UNIVERSITY OF CHITTAGONG

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Medical Database Report

Database Systems Mini-Project Task-2

Course Information

Course Code: 413

Course Title: Database Systems

Semester: 4th

Year: 2025

Submitted By:

Md. Abdullah Al Mamun Emon
ID: 23701028
GitHub: emon4075

Submitted To:

Dr. Rudra Pratap Deb Nath
Associate Professor
Department of Computer Science & Engineering
University of Chittagong
Chittagong-4331

June 20, 2025

Table of Contents

Contents

1 Project Workflow Overview **3**

2 Project Overview **4**

3 Task 1: Relevant Data Identification **4**

4 Task 2: ER Diagram **6**

5 Task 3: Relational Model **8**

6 Database Implementation: Table Creation and Data Insertion **13**

6.1 Database and Schema Creation 13
6.2 Entity Tables Creation 14
6.3 Junction Tables Creation 17
6.4 Data Insertion Process 21
6.5 Relationship Data Population 23

7 Sample Query Solutions **29**

7.1 Query 1: 29
7.2 Query 2: 29
7.3 Query 3: 30
7.4 Query 4: 31
7.5 Query 5: 31
7.6 Query 6: 32
7.7 Query 7: 33
7.8 Query 8: 33
7.9 Query 9: 34
7.10 Query 10: 34
7.11 Query 11: 35
7.12 Query 12: 36

1 Project Workflow Overview

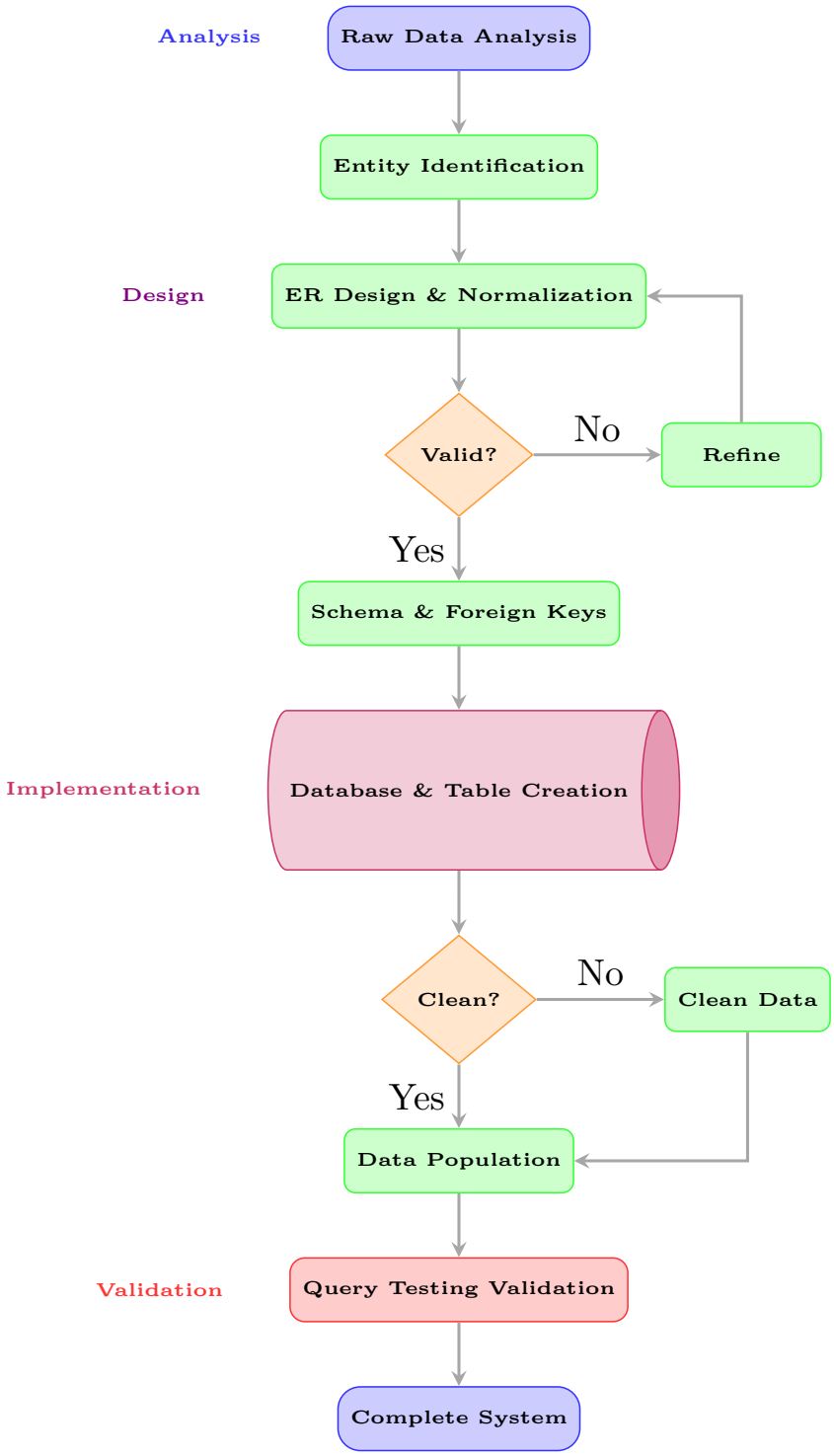


Figure 1: Medical Database Development Workflow

2 Project Overview

Project Goal

The overall goal in the whole mini-project is to create and work with a medical database in several incremental steps. Denormalized and messy data is given in an Excel sheet, and then we need to make that compatible for optimized data. The first step is to identify the information that is going to be represented as well as special requirements. Next, we model the information in an ER diagram and map it to a relational schema. The database schema is then refined and normalized.

After creating a normalized and optimized database, we need to execute the queries specified in the report requirements.

3 Task 1: Relevant Data Identification

Data Analysis Based on Sample Queries and Excel File

The provided Excel file contains comprehensive medical information. After examining the file structure, I identified the following key data entities and relationships:

Primary Entities Identified:

- **Drugs:** Basic information including drug names, categories, and associated product names
- **Side Effects:** Multiple side effects are associated with each drug, which is spread across multiple columns
- **Drug Interactions:** Information about drugs that interact with each other is also organized across multiple columns
- **Diseases:** Disease names and their categories that drugs can treat
- **Companies:** Pharmaceutical companies that produce drugs
- **Clinical Trials:** Information about clinical trials, including title, start dates, completion dates, participants, and status
- **Trial Conditions:** The medical conditions under investigation in the trials
- **Researchers:** Information about researchers conducting clinical trials
- **Locations:** Where clinical trials are conducted
- **Institution:** Where is the clinical trial being conducted

Data Quality and Normalization Requirements

Current Data Structure Issues:
The raw data format is denormalized with repetitive information and multiple columns containing the same type of data (e.g., `side_effect`, `side_effect_0`, `side_effect_1`, etc.). This structure may result in create-update-modify anomalies.

