Automated Database Normalization and Visualization Framework

Subha Shesgin, Nilanjana Das Jui, Sumaiya Nazneen, *Student Member, Computer Science and Engineering , University of Chittagong*

***Abstract*—This paper introduces an advanced database management software tool that enables seamless interaction with the complete normalization workflow through a unified and intuitive interface. The system comprehensively integrates the core database design operations, including automated functional dependency detection, systematic normalization to 1NF, 2NF, and 3NF, rigorous key identification, dependency preservation analysis, lossless decomposition verification, and dynamic generation of entity-relationship (ER) diagrams. To facilitate clarity and operational transparency, the software incorporates an ETL-style workflow visualization that explicitly depicts the interconnections between sequential steps, illustrating how the output of each stage functions as the input for subsequent operations. The interface is structured into four cohesive panels: a tool panel for operation selection, a workflow panel for process visualization, a notification panel for real-time guidance, and a code panel providing the corresponding query or script representations. Fully data-agnostic, the system is capable of processing and normalizing datasets of various types and structures, ensuring broad applicability across academic, industrial, and data-driven environments. By streamlining complex normalization tasks and enhancing interpretability, the tool empowers both database professionals and users with minimal prior knowledge of the database to efficiently understand, execute, and monitor the normalization process, thus promoting data integrity, reducing redundancy, and enabling informed decision making.**

***Index Terms*—Database Normalization, Functional Dependencies, 1NF/2NF/3NF, ER Diagram, Workflow Visualization, Flask, React.**

I. Introduction

Databases constitute a fundamental component of contemporary information systems, serving as the backbone for storing, organizing, and retrieving vast volumes of structured data. The relational database model, introduced by E. F. Codd, remains the dominant paradigm for data management due to its rigorous theoretical foundation and practical flexibility. Yet, the process of designing robust relational schemas is inherently complex and requires meticulous attention to formal principles. Critical operations such as functional dependency (FD) detection, candidate key identification, normalization into higher normal forms (1NF, 2NF, 3NF), and the assurance of dependency preservation and lossless decomposition form the core of this design process. These procedures are indispensable for ensuring that a database schema is consistent, non-redundant, and capable of supporting efficient query execution while safeguarding data integrity and scalability.

The challenge arises from the delicate interplay between theoretical rigor and practical execution. Functional dependencies must be derived accurately to capture the semantic relationships among attributes. Candidate keys, which are minimal sets of attributes capable of uniquely identifying tuples, serve as anchors for defining primary keys and enforcing constraints. Normalization requires the systematic application of rules to reduce redundancy and eliminate update anomalies, but excessive normalization can fragment schemas and impair query performance. Ensuring that decomposition is both lossless (preserving the ability to reconstruct the original relation) and dependency preserving (retaining all original functional constraints) adds yet another layer of difficulty. Performing these tasks manually is cognitively demanding and highly error-prone, as even minor inaccuracies can lead to severe downstream consequences, including redundancy, anomalies, and inconsistent query behavior.

This underscores the pressing need for computational tools that can automate these technical operations while simultaneously making their underlying logic transparent. A purely automated system risks becoming a ”black box,” offering results without insight. Conversely, manual-only approaches are error-prone and time-intensive. Bridging this gap requires an intelligent system that not only carries out complex computations but also communicates its reasoning in a manner that reinforces conceptual understanding.

The system developed in this project addresses this dual requirement by providing an integrated, interactive platform for relational database design. Unlike conventional workflows that compartmentalize tasks across disparate tools or rely heavily on manual derivation, the proposed solution consolidates the complete schema design process within a unified environment. The system supports FD detection, candidate key identification, normalization into successive normal forms, validation of decomposition properties, and entity–relationship (ER) modeling. Collectively, these features allow the tool to cover the essential spectrum of relational schema development, from theoretical analysis to conceptual modeling.

A defining contribution of this system is its workflow-oriented visualization. Drawing inspiration from ETL (Extract - Transform - Load) pipelines, the tool represents database design as a chain of interdependent processes, where the output of one stage becomes the input for the next. For example, functional dependencies derived from the dataset directly inform candidate key identification; these candidate keys, in turn, guide the normalization process. Once normalized, schemas are tested to ensure both dependency preservation and lossless decomposition, after which the validated schemas

transition into ER model construction. This explicit chaining of processes not only clarifies the logical dependencies between stages but also makes abstract concepts more tangible by showing how theoretical constructs evolve into practical schema representations.

