University of Chittagong

Department of Computer Science & Engineering

Database Systems Lab

Name of the assignment:

Database Modeling: Mini Project 1

CSE 414

Assignment 05

Submitted By:
Debashish Chakraborty
ID: 23701034

Submitted To:
Dr. Rudra Pratap Deb Nath
Associate Professor

June 22, 2025

Contents

1 Introduction **3**

2 ER Diagram **3**

3 Relational Model **4**

4 Database Implementation **4**
4.1 Creating the Database Schema: 4
4.2 Creating data table: 5
4.3 Importing Data from CSV into the created data table: 5

5 Create the corresponding tables: **6**

6 Inserting Data into the Database **8**

7 Interacting with the Database **11**
7.1 Question 01 11
7.2 Question 02 11
7.3 Question 03 11
7.4 Question 04 11
7.5 Question 05 12
7.6 Question 06 12
7.7 Question 07 12
7.8 Question 08 13
7.9 Question 09 13
7.10 Question 10 14
7.11 Question 11 14
7.12 Question 12 15

8 What are the differences to your initial design? **15**
8.1 Database Management System Choice 15
8.2 Data Import Strategy 15
8.3 Handling Multiple Values in Single Fields 16
8.4 Clinical Trial Entity Complexity 16
8.5 Data Processing Approach 16

9 Why did these differences occur? **16**
9.1 Real Data Structure Complexity 16
9.2 PostgreSQL Capabilities 16
9.3 Data Quality Challenges 17
9.4 Normalization Requirements 17

10 What advantages does the new design have over the old one? **17**
10.1 Elegant Data Processing 17
10.2 Robust Data Integrity 17
10.3 Query Performance and Flexibility 18
10.4 Scalable Architecture 18

10.5 Professional Database Practices . . . 18

11 Key Insights from Complex Query Implementation 18

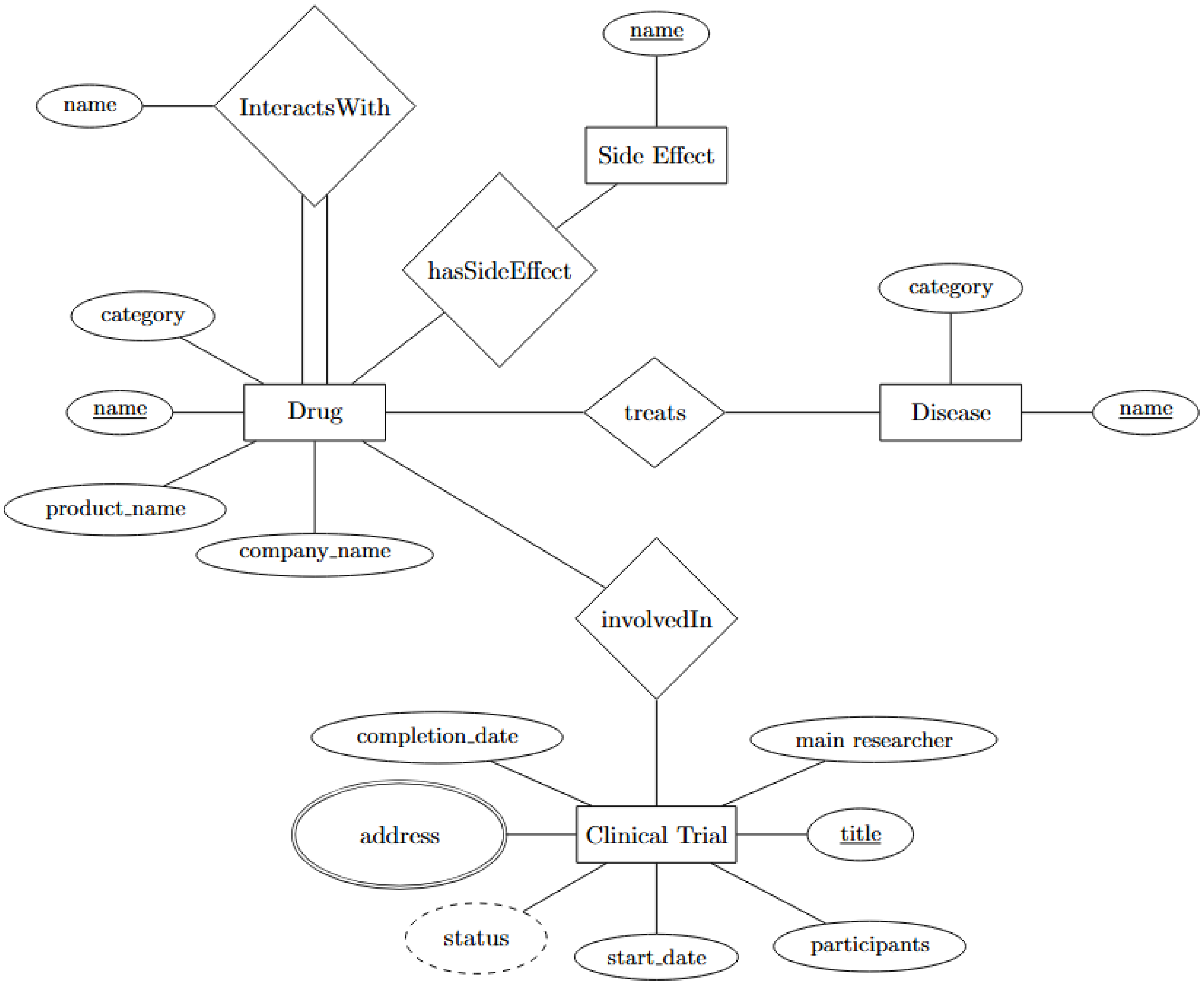
11.1 Complex Analytical Capabilities . . . 18