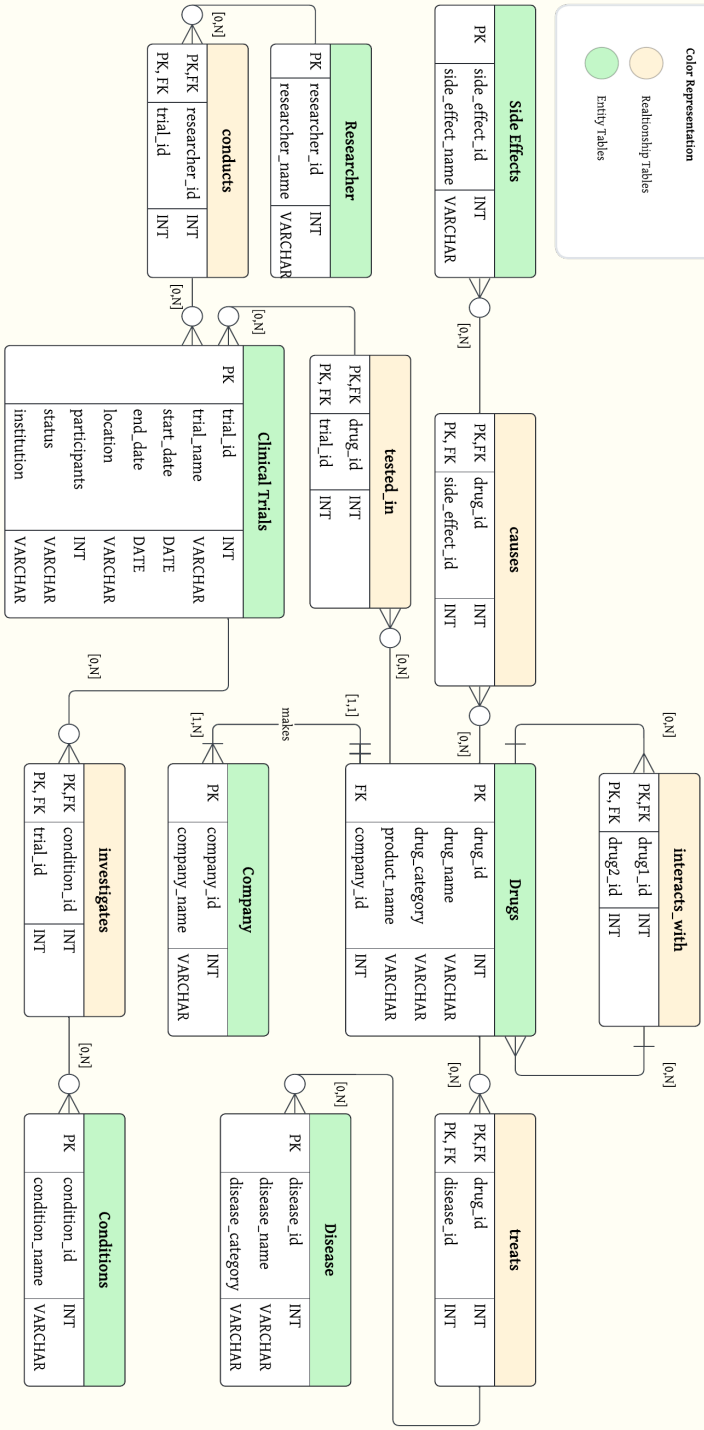
Normalization Strategy:
In order to avoid data anomalies, we need to make this whole dataset normalized. As a consequence, many relationships are identified as many-to-many, requiring careful normalization through the creation of junction tables and proper entity separation.

Key Relationships Identified:

- Drug ↔ Side Effects (Many-to-Many)
- Drug ↔ Drug Interactions (Many-to-Many, Recursive)
- Drug ↔ Diseases (Many-to-Many)
- Drug ↔ Companies (Many-to-One)
- Clinical Trial ↔ Drugs (Many-to-Many)
- Clinical Trial ↔ Researchers (Many-to-Many)
- Clinical Trial ↔ Conditions (Many-to-Many)

4 Task 2: ER Diagram

ER Diagram



ER Diagram Design Analysis

Entity Structure Overview:

Entity	Attributes	Primary Key
Drugs	drug_id, drug_name, drug_category, product_name, company_id (FK)	drug_id (PK)
Disease	disease_id, disease_name, disease_category	disease_id (PK)
Side Effects	side_effect_id, side_effect_name	side_effect_id (PK)
Clinical Trials	trial_id, trial_name, start_date, end_date, location, participants, status, institution	trial_id (PK)
Researcher	researcher_id, researcher_name	researcher_id (PK)
Company	company_id, company_name	company_id (PK)
Conditions	condition_id, condition_name	condition_id (PK)

Relationship Structure Overview:

Relationship	Junction Table Schema	Type	Description
treats	treats(drug_id, disease_id)	M:N	Drug treats disease
causes	causes(drug_id, side_effect_id)	M:N	Drug causes side effect
interacts_with	interacts_with(drug1_id, drug2_id)	M:N	Drug-drug interaction
tested_in	tested_in(drug_id, trial_id)	M:N	Drug tested in trial
conducts	conducts(researcher_id, trial_id)	M:N	Researcher conducts trial
makes	Direct FK in Drugs table	1:N	Company manufactures drug
investigates	investigates(condition_id, trial_id)	M:N	Trial investigates condition

Design Notation:

- **PK** = Primary Key, **FK** = Foreign Key
- **M:N** = Many-to-Many relationship, **1:N** = One-to-Many relationship

5 Task 3: Relational Model

Summary of the Relational Model for Medical Database

Entity/Relationship	Attributes & Key Structure
Companies	company_id (PK), company_name
Drugs	drug_id (PK), drug_name, drug_category, product_name, company_id (FK → Companies)
Diseases	disease_id (PK), disease_name, disease_category
Side_Effects	side_effect_id (PK), side_effect_name
Clinical_Trials	trial_id (PK), trial_name, start_date, end_date, location, participants, status, institution
Researchers	researcher_id (PK), researcher_name
Conditions	condition_id (PK), condition_name
Drug_Disease	drug_id (PK, FK → Drugs), disease_id (PK, FK → Diseases)
Drug_Side_Effects	drug_id (PK, FK → Drugs), side_effect_id (PK, FK → Side_Effects)
Drug_Interactions	drug1_id (PK, FK → Drugs), drug2_id (PK, FK → Drugs), interaction_type
Drug_Trials	drug_id (PK, FK → Drugs), trial_id (PK, FK → Clinical_Trials)
Trial_Conditions	trial_id (PK, FK → Clinical_Trials), condition_id (PK, FK → Conditions)
Trial_Researchers	trial_id (PK, FK → Clinical_Trials), researcher_id (PK, FK → Researchers)

Entity Relations

Companies

- **Companies**: {company_id, company_name}
- This relation stores pharmaceutical companies that produce drugs.
- company_id is a surrogate primary key for unique identification.