Equally significant is the system's user interface, which has been deliberately designed to serve both pedagogical and professional purposes. It integrates four modular panels, each aligned with a specific aspect of database design. The **tool panel** functions as the operational hub, offering users a structured set of commands to initiate processes such as FD detection, schema normalization, or decomposition validation. This structured approach lowers cognitive overhead for learners while ensuring streamlined access for practitioners who require efficiency. The **workflow panel** provides a dynamic, graphical representation of the design pipeline, highlighting dependencies between tasks and making it visually clear how decisions in one stage affect subsequent operations. For educators, this serves as a didactic tool to illustrate design principles, while for researchers and professionals, it enhances traceability and reproducibility. The **notification panel** operates as an intelligent feedback system, delivering real-time responses ranging from confirmations of successful operations to warnings about potential inconsistencies. In academic contexts, this transforms errors into constructive learning moments, while in industry contexts, it acts as a safeguard for maintaining design correctness. Finally, the **code panel** exposes the computational logic behind each operation, displaying algorithmic execution in a transparent manner. This dual-level view-conceptual output alongside algorithmic detail-empowers novice users to connect theory with practice while enabling advanced users to audit, extend, or adapt the underlying algorithms for specialized domains.

In summary, the proposed system advances the practice of relational database design by coupling automation with interpretability. It reduces the cognitive and procedural burdens associated with tasks such as normalization, candidate key derivation, and decomposition validation while simultaneously reinforcing theoretical understanding through interactive visualization. By merging computational precision with pedagogical clarity, the system provides a robust and versatile platform that caters equally to novice learners seeking foundational understanding, educators aiming to illustrate theoretical concepts, and professionals engaged in the design of scalable, efficient, and reliable database systems.

A. *Project Management*

Effective project management was essential for the successful development of this system, though the process was marked by numerous challenges. The project was unprecedented-no prior work had combined the specific theoretical and practical requirements of this system—which meant the team navigated largely uncharted territory. The complexity of database design, coupled with the interdisciplinary demands of the project, required both advanced theoretical knowledge of relational algebra and hands-on software engineering skills.

Throughout development, the team encountered repeated technical hurdles. Backend and frontend code frequently presented issues, including unexpected bugs, integration conflicts, and occasional system crashes. These challenges often necessitated extensive debugging, module rewrites, and reevaluation of architectural decisions. Even minor errors could halt progress, highlighting the intricate interdependence of different system components and the need for careful, iterative testing.

Compounding these difficulties was the limited applicability of existing literature. Related works were often only partially relevant: some focused on theoretical foundations without practical implementation, while others addressed systems with different scope or functionality. As a result, the team relied on a combination of selective adaptation, experimentation, and novel solutions to ensure the system met its objectives.

To manage these challenges effectively, the team implemented a highly structured management strategy. Roles were clearly defined to align with individual expertise, resources and tools were carefully allocated, and a disciplined communication framework ensured that issues were promptly identified and resolved. Strict adherence to schedules, regular design reviews, iterative testing cycles, and focused problem-solving sessions were critical for maintaining progress despite setbacks. Through sustained dedication, rigorous planning, and collaborative effort, the team successfully overcame technical, theoretical, and operational challenges, delivering a fully functional, integrated system that addressed an entirely new set of requirements.

A1: Resources:

To ensure the successful execution of this complex and unprecedented project, the team relied on a carefully curated set of academic and practical resources. Authoritative textbooks on relational database theory, normalization techniques, functional dependency analysis, and algorithmic design provided the foundational knowledge necessary to implement robust backend algorithms. These texts were particularly valuable when designing functional dependency detection and candidate key identification modules, helping the team overcome initial algorithmic errors and inconsistencies.

Peer-reviewed research papers offered insights into advanced normalization strategies, ER modeling approaches, and best practices in database visualization. While no existing system fully combined all the desired features, these papers guided the team in adapting theoretical concepts to the practical demands of a user-friendly interface. This guidance proved critical when addressing complex integration challenges, such as ensuring lossless decomposition validation worked seamlessly across multiple datasets.

