FINAL PROJECT ON DISEASE MODEL

Anuj Kumar Shah
Niyanta Pandey
Dakshit Ashokbhai Golakiya

Executive Summary:

Our project develops a sophisticated database system to manage and analyze disease-related data. Leveraging our knowledge in data modeling, database design, and analysis, we've crafted a tool that offers valuable insights into disease trends and the effectiveness of treatments.

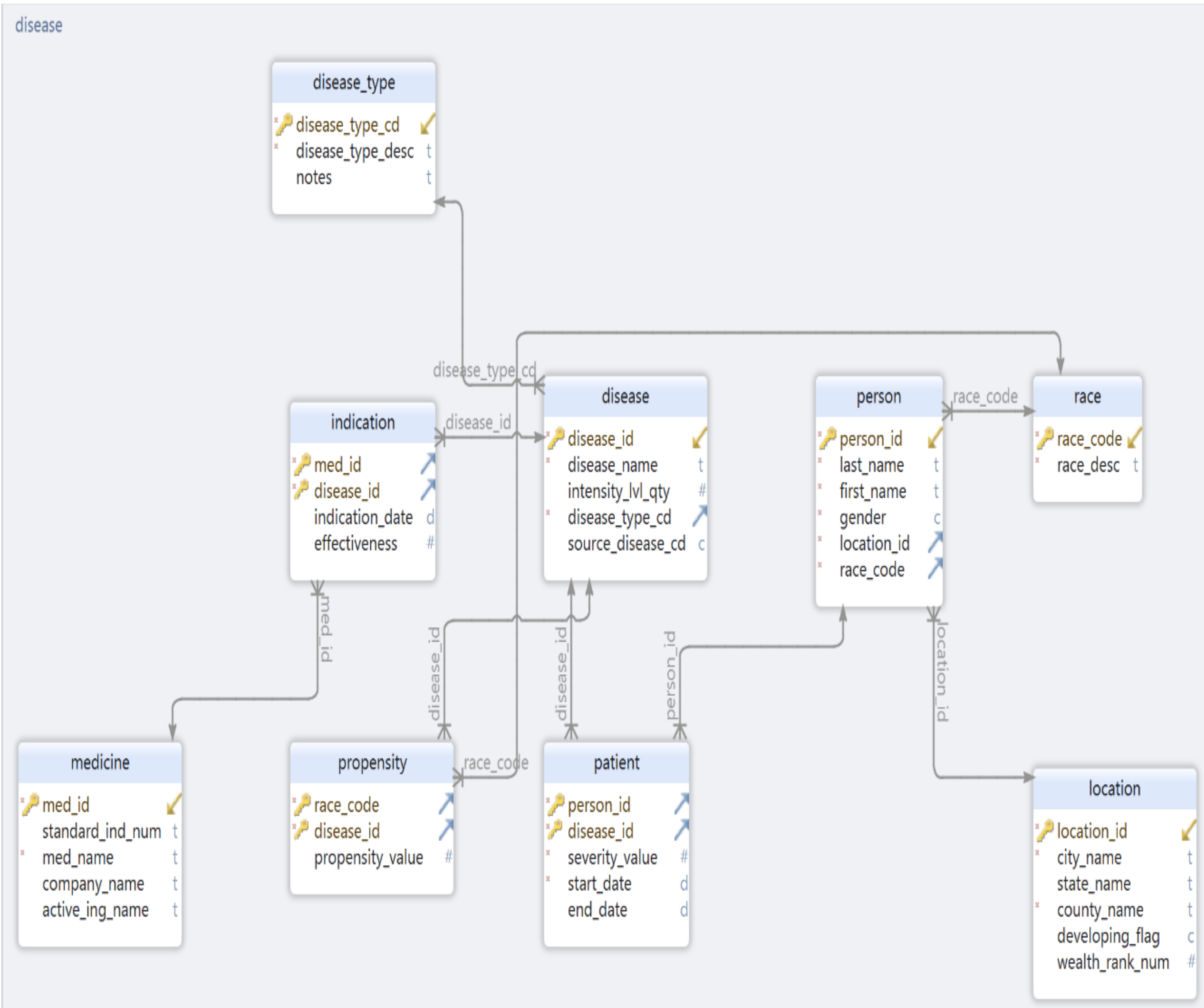
Introduction

The goal of this project is to build a database system adept at handling disease data. We aim to utilize various data modeling and database management techniques learned during our course to derive meaningful insights into patterns in disease spread, treatment efficacy, and demographic distributions.

Background and Context

In this project, we applied fundamental concepts such as Entity-Relationship (ER) modeling, OLTP, OLAP, and ETL processes. These concepts are vital in managing and interpreting extensive health-related data sets. We also explored transitioning to flexible, schema-less NoSQL databases like MongoDB or Neo4J, enhancing scalability and performance. Additionally, we researched using AWS components like RDS/Aurora, DynamoDB, Lambda, and Kinesis for robust, secure, and real-time data processing. For data warehousing, we delved into Snowflake's capabilities for scalability, automated performance tuning, and efficient data sharing.

ER Diagram and Data Dictionary



Entity-Relationship Diagram (ERD) for a database related to healthcare information. It shows several tables with their columns, and the lines between them indicate relationships between the tables. Here's a detailed breakdown of the tables, their columns, and how they're connected:

1. **disease_type:**

    Disease_type_cd: (Primary Key) Code identifying the type of disease.
    disease_type_desc: The description of the disease type.
    notes: Any notes related to the disease type.

2. **disease**:

    disease_id: Unique identifier for the disease.
    disease_name: Name of the disease.
    intensity_lvl_qty: The intensity level quantity of the disease.
    disease_type_cd: A foreign key linking to the disease_type_cd in the disease_type table.
    source_disease_cd: The source code of the disease.