Drugs

- **Drugs**: {drug_id, drug_name, drug_category, product_name, company_id → Companies}
- This relation stores medical drugs and their producing companies.
- Each drug belongs to exactly one company (many-to-one relationship).

Diseases

- **Diseases**: {disease_id, disease_name, disease_category}
- This relation stores diseases that can be treated by drugs.
- Categorization allows for grouping related diseases.

Side_Effects

- **Side_Effects**: {side_effect_id, side_effect_name}
- This relation stores possible side effects that drugs may cause.
- Simple structure focused on side effect identification.

Clinical_Trials

- **Clinical_Trials**: {trial_id, trial_name, start_date, end_date, location, participants, status, institution}
- This relation stores information about medical trials testing drugs.
- Includes temporal data, location, participation details, and current status.

Researchers

- **Researchers**: {researcher_id, researcher_name}
- This relation stores researchers who conduct clinical trials.
- Simple structure focused on researcher identification.

Conditions

- **Conditions**: {condition_id, condition_name}
- This relation stores medical conditions investigated in clinical trials.
- Distinguished from diseases as conditions can be symptoms or states.

Junction Tables (Relationship Relations)

Drug_Disease

- **Drug_Disease**: {drug_id → Drugs, disease_id → Diseases}
- Junction table implementing the "treats" relationship.
- Represents which drugs can treat which diseases (many-to-many).

Drug_Side_Effects

- **Drug_Side_Effects**: {drug_id → Drugs, side_effect_id → Side_Effects}
- Junction table implementing the "causes" relationship.
- Represents which drugs may cause which side effects (many-to-many).

Drug_Interactions

- **Drug_Interactions**: {drug1_id → Drugs, drug2_id → Drugs, interaction_type}
- Junction table implementing drug-to-drug interactions (self-referential).
- Additional attribute describes the nature of the interaction.

Drug_Trials

- **Drug_Trials**: {drug_id → Drugs, trial_id → Clinical_Trials}
- Junction table implementing the "tested_in" relationship.
- Represents which drugs are being studied in which trials (many-to-many).

Trial_Conditions

- **Trial_Conditions**: {trial_id → Clinical_Trials, condition_id → Conditions}
- Junction table implementing the "investigates" relationship.
- Represents which conditions are studied in which trials (many-to-many).

Trial_Researchers

- **Trial_Researchers**: {trial_id → Clinical_Trials, researcher_id → Researchers}
- Junction table implementing the "conducts" relationship.
- Represents which researchers are involved in which trials (many-to-many).

These junction tables effectively capture all many-to-many relationships from the ER diagram, using composite primary keys and foreign key constraints to represent the complex relationships between medical entities. The design supports all the required queries for drug information, clinical trials, and medical research data.

Foreign Key Definitions (Part 1)

1. **Drugs.company_id → Companies.company_id**
   - Ensures that each drug is associated with a valid company.
   - Enforces referential integrity for the production relationship.
2. **Drug_Disease.drug_id → Drugs.drug_id**
   - Ensures that each drug-disease association references a valid drug.
   - Part of the composite key implementing the "treats" relationship.
3. **Drug_Disease.disease_id → Diseases.disease_id**
   - Ensures that each drug-disease association references a valid disease.
   - Part of the composite key implementing the "treats" relationship.
4. **Drug_Side_Effects.drug_id → Drugs.drug_id**
   - Ensures that each drug-side effect association references a valid drug.
   - Part of the composite key implementing the "causes" relationship.
5. **Drug_Side_Effects.side_effect_id → Side_Effects.side_effect_id**
   - Ensures that each drug-side effect association references a valid side effect.
   - Part of the composite key implementing the "causes" relationship.
6. **Drug_Interactions.drug1_id → Drugs.drug_id**
   - Ensures that the first drug in an interaction is a valid drug.
   - Part of the composite key implementing the "interacts_with" relationship.
7. **Drug_Interactions.drug2_id → Drugs.drug_id**
   - Ensures that the second drug in an interaction is a valid drug.
   - Part of the composite key implementing the "interacts_with" relationship.
8. **Drug_Trials.drug_id → Drugs.drug_id**
   - Ensures that each drug-trial association references a valid drug.
   - Part of the composite key implementing the "tested_in" relationship.
9. **Drug_Trials.trial_id → Clinical_Trials.trial_id**
   - Ensures that each drug-trial association references a valid clinical trial.
   - Part of the composite key implementing the "tested_in" relationship.
10. **Trial_Conditions.trial_id → Clinical_Trials.trial_id**
    - Ensures that each trial-condition association references a valid trial.
    - Part of the composite key implementing the "investigates" relationship.
11. **Trial_Conditions.condition_id → Conditions.condition_id**
    - Ensures that each trial-condition association references a valid condition.
    - Part of the composite key implementing the "investigates" relationship.

Foreign Key Definitions (Part 2)

12. **Trial_Researchers.trial_id → Clinical_Trials.trial_id**
    - Ensures that each trial-researcher association references a valid trial.
    - Part of the composite key implementing the "conducts" relationship.
13. **Trial_Researchers.researcher_id → Researchers.researcher_id**
    - Ensures that each trial-researcher association references a valid researcher.
    - Part of the composite key implementing the "conducts" relationship.

Summary of Foreign Key Constraints:

- **Total Foreign Key Relationships:** 13 constraints ensuring complete referential integrity
- **Entity-to-Entity References:** 1 direct foreign key (Drugs → Companies)
- **Junction Table References:** 12 foreign keys supporting many-to-many relationships
- **Constraint Types:** All foreign keys enforce NOT NULL and RESTRICT on delete for data integrity

Referential Integrity Benefits:

- **Data Consistency:** Prevents orphaned records and maintains logical relationships
- **Query Reliability:** Ensures JOIN operations return meaningful results
- **Business Rule Enforcement:** Database structure naturally enforces medical data relationships
- **Performance Optimization:** Foreign key indexes improve query execution for related data

6 Database Implementation: Table Creation and Data Insertion

Data Processing Overview

Data Processing Approach: To work with the provided dataset, the following approach was used:

1. First, the raw CSV data was loaded into a temporary table (`tb.raw_csv_data`)
2. This temporary table preserved the denormalized structure of the original CSV
3. From this temporary table, data was extracted and transformed into the normalized structure
4. SQL INSERT statements with appropriate JOINs and subqueries were used to distribute data across tables

6.1 Database and Schema Creation

Database Initialization

Listing 1: Database Creation

```
1 CREATE DATABASE medical;
```

Database Creation Explanation

Purpose and Implementation:

- Creates a dedicated database named `medical` for the entire medical information system
- Establishes the foundational workspace for all medical database objects
- Provides isolation and organization for the medical data management system

Database Selection

Listing 2: Database Context Setting

```
1 USE medical;
```

Database Selection Explanation

Context Setting:

- Sets the active database context for subsequent table creation operations
- Ensures all following DDL and DML statements execute within the medical database
- Eliminates the need to qualify table names with database prefix

