healthcare organizations, research institutions, and government agencies can gain valuable insights into disease transmission patterns, high-risk areas, and the efficacy of different treatments in controlling outbreaks. This information can be used to allocate resources, inform public health policies, and improve patient outcomes.

Consider including the following entities in your ER diagram:

- Disease (DiseaseID, Name, Type, Symptoms)
- Patient (PatientID, Name, DateOfBirth, Gender, Address, ContactNumber)
- Location (LocationID, City, State, Country, Latitude, Longitude)
- Treatment (TreatmentID, Name, Description, Type)
- Diagnosis (DiagnosisID, DiseaseID, PatientID, LocationID, DiagnosisDate)
- TreatmentOutcome (OutcomeID, TreatmentID, DiagnosisID, StartDate, EndDate, Result)
- HealthcareProvider (ProviderID, Name, Type, Address, ContactNumber)
- ProviderTreatment (ProviderTreatmentID, ProviderID, TreatmentID, Cost)

Create relationships, cardinality/optionality, and relationship phrases for the entities in the ER diagram. For example:

- A Patient can have one or many Diagnoses.
- A Diagnosis is related to one Disease, one Patient, and one Location.
- A TreatmentOutcome is related to one Treatment and one Diagnosis.
- A HealthcareProvider can offer one or many Treatments.
- A Treatment can be offered by one or many HealthcareProviders.

Data Dictionary:

- In the data dictionary, define each entity, non-obvious attributes, and additional details about entity identities and datatypes. For example:

- Disease:
  DiseaseID: Unique identifier for a disease (Integer)
  Name: Name of the disease (String)
  Type: Category of the disease (e.g., viral, bacterial) (String)
  Symptoms: Common symptoms of the disease (String)
  Relationship: One Disease can have many Diagnoses (1:N)
  Relationship phrase: A Disease may be diagnosed in many Diagnoses, but a Diagnosis is related to only one Disease.

- Patient:
  PatientID: Unique identifier for a patient (Integer)
  Name: Full name of the patient (String)
  DateOfBirth: Patient's date of birth (Date)
  Gender: Patient's gender (String)
  Address: Patient's home address (String)
  ContactNumber: Patient's contact number (String)
  Relationship: One Patient can have many Diagnoses (1:N)
  Relationship phrase: A Patient may have many Diagnoses, but a Diagnosis is related to only one Patient.

- Location:
  LocationID: Unique identifier for a location (Integer)
  City: Name of the city (String)
  State: Name of the state (String)
  Country: Name of the country (String)
  Latitude: Geographical latitude of the location (Decimal)
  Longitude: Geographical longitude of the location (Decimal)
  Relationship: One Location can have many Diagnoses (1:N)
  Relationship phrase: A Location may have many Diagnoses, but a Diagnosis is related to only one Location.

- Treatment:
  TreatmentID: Unique identifier for a treatment (Integer)
  Name: Name of the treatment (String)
  Description: Description of the treatment, including method and purpose (String)
  Type: Category of the treatment (e.g., medication, therapy) (String)
  Relationship: Many-to-many relationship between Treatment and HealthcareProvider through ProviderTreatment (M:N)
  Relationship phrase: A Treatment can be offered by many HealthcareProviders, and a HealthcareProvider can offer many Treatments.