3. **indication**:

    med_id: The medication identifier.
    disease_id: A foreign key linking to the disease_id in the disease table.
    indication_date: The date when the medication was indicated.
    effectiveness: The effectiveness of the medication.

4. **medicine:**

    med_id: Unique identifier for the medicine.
    standard_id_num: A standard identification number for the medicine.
    med_name: Name of the medicine.
    company_name: The name of the company that produces the medicine.
    active_ing_name: The name of the active ingredient in the medicine.

5. **person:**

    person_id: Unique identifier for the person.
    last_name: The last name of the person.

first_name: The first name of the person.
gender: The gender of the person.
location_id: A foreign key linking to the location_id in the location table.
race_code: A foreign key linking to the race_code in the race table.

6. **race:**

   race_code: The code for the race.
   race_desc: The description of the race.

7. **patient:**

   person_id: A foreign key linking to the person_id in the person table.
   disease_id: A foreign key linking to the disease_id in the disease table.
   severity_value: The value indicating the severity of the patient's condition.
   start_date: The start date of the disease or condition.
   end_date: The end date of the disease or condition.

8. **location:**

   location_id: Unique identifier for the location.
   city_name: The name of the city.
   state_name: The name of the state.
   county_name: The name of the county.
   developing_flag: A flag indicating whether the location is in a developing area.
   wealth_rank_num: A numerical rank of the location's wealth.

9. **propensity:**

   race_code: A foreign key linking to the race_code in the race table.
   disease_id: A foreign key linking to the disease_id in the disease table.
   propensity_value: The value indicating the propensity of the race to a disease.

The connections between these tables are represented by the lines, which indicate the relationships based on foreign keys. For example, the disease_id in the disease table is linked to the disease_id in the indication table, meaning that the indications are related to specific diseases. Similarly, the person_id in the person table is linked to the person_id in the patient table, suggesting that the patient records are related to specific people in the person table. The above ERD shows how data related to diseases, patients, medications, and demographics are structured and interrelated in this database.

Database and Table Creation in PostgreSQL

We developed a relational database in PostgreSQL, focusing on referential integrity and relational design. The schema was crafted to optimize query performance and data accuracy, reflecting the relationships established in the ER model.

Query   Query History

```sql
-- Creation of table for storing geographical locations
CREATE TABLE location(
    location_id SERIAL PRIMARY KEY, -- Using SERIAL for auto-incrementing primary key
    city_name VARCHAR(100) NOT NULL,
    state_name VARCHAR(100),
    county_name VARCHAR(100) NOT NULL,
    developing_flag CHAR(1), -- Flag to indicate if the location is developing
    wealth_rank_num INT CHECK (wealth_rank_num >= 1) -- Check constraint for wealth rank
);

-- Creation of table for storing personal details
CREATE TABLE person(
    person_id SERIAL PRIMARY KEY, -- Using SERIAL for auto-incrementing primary key
    last_name VARCHAR(50) NOT NULL,
    first_name VARCHAR(50) NOT NULL,
    gender CHAR(1) NOT NULL CHECK (gender IN ('M', 'F', 'U')), -- Check constraint for gender
    location_id INT NOT NULL,
    race_code CHAR(5) NOT NULL,
    FOREIGN KEY(race_code) REFERENCES race(race_code)
        ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY(location_id) REFERENCES location(location_id)
        ON UPDATE CASCADE ON DELETE CASCADE
);

-- Creation of table for storing patient disease records
CREATE TABLE patient(
    person_id INT NOT NULL,
    disease_id INT NOT NULL,
    severity_value INT NOT NULL DEFAULT 1, -- Default severity
    start_date DATE NOT NULL,
    end_date DATE,
    PRIMARY KEY(person_id, disease_id), -- Composite primary key
    FOREIGN KEY(disease_id) REFERENCES disease(disease_id)
        ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY(person_id) REFERENCES person(person_id)
        ON UPDATE CASCADE ON DELETE CASCADE
);
```

Data Output   Messages   Notifications

```
CREATE TABLE

Query returned successfully in 318 msec.
```

Data Loading Process and Operational Queries on OLTP

We sourced data from credible healthcare datasets in CSV format and loaded it into our PostgreSQL database. This setup enables efficient querying and analysis, as demonstrated by the operational queries we executed:

```sql
-- Query to list all medicines and their indications for 'Tuberculosis'
SELECT
    m.med_name, -- Selecting the name of the medicine
    m.active_ing_name, -- Selecting the active ingredient of the medicine
    i.effectiveness, -- Selecting the effectiveness of the medicine for the disease
    i.indication_date -- Selecting the date when the medicine was indicated
FROM
    medicine m -- Starting from the medicine table
    JOIN indication i ON m.med_id = i.med_id -- Joining with the indication table to match medicines with their indications
WHERE
    i.disease_id = (SELECT disease_id FROM disease WHERE disease_name = 'Tuberculosis'); -- Filtering to include only those medicines indicated for Tu
```

Output Messages Notifications

med_name character varying (25)	active_ing_name character varying (150)	effectiveness numeric (5,2)	indication_date date
Dovprela	Pretomanid	90.00	1998-12-06

```sql
-- Query to summarize the number of disease incidences by location
SELECT
    l.city_name, -- Selecting the city name
    l.state_name, -- Selecting the state name
    COUNT(p.disease_id) AS disease_count -- Counting the number of diseases for each location
FROM
    patient p -- Starting from the patient table
    JOIN person per ON p.person_id = per.person_id -- Joining with the person table to get the location ID
    JOIN location l ON per.location_id = l.location_id -- Joining with the location table to get city and state names
GROUP BY
    l.city_name, l.state_name; -- Grouping the results by city and state names
```