6.2 Entity Tables Creation

Companies Table Creation

Listing 3: Companies Table Schema

```sql
CREATE TABLE Companies (
    company_id INT PRIMARY KEY AUTO_INCREMENT,
    company_name VARCHAR(255) NOT NULL UNIQUE
);
```

Companies Table Explanation

Design Rationale:

- `company_id`: Auto-incrementing primary key for unique identification
- `company_name`: Unique constraint ensures no duplicate pharmaceutical companies
- `NOT NULL` constraint enforces data integrity for company names
- Simple structure optimized for foreign key relationships with drugs
- Foundation table that must be populated before drugs table due to referential integrity

Drugs Table Creation

Listing 4: Drugs Table Schema

```sql
CREATE TABLE Drugs (
    drug_id INT PRIMARY KEY AUTO_INCREMENT,
    drug_name VARCHAR(100) NOT NULL,
    drug_category VARCHAR(100),
    product_name VARCHAR(255),
    company_id INT,
    FOREIGN KEY (company_id) REFERENCES Companies(company_id)
);
```

Drugs Table Explanation

Design Features:

- Central entity storing drug information with categorical classification
- Foreign key relationship establishes drug-company association (many-to-one)
- `product_name` allows for brand name variations of the same drug
- `drug_category` enables classification-based queries and analysis
- Referential integrity maintained through foreign key constraint to Companies table

Diseases Table Creation

Listing 5: Diseases Table Schema

```sql
CREATE TABLE Diseases (
    disease_id INT PRIMARY KEY AUTO_INCREMENT,
    disease_name VARCHAR(255) NOT NULL,
    disease_category VARCHAR(100)
);
```

Diseases Table Explanation

Structure Analysis:

- Stores medical conditions that can be treated by drugs
- `disease_category` supports hierarchical disease classification
- Simple structure optimized for treatment relationship queries
- Enables categorization for medical research and analysis
- Independent entity that can be populated without dependencies

Side Effects Table Creation

Listing 6: Side Effects Table Schema

```sql
CREATE TABLE Side_Effects (
    side_effect_id INT PRIMARY KEY AUTO_INCREMENT,
    side_effect_name VARCHAR(100) NOT NULL UNIQUE
);
```

Side Effects Table Explanation

Implementation Details:

- Maintains a controlled vocabulary of possible drug side effects
- `UNIQUE` constraint prevents duplicate side effect entries
- Optimized for many-to-many relationships with drugs
- Supports comprehensive adverse effect tracking and analysis
- Normalized approach eliminates redundancy in side effect data

Clinical Trials Table Creation

Listing 7: Clinical Trials Table Schema

```sql
CREATE TABLE Clinical_Trials (
    trial_id INT PRIMARY KEY AUTO_INCREMENT,
    trial_name TEXT,
    start_date DATE,
    end_date DATE,
    participants INT,
    status VARCHAR(50),
    location VARCHAR(100)
);
```

Clinical Trials Table Explanation

Comprehensive Trial Management:

- `TEXT` data type for trial names accommodates long descriptive titles
- `DATE` fields enable temporal analysis and trial duration calculations
- `participants` field supports statistical analysis of study sizes
- `status` and `location` provide operational and geographical context
- Supports longitudinal research and regulatory compliance tracking

Conditions Table Creation

Listing 8: Conditions Table Schema

```sql
CREATE TABLE Conditions (
    condition_id INT PRIMARY KEY AUTO_INCREMENT,
    condition_name VARCHAR(255) NOT NULL UNIQUE
);
```

Conditions Table Explanation

Medical Conditions Management:

- Stores medical conditions investigated in clinical trials
- Distinguished from diseases as conditions can be symptoms or states
- `UNIQUE` constraint ensures no duplicate condition entries
- Simple structure optimized for trial-condition relationship management
- Supports comprehensive research categorization and analysis

Researchers Table Creation

Listing 9: Researchers Table Schema

```sql
CREATE TABLE Researchers (
    researcher_id INT PRIMARY KEY AUTO_INCREMENT,
    researcher_name VARCHAR(255) NOT NULL
);
```

Researchers Table Explanation

Research Personnel Management:

- Maintains researcher information for trial attribution
- Simple structure focused on researcher identification
- Supports comprehensive research tracking and academic attribution
- Enables queries about researcher involvement across multiple trials
- Foundation for research collaboration network analysis

6.3 Junction Tables Creation

Trial Conditions Junction Table

Listing 10: Trial-Conditions Relationship Table

```sql
CREATE TABLE Trial_Conditions (
    trial_id INT,
    condition_id INT,
    PRIMARY KEY (trial_id, condition_id),
    FOREIGN KEY (trial_id) REFERENCES Clinical_Trials(trial_id) ON DELETE CASCADE,
    FOREIGN KEY (condition_id) REFERENCES Conditions(condition_id) ON DELETE CASCADE
);
```

Trial Conditions Junction Explanation

Trial-Condition Association:

- Links clinical trials to investigated medical conditions
- Composite primary key ensures unique trial-condition pairings
- CASCADE DELETE maintains referential integrity during entity removal
- Supports many-to-many relationship between trials and conditions
- Enables queries about conditions studied in specific trials

Drug Disease Junction Table

Listing 11: Drug-Disease Treatment Relationship

```sql
CREATE TABLE Drug_Disease (
    drug_id INT,
    disease_id INT,
    PRIMARY KEY (drug_id, disease_id),
    FOREIGN KEY (drug_id) REFERENCES Drugs(drug_id) ON DELETE CASCADE,
    FOREIGN KEY (disease_id) REFERENCES Diseases(disease_id) ON DELETE CASCADE
);
```

Drug Disease Junction Explanation

Treatment Relationship Implementation:

- Implements "treats" relationship between drugs and diseases
- Composite primary key prevents duplicate treatment associations
- CASCADE DELETE ensures cleanup when drugs or diseases are removed
- Supports many-to-many relationship for comprehensive treatment mapping
- Foundation for therapeutic efficacy queries and analysis

Drug Side Effects Junction Table

Listing 12: Drug-Side Effects Adverse Relationship

```sql
CREATE TABLE Drug_Side_Effects (
    drug_id INT,
    side_effect_id INT,
    PRIMARY KEY (drug_id, side_effect_id),
    FOREIGN KEY (drug_id) REFERENCES Drugs(drug_id) ON DELETE CASCADE,
    FOREIGN KEY (side_effect_id) REFERENCES Side_Effects(side_effect_id) ON DELETE CASCADE
);
```

Drug Side Effects Junction Explanation

Adverse Effects Association:

- Captures "causes" relationship for drug adverse effects
- Composite primary key ensures unique drug-side effect pairings
- CASCADE DELETE maintains data consistency during updates
- Supports pharmacovigilance and drug safety analysis
- Enables comprehensive adverse effect profiling for each drug

