Design and Implementation of a Relational Database for Bioinformatics Data Integration

Abstract:

This capstone project delivers an end-to-end relational database design and deployment pipeline for the integration of heterogeneous bioinformatics datasets from UniProt, HGNC, and Disease Ontology. It captures the complete workflow—from source data extraction, preprocessing, and conceptual modeling to advanced normalization (1NF through 3NF), SQL-based schema development, and full-scale staging-to-production migration. By leveraging principles of relational theory, this work ensures optimal data accuracy, consistency, scalability, and operational resilience. The resulting system models essential biological entities and relationships, enforces referential integrity through strong PK/FK constraints, and concludes with tested backup and restore procedures, positioning the project as a template for production-grade bioinformatics database systems.

1. Introduction:

With the rise of omics technologies and next-generation sequencing, biological data has become increasingly massive, intricate, and diverse. Managing this complexity demands database systems that can structure, retrieve, and maintain data with high integrity. Relational databases—built on mathematical foundations and governed by normalization rules—offer a robust framework for tackling these demands. However, naive database implementations often lead to redundancy, update anomalies, and scalability challenges. This project addresses these issues by designing and implementing a normalized, multi-schema relational database using MySQL for handling protein, gene, and disease data. The project incorporates best practices in entity-relationship modeling, schema partitioning (development, staging, and production), SQL scripting, and rigorous data validation to simulate real-world, enterprise-grade deployment pipelines in bioinformatics.

2. Objectives

- Conduct in-depth profiling of raw bioinformatics datasets and identify normalization violations.

- Design a comprehensive ERD that captures core biological entities and their inter-relationships.
- Normalize data progressively from unstructured 1NF form to fully resolved 3NF structures.
- Implement MySQL database schemas for development, staging, and production with consistent audit policies.
- Create SQL-based workflows for table creation, data insertion, FK/PK validation, and view generation.
- Incorporate data curation mechanisms using is_valid, create_time, and update_time fields.
- Develop error-resilient backup and recovery processes to simulate production-level fault tolerance.

3. Methodology

3.1 Data Profiling and Atomicity Assessment

The project began by curating data from UniProt based on disease-specific keywords such as "Parkinson" and "Dystonia." Each resulting entry was manually tagged with a unique item number, disease search term, and data collector metadata. The raw datasets were reviewed to detect normalization anomalies including:

- **Multi-valued attributes**: Columns such as "Alternative Names" and "Gene Synonyms" included multiple entries within a single cell, violating atomicity.
- **Derived/redundant fields**: URL fields embedding existing keys (e.g., UniProtKB ID or HGNC ID) were removed due to derivability.
- **Short-name embeddings**: Disease names with abbreviated tags in parentheses (e.g., "Dystonia (DYT11)") were decomposed into discrete fields.
- **Identifier duplication**: UniProt gene names and HGNC approved symbols were found to be identical in many cases, leading to removal of redundant columns.

These violations were corrected using string parsing, field decomposition, and refactoring, ensuring strict adherence to 1NF principles.

3.2 Conceptual Entity-Relationship Modeling

Conceptual modeling abstracted essential biological entities as follows:

- **Protein**: Defined by UniProtKB entry, includes sequence length, mass, and alternative names.
- **Gene**: Identified by HGNC ID, along with gene name, synonyms, and chromosomal location.
- **Disease**: Mapped using UniProt disease names and linked phenotype MIM numbers.
- **DO Disease**: Retrieved via DOID and includes disease name, definition, and relationships.

- **Search Activity**: Captures user-level data acquisition.

Functional dependencies such as HGNC ID → Gene Metadata and UniProt Disease Name → MIM + DOID were used to define relationships and dependencies among entities. Cardinalities were extracted by profiling pairwise associations between keys and attributes.