11.2 Time-Based Analysis Support . . . 19

Introduction

The requirements of this mini-project is to create and work with a medical database. The first step is to identify the information that is going to be represented in a database model. The information is modeled in an ER diagram and mapped to a relational schema. The database should store information about drugs, their categories, the diseases they can treat, their side effects, their commercial products, their interactions with other drugs, and possibly information about related clinical trials.
Consider the following sample information available in the given *Excel file*.

ER Diagram



Relational Model

This model translates the ER Diagram into a relational schema, specifying tables, attributes, primary keys, foreign keys, and data types with standard professional notation.

1. **Drug**(drug_name, drug_category, product_name, company_name)
2. **SideEffect**(name)
3. **Drug_hasSideEffect**(drug_name, side_effect_name)
   - Foreign key: drug_name $\longrightarrow$ Drug
   - Foreign key: side_effect_name $\longrightarrow$ SideEffect
4. **Interaction**(name)
5. **Drug_Interaction**(drug_name, interaction_name)
   - Foreign key: drug_name $\longrightarrow$ Drug
   - Foreign key: interaction_name $\longrightarrow$ Interaction
6. **Disease**(disease_name, disease_category)
7. **Treats**(drug_name, disease_name)
   - Foreign key: drug_name $\longrightarrow$ Drug
   - Foreign key: disease_name $\longrightarrow$ Disease
8. **ClinicalTrial**(clinical_trial_title, clinical_trial_start_date, clinical_trial_completion_date, clinical_trial_participants, clinical_trial_status, clinical_trial_address, clinical_trial_institution, clinical_trial_address_1, clinical_trial_main_researcher)
9. **Drug_ClinicalTrial**(drug_name, clinical_trial_title)
   - Foreign key: drug_name $\longrightarrow$ Drug
   - Foreign key: clinical_trial_title $\longrightarrow$ ClinicalTrial

Database Implementation

In this assignment, we are given a sample of medical drug related data. We will use postgreSQL here. Consider the following sample information available in the given *Excel file*.

Creating the Database Schema:

At first, we have to create a database named DrugSchema.

```sql
create database DrugSchema;
```

Creating data table:

```sql
CREATE TABLE data (
    drug_name TEXT,
    side_effect TEXT,
    side_effect_1 TEXT,
    side_effect_2 TEXT,
    side_effect_3 TEXT,
    side_effect_4 TEXT,
    interacts_with TEXT,
    interacts_with_1 TEXT,
    interacts_with_2 TEXT,
    disease_name TEXT,
    disease_category TEXT,
    drug_category TEXT,
    product_name TEXT,
    company_name TEXT,
    clinical_trial_title TEXT,
    clinical_trial_start_date TEXT,
    clinical_trial_completion_date TEXT,
    clinical_trial_participants NUMERIC,
    clinical_trial_status TEXT,
    clinical_trial_condition TEXT,
    clinical_trial_condition_1 TEXT,
    clinical_trial_address TEXT,
    clinical_trial_institution TEXT,
    clinical_trial_address_1 TEXT,
    clinical_trial_main_researcher TEXT,
    clinical_trial_condition_2 TEXT
);
```

Importing Data from CSV into the created data table:

Let's convert the provided Tutorial1_data.xlsx file into a CSV format that can be used to populate the database tables. The CSV file's columns will be named according to the tables they correspond to.