Drug Interactions Junction Table

Listing 13: Drug-Drug Interaction Relationship

```sql
CREATE TABLE Drug_Interactions (
    drug1_id INT,
    drug2_id INT,
    interaction_type VARCHAR(100),
    PRIMARY KEY (drug1_id, drug2_id),
    FOREIGN KEY (drug1_id) REFERENCES Drugs(drug_id) ON DELETE CASCADE,
    FOREIGN KEY (drug2_id) REFERENCES Drugs(drug_id) ON DELETE CASCADE
);
```

Drug Interactions Junction Explanation

Drug Interaction Modeling:

- Self-referential relationship modeling drug-to-drug interactions
- `interaction_type` attribute provides classification context
- Composite primary key prevents duplicate interaction records
- CASCADE DELETE ensures integrity when drugs are removed
- Critical for clinical decision support and drug safety analysis

Drug Trials Junction Table

Listing 14: Drug-Clinical Trials Testing Relationship

```sql
CREATE TABLE Drug_Trials (
    drug_id INT,
    trial_id INT,
    PRIMARY KEY (drug_id, trial_id),
    FOREIGN KEY (drug_id) REFERENCES Drugs(drug_id) ON DELETE CASCADE,
    FOREIGN KEY (trial_id) REFERENCES Clinical_Trials(trial_id) ON DELETE CASCADE
);
```

Drug Trials Junction Explanation

Clinical Testing Association:

- Links drugs to their testing environments and protocols
- Represents which drugs are being studied in which trials
- Composite primary key ensures unique drug-trial associations
- CASCADE DELETE maintains consistency during data updates
- Supports research timeline analysis and drug development tracking

Trial Researchers Junction Table

Listing 15: Trial-Researchers Conduct Relationship

```sql
CREATE TABLE Trial_Researchers (
    trial_id INT,
    researcher_id INT,
    PRIMARY KEY (trial_id, researcher_id),
    FOREIGN KEY (trial_id) REFERENCES Clinical_Trials(trial_id) ON DELETE CASCADE,
    FOREIGN KEY (researcher_id) REFERENCES Researchers(researcher_id) ON DELETE CASCADE
);
```

Trial Researchers Junction Explanation

Research Attribution Implementation:

- Associates researchers with their conducted studies
- Implements "conducts" relationship between researchers and trials
- Composite primary key prevents duplicate researcher-trial associations
- CASCADE DELETE ensures data integrity during personnel changes
- Supports academic attribution and collaboration network analysis

6.4 Data Insertion Process

Companies Data Population

Listing 16: Companies Data Insertion

```sql
INSERT INTO companies (company_name)
SELECT DISTINCT company_name
FROM tb.raw_csv_data
WHERE company_name IS NOT NULL AND company_name != '';
```

Companies Data Insertion Explanation

Data Quality and Extraction:

- Extracts unique pharmaceutical company names from raw CSV data
- `DISTINCT` keyword eliminates duplicate company entries
- `WHERE` clause filters out NULL and empty string values
- Ensures clean, normalized company data for foreign key relationships
- Must be executed first due to foreign key dependencies in Drugs table

Diseases Data Population

Listing 17: Diseases Data Insertion

```sql
INSERT INTO diseases (disease_name, disease_category)
SELECT DISTINCT disease_name, disease_category
FROM tb.raw_csv_data
WHERE disease_name IS NOT NULL AND disease_name != '';
```

Diseases Data Insertion Explanation

Categorical Disease Management:

- Populates diseases with both name and category information
- Maintains categorical structure for hierarchical disease classification
- Ensures data integrity through comprehensive NULL value filtering
- Supports medical taxonomy and research categorization
- Independent insertion without foreign key dependencies

Side Effects Data Consolidation

Listing 18: Side Effects Multi-Column Extraction

```sql
INSERT INTO side_Effects (side_effect_name)
SELECT DISTINCT side_effect
FROM (
    SELECT side_effect FROM tb.raw_csv_data WHERE side_effect IS NOT NULL AND side_effect
        != ''
    UNION
    SELECT side_effect_0 FROM tb.raw_csv_data WHERE side_effect_0 IS NOT NULL AND
        side_effect_0 != ''
    UNION
    SELECT side_effect_1 FROM tb.raw_csv_data WHERE side_effect_1 IS NOT NULL AND
        side_effect_1 != ''
    UNION
    SELECT side_effect_2 FROM tb.raw_csv_data WHERE side_effect_2 IS NOT NULL AND
        side_effect_2 != ''
    UNION
    SELECT side_effect_3 FROM tb.raw_csv_data WHERE side_effect_3 IS NOT NULL AND
        side_effect_3 != ''
) AS all_side_effects;
```

Side Effects Data Consolidation Explanation

Denormalized Data Consolidation:

- Handles denormalized side effect data across multiple CSV columns
- `UNION` operations consolidate all side effects into a single result set
- Eliminates duplicates across all side effect columns using `DISTINCT`
- Comprehensive data extraction from multiple source columns
- Transforms denormalized structure into normalized entity table

Drugs Data Population with Foreign Keys

Listing 19: Drugs Data Insertion with Company References

```sql
INSERT INTO drugs (drug_name, drug_category, product_name, company_id)
SELECT DISTINCT
    r.drug_name,
    r.drug_category,
    r.product_name,
    c.company_id
FROM tb.raw_csv_data r
JOIN companies c ON r.company_name = c.company_name
WHERE r.drug_name IS NOT NULL AND r.drug_name != '';
```

Drugs Data Population Explanation

Foreign Key Relationship Establishment:

- Establishes foreign key relationships by joining with the Companies table
- Maintains referential integrity through proper company_id assignment
- Includes all drug attributes: name, category, product name, and company association
- JOIN operation ensures only drugs with valid companies are inserted
- Creates foundation for all drug-related relationship tables

Clinical Trials Data Population with Date Handling

Listing 20: Clinical Trials Data Insertion with Date Conversion

```
INSERT INTO clinical_trials (trial_name, start_date, end_date, participants, status,
    location, institution)
SELECT DISTINCT
    clinical_trial_title,
    STR_TO_DATE(clinical_trial_start_date, '%Y.%m.%d'),
    STR_TO_DATE(clinical_trial_completion_date, '%Y.%m.%d'),
    CAST(clinical_trial_participants AS UNSIGNED),
    clinical_trial_status,
    CONCAT_WS(', ', clinical_trial_address, clinical_trial_address_0),
    clinical_trial_institution
FROM tb.raw_csv_data
WHERE clinical_trial_title IS NOT NULL AND clinical_trial_title != '';
```

Clinical Trials Data Population Explanation

Complex Data Transformation:

- STR_TO_DATE: Converts string dates to proper DATE format for temporal operations
- CAST AS UNSIGNED: Ensures participant counts are stored as integers
- CONCAT_WS: Combines multiple address fields with comma separation
- Comprehensive trial information including temporal, geographical, and institutional data
- Enables complex temporal queries and trial timeline analysis