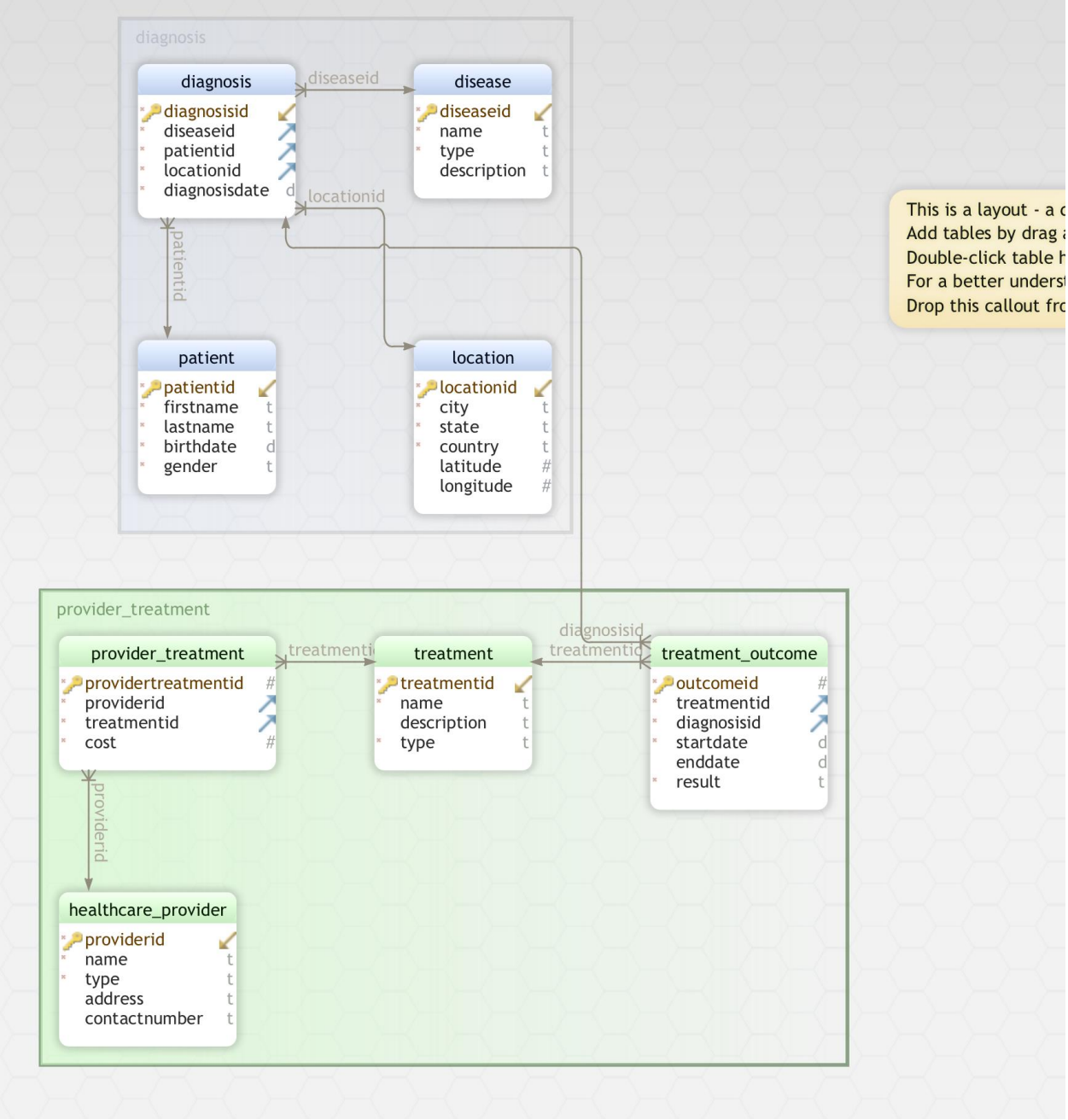
- Diagnosis:
  DiagnosisID: Unique identifier for a diagnosis (Integer)
  DiseaseID: Foreign key referencing Disease (Integer)
  PatientID: Foreign key referencing Patient (Integer)
  LocationID: Foreign key referencing Location (Integer)
  DiagnosisDate: Date when the diagnosis was made (Date)
  Relationship: One Diagnosis can have many TreatmentOutcomes (1:N)
  Relationship phrase: A Diagnosis may have many TreatmentOutcomes, but a **TreatmentOutcome** is related to only one Diagnosis.

- TreatmentOutcome:
  OutcomeID: Unique identifier for a treatment outcome (Integer)
  TreatmentID: Foreign key referencing Treatment (Integer)
  DiagnosisID: Foreign key referencing Diagnosis (Integer)
  StartDate: Date when the treatment started (Date)
  EndDate: Date when the treatment ended (Date)
  Result: Outcome of the treatment (e.g., successful, failed, ongoing) (String)
  Relationship: One Treatment can have many TreatmentOutcomes (1:N)
  Relationship phrase: A Treatment may have many TreatmentOutcomes, but a TreatmentOutcome is related to only one Treatment.

- HealthcareProvider:
  ProviderID: Unique identifier for a healthcare provider (Integer)
  Name: Name of the healthcare provider (String)
  Type: Type of healthcare provider (e.g., hospital, clinic, private practice) (String)
  Address: Address of the healthcare provider (String)
  ContactNumber: Contact number of the healthcare provider (String)
  Relationship: Many-to-many relationship between HealthcareProvider and Treatment through ProviderTreatment (M:N)
  Relationship phrase: A HealthcareProvider can offer many Treatments, and a Treatment can be offered by many HealthcareProviders.