Online tutorials, coding guides, and documentation were employed extensively to address frontend and backend implementation issues. They were instrumental in resolving bugs, debugging system crashes, and implementing responsive

interface components. These resources allowed the team to quickly learn new libraries, troubleshoot integration conflicts, and optimize system performance under load.

Finally, example datasets spanning diverse real-world scenarios were used to validate the system's dataset-agnostic functionality, ensuring reliability and versatility across multiple domains. Testing with these datasets was key in identifying edge-case failures, preventing system crashes, and refining both backend algorithms and frontend displays.

By strategically combining theoretical references, applied research, practical coding guides, and real-world testing datasets, the team systematically overcame technical and conceptual challenges. This comprehensive approach enabled the creation of a robust, interactive system that bridges abstract database principles with practical implementation. Here some resources links below:

- https://stackoverflow.com/questions/51356402/how-to-upload-excel-or-csv-file-to-flask-as-a-pandas-data-frame
- https://www.kdnuggets.com/data-cleaning-with-pandas
- https://medium.com/40tubelwj/python-pandas-advanced-techniques-for-data-cleaning-and-transformation-part-1-471f007367fd
- https://levelup.gitconnected.com/checking-mining-and-exploring-functional-dependencies-in-python-903bb0e26d5d
- https://learn.microsoft.com/en-us/fabric/data-science/tutorial-data-cleaning-functional-dependencies
- https://realpython.com/python-data-cleaning-numpy-pandas/
- https://andygrigg.com/?tag=pandas
- https://github.com/pandas-dev/pandas/blob/main/pandas/core/groupby/generic.py
- https://github.com/app-generator/sample-flask-pandas-dataframe?tab=readme-ov-file
- https://blog.stackademic.com/16-data-feature-normalization-methods-using-python-with-examples-part-1-of-3-26578b2b8ba6
- https://www.digitalocean.com/community/tutorials/normalize-data-in-python
- https://medium.com/40nuburoojkhattak/connecting-your-react-app-to-your-flask-api-a-step-by-step-guide-3daa8ce9d3f2
- https://javascript.plainenglish.io/hooking-your-react-frontend-to-your-flask-api-fa83b14c9684
- https://medium.com/40colinatjku/integrating-flask-and-react-using-vite-for-development-and-nginx-in-production-a-microservices-9df7a21ca8f5
- https://arxiv.org/abs/1602.00563
- https://extbi.cs.aau.dk/SETLBI/index.php
- https://reactflow.dev/
- https://tailwindcss.com/docs/installation/using-vite

Tools:

Effective execution of this complex and unprecedented project required a carefully selected suite of tools to ensure robust collaboration, maintain code quality, and facilitate seamless integration between backend and frontend components.

1) **GitHub** served as the backbone for version control and collaborative development. By managing multiple development branches, GitHub allowed the team to experiment with different algorithms or interface designs without risking the stability of the main system. The platform's issue-tracking and pull-request features enabled meticulous code reviews, helping to identify subtle bugs in backend algorithms, such as incorrect candidate key detection or faulty functional dependency validation, before they propagated into the integrated system. GitHub's commit history also proved invaluable for diagnosing system crashes and rolling back problematic changes.

   For reference and reproducibility, the complete project work has been made publicly available at the provided GitHub link.

   https://github.com/nilanjanajui/Project–DataBase-Design-Studio.git

2) **Trello** was employed as a project management dashboard to organize tasks, track progress, and monitor milestones. By creating clear task cards for specific challenges-such as debugging frontend workflow crashes or implementing lossless decomposition verification-Trello allowed the team to prioritize work, allocate resources efficiently, and maintain accountability. It also helped visualize dependencies between tasks, ensuring that critical backend or frontend updates were completed in sequence to prevent integration conflicts.

   For task management, collaboration, and progress tracking, the team utilized Trello as an organizational tool. The complete project workflow, task allocations, and meeting schedules are documented and available at the provided Trello link.

   https://trello.com/invite/b/686d400a234c5cd159c9e544/ATTI69f416fc0dd899c3b83f929192bac59f6E2E33C8/dbms-project

3) **Discord** enabled real-time communication, essential for collaborative debugging and problem-solving. When the system experienced unexpected crashes, such as interface freezes or dataset-related errors, the team used Discord to conduct live troubleshooting sessions. Its voice channels facilitated rapid discussions of complex algorithmic issues, while its screen-sharing capability allowed members to collaboratively inspect code, trace errors, and implement fixes in real-time, reducing downtime caused by technical setbacks.