6.5 Relationship Data Population

Drug-Disease Treatment Relationships

Listing 21: Drug-Disease Association Population

```
INSERT INTO drug_disease (drug_id, disease_id)
SELECT DISTINCT d.drug_id, dis.disease_id
FROM tb.raw_csv_data r
JOIN Drugs d ON r.drug_name = d.drug_name
JOIN Diseases dis ON r.disease_name = dis.disease_name;
```

Drug-Disease Relationships Explanation

Treatment Association Implementation:

- Establishes "treats" relationships between drugs and diseases
- Double JOIN ensures both drug and disease exist before creating association
- `DISTINCT` prevents duplicate treatment relationships
- Forms the foundation for therapeutic query capabilities
- Critical for drug efficacy and indication analysis

Drug-Side Effects Adverse Relationships

Listing 22: Drug-Side Effects Association Population

```
INSERT INTO drug_side_effects (drug_id, side_effect_id)
SELECT DISTINCT d.drug_id, se.side_effect_id
FROM tb.raw_csv_data r
JOIN Drugs d ON r.drug_name = d.drug_name
JOIN Side_Effects se ON (
    r.side_effect = se.side_effect_name OR
    r.side_effect_0 = se.side_effect_name OR
    r.side_effect_1 = se.side_effect_name OR
    r.side_effect_2 = se.side_effect_name OR
    r.side_effect_3 = se.side_effect_name
);
```

Drug-Side Effects Relationships Explanation

Multi-Column Relationship Mapping:

- Complex `OR` condition handles multiple side effect columns from denormalized data
- Creates comprehensive drug-side effect associations across all source columns
- Ensures complete capture of adverse effect relationships
- Supports pharmacovigilance and drug safety analysis
- Essential for contraindication and safety profile development

Missing Drugs Resolution for Interactions

Listing 23: Additional Drugs for Interaction Completeness

```sql
INSERT INTO drugs (drug_name)
SELECT DISTINCT interacts_with
FROM tb.raw_csv_data
WHERE interacts_with IS NOT NULL AND interacts_with != ''
  AND interacts_with NOT IN (SELECT drug_name FROM drugs)
UNION
SELECT DISTINCT interacts_with_0
FROM tb.raw_csv_data
WHERE interacts_with_0 IS NOT NULL AND interacts_with_0 != ''
  AND interacts_with_0 NOT IN (SELECT drug_name FROM drugs)
UNION
SELECT DISTINCT interacts_with_1
FROM tb.raw_csv_data
WHERE interacts_with_1 IS NOT NULL AND interacts_with_1 != ''
  AND interacts_with_1 NOT IN (SELECT drug_name FROM drugs);
```

Missing Drugs Resolution Explanation

Data Consistency and Integrity:

- Identifies drugs mentioned in interaction data but not in main drug list
- `NOT IN` subquery prevents duplicate insertions
- Ensures referential integrity for drug interaction relationships
- Handles data inconsistencies between main drug list and interaction data
- Critical for maintaining complete drug interaction networks

Drug Interactions Implementation

Listing 24: Drug-Drug Interaction Population with Optimization

```sql
INSERT INTO drug_interactions (drug1_id, drug2_id, interaction_type)
SELECT DISTINCT
    LEAST(d1.drug_id, d2.drug_id),
    GREATEST(d1.drug_id, d2.drug_id),
    'interacts_with'
FROM tb.raw_csv_data r
JOIN drugs d1 ON r.drug_name = d1.drug_name
JOIN drugs d2 ON (
    r.interacts_with = d2.drug_name OR
    r.interacts_with_0 = d2.drug_name OR
    r.interacts_with_1 = d2.drug_name
)
WHERE d1.drug_id <> d2.drug_id
  AND NOT EXISTS (
      SELECT 1 FROM drug_interactions di
      WHERE di.drug1_id = LEAST(d1.drug_id, d2.drug_id)
        AND di.drug2_id = GREATEST(d1.drug_id, d2.drug_id)
  );
```

Drug Interactions Implementation Explanation

Advanced Interaction Logic:

- `LEAST/GREATEST`: Ensures consistent ordering to prevent duplicate bidirectional entries
- `NOT EXISTS`: Prevents duplicate interaction records through existence checking
- `d1.drug_id <> d2.drug_id`: Prevents self-interaction entries
- Handles multiple interaction columns from denormalized source data
- Creates symmetric interaction relationships for comprehensive drug safety analysis

Conditions Data Population

Listing 25: Clinical Trial Conditions Extraction

```
INSERT INTO conditions (condition_name)
SELECT DISTINCT cond
FROM (
    SELECT clinical_trial_condition AS cond FROM tb.raw_csv_data WHERE
        clinical_trial_condition IS NOT NULL AND clinical_trial_condition != ''
    UNION
    SELECT clinical_trial_condition_0 AS cond FROM tb.raw_csv_data WHERE
        clinical_trial_condition_0 IS NOT NULL AND clinical_trial_condition_0 != ''
    UNION
    SELECT clinical_trial_condition_1 AS cond FROM tb.raw_csv_data WHERE
        clinical_trial_condition_1 IS NOT NULL AND clinical_trial_condition_1 != ''
) AS all_conditions;
```

Conditions Data Population Explanation

Multi-Column Condition Consolidation:

- Consolidates trial conditions from multiple denormalized columns
- `UNION` operations merge all condition variations into single result set
- `DISTINCT` eliminates duplicate conditions across all source columns
- Comprehensive extraction ensures no condition data is lost
- Prepares normalized condition data for trial-condition relationships

Trial-Conditions Relationship Population

Listing 26: Trial-Conditions Association Creation

```
INSERT INTO trial_conditions (trial_id, condition_id)
SELECT DISTINCT ct.trial_id, c.condition_id
FROM tb.raw_csv_data r
JOIN clinical_trials ct ON r.clinical_trial_title = ct.trial_name
JOIN conditions c ON (
    r.clinical_trial_condition = c.condition_name OR
    r.clinical_trial_condition_0 = c.condition_name OR
    r.clinical_trial_condition_1 = c.condition_name
)
WHERE (r.clinical_trial_condition IS NOT NULL AND r.clinical_trial_condition != '') OR
    (r.clinical_trial_condition_0 IS NOT NULL AND r.clinical_trial_condition_0 != '') OR
    (r.clinical_trial_condition_1 IS NOT NULL AND r.clinical_trial_condition_1 != '');
```

Trial-Conditions Relationship Explanation

Complex Multi-Column Association:

- Links clinical trials to their investigated conditions using complex OR matching
- Handles denormalized condition data across multiple CSV columns
- Comprehensive WHERE clause ensures valid data in any condition column
- Double JOIN ensures both trial and condition exist before creating association
- Foundation for research focus and medical condition investigation analysis

Drug-Trials Relationship Population

Listing 27: Drug-Trials Testing Association Creation

```
INSERT INTO drug_trials (drug_id, trial_id)
SELECT DISTINCT d.drug_id, ct.trial_id
FROM tb.raw_csv_data r
JOIN drugs d ON r.drug_name = d.drug_name
JOIN clinical_trials ct ON r.clinical_trial_title = ct.trial_name
WHERE r.drug_name IS NOT NULL AND r.clinical_trial_title IS NOT NULL;
```