Output Messages Notifications

city_name character varying (100)	state_name character varying (100)	disease_count bigint
Charlottesville	Virginia	1
Danbury	Connecticut	1
Shoreline	Washington	1
Waterbury	Connecticut	1
Lynchburg	Virginia	1
Maricopa	Arizona	1
Youngstown	Ohio	1
Daly City	California	1
Eagan	Minnesota	1
Coral Gables	Florida	1
Burnsville	Minnesota	1
Rancho Palos Verdes	California	1
Lafayette	Louisiana	1

```sql
-- Query to analyze the propensity of races for different diseases
SELECT
    r.race_desc, -- Selecting the race description
    d.disease_name, -- Selecting the disease name
    pr.propensity_value -- Selecting the propensity value indicating how common the disease is among the race
FROM
    propensity pr -- Starting from the propensity table
    JOIN race r ON pr.race_code = r.race_code -- Joining with the race table to get race descriptions
    JOIN disease d ON pr.disease_id = d.disease_id -- Joining with the disease table to get disease names
ORDER BY
    pr.propensity_value DESC; -- Ordering by propensity value to see which combinations have the highest propensity
```

Output Messages Notifications

race_desc character varying (100)	disease_name character varying (100)	propensity_value integer
Black or African American, not of Hispanic origin	Malaria	10
Unspecified	Dengue	10
Hispanic	Trachoma	10
Unspecified	Other intestinal infections	10
Unspecified	Lower respiratory infections	10
Asian	HIV/AIDS	10
White, not of Hispanic origin	Leprosy	10
White, not of Hispanic origin	Intestinal nematode infections	10
Hispanic	Diarrhoeal diseases	9
Unspecified	Hepatitis C (d)	9
Unspecified	Trachoma	9
Unspecified	HIV/AIDS	9
White, not of Hispanic origin	Diarrhoeal diseases	9

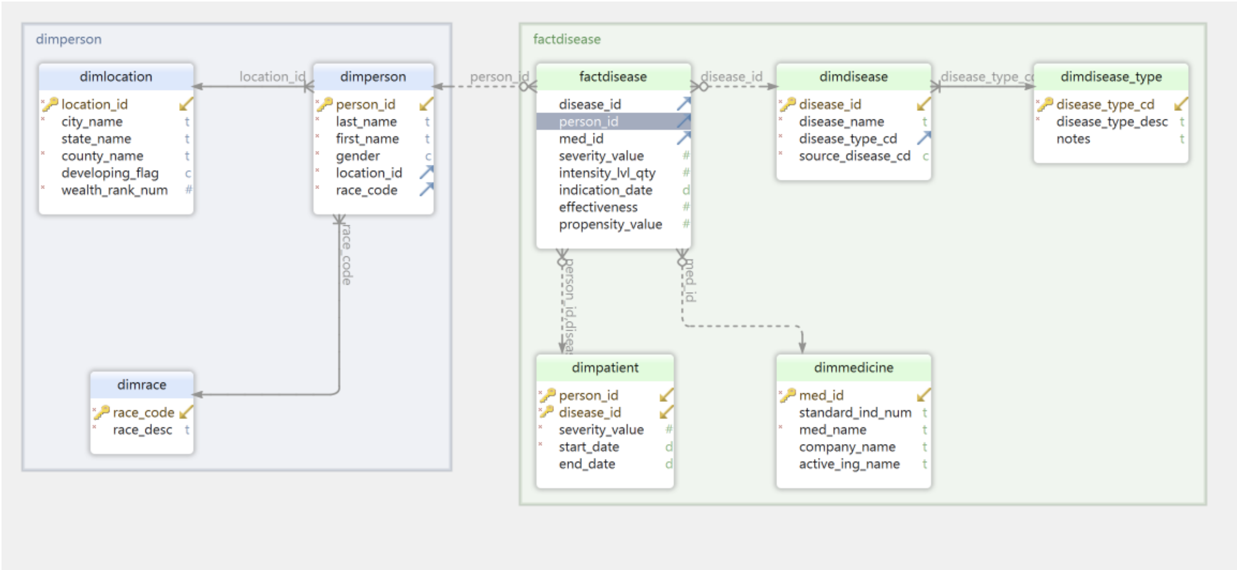
```sql
-- Query to count the number of diseases by their type
SELECT
    dt.disease_type_desc, -- Selecting the description of the disease type
    COUNT(d.disease_id) AS disease_count -- Counting the number of diseases for each type
FROM
    disease d -- Starting from the disease table
    JOIN disease_type dt ON d.disease_type_cd = dt.disease_type_cd -- Joining with the disease_type table to get type descriptions
GROUP BY
    dt.disease_type_desc; -- Grouping the results by disease type descriptions
```

Output Messages Notifications

disease_type_desc character varying (1000)	disease_count bigint
Hepatitis B (d)	1
Meningitis	1
Trachoma	1
Tuberculosis	1
Childhood-cluster diseases	1
Obstructed labour	1
STDs excluding HIV	1
Hypertensive disorders	1
Maternal haemorrhage	1
Japanese encephalitis	1
Intestinal nematode infections	1
Diarrhoeal diseases	1
HIV/AIDS	1
Dengue	1
Maternal sepsis	1

Dimensional Data Model and ELT in Data Warehouse:

In our data warehouse, we have adopted the Star Schema, comprising both fact and dimension tables. This structure enhances the efficiency of queries and analytical operations, enabling us to handle intricate analytical queries effortlessly.



The data workflow in our system follows an ETL (Extract, Transform, Load) approach. Initially, we extract data from the OLTP (Online Transaction Processing) system. Following this, the data undergoes transformation to align with the dimensional model of our data warehouse. Finally, it's loaded into the warehouse. This methodology guarantees that the data used for analytical queries is tailored for optimal analysis.