Listing 1: Run in psql terminal

```sql
COPY data
FROM 'C:\Users\user\Downloads\Telegram
   Desktop\Database\Tutorial_1\Tutorial1_data.csv'
DELIMITER ',' CSV HEADER;
```

Create the corresponding tables:

```sql
-- Table for Drug Entity
CREATE TABLE Drug (
    drug_name VARCHAR(255) PRIMARY KEY,
    drug_category VARCHAR(100),
    product_name VARCHAR(255),
    company_name VARCHAR(255)
);

-- Table for SideEffect Entity
CREATE TABLE SideEffect (
    name VARCHAR(255) PRIMARY KEY
);

-- Table for Interaction Entity
CREATE TABLE Interaction (
    name VARCHAR(255) PRIMARY KEY
);

-- Table for Disease Entity
CREATE TABLE Disease (
    disease_name VARCHAR(255) PRIMARY KEY,
    disease_category VARCHAR(100)
);

-- Table for ClinicalTrial Entity (dates as TEXT now)
CREATE TABLE ClinicalTrial (
    clinical_trial_title VARCHAR(500) PRIMARY KEY,
    clinical_trial_start_date TEXT,
    clinical_trial_completion_date TEXT,
    clinical_trial_participants NUMERIC,
    clinical_trial_status VARCHAR(50),
    clinical_trial_address VARCHAR(500),
    clinical_trial_institution VARCHAR(255),
    clinical_trial_address_1 VARCHAR(500),
    clinical_trial_main_researcher VARCHAR(255)
);

-- Table for ClinicalTrialCondition Entity
CREATE TABLE ClinicalTrialCondition (
    name VARCHAR(255) PRIMARY KEY
);

-- Junction Table for Drug-SideEffect relationship
CREATE TABLE Drug_SideEffect (
    drug_name VARCHAR(255) REFERENCES Drug(drug_name) ON DELETE
       CASCADE,
    side_effect_name VARCHAR(255) REFERENCES SideEffect(name) ON
       DELETE CASCADE,
    PRIMARY KEY (drug_name, side_effect_name)
);

-- Junction Table for Drug-Interaction relationship
CREATE TABLE Drug_Interaction (
    drug_name VARCHAR(255) REFERENCES Drug(drug_name) ON DELETE
       CASCADE,
    interaction_name VARCHAR(255) REFERENCES Interaction(name)
       ON DELETE CASCADE,
    PRIMARY KEY (drug_name, interaction_name)
);

-- Junction Table for Drug-Disease relationship
CREATE TABLE Drug_Disease (
    drug_name VARCHAR(255) REFERENCES Drug(drug_name) ON DELETE
       CASCADE,
    disease_name VARCHAR(255) REFERENCES Disease(disease_name)
       ON DELETE CASCADE,
    PRIMARY KEY (drug_name, disease_name)
);

-- Junction Table for Drug-ClinicalTrial relationship
CREATE TABLE Drug_ClinicalTrial (
    drug_name VARCHAR(255) REFERENCES Drug(drug_name) ON DELETE
       CASCADE,
    clinical_trial_title VARCHAR(500) REFERENCES
       ClinicalTrial(clinical_trial_title) ON DELETE CASCADE,
    PRIMARY KEY (drug_name, clinical_trial_title)
);

-- Junction Table for ClinicalTrial-Condition relationship
CREATE TABLE ClinicalTrial_Condition (
    clinical_trial_title VARCHAR(500) REFERENCES
       ClinicalTrial(clinical_trial_title) ON DELETE CASCADE,
    condition_name VARCHAR(255) REFERENCES
       ClinicalTrialCondition(name) ON DELETE CASCADE,
    PRIMARY KEY (clinical_trial_title, condition_name)
);
```