4) **Google Calendar** provided structured scheduling and time management support, ensuring that development sessions, debugging periods, and team meetings were conducted consistently. By allocating dedicated blocks for backend algorithm refinement, frontend interface

testing, and system integration, the team was able to maintain disciplined progress despite the technical difficulties encountered. Calendar reminders ensured that no critical milestone or review session was missed, fostering accountability and sustained dedication throughout the project.

By leveraging this integrated suite of tools, the team maintained coordination across multiple development layers, efficiently addressed both anticipated and unforeseen technical challenges, and systematically resolved issues ranging from algorithmic inconsistencies to system crashes. This strategic approach enabled the successful delivery of a stable, interactive, and pedagogically effective system, bridging theoretical database concepts with practical implementation.

A2 : Specific Roles:

Nilanjana Das Jui (23701011)
Nilanjana contributed to the backend by implementing functional dependency detection, normalization, and ER diagram generation. On the frontend, she worked on shared components including stateContext, api.js, and app.jsx, as well as developing the ETL workflow and Tool Panel. She also contributed to app.py, implementing the backend routes and logic necessary for executing these operations and ensuring smooth interaction with the frontend. Her collaboration responsibilities included managing the GitHub repository, maintaining version control, and synchronizing code contributions across the team.

Subha Shesgin (23701036)
Subha handled backend tasks such as dataset conversion to CSV, data cleaning, and key detection. On the frontend, she implemented the Message Panel, which included dynamic buttons for functional dependencies and 3NF tables, and contributed to shared components (stateContext, api.js, app.jsx). Subha also worked on app.py, integrating backend functionalities for CSV conversion, cleaning, and key detection into the application. Her collaboration efforts focused on maintaining project documentation, recording workflows, algorithms, and interface specifications for reference and consistency.

Sumaiya Nazneen (23701042)
Sumaiya handled backend operations including lossless decomposition verification and dependency preservation checks. On the frontend, she designed the Code Panel, which displayed backend computations in real time, and contributed to shared components (stateContext, api.js, app.jsx). She also contributed to app.py, ensuring proper backend implementation of lossless decomposition and dependency preservation logic and connecting these functions to the frontend interface. Her collaboration involved managing the Trello board, tracking tasks, milestones, and overall project progress to keep the team organized.

All three members jointly conducted task allocation, progress monitoring, and integration of frontend and backend modules, while organizing team meetings and scheduling development activities using Google Calendar. These collaborative efforts ensured a consistent workflow, timely problem-solving, and the overall reliability, robustness, and effectiveness of the final system.

A3 : Meeting Structure:

Given the ambitious scope and technical complexity of this project, maintaining a disciplined and well-structured meeting schedule was critical to ensure steady progress and timely problem resolution. The team adopted a rigorous cadence that balanced extended development sessions with regular checkpoints for discussion, review, and debugging.

On weekends, two dedicated meetings were held daily-one in the morning and one in the evening. These longer sessions allowed the team to focus on complex algorithmic development, integration of backend and frontend modules, and troubleshooting of critical issues such as system crashes, unresponsive interface components, or incorrect normalization outputs. Extended weekend meetings provided uninterrupted blocks of time for iterative testing, collaborative debugging, and refinement of workflows, ensuring that persistent technical challenges were addressed promptly.

During weekdays, a single daily meeting was held to monitor ongoing progress, allocate tasks, and discuss issues that emerged during individual development sessions. These shorter meetings emphasized accountability, allowing the team to quickly resolve minor bugs, clarify integration priorities, and coordinate subsequent development steps. All meetings followed a structured agenda, including:

1) Review of completed tasks and milestones.
2) Discussion of technical challenges, particularly backend or frontend bugs and system crashes.
3) Allocation of new tasks and setting deadlines for the next checkpoint.
4) Collaborative debugging sessions or live code review where necessary.