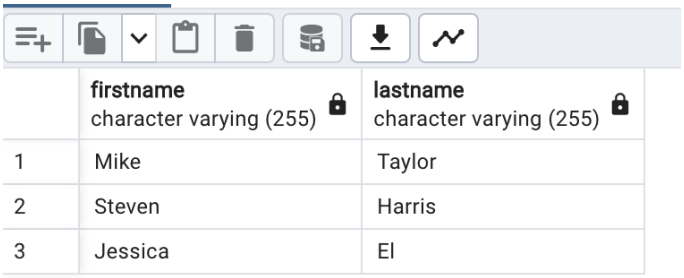
- ProviderTreatment:
  ProviderTreatmentID: Unique identifier for a provider treatment relationship (Integer)
  ProviderID: Foreign key referencing HealthcareProvider (Integer)
  TreatmentID: Foreign key referencing Treatment (Integer)
  Cost: Cost of the treatment at the healthcare provider (Decimal)
  Relationship: One ProviderTreatment connects one HealthcareProvider and one Treatment (1:1)
  Relationship phrase: A ProviderTreatment connects a single HealthcareProvider with a single Treatment.

ER Diagram:


diagnosis
diagnosis
diagnosisid
diseaseid
patientid
locationid
diagnosisdate
diseaseid
locationid
patientid
disease
diseaseid
name
type
description
patient
patientid
firstname
lastname
birthdate
gender
location
locationid
city
state
country
latitude
longitude
This is a layout - a
Add tables by drag
Double-click table h
For a better unders
Drop this callout fr
provider_treatment
provider_treatment
providertreatmentid
providerid
treatmentid
cost
treatmenti
treatment
treatmentid
name
description
type
diagnosisid
treatmentid
treatment_outcome
outcomeid
treatmentid
diagnosisid
startdate
enddate
result
providerid
healthcare_provider
providerid
name
type
address
contactnumber

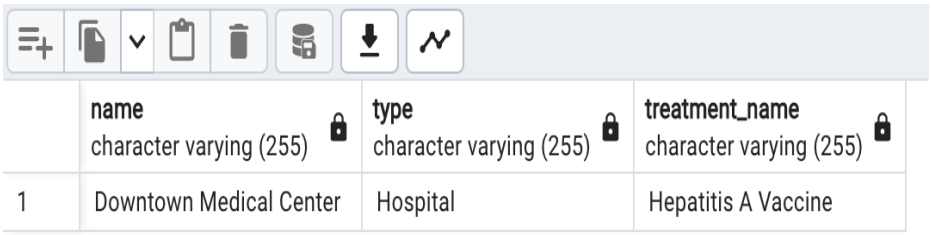
Here are some of the interesting use cases and analysis demonstrated with the data and queries:

- Retrieving all patients diagnosed with Malaria.



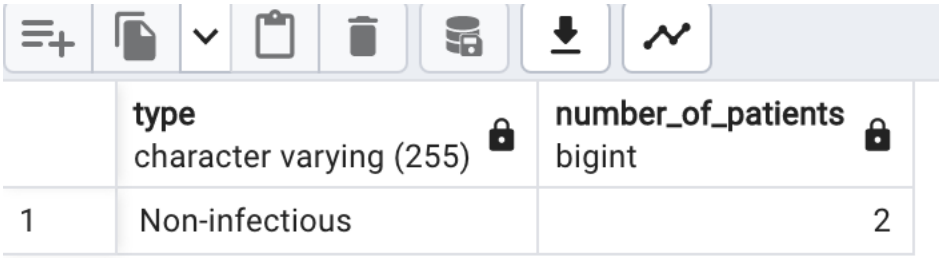
	firstname character varying (255)	lastname character varying (255)
1	Mike	Taylor
2	Steven	Harris
3	Jessica	El

- Listing all healthcare providers offering Hepatitis A Vaccine.