Inserting Data into the Database

Now that we have the data table with all the column from the csv file, we can insert the data into the corresponding tables in our DrugSchema database. Below is the SQL code for every insert statement to populate all the tables.

```sql
--Insert into Drug
INSERT INTO Drug (drug_name, drug_category, product_name,
   company_name)
SELECT DISTINCT drug_name, drug_category, product_name,
   company_name
FROM data
WHERE drug_name IS NOT NULL
ON CONFLICT (drug_name) DO NOTHING;


-- Insert into SideEffect
INSERT INTO SideEffect (name)
SELECT DISTINCT TRIM(val) AS name
FROM data,
LATERAL UNNEST(ARRAY[
    side_effect, side_effect_1, side_effect_2, side_effect_3,
       side_effect_4
]) AS val
WHERE val IS NOT NULL AND TRIM(val) <> ''
ON CONFLICT (name) DO NOTHING;


Insert into Interaction
INSERT INTO Interaction (name)
SELECT DISTINCT TRIM(val) AS name
FROM data,
LATERAL UNNEST(ARRAY[
    interacts_with, interacts_with_1, interacts_with_2
]) AS val
WHERE val IS NOT NULL AND TRIM(val) <> ''
ON CONFLICT (name) DO NOTHING;


-- Insert into Disease
INSERT INTO Disease (disease_name, disease_category)
SELECT DISTINCT disease_name, disease_category
FROM data
WHERE disease_name IS NOT NULL
ON CONFLICT (disease_name) DO NOTHING;


-- Insert into ClinicalTrial
INSERT INTO ClinicalTrial (
    clinical_trial_title,
    clinical_trial_start_date,
    clinical_trial_completion_date,
    clinical_trial_participants,
    clinical_trial_status,
    clinical_trial_address,
    clinical_trial_institution,
    clinical_trial_address_1,
    clinical_trial_main_researcher
)
SELECT DISTINCT
    clinical_trial_title,
    clinical_trial_start_date,
    clinical_trial_completion_date,
    clinical_trial_participants,
    clinical_trial_status,
    clinical_trial_address,
    clinical_trial_institution,
    clinical_trial_address_1,
    clinical_trial_main_researcher
FROM data
WHERE clinical_trial_title IS NOT NULL
ON CONFLICT (clinical_trial_title) DO NOTHING;


-- Insert into ClinicalTrialCondition
INSERT INTO ClinicalTrialCondition (name)
SELECT DISTINCT TRIM(val) AS name
FROM data,
LATERAL UNNEST(ARRAY[
    clinical_trial_condition, clinical_trial_condition_1,
       clinical_trial_condition_2
]) AS val
WHERE val IS NOT NULL AND TRIM(val) <> ''
ON CONFLICT (name) DO NOTHING;


-- Insert into Drug_SideEffect
INSERT INTO Drug_SideEffect (drug_name, side_effect_name)
SELECT DISTINCT drug_name, TRIM(val)
FROM data,
LATERAL UNNEST(ARRAY[
    side_effect, side_effect_1, side_effect_2, side_effect_3,
       side_effect_4
]) AS val
WHERE drug_name IS NOT NULL AND val IS NOT NULL AND TRIM(val) <>
   ''
ON CONFLICT (drug_name, side_effect_name) DO NOTHING;


-- Insert into Drug_Interaction
INSERT INTO Drug_Interaction (drug_name, interaction_name)
SELECT DISTINCT drug_name, TRIM(val)
```

```sql
FROM data,
LATERAL UNNEST(ARRAY[
    interacts_with, interacts_with_1, interacts_with_2
]) AS val
WHERE drug_name IS NOT NULL AND val IS NOT NULL AND TRIM(val) <>
    ''
ON CONFLICT (drug_name, interaction_name) DO NOTHING;


-- Insert into Drug_Disease
INSERT INTO Drug_Disease (drug_name, disease_name)
SELECT DISTINCT drug_name, disease_name
FROM data
WHERE drug_name IS NOT NULL AND disease_name IS NOT NULL
ON CONFLICT (drug_name, disease_name) DO NOTHING;


-- Insert into Drug_ClinicalTrial
INSERT INTO Drug_ClinicalTrial (drug_name, clinical_trial_title)
SELECT DISTINCT drug_name, clinical_trial_title
FROM data
WHERE drug_name IS NOT NULL AND clinical_trial_title IS NOT NULL
ON CONFLICT (drug_name, clinical_trial_title) DO NOTHING;


-- Insert into ClinicalTrial_Condition
INSERT INTO ClinicalTrial_Condition (clinical_trial_title,
    condition_name)
SELECT DISTINCT clinical_trial_title, TRIM(val)
FROM data,
LATERAL UNNEST(ARRAY[
    clinical_trial_condition, clinical_trial_condition_1,
        clinical_trial_condition_2
]) AS val
WHERE clinical_trial_title IS NOT NULL AND val IS NOT NULL AND
    TRIM(val) <> ''
ON CONFLICT (clinical_trial_title, condition_name) DO NOTHING;
```