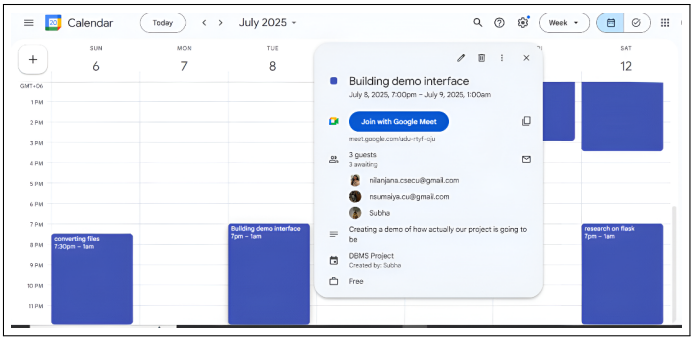

Fig. 1: Google Calendar Schedule view showing project-related events.

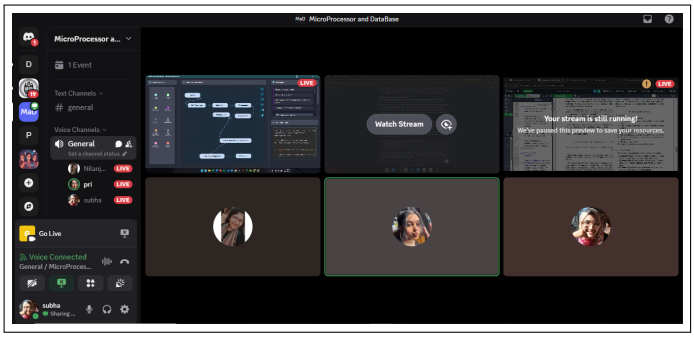

Fig. 2: Discord project meeting.

This disciplined meeting structure, combined with strict adherence to deadlines and consistent communication, ensured that the team remained focused, coordinated, and productive despite recurring technical obstacles. By fostering a culture of accountability, collaboration, and sustained dedication, the team was able to systematically address challenges, optimize system performance, and maintain momentum throughout the development process.

II. Related Work

Previous works in both academic and industrial domains have explored methods and tools for relational database design, focusing on normalization and schema optimization. In the educational domain, several normalization teaching tools have been developed to help students understand concepts such as functional dependencies (FDs), candidate keys, and higher normal forms. These tools often provide step-by-step guidance but typically require manual FD entry and offer limited or no visualization, which can hinder intuitive understanding of the workflow and interdependencies between design stages.

In industrial settings, commercial DBMS systems like Oracle and SQL Server support normalization indirectly through schema design utilities and constraints management. While these enterprise solutions offer robust schema management and enforce data integrity, they generally do not provide fully automated, teaching-friendly pipelines that demonstrate the sequential relationship between FD detection, candidate key identification, normalization, and ER diagram generation. Consequently, users must interpret the underlying logic themselves, which can be challenging for learners or less experienced designers.

Several research prototypes have attempted to address these gaps by combining automation with visualization. These systems aim to automate FD detection, candidate key derivation, and normalization while representing the workflow in a clear, stepwise manner. However, many of these solutions either remain limited to small datasets, lack full integration of decomposition validation and ER modeling, or fail to provide an interface suitable for educational purposes.

The system proposed in this project bridges these gaps by offering a fully automated and interactive platform that integrates FD detection, candidate key identification, normalization into successive normal forms, dependency preservation checks, lossless decomposition validation, and ER modeling. Its workflow-oriented visualization demonstrates the logical progression from raw data to normalized schemas, creating a teaching-friendly pipeline that is both pedagogically effective and professionally relevant. By unifying automation and visualization, this system addresses limitations of both academic teaching tools and commercial DBMS systems, providing a versatile solution suitable for learners and practitioners alike.

1) https://erdplus.com/
2) https://arxiv.org/abs/2507.23470
3) https://www.behance.net/gallery/218046239/ETLpro-ETL-Drag-Drop-React-flow-app

III. Use Case

The proposed system supports end-to-end relational database design and normalization for complex biomedical datasets. A representative use case involves a drug dataset comprising attributes such as Drug Name, Side Effects, Interacts With, Disease Name, Disease Category, Drug Category, Product Name, Company Name, Clinical Trial Title, Clinical Trial Dates, Participants, Status, Conditions, Country, Institution, Address, and Main Researcher.