Query Query History

```
-- Creating dimension table for Person in the data warehouse
CREATE TABLE disease_dw.dimPerson(
    Person_Id INT PRIMARY KEY, -- Primary Key for Person
    Last_Name VARCHAR(50) NOT NULL, -- Last name of the person
    First_Name VARCHAR(50) NOT NULL, -- First name of the person
    Gender CHAR(1) NOT NULL, -- Gender of the person
    Location_Id INT NOT NULL, -- Foreign key to Location
    Race_Code CHAR(5) NOT NULL, -- Foreign key to Race
    FOREIGN KEY(Race_Code) REFERENCES disease_dw.dimRace(Race_Code)
        ON UPDATE CASCADE ON DELETE CASCADE, -- Cascade updates/deletes to Race
    FOREIGN KEY(Location_Id) REFERENCES disease_dw.dimLocation(Location_Id)
        ON UPDATE CASCADE ON DELETE CASCADE -- Cascade updates/deletes to Location
);

-- Populating the dimPerson table from the existing person table
INSERT INTO disease_dw.dimPerson
SELECT Person_Id, Last_Name, First_Name, Gender, Location_Id, Race_Code FROM person;

```

```
-- Creating dimension table for Medicine in the data warehouse
CREATE TABLE disease_dw.dimMedicine(
    Med_Id INT PRIMARY KEY, -- Primary Key for Medicine
    Standard_Ind_Num VARCHAR(250), -- Standard Industry Number
    Med_Name VARCHAR(25) NOT NULL, -- Name of the Medicine
    Company_Name VARCHAR(150), -- Name of the Company that manufactures the Medicine
    Active_Ing_Name VARCHAR(150) -- Active Ingredient in the Medicine
);

-- Populating the dimMedicine table from the existing medicine table
INSERT INTO disease_dw.dimMedicine
SELECT Med_Id, Standard_Ind_Num, Med_Name, Company_Name, Active_Ing_Name FROM medicine;


-- Drop the existing dimPatient table if it exists
DROP TABLE IF EXISTS disease_dw.dimPatient;


-- Creating dimension table for Patient in the data warehouse
```

Servers (1)
- PostgreSQL 15
  - Databases (14)
    - Assignment3
    - Disease
      - Casts
      - Catalogs
      - Event Triggers
      - Extensions
      - Foreign Data Wrappers
      - Languages
      - Publications
      - Schemas (1)
        - public
          - Aggregates
          - Collations
          - Domains
          - FTS Configurations
          - FTS Dictionaries
          - FTS Parsers
          - FTS Templates
          - Foreign Tables
          - Functions
          - Materialized Views
          - Operators
          - Procedures
          - Sequences
          - Tables (9)
          - Trigger Functions
          - Types
          - Views
      - Subscriptions
    - GoTData
    - aboutme
    - afceast
    - books
    - college_grouping
    - flights
    - library
    - mydb
    - postgres

Disease/postgres@PostgreSQL 15

No limit

Query | Query History

```sql
    a.Severity_Value AS Severity_Value,
    b.Intensity_Lvl_Qty AS Intensity_Lvl_Qty,
    c.Indication_Date AS Indication_Date,
    c.Effectiveness AS Effectiveness,
    d.Propensity_Value AS Propensity_Value
FROM
    patient a
    INNER JOIN disease b ON a.Disease_Id = b.Disease_Id
    INNER JOIN indication c ON a.Disease_Id = c.Disease_Id
    INNER JOIN medicine e ON c.Med_Id = e.Med_Id
    INNER JOIN propensity d ON a.Disease_Id = d.Disease_Id;

-- Sample queries to view data from the dimensional model and fact table
SELECT * FROM disease_dw.dimDisease;
SELECT * FROM disease_dw.factDisease;
```

Data Output | Messages | Notifications

	disease_id [PK] integer	disease_name character varying (100)	disease_type_cd character	source_disease_cd character
1	1	Tuberculosis	U003	TB
2	2	STDs excluding HIV	U004	STD
3	3	HIV/AIDS	U009	HIV
4	4	Diarrhoeal diseases	U010	DHD
5	5	Meningitis	U011	MGTS
6	6	Hepatitis B (d)	U017	HPB
7	7	Hepatitis C (d)	U018	HPC
8	8	Malaria	U029	MLR
9	9	Leprosy	U030	LP
10	10	Dengue	U031	DG
11	11	Trachoma	U032	TRC
12	12	Intestinal nematode infections	U043	INI
13	13	Other intestinal infections	U044	OII
14	14	Lower respiratory infections	U045	LRI
15	15	Upper respiratory infections	U046	URI


```sql
    a.Severity_Value AS Severity_Value,
    b.Intensity_Lvl_Qty AS Intensity_Lvl_Qty,
    c.Indication_Date AS Indication_Date,
    c.Effectiveness AS Effectiveness,
    d.Propensity_Value AS Propensity_Value
FROM
    patient a
    INNER JOIN disease b ON a.Disease_Id = b.Disease_Id
    INNER JOIN indication c ON a.Disease_Id = c.Disease_Id
    INNER JOIN medicine e ON c.Med_Id = e.Med_Id
    INNER JOIN propensity d ON a.Disease_Id = d.Disease_Id;

-- Sample queries to view data from the dimensional model and fact table
SELECT * FROM disease_dw.dimDisease;
SELECT * FROM disease_dw.factDisease;
```