Drug-Trials Relationship Explanation

Clinical Testing Association:

- Establishes which drugs are being tested in which clinical trials
- Double JOIN ensures both drug and trial exist before creating association
- `DISTINCT` prevents duplicate drug-trial testing relationships
- Critical for drug development timeline and regulatory approval tracking
- Supports research pipeline analysis and drug efficacy studies

Researchers Data Population

Listing 28: Researchers Data Extraction

```
INSERT INTO medical.researchers (researcher_name)
SELECT DISTINCT clinical_trial_main_researcher
FROM tb.raw_csv_data
WHERE clinical_trial_main_researcher IS NOT NULL AND clinical_trial_main_researcher != '';
```

Researchers Data Population Explanation

Research Personnel Extraction:

- Extracts unique researcher names from clinical trial data
- `DISTINCT` eliminates duplicate researcher entries
- Comprehensive filtering ensures only valid researcher names are included
- Uses explicit database qualification for cross-schema clarity
- Foundation for research attribution and collaboration analysis

Trial-Researchers Relationship Population

Listing 29: Trial-Researchers Conduct Association Creation

```
INSERT INTO medical.trial_researchers (trial_id, researcher_id)
SELECT DISTINCT ct.trial_id, res.researcher_id
FROM tb.raw_csv_data r
JOIN medical.clinical_trials ct ON r.clinical_trial_title = ct.trial_name
JOIN medical.researchers res ON r.clinical_trial_main_researcher = res.researcher_name
WHERE r.clinical_trial_main_researcher IS NOT NULL AND r.clinical_trial_title IS NOT NULL;
```

Trial-Researchers Relationship Explanation

Research Attribution Implementation:

- Creates research attribution and collaboration networks
- Links researchers to their respective conducted trials
- Double JOIN ensures both trial and researcher exist before association
- Uses explicit database qualification for consistency
- Enables academic collaboration analysis and research impact studies

7 Sample Query Solutions

Query Implementation Overview

The following section presents SQL solutions for the required database queries. Each query demonstrates the practical application of the normalized database schema and validates the design's capability to answer complex medical research questions.

7.1 Query 1:

Query 1 - Question

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

Query 1 - SQL Solution

Listing 30: Count Drugs with Nausea Side Effect

```sql
SELECT COUNT(d.drug_id)
FROM drugs d
JOIN drug_side_effects dse ON d.drug_id = dse.drug_id
JOIN side_effects se ON dse.side_effect_id = se.side_effect_id
WHERE se.side_effect_name = 'nausea';
```

Query 1 - Explanation

Approach and Logic:

- Uses JOIN operations to connect drugs with their associated side effects through the junction table
- Filters for the specific side effect 'nausea' using WHERE clause
- COUNT function provides the total number of distinct drugs causing nausea
- Demonstrates the effectiveness of the normalized many-to-many relationship design

7.2 Query 2:

Query 2 - Question

Find the drugs that interact with butabarbital.

Query 2 - SQL Solution

Listing 31: Drugs Interacting with Butabarbital

```sql
SELECT DISTINCT d.drug_name
FROM drugs d
JOIN drug_interactions di ON (d.drug_id = di.drug1_id OR di.drug2_id = d.drug_id)
WHERE (di.drug1_id = (
    SELECT d1.drug_id
    FROM drugs d1
    WHERE d1.drug_name = 'butabarbital'
)
OR
 di.drug2_id = (
    SELECT d2.drug_id
    FROM drugs d2
    WHERE d2.drug_name = 'butabarbital'
)) AND d.drug_name <> 'butabarbital';
```

Query 2 - Explanation

Approach and Logic:

- Handles bidirectional drug interactions using OR conditions in JOIN
- Subqueries find the drug_id for 'butabarbital' to match against both interaction positions
- Excludes 'butabarbital' itself from results using WHERE condition
- DISTINCT ensures each interacting drug appears only once in results
- Demonstrates the self-referential relationship implementation for drug interactions

7.3 Query 3:

Query 3 - Question

Find the drugs with side effects cough and headache.

Query 3 - SQL Solution

Listing 32: Drugs with Both Cough and Headache Side Effects

```sql
SELECT d.drug_name
FROM drugs d
JOIN drug_side_effects dse1 ON dse1.drug_id = d.drug_id
JOIN side_effects se1 ON dse1.side_effect_id = se1.side_effect_id
WHERE se1.side_effect_name = 'cough'
INTERSECT
SELECT d.drug_name
FROM drugs d
JOIN drug_side_effects dse1 ON dse1.drug_id = d.drug_id
JOIN side_effects se1 ON dse1.side_effect_id = se1.side_effect_id
WHERE se1.side_effect_name = 'headache';
```

Query 3 - Explanation

Approach and Logic:

- Uses INTERSECT operation to find drugs that cause both specified side effects
- Two separate SELECT statements filter for 'cough' and 'headache' respectively
- INTERSECT ensures only drugs appearing in both result sets are returned
- Demonstrates set operations for complex filtering requirements
- Validates the many-to-many relationship between drugs and side effects

7.4 Query 4:

Query 4 - Question

Find the drugs that can be used to treat endocrine diseases.

Query 4 - SQL Solution

Listing 33: Drugs for Endocrine Disease Treatment

```
SELECT DISTINCT(d.drug_name)
FROM drugs d
JOIN drug_disease dd ON d.drug_id = dd.drug_id
JOIN diseases dis ON dis.disease_id = dd.disease_id
WHERE dis.disease_category = 'Endocrine';
```

Query 4 - Explanation

Approach and Logic:

- Joins drugs with diseases through the drug_disease junction table
- Filters results based on disease category classification
- DISTINCT ensures each drug appears only once despite multiple endocrine diseases
- Demonstrates the effectiveness of categorical disease classification
- Supports therapeutic area analysis and drug discovery research

7.5 Query 5:

Query 5 - Question

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

Query 5 - SQL Solution

Listing 34: Immunological Treatments Excluding Hematological Use

```sql
SELECT d2.drug_name, COUNT(DISTINCT(dis2.disease_id))
FROM drugs d2
JOIN drug_disease dd2 ON dd2.drug_id = d2.drug_id
JOIN diseases dis2 ON dis2.disease_id = dd2.disease_id
WHERE dis2.disease_category = 'Immunological' AND d2.drug_id NOT IN(
    SELECT d1.drug_id
    FROM drugs d1
    JOIN drug_disease dd1 ON dd1.drug_id = d1.drug_id
    JOIN diseases dis1 ON dis1.disease_id = dd1.disease_id
    WHERE disease_category = 'Hematological'
)
GROUP BY d2.drug_name;
```

Query 5 - Explanation

Approach and Logic:

- Uses NOT IN subquery to exclude drugs used for hematological diseases
- Filters for immunological disease category in main query
- COUNT with DISTINCT provides frequency of treatment across different diseases
- GROUP BY organizes results by drug name for comparison
- Demonstrates complex filtering with multiple disease categories

7.6 Query 6:

Query 6 - Question

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

Query 6 - SQL Solution

Listing 35: Diseases Treatable by Hydrocortisone but not Etanercept

```sql
SELECT DISTINCT dis1.disease_name
FROM diseases dis1
JOIN drug_disease dd1 ON dis1.disease_id = dd1.disease_id
JOIN drugs d1 ON d1.drug_id = dd1.drug_id
WHERE d1.drug_name = 'hydrocortisone' AND dis1.disease_id NOT IN(
    SELECT DISTINCT dis2.disease_id
    FROM diseases dis2
    JOIN drug_disease dd2 ON dis2.disease_id = dd2.disease_id
    JOIN drugs d2 ON d2.drug_id = dd2.drug_id
    WHERE d2.drug_name = 'etanercept'
);
```

Query 6 - Explanation

Approach and Logic:

- Main query finds diseases treatable by hydrocortisone
- NOT IN subquery excludes diseases also treatable by etanercept
- DISTINCT ensures each disease appears only once in results
- Demonstrates comparative drug efficacy analysis capabilities
- Useful for identifying drug-specific therapeutic niches

7.7 Query 7:

Query 7 - Question

Find the top-10 side effects that drugs used to treat asthma related diseases have.

Query 7 - SQL Solution

Listing 36: Top 10 Side Effects for Asthma Treatment Drugs

```
SELECT se.side_effect_name AS 'Side Effect', COUNT(se.side_effect_name) AS 'Frequency'
FROM diseases dis
JOIN drug_disease dd ON dd.disease_id = dis.disease_id
JOIN drug_side_effects dse ON dse.drug_id = dd.drug_id
JOIN side_effects se ON se.side_effect_id = dse.side_effect_id
WHERE dis.disease_name LIKE '%Asthma%'
GROUP BY se.side_effect_name
ORDER BY Frequency DESC
LIMIT 10;
```

Query 7 - Explanation

Approach and Logic:

- Uses LIKE operator with wildcards to find asthma-related diseases
- Chains multiple JOINs to connect diseases → drugs → side effects
- COUNT and GROUP BY provide frequency analysis of side effects
- ORDER BY DESC with LIMIT 10 returns top side effects by frequency
- Demonstrates pattern matching and statistical analysis capabilities

7.8 Query 8:

Query 8 - Question

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

Query 8 - SQL Solution

Listing 37: Extensively Studied Drugs in Large Trials

```sql
SELECT d.drug_name AS 'Drug Name', COUNT(DISTINCT dt.trial_id) AS Trials
FROM clinical_trials ct
JOIN drug_trials dt ON ct.trial_id = dt.trial_id
JOIN drugs d ON dt.drug_id = d.drug_id
WHERE ct.participants > 30
GROUP BY dt.drug_id, d.drug_name
HAVING COUNT(DISTINCT dt.trial_id) > 3;
```

Query 8 - Explanation

Approach and Logic:

- Filters trials with more than 30 participants using WHERE clause
- GROUP BY organizes results by individual drugs
- HAVING clause filters for drugs with more than 3 qualifying trials
- COUNT DISTINCT ensures each trial is counted only once per drug
- Identifies well-researched drugs with substantial clinical evidence

7.9 Query 9:

Query 9 - Question

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

Query 9 - Status

Not solved yet

7.10 Query 10:

Query 10 - Question

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

Query 10 - SQL Solution

Listing 38: Researchers Studying Cross-Category Treatments

```sql
SELECT DISTINCT r.researcher_name
FROM diseases dis
JOIN drug_disease dd ON dis.disease_id = dd.disease_id
JOIN drug_trials dt ON dt.drug_id = dd.drug_id
JOIN trial_researchers tr ON tr.trial_id = dt.trial_id
JOIN researchers r ON r.researcher_id = tr.researcher_id
WHERE dis.disease_category = 'Respiratory'
INTERSECT
SELECT DISTINCT r.researcher_name
FROM diseases dis
JOIN drug_disease dd ON dis.disease_id = dd.disease_id
JOIN drug_trials dt ON dt.drug_id = dd.drug_id
JOIN trial_researchers tr ON tr.trial_id = dt.trial_id
JOIN researchers r ON r.researcher_id = tr.researcher_id
WHERE dis.disease_category = 'Cardiovascular';
```

Query 10 - Explanation

Approach and Logic:

- Uses INTERSECT to find researchers working in both disease categories
- Complex JOIN chain connects researchers to disease categories through trials and drugs
- Two separate queries filter for respiratory and cardiovascular diseases respectively
- INTERSECT ensures only researchers appearing in both result sets are returned
- Identifies interdisciplinary researchers working across therapeutic areas

7.11 Query 11:

Query 11 - Question

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.

Query 11 - SQL Solution

Listing 39: Top 3 Cross-Category Researchers by Trial Count

```sql
SELECT r.researcher_name, COUNT(DISTINCT tr.trial_id) AS 'T_Count'
FROM diseases dis
JOIN drug_disease dd ON dd.disease_id = dis.disease_id
JOIN drug_trials dt ON dt.drug_id = dd.drug_id
JOIN trial_researchers tr ON tr.trial_id = dt.trial_id
JOIN researchers r ON r.researcher_id = tr.researcher_id
WHERE dt.drug_id IN (
    SELECT d.drug_id
    FROM diseases ds
    JOIN drug_disease d ON d.disease_id = ds.disease_id
    WHERE ds.disease_category = 'Respiratory'
) AND dt.drug_id IN (
    SELECT d.drug_id
    FROM diseases ds
    JOIN drug_disease d ON d.disease_id = ds.disease_id
    WHERE ds.disease_category = 'Cardiovascular'
)
GROUP BY r.researcher_name
ORDER BY T_Count DESC
LIMIT 3;
```

Query 11 - Explanation

Approach and Logic:

- Uses nested subqueries to identify drugs treating both respiratory and cardiovascular diseases
- COUNT DISTINCT provides trial count per researcher for cross-category drugs
- ORDER BY DESC with LIMIT 3 returns top researchers by trial volume
- GROUP BY organizes results by individual researchers
- Identifies the most prolific interdisciplinary researchers in the field

7.12 Query 12:

Query 12 - Question

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

Query 12 - SQL Solution

Listing 40: Drug Categories Studied Only in US

```sql
SELECT DISTINCT d.drug_category
FROM  drugs d
JOIN drug_trials dt USING(drug_id)
JOIN clinical_trials ct USING(trial_id)
WHERE ct.location LIKE '%United States%';
```

Query 12 - Explanation

Approach and Logic:

- Uses USING clause for natural joins between related tables
- LIKE operator with wildcards matches location containing 'United States'
- DISTINCT ensures each drug category appears only once
- Identifies geographic patterns in drug research and development