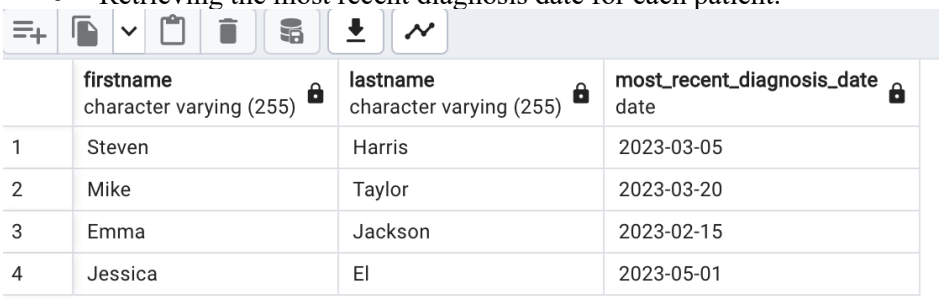
	name character varying (255)	type character varying (255)	treatment_name character varying (255)
1	Downtown Medical Center	Hospital	Hepatitis A Vaccine

- Counting the number of patients diagnosed with non-infectious diseases.



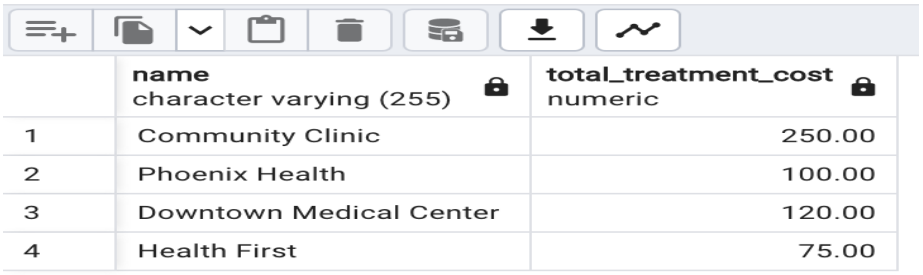
	type character varying (255)	number_of_patients bigint
1	Non-infectious	2

- Retrieving the most recent diagnosis date for each patient.



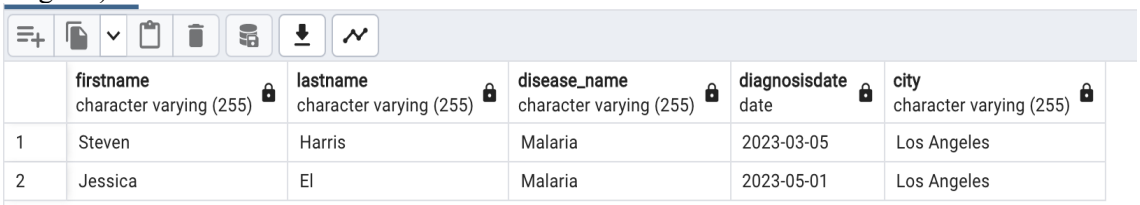
	firstname character varying (255)	lastname character varying (255)	most_recent_diagnosis_date date
1	Steven	Harris	2023-03-05
2	Mike	Taylor	2023-03-20
3	Emma	Jackson	2023-02-15
4	Jessica	El	2023-05-01

- Finding the total cost of treatments provided by each healthcare provider.



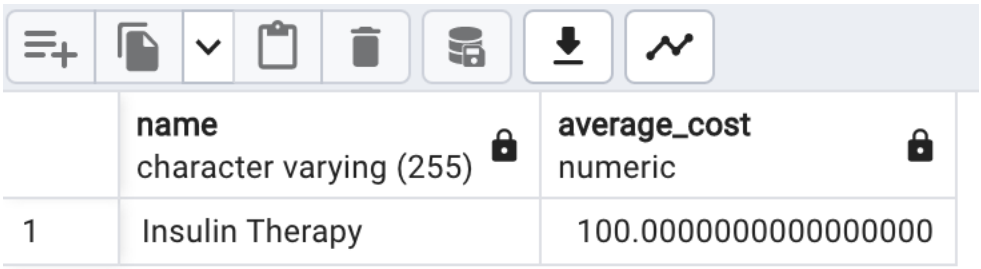
	name character varying (255)	total_treatment_cost numeric
1	Community Clinic	250.00
2	Phoenix Health	100.00
3	Downtown Medical Center	120.00
4	Health First	75.00

- Retrieving patient details and diagnosis information for patients diagnosed in a specific city (Los Angeles).



	firstname character varying (255)	lastname character varying (255)	disease_name character varying (255)	diagnosisdate date	city character varying (255)
1	Steven	Harris	Malaria	2023-03-05	Los Angeles
2	Jessica	El	Malaria	2023-05-01	Los Angeles