Data Output | Messages | Notifications

	disease_id integer	person_id integer	med_id integer	severity_value integer	intensity_lvl_qty integer	indication_date date	effectiveness double precision	propensity_value integer
1	1	61	3	5	6	1998-12-06	90	7
2	1	61	3	5	6	1998-12-06	90	4
3	1	61	3	5	6	1998-12-06	90	8
4	1	61	3	5	6	1998-12-06	90	5
5	1	61	3	5	6	1998-12-06	90	2
6	1	46	3	1	6	1998-12-06	90	7
7	1	46	3	1	6	1998-12-06	90	4
8	1	46	3	1	6	1998-12-06	90	8
9	1	46	3	1	6	1998-12-06	90	5
10	1	46	3	1	6	1998-12-06	90	2
11	1	31	3	1	6	1998-12-06	90	7
12	1	31	3	1	6	1998-12-06	90	4
13	1	31	3	1	6	1998-12-06	90	8
14	1	31	3	1	6	1998-12-06	90	5
15	1	31	3	1	6	1998-12-06	90	2
16	1	16	3	1	6	1998-12-06	90	7
17	1	16	3	1	6	1998-12-06	90	4
18	1	16	3	1	6	1998-12-06	90	8
19	1	16	3	1	6	1998-12-06	90	5
20	1	16	3	1	6	1998-12-06	90	2

Analytical Queries:

Our analytical queries delve into diverse facets of disease data. For instance, one such query focuses on examining the occurrence of diseases over time, uncovering trends and patterns. The valuable insights derived from these queries are crucial for shaping healthcare policies and strategies.

1. ***Trend Analysis of Disease Incidence Over Time:***
   Our analysis involves tracking the annual trends of various diseases, based on the recorded number of cases.

```
-- Analytical Queries

-- Trend Analysis of Disease Incidence Over Time:
-- This query analyzes the yearly trend of different diseases based on the number of cases.

SELECT d.disease_name, COUNT(pt.disease_id) AS incidence_count, EXTRACT(YEAR FROM pt.start_date) AS year
FROM patient pt
JOIN disease d ON pt.disease_id = d.disease_id
GROUP BY d.disease_name, EXTRACT(YEAR FROM pt.start_date)
ORDER BY year, incidence_count DESC;
```

Data Output Messages Notifications

	disease_name character varying (100)	incidence_count bigint	year numeric
1	Tuberculosis	5	2023
2	STDs excluding HIV	5	2023
3	Hepatitis C (d)	4	2023
4	Leprosy	4	2023
5	Meningitis	4	2023
6	Lower respiratory infections	4	2023
7	HIV/AIDS	4	2023
8	Trachoma	4	2023
9	Hepatitis B (d)	4	2023
10	Intestinal nematode infections	4	2023
11	Malaria	4	2023
12	Other intestinal infections	4	2023
13	Upper respiratory infections	4	2023
14	Diarrhoeal diseases	4	2023
15	Dengue	4	2023

2. ***Geographical Distribution of Specific Diseases***

   We implemented analytical queries to identify the distribution of diseases across cities and states in the United States.

Query Query History

```sql
-- Geographical Distribution of Specific Diseases:
-- This query identifies the distribution of diseases across cities and states.

SELECT l.city_name, l.state_name, d.disease_name, COUNT(p.disease_id) AS cases
FROM patient p
JOIN person pr ON p.person_id = pr.person_id
JOIN location l ON pr.location_id = l.location_id
JOIN disease d ON p.disease_id = d.disease_id
GROUP BY l.city_name, l.state_name, d.disease_name
ORDER BY cases DESC;
```

Data Output Messages Notifications

	city_name character varying (100)	state_name character varying (100)	disease_name character varying (100)	cases bigint
1	Elkhart	Indiana	Hepatitis C (d)	1
2	Santa Rosa	California	Tuberculosis	1
3	Rancho Santa Margarita	California	Intestinal nematode infections	1
4	Worcester	Massachusetts	Leprosy	1
5	Cleveland	Ohio	Malaria	1
6	Nampa	Idaho	Diarrhoeal diseases	1
7	Perris	California	Hepatitis C (d)	1
8	Fargo	North Dakota	Leprosy	1
9	Montgomery	Alabama	Tuberculosis	1
10	Roswell	New Mexico	Upper respiratory infections	1
11	Charlottesville	Virginia	Lower respiratory infections	1
12	Hayward	California	Dengue	1
13	Burnsville	Minnesota	Other intestinal infections	1
14	Omaha	Nebraska	Trachoma	1
15	Waterbury	Connecticut	STDs excluding HIV	1
16	Marietta	Georgia	Leprosy	1
17	Daly City	California	Leprosy	1
18	Taylor	Michigan	Dengue	1
19	Fort Wayne	Indiana	Diarrhoeal diseases	1
20	Woburn	Massachusetts	HIV/AIDS	1
21	Palm Springs	California	Lower respiratory infections	1
22	Shoreline	Washington	Meningitis	1

3. *Medication Effectiveness Analysis*

This analytical query evaluates the effectiveness of medications for different diseases.

```sql
-- Medication Effectiveness Analysis:
-- This query evaluates the effectiveness of medications for different diseases.

SELECT d.disease_name, m.med_name, AVG(i.effectiveness) AS average_effectiveness
FROM indication i
JOIN disease d ON i.disease_id = d.disease_id
JOIN medicine m ON i.med_id = m.med_id
GROUP BY d.disease_name, m.med_name
ORDER BY d.disease_name, average_effectiveness DESC;
```

Data Output Messages Notifications

	disease_name character varying (100)	med_name character varying (25)	average_effectiveness numeric
1	Dengue	Dengvaxia	94.0000000000000000
2	Diarrhoeal diseases	Truberzi	93.0000000000000000
3	Hepatitis B (d)	Procomvax	88.0000000000000000
4	Hepatitis C (d)	Hexyon	85.0000000000000000
5	HIV/AIDS	Biktarvy	70.0000000000000000
6	Intestinal nematode infections	Ayvakyt	87.0000000000000000
7	Leprosy	Synagis	75.0000000000000000
8	Lower respiratory infections	Esbriet	97.0000000000000000
9	Malaria	Artesunate Amivas	93.0000000000000000
10	Meningitis	Trumenba	85.0000000000000000
11	Other intestinal infections	Sutent	89.0000000000000000
12	STDs excluding HIV	penicillin g benzathine	89.0000000000000000
13	Trachoma	Evotaz	83.0000000000000000
14	Tuberculosis	Dovprela	90.0000000000000000
15	Upper respiratory infections	Cayston	90.0000000000000000