Upon uploading the dataset in CSV format, the system initiates automated data cleaning, which includes removing duplicate entries, consolidating multiple columns representing side effects and interactions, and standardizing attribute values. This ensures that the dataset is prepared for reliable analysis and functional dependency computation.

Next, functional dependencies (FDs) are automatically identified. Examples of detected dependencies include:

$$drug\_name \rightarrow side\_effect \tag{1}$$

$$disease\_name \rightarrow disease\_catagory \tag{2}$$

$$drug\_name \rightarrow clinical\_trial\_title \tag{3}$$

These dependencies serve as the basis for determining candidate keys, which uniquely identify tuples within the relations.

The system then performs normalization, decomposing the dataset into First Normal Form (1NF), Second Normal Form (2NF), and Third Normal Form (3NF). Each normalized form is preserved and made accessible, enabling stepwise inspection.

Validation procedures are applied to ensure the correctness of the decomposition:

Dependency Preservation Check: Verifies that all original functional dependencies are retained in the decomposed relations.

Lossless Join Check: Confirms that the original dataset can be reconstructed without loss from the normalized relations.

After normalization, a comprehensive Entity-Relationship (ER) diagram is generated, representing key entities such as Drug, Disease, ClinicalTrial, and Institution, along with their interrelationships. This diagram bridges conceptual and logical design and facilitates intuitive understanding of the database structure.

An interactive workflow interface provides feedback at each stage. The message panel displays system status messages after every step. Contextual buttons allow users to explore:

1) Detected functional dependencies
2) Normalized 1NF, 2NF, and 3NF tables
3) Generated ER diagram

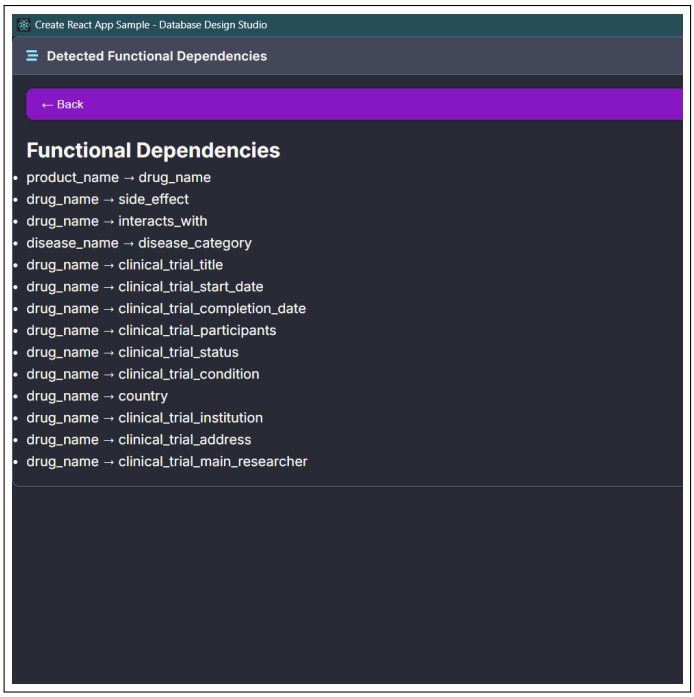

Fig. 3: Detected FDs for sample file.

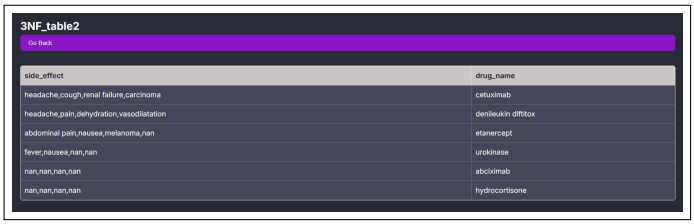

Fig. 4: one of 3nf tables for sample file.

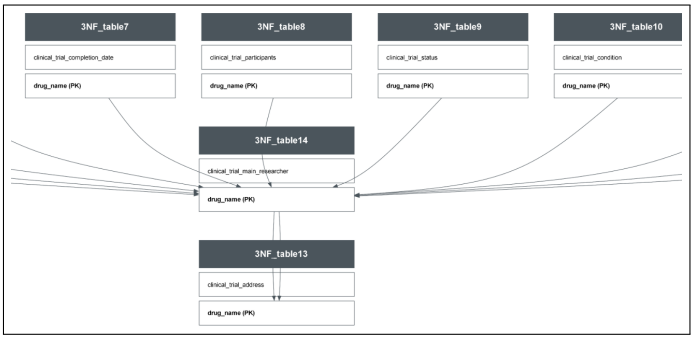

Fig. 5: ER Diagram for sample file.