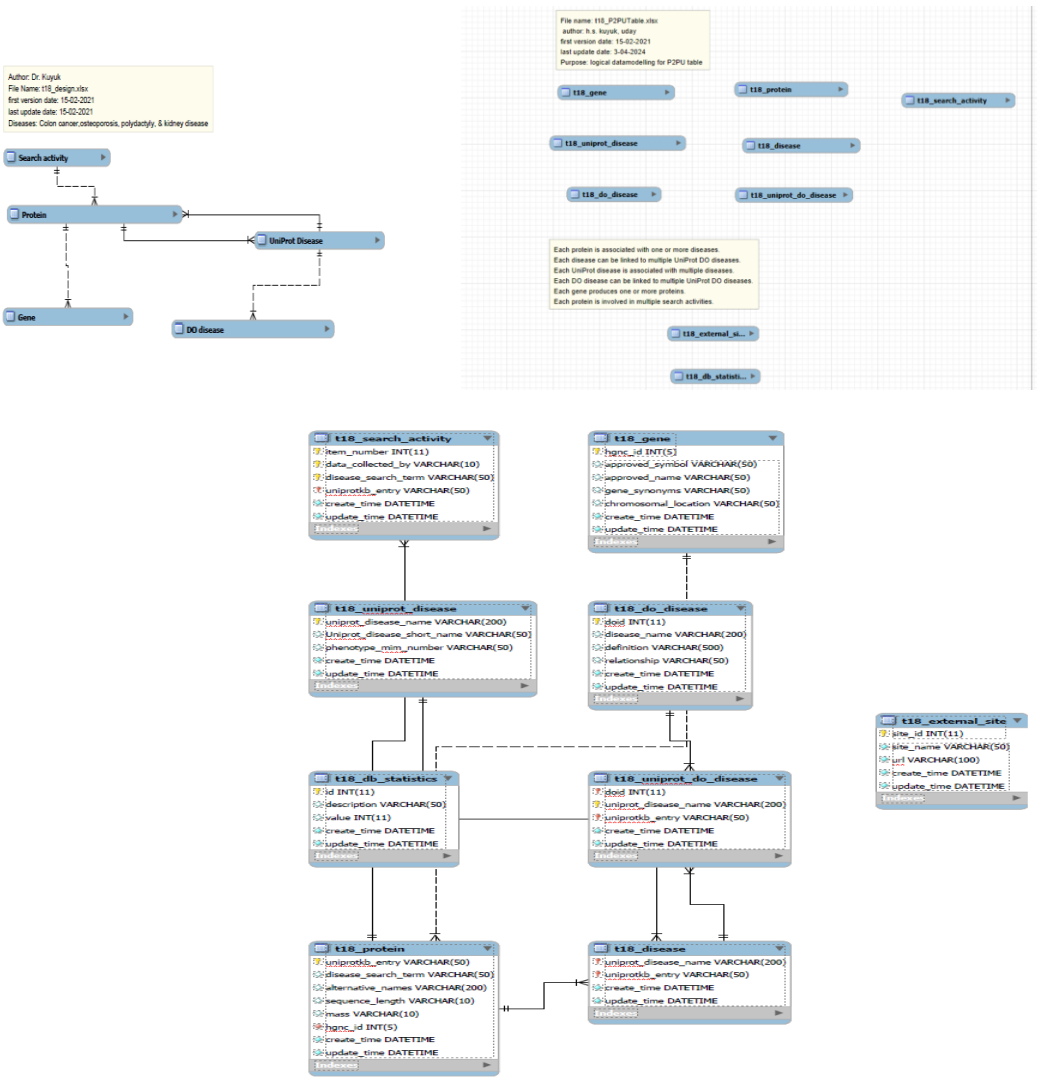
3.3 Cardinality and Integrity Constraints