4. *Correlation between Race and Disease Propensity.*

We tried to examine if certain races have higher propensities for specific diseases.

```sql
--Correlation Between Race and Disease Propensity:
-- This query examines if certain races have higher propensities for specific diseases.

SELECT r.race_desc, d.disease_name, AVG(p.propensity_value) AS average_propensity
FROM propensity p
JOIN race r ON p.race_code = r.race_code
JOIN disease d ON p.disease_id = d.disease_id
GROUP BY r.race_desc, d.disease_name
ORDER BY average_propensity DESC;
```

Data Output Messages Notifications

	race_desc character varying (100)	disease_name character varying (100)	average_propensity numeric
1	White, not of Hispanic origin	Intestinal nematode infections	10.0000000000000000
2	Black or African American, not of Hispanic origin	Malaria	10.0000000000000000
3	Unspecified	Lower respiratory infections	10.0000000000000000
4	Unspecified	Dengue	10.0000000000000000
5	Asian	HIV/AIDS	10.0000000000000000
6	Unspecified	Other intestinal infections	10.0000000000000000
7	Hispanic	Trachoma	10.0000000000000000
8	White, not of Hispanic origin	Leprosy	10.0000000000000000
9	Unspecified	HIV/AIDS	9.0000000000000000
10	White, not of Hispanic origin	Diarrhoeal diseases	9.0000000000000000
11	Unspecified	Hepatitis C (d)	9.0000000000000000
12	Hispanic	Diarrhoeal diseases	9.0000000000000000
13	Unspecified	Trachoma	9.0000000000000000
14	Black or African American, not of Hispanic origin	Meningitis	8.0000000000000000
15	Black or African American, not of Hispanic origin	Other intestinal infections	8.0000000000000000
16	Black or African American, not of Hispanic origin	Diarrhoeal diseases	8.0000000000000000
17	Black or African American, not of Hispanic origin	STDs excluding HIV	8.0000000000000000
18	Black or African American, not of Hispanic origin	Tuberculosis	8.0000000000000000
19	Hispanic	Malaria	8.0000000000000000
20	White, not of Hispanic origin	Malaria	8.0000000000000000
21	Unspecified	Tuberculosis	7.0000000000000000
22	Asian	Malaria	7.0000000000000000

5. ***Patient Demographics for Specific Disease***

   The below analytical query provides demographical insights such as gender and race for patients with specific diseases.

```sql
-- Patient Demographics for Specific Diseases:
-- This query provides demographic insights (gender, race) for patients with specific diseases.
SELECT d.disease_name, pr.gender, r.race_desc, COUNT(*) AS patient_count
FROM patient p
JOIN person pr ON p.person_id = pr.person_id
JOIN race r ON pr.race_code = r.race_code
JOIN disease d ON p.disease_id = d.disease_id
GROUP BY d.disease_name, pr.gender, r.race_desc;
```

Data Output Messages Notifications

	disease_name character varying (100)	gender character	race_desc character varying (100)	patient_count bigint
1	HIV/AIDS	M	White, not of Hispanic origin	1
2	Lower respiratory infections	F	Black or African American, not of Hispanic origin	1
3	STDs excluding HIV	F	Unspecified	1
4	Malaria	F	Black or African American, not of Hispanic origin	1
5	STDs excluding HIV	F	Black or African American, not of Hispanic origin	1
6	Dengue	M	Asian	1
7	Other intestinal infections	F	Hispanic	1
8	Dengue	F	Unspecified	1
9	Leprosy	M	Hispanic	1
10	Leprosy	M	Black or African American, not of Hispanic origin	1
11	Trachoma	M	Black or African American, not of Hispanic origin	2
12	Tuberculosis	M	White, not of Hispanic origin	1
13	Dengue	M	Black or African American, not of Hispanic origin	1
14	Meningitis	M	White, not of Hispanic origin	1
15	Lower respiratory infections	F	Hispanic	1
16	Malaria	M	White, not of Hispanic origin	1
17	STDs excluding HIV	F	White, not of Hispanic origin	1
18	Hepatitis C (d)	F	White, not of Hispanic origin	1
19	Intestinal nematode infections	M	Black or African American, not of Hispanic origin	1
20	Hepatitis C (d)	M	Black or African American, not of Hispanic origin	2
21	Hepatitis B (d)	M	Black or African American, not of Hispanic origin	2
22	Diarrhoeal diseases	M	Black or African American, not of Hispanic origin	2

NoSQL Database Structure for Disease Model

MongoDB (Document-Oriented NoSQL Database):

In MongoDB, instead of tables, we have collections of documents. Each document can have a different structure. A patient document in MongoDB can look like the following:

```
{
  "patient_id": "P123",
  "name": "John Doe",
  "age": 45,
  "diseases": [
    {
      "disease_id": "D456",
      "name": "Diabetes",
      "treatments": [
        {
          "treatment_id": "T789",
          "name": "Insulin Therapy",
          "effectiveness": 85
        }
      ]
    }
  ]
}
```

- For instance, a Patient document could include embedded documents for Diseases and Treatments, allowing for a more holistic view of the patient's health record in a single document. This structure allows adding new diseases or treatments to a patient's record without altering the schema for other patients.

- This model is flexible; as new diseases or treatment modalities are introduced, they can easily be added to the patient's record without altering the structure of other documents or collections.

Advantages of MongoDB (Document-Oriented NoSQL Database):

Documents and Collections: Each document can have a varied structure, containing all the data related to a single entity, which allows embedding arrays and other documents directly.
Denormalization: Related data might be stored together in the same document for quicker read access, reducing the need for joins
.**Schema-less Design**: The schema can evolve over time without requiring migrations, making it easier to adapt to changing data needs.