This use case demonstrates the integration of automation and visualization in database normalization. The system provides a teaching-friendly pipeline that allows researchers and learners to interactively trace normalization steps while producing a professionally validated schema suitable for practical applications.

IV. System Framework

A. *Operations*

The execution of this project involved a structured sequence of operations, each addressing a distinct stage of relational database design while ensuring that theoretical principles were translated into practical, implementable solutions. These operations can be broadly categorized into **analytical tasks**, **algorithmic processing**, **system implementation**, and **validation activities**.

1) **Problem Definition and Requirement Analysis**:
The project commenced with a clear identification of the research gap: the absence of a comprehensive system that integrates functional dependency detection, candidate key identification, normalization, decomposition validation, and ER modeling into a single workflow. Existing tools were studied for reference, but their limitations—such as isolated functionalities or lack of workflow visualization—shaped the requirements of this project.

2) **Functional Dependency (FD) Extraction and Analysis:**
The first technical operation focused on detecting and analyzing functional dependencies from the input schema. Since FDs are foundational to normalization and candidate key derivation, the project devoted significant effort to designing algorithms that ensured accurate identification of dependencies. This step laid the groundwork for all subsequent operations.

3) **Candidate Key Identification:**
Once FDs were established, algorithms were implemented to determine candidate keys. This involved testing attribute closures against the universal relation, ensuring that only minimal keys capable of uniquely identifying tuples were retained. These

candidate keys later guided primary key selection and normalization procedures.

4) **Schema Normalization into Higher Normal Forms:** The system then automated normalization, progressing systematically from 1NF to 3NF (and in certain cases BCNF). Each step applied theoretical rules to eliminate redundancy, reduce anomalies, and maintain semantic integrity. Careful attention was given to balance between normalization and performance trade-offs, ensuring that over-fragmentation was avoided.

5) **Lossless and Dependency-Preserving Decomposition Validation:**
After normalization, every decomposition was validated against two critical properties:

- **Losslessness**, ensuring that the original relation could be reconstructed without data loss.
- **Dependency Preservation**, ensuring that all original functional dependencies remained enforceable in the decomposed schema.

Automated checks were implemented to rigorously evaluate these conditions, preventing structural weaknesses in the resulting design.

6) **Entity–Relationship (ER) Modeling:**
A conceptual ER model was then derived from the normalized schema. This stage emphasized visualization, providing a bridge between the abstract relational structure and an intuitive representation of entities, attributes, and relationships. Unlike conventional ER tools, the system ensured that the ER diagram was informed by mathematically validated schemas, maintaining correctness from the ground up.

7) **User Interface Integration and Workflow Visualization:**
The analytical operations were consolidated into a user-facing system composed of four panels: the tool panel, workflow panel, notification panel, and code panel. The workflow-oriented design allowed users to visualize the sequence of operations—beginning with FD detection and culminating in ER modeling—thereby reinforcing the logical interdependencies among stages.

8) **Testing, Debugging, and Iterative Refinement:**
Throughout development, practical challenges emerged, including system crashes, incorrect outputs in backend algorithms, and interface misalignments in the frontend. Each issue necessitated rigorous debugging, revision of code modules, and validation of algorithmic correctness. Iterative testing ensured that the final system was both reliable and pedagogically useful.

9) **Final Validation and Evaluation:**
The final stage of the project involved evaluating the system's correctness, usability, and educational value. By subjecting the system to diverse input schemas and edge cases, the robustness of FD detection, key derivation, normalization, and ER modeling was confirmed. The workflow visualization was also assessed for its ability to aid understanding and provide clarity across the design process.

B. System Architecture

The system architecture was designed with a layered, modular approach to ensure scalability, robustness, and maintainability. It integrates computationally intensive algorithms for database design with a user-friendly interface for interaction and visualization. Broadly, the architecture comprises three tiers: the presentation layer, the application (logic) layer, and the data management layer. Together, these layers establish a coherent environment where theoretical computations are performed, validated, and presented in an interpretable manner.