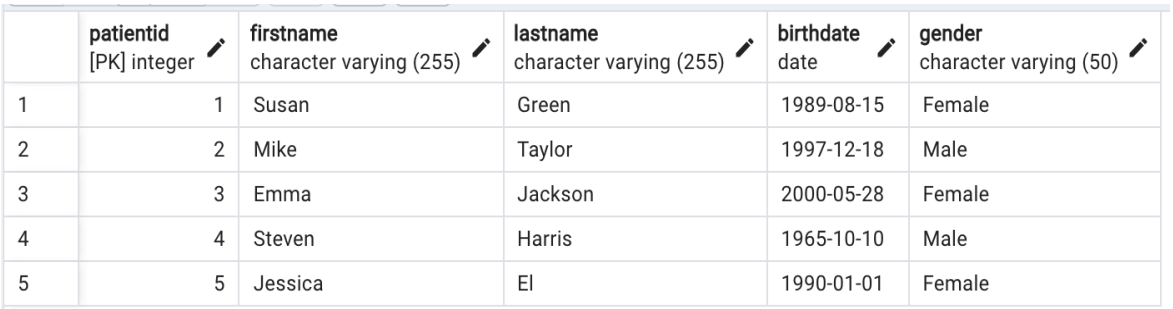
- Finding the average cost of a Insulin Therapy across all healthcare providers.



	name character varying (255)	average_cost numeric
1	Insulin Therapy	100.0000000000000000

Additionally, It showcases data manipulation, such as:

- We Add a new patient, Jessica El, to the patient table.



	patientid [PK] integer	firstname character varying (255)	lastname character varying (255)	birthdate date	gender character varying (50)
1	1	Susan	Green	1989-08-15	Female
2	2	Mike	Taylor	1997-12-18	Male
3	3	Emma	Jackson	2000-05-28	Female
4	4	Steven	Harris	1965-10-10	Male
5	5	Jessica	El	1990-01-01	Female

- Added a new diagnosis for Jessica El with Malaria in Los Angeles on May 1st, 2023.

	diagnosisid [PK] integer	diseaseid integer	patientid integer	locationid integer	diagnosisdate date
1	5	1	2	4	2023-03-20
2	6	3	3	3	2023-02-15
3	7	3	4	4	2023-01-10
4	8	1	4	1	2023-03-05
5	9	1	5	1	2023-05-01

- Added a new treatment, "Malaria Prevention Vaccine", to the treatment table.

	treatmentid [PK] integer	name character varying (255)	description text	type character varying (255)
1	1	Antimalarial Medication	Medications to treat Malaria	Pharmacological
2	2	Hepatitis A Vaccine	Vaccine to prevent Hepatitis A	Immunization
3	3	Inhalers	Medications for Asthma	Pharmacological
4	4	Insulin Therapy	Treatment to regulate blood sugar levels	Pharmacological
5	5	Malaria Prevention Vaccine	A vaccine to prevent Malaria	Immunization

- Added a new provider treatment for the Community Clinic offering the Malaria Prevention Vaccine at a cost of $90.

	providertreatmentid [PK] integer	providerid integer	treatmentid integer	cost numeric (10,2)
1	9	2	2	120.00
2	10	3	3	75.00
3	11	4	4	100.00
4	12	1	1	150.00
5	13	1	5	100.00
6	16	1	5	90.00

- Added a new treatment outcome for Jessica El, indicating she recovered after receiving the Malaria Prevention Vaccine treatment.