Interacting with the Database

Answering the questions provided in the assignment requires querying the database we just created. Below are the questions and their corresponding SQL queries to retrieve the required information.

Question 01

Find the number of drugs that have nausea as a side effect

Solution:

```sql
SELECT COUNT(DISTINCT drug_name) AS drug_count
FROM Drug_SideEffect
WHERE side_effect_name ILIKE 'nausea';
```

Question 02

Find the drugs that interact with butabarbital

Solution:

```sql
SELECT DISTINCT drug_name
FROM Drug_Interaction
WHERE interaction_name ILIKE 'butabarbital';
```

Question 03

Find the drugs with side effects cough and headache

Solution:

```sql
SELECT drug_name
FROM Drug_SideEffect
WHERE side_effect_name ILIKE 'cough'
INTERSECT
SELECT drug_name
FROM Drug_SideEffect
WHERE side_effect_name ILIKE 'headache';
```

Question 04

Find the drugs that can be used to treat endocrine diseases

Solution:

```sql
SELECT DISTINCT drug_name
FROM Drug_Disease
JOIN Disease USING (disease_name)
WHERE disease_category ILIKE 'endocrine%';
```

Question 05

Find the most common treatment for immunological diseases that have not been used for hematological diseases

Solution:

```sql
SELECT drug_name
FROM Drug_Disease
JOIN Disease USING (disease_name)
WHERE disease_category ILIKE 'immunological'
  AND drug_name NOT IN (
    SELECT drug_name
    FROM Drug_Disease
    JOIN Disease USING (disease_name)
    WHERE disease_category ILIKE 'hematological'
  )
GROUP BY drug_name
ORDER BY COUNT(*) DESC
LIMIT 1;
```

Question 06

Find the diseases that can be treated with hydrocortisone but not with etanercept

Solution:

```sql
SELECT disease_name
FROM Drug_Disease
WHERE drug_name ILIKE 'hydrocortisone'
EXCEPT
SELECT disease_name
FROM Drug_Disease
WHERE drug_name ILIKE 'etanercept';
```

Question 07

Find the top-10 side effects that drugs used to treat asthma related diseases have not been used for hematological diseases

Solution:

```sql
SELECT side_effect_name, COUNT(*) AS freq
FROM Drug_SideEffect
WHERE drug_name IN (
    SELECT drug_name
    FROM Drug_Disease
    JOIN Disease USING (disease_name)
    WHERE disease_name ILIKE '%asthma%'
)
AND drug_name NOT IN (
    SELECT drug_name
    FROM Drug_Disease
    JOIN Disease USING (disease_name)
    WHERE disease_category ILIKE 'hematological'
)
GROUP BY side_effect_name
ORDER BY freq DESC
LIMIT 10;
```

Question 08

Find the drugs that have been studied in more than three clinical trials with more than 30 participants

Solution:

```
SELECT drug_name
FROM Drug_ClinicalTrial
JOIN ClinicalTrial USING (clinical_trial_title)
WHERE clinical_trial_participants > 30
GROUP BY drug_name
HAVING COUNT(DISTINCT clinical_trial_title) > 3;
```

Question 09

Find the largest number of clinical trials and the drugs they have studied that have been active in the same period of time