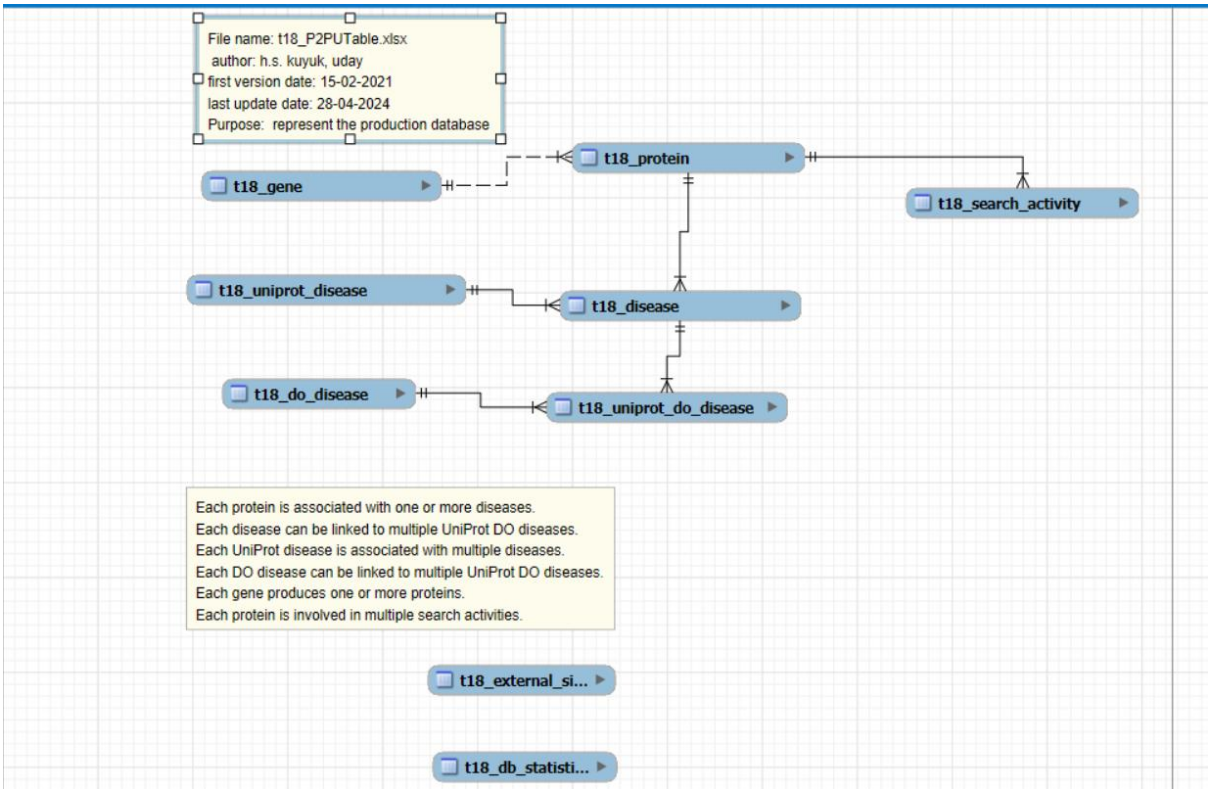
Entity relationships were explicitly modeled in MySQL Workbench, including:

- **1:1 (Protein–Gene)**: Each protein is derived from one gene.
- **1:N (Protein–Search Activity)**: A protein may be referenced in multiple searches.
- **1:N (Disease–UniProt Disease)** and **N:1 (UniProt Disease–DO Disease)**.

All relationships were annotated with identifying or non-identifying tags based on primary key propagation rules.

Associated ER diagrams include:





3.4 Normalization to 3NF

A systematic six-step normalization process was adopted:

- **1NF**: Flattened multi-value columns into atomic fields.
- **2NF**: Resolved partial dependencies from composite keys in Search Activity.
- **3NF**: Eliminated transitive dependencies by segregating derivable fields into separate entities.

This yielded nine normalized tables, each with unique primary keys and referential integrity maintained via foreign keys.

3.5 Schema Engineering and Layering

Schemas were implemented across three distinct database environments:

- **Development (t18_dev_p2p)**: Constraint-enforced 3NF schema.
- **Staging (t18_staging_p2p)**: Intermediate schema for data transfer and transformation.
- **Raw Staging (t18_staging_1)**: Initial data load with surrogate PKs and no constraints.

Each schema was backed by .mwb model files and populated with SQL scripts for forward engineering.

3.6 SQL Pipeline Development

SQL scripts were developed for all core database operations:

- **DDL Scripts**: Table definitions with audit columns and FK constraints.
- **ETL Scripts**: INSERT INTO ... SELECT with is_valid flags.
- **Validation Scripts**: FK consistency checks via LEFT JOIN and NULL detection.
- **Materialized Views**: Generated via CREATE VIEW to consolidate protein-gene-disease mappings.