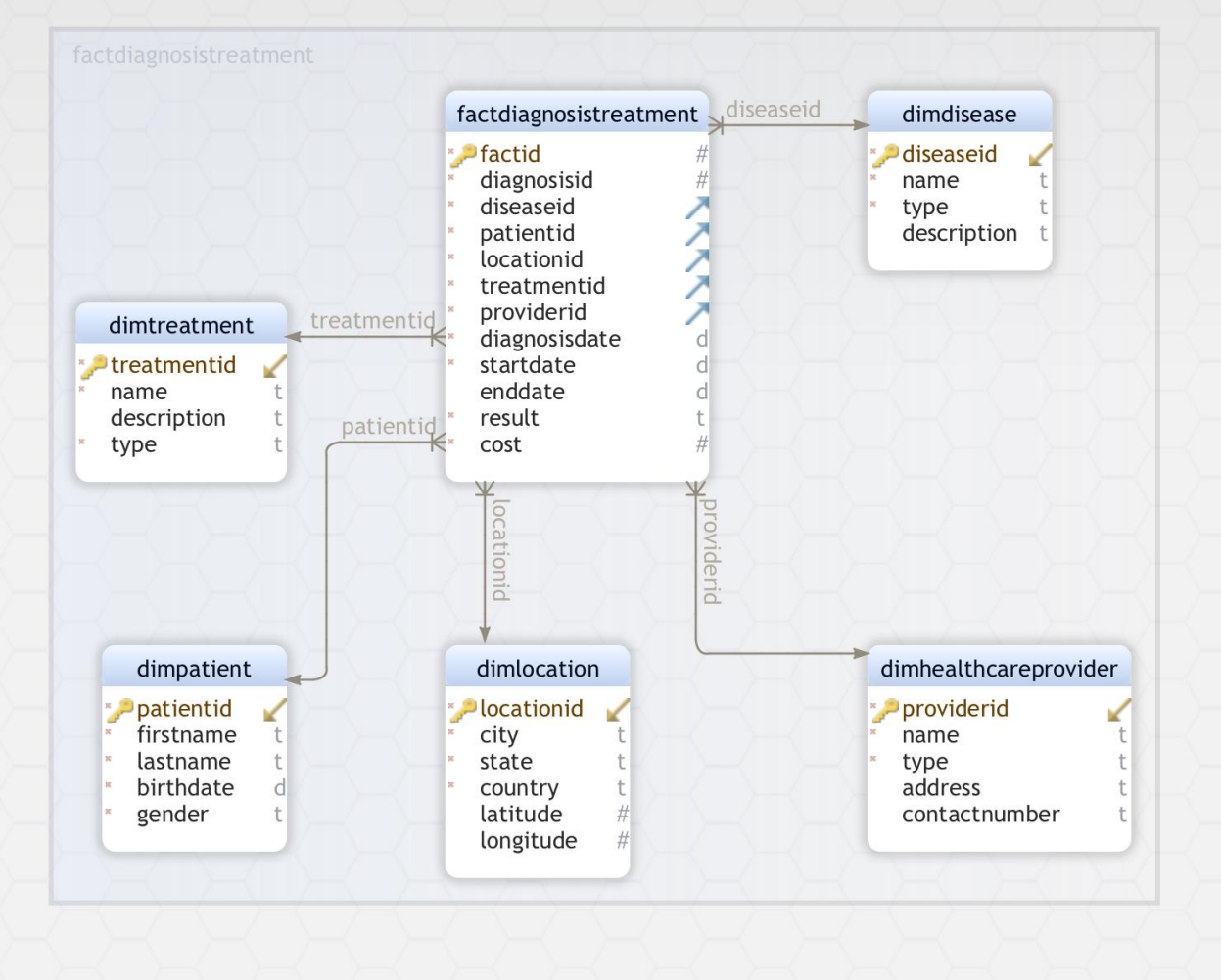
	outcomeid [PK] integer	treatmentid integer	diagnosisid integer	startdate date	enddate date	result character varying (255)
1	21	1	5	2023-03-21	2023-03-30	Recovered
2	22	3	7	2023-02-16	2023-02-25	Stable
3	23	4	6	2023-01-11	2023-01-20	Improved
4	24	1	8	2023-03-06	2023-03-15	Recovered
5	25	5	9	2023-05-02	2023-05-12	Recovered

- Updated the cost of the Malaria Prevention Vaccine at the Community Clinic by increasing it by $10.

	providertreatmentid [PK] integer	providerid integer	treatmentid integer	cost numeric (10,2)
1	9	2	2	120.00
2	10	3	3	75.00
3	11	4	4	100.00
4	12	1	1	150.00
5	13	1	5	100.00

Disease schema

In the dimensional model, we have one fact table (**FactDiagnosisTreatment**) derived from the Diagnosis, ProviderTreatment, and TreatmentOutcome fact entities. This fact table captures the quantitative data related to diagnoses, treatment costs, and treatment outcomes, and is linked to the dimension tables (DimDisease, DimPatient, DimLocation, DimTreatment, and DimHealthcareProvider) through foreign keys.



Created five dimension tables in the data_warehouse schema: DimDisease: Stores information about different diseases.

- DimPatient: Stores patient details.
- DimLocation: Stores location details, such as city, state, and country.
- DimTreatment: Stores treatment-related information.
- DimHealthcareProvider: Stores healthcare provider details, such as name, type, address, and contact number.