Solution:

```
WITH all_dates AS (
    SELECT clinical_trial_start_date AS date FROM ClinicalTrial
    UNION
    SELECT clinical_trial_completion_date AS date FROM
       ClinicalTrial
),
active_counts AS (
    SELECT date,
           (SELECT COUNT(*)
            FROM ClinicalTrial
            WHERE clinical_trial_start_date <= date
              AND clinical_trial_completion_date >= date) AS
                 active_trial_count
    FROM all_dates
),
max_date AS (
    SELECT date, active_trial_count
    FROM active_counts
    ORDER BY active_trial_count DESC
    LIMIT 1
),
active_trials_on_max_date AS (
    SELECT clinical_trial_title
    FROM ClinicalTrial
```

```sql
    WHERE clinical_trial_start_date <= (SELECT date FROM max_date)
      AND clinical_trial_completion_date >= (SELECT date FROM max_date)
)
SELECT
    (SELECT active_trial_count FROM max_date) AS max_active_trials,
    dct.drug_name
FROM Drug_ClinicalTrial dct
JOIN active_trials_on_max_date at ON dct.clinical_trial_title = at.clinical_trial_title;
```

Question 10

Find the main researchers that have conducted clinical trials that study drugs that can be used to treat both respiratory and cardiovascular diseases

Solution:

```sql
SELECT DISTINCT clinical_trial_main_researcher
FROM ClinicalTrial
JOIN Drug_ClinicalTrial USING (clinical_trial_title)
JOIN Drug_Disease USING (drug_name)
JOIN Disease USING (disease_name)
WHERE disease_category ILIKE 'respiratory'
INTERSECT
SELECT DISTINCT clinical_trial_main_researcher
FROM ClinicalTrial
JOIN Drug_ClinicalTrial USING (clinical_trial_title)
JOIN Drug_Disease USING (drug_name)
JOIN Disease USING (disease_name)
WHERE disease_category ILIKE 'cardiovascular';
```

Question 11

Find up to three main researchers that have conducted the larger number of clinical trials that study drugs that can be used to treat both respiratory and cardiovascular diseases

Solution:

```sql
WITH both_disease_drugs AS (
  SELECT drug_name
  FROM Drug_Disease
  JOIN Disease USING (disease_name)
  WHERE disease_category ILIKE 'respiratory'
  INTERSECT
  SELECT drug_name
  FROM Drug_Disease
  JOIN Disease USING (disease_name)
  WHERE disease_category ILIKE 'cardiovascular'
),
relevant_trials AS (
  SELECT clinical_trial_main_researcher
  FROM Drug_ClinicalTrial
  JOIN ClinicalTrial USING (clinical_trial_title)
  WHERE drug_name IN (SELECT drug_name FROM both_disease_drugs)
)
SELECT clinical_trial_main_researcher, COUNT(*) AS trial_count
FROM relevant_trials
GROUP BY clinical_trial_main_researcher
ORDER BY trial_count DESC
LIMIT 3;
```

Question 12

Find the categories of drugs that have been only studied in clinical trials based in United States

Solution:

```sql
SELECT DISTINCT drug_category
FROM Drug
WHERE drug_name IN (
  SELECT drug_name
  FROM Drug_ClinicalTrial
  JOIN ClinicalTrial USING (clinical_trial_title)
  WHERE clinical_trial_address ILIKE '%united states%'
)
AND drug_name NOT IN (
  SELECT drug_name
  FROM Drug_ClinicalTrial
  JOIN ClinicalTrial USING (clinical_trial_title)
  WHERE clinical_trial_address NOT ILIKE '%united states%'
);
```

What are the differences to your initial design?

Database Management System Choice

- **Initial Design:** Planned to use simpler database operations and basic SQL
- **Final Implementation:** Used PostgreSQL with advanced features like `LATERAL UNNEST` and array handling capabilities

Data Import Strategy

- **Initial Design:** Direct insertion from normalized data sources
- **Final Implementation:** Created an intermediate `data` table to hold raw CSV data, then used complex SQL queries to normalize and populate target tables

Handling Multiple Values in Single Fields

- **Initial Design:** Assumed one-to-one relationships for attributes
- **Final Implementation:** Discovered multiple side effects, interactions, and conditions per drug, requiring columns like `side_effect_1`, `side_effect_2`, etc.

Clinical Trial Entity Complexity

- **Initial Design:** Simple clinical trial entity with basic attributes
- **Final Implementation:** Added `ClinicalTrialCondition` as a separate entity with many-to-many relationship through `ClinicalTrial_Condition` junction table

Data Processing Approach

- **Initial Design:** Basic SQL operations with multiple separate queries for each data type
- **Final Implementation:** Set-based SQL operations using `LATERAL UNNEST` to handle array-like data efficiently

Why did these differences occur?