Examples:

```sql
SELECT 't18_disease' AS child, 't18_protein' AS parent
FROM t18_disease d
LEFT OUTER JOIN t18_protein p ON d.uniprotkb_entry = p.uniprotkb_entry
WHERE p.uniprotkb_entry IS NULL;

SELECT 't18_do_disease' AS child, 't18_disease' AS parent
FROM t18_do_disease dd
LEFT OUTER JOIN t18_disease d ON dd.disease_name = d.uniprot_disease_name
WHERE d.uniprot_disease_name IS NULL;

SELECT 't18_external_site' AS child, 't18_protein' AS parent
FROM t18_external_site es
LEFT OUTER JOIN t18_protein p ON es.site_id = p.hgnc_id
WHERE p.hgnc_id IS NULL;

SELECT 't18_gene' AS child, 't18_protein' AS parent
FROM t18_gene g
LEFT OUTER JOIN t18_protein p ON g.hgnc_id = p.hgnc_id
WHERE p.hgnc_id IS NULL;

SELECT 't18_search_activity' AS child, 't18_protein' AS parent
FROM t18_search_activity sa
LEFT OUTER JOIN t18_protein p ON sa.uniprotkb_entry = p.uniprotkb_entry
WHERE p.uniprotkb_entry IS NULL;

SELECT 't18_uniprot_do_disease' AS child, 't18_do_disease' AS parent
FROM t18_uniprot_do_disease udd
LEFT OUTER JOIN t18_do_disease dd ON udd.doid = dd.doid
WHERE dd.doid IS NULL;
```

```sql
CREATE VIEW t18_dev_p2p.t18_vw_p2p AS
SELECT distinct
    p.uniprotkb_entry AS 'UniProtKB Entry',
    p.alternative_names AS 'Protein Name',
    p.alternative_names AS 'Alternative Protein Names',
    g.hgnc_id AS 'HGNC ID',
    g.approved_symbol AS 'Approved Gene Symbol',
    g.chromosomal_location AS 'Chromosomal Location',
    d.uniprot_disease_name AS 'UniProt Disease Name',
    u.phenotype_mim_number AS 'Phenotype MIM #',
    do.doid AS 'DOID',
    do.disease_name AS 'DO Disease Name'
FROM
    t18_protein p
LEFT OUTER JOIN
    t18_gene g ON p.hgnc_id = g.hgnc_id
LEFT OUTER JOIN
    t18_disease d ON p.uniprotkb_entry = d.uniprotkb_entry
LEFT OUTER JOIN
    t18_uniprot_disease u ON d.uniprot_disease_name = u.uniprot_disease_name
LEFT OUTER JOIN
    t18_uniprot_do_disease udo ON udo.uniprotkb_entry = p.uniprotkb_entry
LEFT OUTER JOIN
    t18_do_disease do ON udo.doid = do.doid;
```

3.7 Backup, Restore, and Error Resolution

Full schema backups were performed using mysqldump for all layers. Recovery scripts embedded CREATE DATABASE and USE statements to streamline restoration. Invalid data was flagged with is_valid = 0, and full round-trip validation was confirmed via schema deletion and re-import procedures.

4. Results and Final Schema Overview

The final schema includes the following normalized tables:

Table	Primary Key	Foreign Keys
Gene	hgnc_id	N/A
Protein	uniprotkb_entry	hgnc_id → Gene
Disease	uniprot_disease_name	uniprotkb_entry → Protein
UniProt_Disease	uniprot_disease_name	N/A
DO_Disease	doid	N/A
UniProt_DO_Disease	doid + uniprot_disease_name + uniprotkb_entry	doid → DO_Disease, uniprot_disease_name → Disease
Search_Activity	item_number + data_collected_by + disease_search_term	uniprotkb_entry → Protein
External_Site	site_id	N/A
DB_Statistics	id	N/A

All foreign key constraints passed validation checks, and the view t18_vw_p2p returned the expected 13 composite mappings.

5. Conclusion

This project exemplifies the practical application of relational theory to complex bioinformatics domains. It delivers a robust, validated, and production-ready relational schema capable of supporting large-scale biological data integration tasks. Through normalization, schema partitioning, SQL scripting, and data governance practices, the database ensures integrity, performance, and extensibility. The design can be leveraged as a foundational framework for future expansions in clinical informatics, genomics, and translational medicine use cases.