→ Each table has a primary key and appropriate column to store the respective data.

→ Created the FactDiagnosisTreatment table in the data_warehouse schema. The fact table stores information about diagnosis and treatment events, with foreign key references to the dimension tables. It also has additional columns to store diagnosis-related data, such as dates, result, and cost.

- Load data from the OLTP (Online Transaction Processing) tables to the corresponding dimension tables in the data warehouse schema. This is done using

```
INSERT INTO data_warehouse.dimdisease SELECT * FROM disease;
INSERT INTO data_warehouse.dimpatient SELECT * FROM patient;
INSERT INTO data_warehouse.dimlocation SELECT * FROM location;
INSERT INTO data_warehouse.dimtreatment SELECT * FROM treatment;
INSERT INTO data_warehouse.dimhealthcareprovider SELECT * FROM healthcare_provider;
```

statements, which copy data from the OLTP tables to the dimension tables.

- We load data from the OLTP tables to the FactDiagnosisTreatment table in the data warehouse schema.

- This is done by joining the diagnosis, treatment_outcome, and provider_treatment tables from the OLTP schema, and inserting the relevant information into the fact table.

```
INSERT INTO data_warehouse.factdiagnosistreatment (
   DiagnosisID, DiseaseID, PatientID, LocationID, TreatmentID, ProviderID, DiagnosisDate,
StartDate, EndDate, Result, Cost)
SELECT d.DiagnosisID, d.DiseaseID, d.PatientID, d.LocationID, t.TreatmentID, p.ProviderID,
d.DiagnosisDate, t.StartDate, t.EndDate, t.Result, p.Cost
FROM diagnosis d
JOIN treatment_outcome t ON d.DiagnosisID = t.DiagnosisID
JOIN provider_treatment p ON t.TreatmentID = p.TreatmentID;
```

WE Run analytical queries to showcase insights from the data warehouse.

- These queries include:

a. Number of diagnoses per disease: This query shows the number of diagnoses for each disease, ordered by the count of diagnoses in descending order.

	disease character varying (255)	diagnoses_count bigint
1	Malaria	3
2	Asthma	2

b. Average cost of treatments per disease: This query calculates the average cost of treatments for each disease, ordered by the average cost in descending order.

	disease character varying (255)	average_cost numeric
1	Malaria	133.3333333333333333
2	Asthma	87.5000000000000000

c. Treatment outcomes per disease: This query shows the treatment outcomes (e.g., cured, not cured) for each disease, along with their counts, ordered by disease and outcome count in descending order.

	disease character varying (255)	outcome character varying (255)	outcome_count bigint
1	Asthma	Stable	1
2	Asthma	Improved	1
3	Malaria	Recovered	3

d. Top healthcare providers by the number of treatments performed: This query shows the top 5 healthcare providers based on the number of treatments they have performed, ordered by the count of treatments in descending order.

	provider character varying (255)	treatments_count bigint
1	Community Clinic	3
2	Phoenix Health	1
3	Health First	1

NoSQL database structure and contrast:

In a NoSQL database, the structure differs significantly from that of a relational database. The data model depends on the type of NoSQL database chosen. Here, we will provide examples for two different types of NoSQL databases: MongoDB (a document-oriented database) and Neo4j (a graph database).

MongoDB (Document-oriented database):

In MongoDB, data is stored as documents in collections, with each document being a flexible, self-contained JSON-like structure. This allows for a more flexible schema that can easily adapt to changes in the data model. Here's an example of how the data structure could look like for our healthcare use case:

```
{
  "patient": {
    "_id": ObjectId("..."),
    "firstName": "Susan",
    "lastName": "Green",
    "birthDate": "1989-08-15",
    "gender": "Female",
    "diagnoses": [
      {
        "disease": {
          "name": "Malaria",
          "type": "Parasitic",
          "description": "A life-threatening disease caused by parasites transmitted through the bite of infected mosquitoes."
        },
```

```
      "location": {
        "city": "Los Angeles",
        "state": "California",
        "country": "USA",
        "latitude": 34.0522,
        "longitude": -118.2437
      },
      "diagnosisDate": "2023-03-05",
      "treatmentOutcome": {
        "treatment": {
          "name": "Antimalarial Medication",
          "description": "Medications to treat and prevent Malaria",
          "type": "Pharmacological"
        },
        "startDate": "2023-03-06",
        "endDate": "2023-03-15",
        "result": "Recovered",
        "healthcareProvider": {
          "name": "Community Clinic",
          "type": "Clinic",
          "address": "789 Main St, Los Angeles, CA",
          "contactNumber": "(555) 987-6543",
          "cost": 120.00
        }
      }
    }
  ]
 }
}
```


Value and contrast of MongoDB:

•Flexible schema: MongoDB can handle changes in the data model more easily compared to a relational database.