Real Data Structure Complexity

The actual CSV data revealed:

- Multiple values stored in separate columns (side_effect, side_effect_1, side_effect_2, etc.)
- Inconsistent data entry requiring `TRIM()` and `ILIKE` operations
- NULL and empty string handling throughout the dataset
- Clinical trial conditions as a separate categorical data requiring normalization

PostgreSQL Capabilities

- **Advanced SQL Features:** `LATERAL UNNEST` allowed elegant handling of multiple columns as arrays
- **Conflict Resolution:** `ON CONFLICT DO NOTHING` provided robust duplicate handling
- **Array Operations:** PostgreSQL's array handling simplified the normalization process

Data Quality Challenges

- Discovered inconsistent casing requiring case-insensitive queries (`ILIKE`)
- Found whitespace issues necessitating `TRIM()` functions
- Encountered empty strings that needed filtering with `TRIM(val) <> ''` conditions

Normalization Requirements

- Multiple side effects per drug required proper many-to-many relationships
- Clinical trial conditions needed separate entity modeling
- Drug interactions required bidirectional relationship handling

What advantages does the new design have over the old one?

Elegant Data Processing

Listing 2: PostgreSQL Array Processing vs Row-by-Row Approach

```sql
-- New approach: Process multiple columns as arrays
INSERT INTO SideEffect (name)
SELECT DISTINCT TRIM(val) AS name
FROM data,
LATERAL UNNEST(ARRAY[
    side_effect, side_effect_1, side_effect_2, side_effect_3,
       side_effect_4
]) AS val
WHERE val IS NOT NULL AND TRIM(val) <> ''
ON CONFLICT (name) DO NOTHING;

-- Old approach would require multiple INSERT statements
-- INSERT INTO SideEffect SELECT DISTINCT side_effect FROM
   data...
-- INSERT INTO SideEffect SELECT DISTINCT side_effect_1 FROM
   data...
-- etc.
```

Robust Data Integrity

- **Conflict Handling:** `ON CONFLICT DO NOTHING` prevents duplicate key errors
- **Referential Integrity:** `ON DELETE CASCADE` maintains consistency
- **Data Validation:** Comprehensive NULL and empty string checking
- **Type Safety:** Proper data types for numeric fields like `clinical_trial_participants`

Query Performance and Flexibility

- **Case-Insensitive Searches:** `ILIKE` operators handle inconsistent data entry
- **Set Operations:** `INTERSECT`, `EXCEPT` for complex queries
- **Advanced Joins:** Natural joins using `USING` clause for cleaner syntax
- **Window Functions:** Support for complex analytical queries

Scalable Architecture

- **Modular Design:** Separate entities for each concept (Drug, SideEffect, Interaction, etc.)
- **Junction Tables:** Proper many-to-many relationships with composite primary keys
- **Extensibility:** Easy to add new attributes or entities without breaking existing structure
- **Data Warehousing:** Intermediate `data` table serves as staging area for ETL processes

Professional Database Practices

The final implementation demonstrates:

- Proper normalization following 3NF principles
- Comprehensive foreign key relationships with appropriate cascade rules
- Efficient bulk data loading strategies
- Production-ready error handling and data validation

Key Insights from Complex Query Implementation

The tutorial's advanced queries revealed additional design benefits:

Complex Analytical Capabilities

Listing 3: Advanced Query Example

```sql
-- Finding drugs treating both respiratory AND cardiovascular diseases
SELECT drug_name
FROM Drug_Disease
JOIN Disease USING (disease_name)
WHERE disease_category ILIKE 'respiratory'
INTERSECT
SELECT drug_name
```

```
FROM Drug_Disease
JOIN Disease USING (disease_name)
WHERE disease_category ILIKE 'cardiovascular';
```

Time-Based Analysis Support

The clinical trial date handling supports complex temporal queries for finding overlapping trials and active research periods, demonstrating the database's capability for longitudinal medical research analysis.

The transformation from initial design to final implementation taught several crucial database development principles:

- **Data Exploration First:** Always examine actual data before finalizing schema design.
- **Staged Implementation:** Using intermediate tables for complex data transformations.
- **PostgreSQL Power:** Advanced SQL features can dramatically simplify complex data processing.
- **Normalization Benefits:** Proper entity separation enables sophisticated analytical queries.
- **Error Handling:** Production databases require comprehensive conflict resolution strategies.