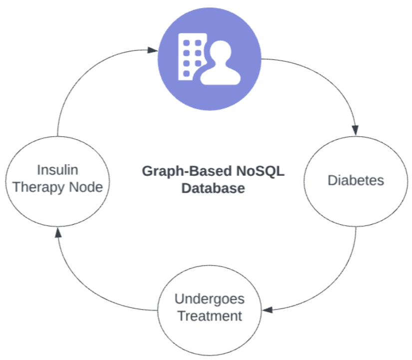
Neo4J (Graph-Based NoSQL Database):

In Neo4J, data is stored as nodes and edges. Nodes represent entities and edges represent relationships. For the disease model, patients, diseases, and treatments would be nodes. Relationships like "HAS_DISEASE" or "UNDERGOES_TREATMENT" would be edges connecting these nodes. This structure is particularly powerful for querying complex relationships, like tracing the spread of a disease through a network of patients or analyzing the efficacy of treatments across various demographics.

Patient Node: Labeled as "Patient" with properties such as name: "John Doe".
Disease Node: Labeled as "Disease" with properties like name: "Diabetes".
Treatment Node: Labeled as "Treatment" with properties such as name: "Insulin Therapy".

Relationships:
HAS_DISEASE Relationship connects "John Doe" (Patient Node) to "Diabetes" (Disease Node). This relationship indicates that John Doe has been diagnosed with Diabetes.
UNDERGOES_TREATMENT Relationship connects "Diabetes" (Disease Node) to "Insulin Therapy" (Treatment Node). This relationship shows that the standard treatment for Diabetes in this case is Insulin Therapy.

The overall structure forms a chain from the patient, through their disease, to their treatment, clearly and intuitively representing the relationships between these entities.



Neo4J (Graph-Based NoSQL Database):

Nodes and Relationships: Data is structured as nodes (entities) and edges (relationships), which is ideal for datasets where relationships are as important as the data itself.
Property Graph: Both nodes and relationships can have properties, allowing them to store data.
Schema-free: There's no fixed schema, enabling you to add different properties to nodes and relationships as needed.

AWS Architecture for Disease Model

Components of AWS Architecture

1. Amazon EC2: Host application in the cloud.

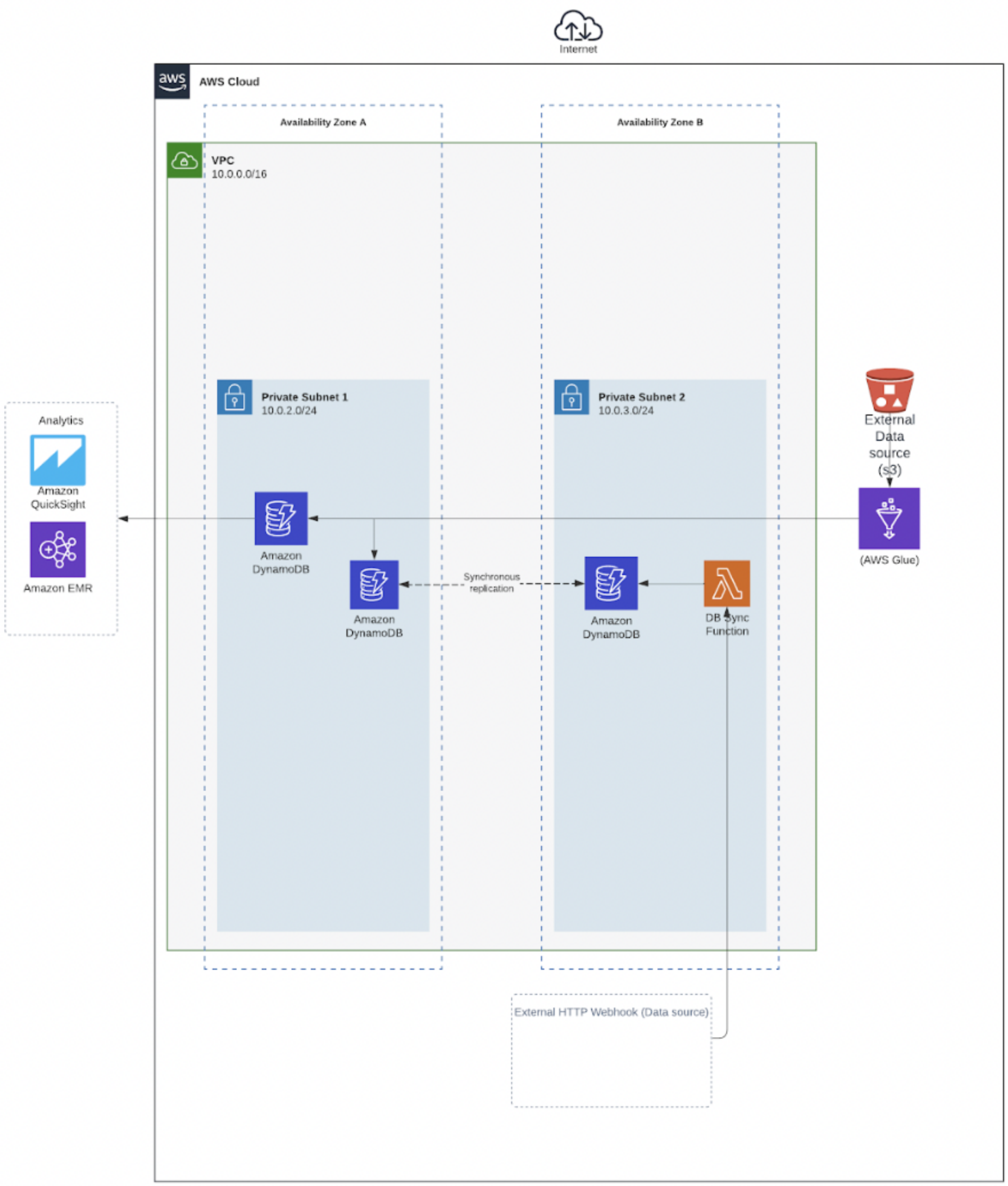
2. Amazon S3: Store backups, logs, and other files.