•Scalability: MongoDB provides horizontal scaling by allowing data to be distributed across multiple servers, which is beneficial for large datasets and high-traffic applications.

•Reduced need for JOINs: As data is stored in a nested format, there's less need for complex JOIN operations during queries.

Neo4j (Graph database):

In Neo4j, data is represented as nodes, relationships, and properties in a graph model. Nodes represent entities, relationships represent connections between entities, and properties store attributes of entities or relationships.

Here's a conceptual representation of our healthcare use case in Neo4j:

- Nodes: Patient, Disease, Location, Treatment, HealthcareProvider
- Relationships: DIAGNOSED_WITH, TREATED_WITH, LOCATED_IN, TREATS, PROVIDED_BY

Value and contrast of Neo4j:

- Optimized for connected data: Neo4j is designed for querying and traversing connected data, making it well-suited for use cases that involve complex relationships.

- Flexibility: Graph databases can easily accommodate changes in the data model without requiring significant schema alterations.

- Query performance: Neo4j provides efficient querying for connected data, with performance that remains relatively consistent as the dataset grows.

In summary, NoSQL databases like MongoDB and Neo4j offer different data structures that cater to varying use cases and requirements. While MongoDB is well-suited for storing hierarchical or semi-structured data, Neo4j excels at handling complex relationships and connected data. Both MongoDB and Neo4j provide flexibility in schema design, allowing for easier adaptation to changes in the data model.

AWS Architecture Design:

To design an architecture for your database and application in AWS with a focus on resilience, performance, and security, we can use the following AWS services:

- Amazon RDS (Relational Database Service): To host the PostgreSQL database with multi-AZ deployment for high availability and automated backups for resilience. It also supports encryption at rest and in transit.

- Amazon EC2 (Elastic Compute Cloud): To host the application server, which can be auto scaled based on traffic, ensuring performance and cost optimization.

- Amazon S3 (Simple Storage Service): To store ETL scripts and any other artifacts needed for the data processing pipeline.

- Amazon Lambda: To handle both batch and real-time data processing using the Lambda Architecture.

- Amazon API Gateway: To create, publish, and secure APIs for the application.

- AWS IAM (Identity and Access Management): To manage access control and user permissions for security.

- Amazon VPC (Virtual Private Cloud): To create an isolated network for your infrastructure, with security groups and network access control lists for enhanced security.

2. Loading Data Using AWS Lambda Architecture:

- **Batch Processing:** Use AWS Lambda functions to perform ETL operations on data stored in Amazon S3. Schedule the functions to run periodically using Amazon CloudWatch Events. The processed data can then be loaded into the PostgreSQL database hosted on Amazon RDS.

- **Real-time Processing:** Stream data into AWS Kinesis Data Streams, which can trigger AWS Lambda functions to process the data in real-time. The processed data can then be stored in the PostgreSQL database hosted on Amazon RDS.

❖ Snowflake vs. PostgreSQL:

If your data warehouse was in Snowflake instead of PostgreSQL, you would experience several advantages and differentiating factors:

1. Snowflake is designed to handle large-scale data workloads, providing virtually unlimited storage and compute resources to scale independently.
2. Snowflake's architecture optimizes query execution by leveraging its unique multi-cluster shared data architecture, which allows for better performance on analytical workloads.
3. Snowflake enables seamless and secure data sharing with other Snowflake users, without requiring data copies or transfers.
4. Snowflake can handle many concurrent queries without impacting performance, thanks to its multi-cluster architecture.
5. Snowflake uses a consumption-based pricing model, where you only pay for the storage and compute resources you use.
6. Snowflake is a fully managed service, which means you don't need to worry about infrastructure maintenance, software updates, or database administration.
7. Snowflake provides native support for semi-structured data like JSON, making it easier to store and query JSON data without any transformations.
8. Overall, while both PostgreSQL and Snowflake can be used as data warehouses, Snowflake offers better scalability, performance, and flexibility for large-scale analytical workloads.