3. AWS Glue: For ETL process

4. AWS DynamoDB: As NoSQL databases for different types of data storage needs.

5. AWS Lambda: To run code in response to triggers (such as changes in data or system states).

6. Amazon EMR on AWS: To handle data-processing jobs



Data Processing and Loading:

1. **Batch Processing (AWS Glue)**: Regularly scheduled ETL jobs that process historical patient data for analysis, for example, aggregating monthly treatment success rates.

2. **Real-Time Processing (AWS Lambda and Amazon Kinesis)**: Real-time monitoring of patient vitals streamed through Kinesis, with Lambda functions triggering alerts for abnormal readings.

3. **Ensuring Resilience, Performance, and Security:**

    a) **Resilience**: Use of Amazon RDS Multiple availability zones (AZ) deployments ensures that if one AZ goes down, the database can failover to another, ensuring continuity of critical healthcare applications.

    b) **Performance**: Utilize Amazon ElastiCache to store frequently accessed data like common disease-treatment mappings, reducing the load on the main database.

    c) **Security**: Implement strict IAM roles to control access to patient data, ensuring that only authorized personnel can view or modify sensitive information.

Scalability

Snowflake	PostgreSQL
Scenario*: Suppose our disease model database needs to rapidly scale up for a project analyzing extensive data from thousands of patients to identify genetic predispositions to certain diseases.*	***Scenario****: The same genomic data analysis in a PostgreSQL-based warehouse.*
Snowflake's Approach: In Snowflake, we can instantly scale up computational resources (warehouses) to handle this increased demand without impacting your storage costs. This means we can run complex genomic data analyses and queries concurrently without performance degradation.	**PostgreSQL's Approach**: To manage this, we might need to provision additional hardware or implement sharding (distributing the database load across multiple machines or partitions), which can be complex and time-consuming.

\

Maintenance and Performance

Snowflake	PostgreSQL
***Scenario**: Running a complex query to correlate genetic markers with the efficacy of cancer treatments.*	***Scenario**: The same complex query in a PostgreSQL environment.*
Snowflake's Approach: Snowflake automatically optimizes query performance. Its architecture separates compute and storage, enabling efficient query execution without manual tuning.	**PostgreSQL's Approach**: This would likely require manual performance tuning, such as creating and maintaining indexes, which can be labor-intensive and require deep database expertise.
Benefit: This is particularly advantageous for dynamically changing query patterns often seen in medical research.	**Challenge**: For complex and evolving queries, this manual tuning can become a bottleneck in data analysis.

Data Sharing and Security

Snowflake	PostgreSQL
***Scenario**: Sharing anonymized patient treatment outcomes with external research institutions for collaborative studies.*	***Scenario**: Attempting the same data sharing using PostgreSQL.*
Snowflake's Approach: Snowflake allows us to securely share a portion of our data warehouse, like specific datasets, without replicating the data. We can grant access to external researchers to these datasets with fine-grained control.	**PostgreSQL's Approach**: Typically, this involves creating database replicas or exporting data, which can be complex to manage and may introduce security vulnerabilities.
Benefit: This facilitates secure and efficient collaboration without increasing the risk of data breaches or duplicating data.	**Challenge**: This method increases the complexity of data management and can pose challenges in ensuring data security and compliance, especially crucial in healthcare data handling.

In summary, while PostgreSQL is a powerful and widely-used relational database system, Snowflake offers specific advantages in a data warehousing context, particularly for large-scale, complex, and collaborative disease model analyses. Snowflake's architecture provides scalability, ease of maintenance, and secure data sharing capabilities, which are highly beneficial for dynamic and data-intensive fields like healthcare research and genomics.

Conclusion:

In conclusion, our project successfully designed and implemented a comprehensive database system focused on managing and analyzing disease-related data. By integrating principles of data modeling, database design, and analytical techniques, we developed a robust tool capable of providing critical insights into disease patterns, treatment effectiveness, and demographic distributions.

Utilizing a relational database setup in PostgreSQL allowed us to maintain referential integrity and optimize query performance. Our structured approach, from the conceptual ER model to the physical data dictionary, ensured a clear understanding of the database's organization, enabling efficient data handling and accurate insights.

The implementation of ETL processes facilitated a seamless flow of data from the OLTP system to our data warehouse, adhering to a dimensional model that supports complex analytical queries. This structured data workflow proved to be essential for optimizing the data for analysis, allowing for sophisticated trend analysis and demographic studies.

Our exploration into NoSQL databases like MongoDB and Neo4J demonstrated the flexibility and scalability of schema-less and graph-based models, respectively. These systems provided alternative methods for managing and querying data, showing promise for handling complex, unstructured datasets.

Incorporating AWS cloud services into our architecture offered robust, secure, and real-time data processing capabilities. Services like AWS Glue, Lambda, Kinesis, RDS, and ElastiCache contributed to a highly resilient, performant, and secure infrastructure.

Furthermore, the comparison between Snowflake and PostgreSQL highlighted the distinct advantages of Snowflake in terms of scalability, maintenance, and data sharing, particularly in the context of large-scale, collaborative research efforts.

Overall, our project stands as a testament to the power of modern database technologies in transforming healthcare data into actionable insights, thus supporting the advancement of healthcare decisions and strategies. The methodologies and technologies we've employed demonstrate the potential to significantly impact the healthcare industry by enhancing the ability to conduct in-depth analyses of disease trends and treatment outcomes.