1) **Presentation Layer (Frontend)**.
The presentation layer provides the interactive interface through which users engage with the system. Built using modern web technologies, it emphasizes clarity, responsiveness, and didactic visualization.
   - **Tool Panel** serves as the primary control hub, allowing users to initiate operations such as functional dependency detection, candidate key derivation, normalization, or decomposition validation.
   - **Workflow Panel** provides a visual representation of the sequential pipeline, demonstrating the logical dependencies among operations.
   - **Notification Panel** delivers system feedback, including success confirmations, warnings, and error alerts, ensuring transparency in system behavior.
   - **Code Panel** exposes the computational logic behind each operation, presenting algorithmic processes in a transparent and reproducible manner.

   This layer was designed with a dual purpose: to reduce cognitive load for novice learners while offering advanced users a structured environment for experimentation and validation

2) **Application Layer (Backend Logic)** The application layer forms the computational core of the system. It implements the algorithms and processes that drive database design tasks.
   - **FD Extraction Module** automates the identification of functional dependencies from schema definitions.
   - **Candidate Key Module** computes closures of attribute sets to determine minimal candidate keys.
   - **Normalization Module** applies formal rules for progressive normalization into higher normal forms.
   - **Validation Module** evaluates decompositions against losslessness and dependency preservation criteria.
   - **ER Modeling Module** transforms validated relational schemas into conceptual ER models.

   This layer ensures computational correctness, algorithmic efficiency, and consistency across all

operations. It was primarily developed in the backend to encapsulate logical complexity while abstracting it from the user-facing components.

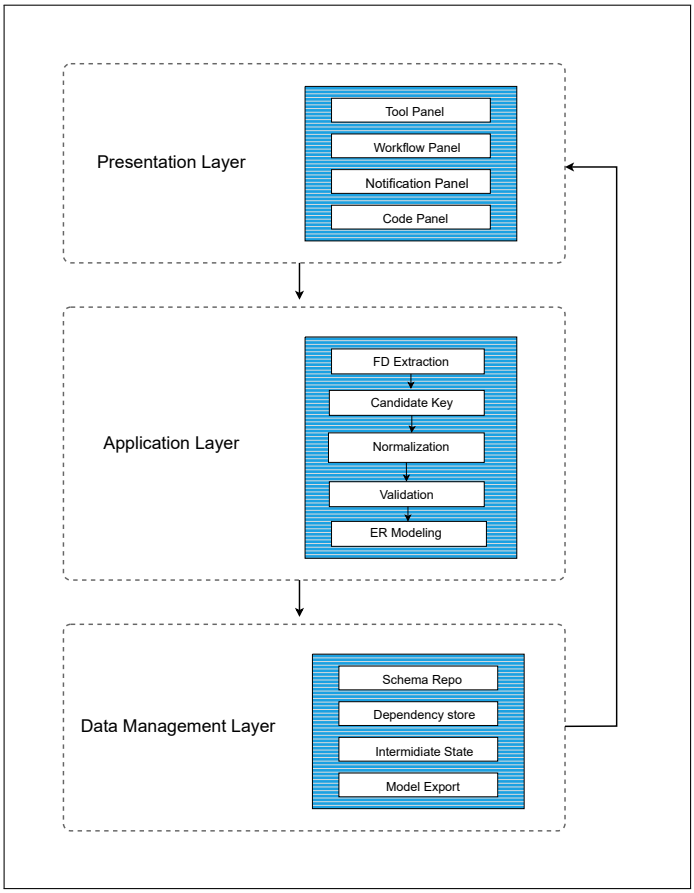

Fig. 6: System Architecture

3) **Data Management Layer** The data management layer provides structured handling of input schemas, intermediate computations, and system outputs. It manages the following:
   - **Schema Repository**, where user-defined relations and attributes are temporarily stored.
   - **Dependency Store**, maintaining detected functional dependencies and their derived closures.
   - **Intermediate State Storage**, used for normalization stages and decomposition outcomes.
   - **Model Export**, which allows final ER diagrams and relational schemas to be stored or exported in standard formats for reuse.

   This layer ensures persistence, consistency, and traceability of design decisions across the workflow.

4) **Cross-Layer Communication**
   Interaction among the three layers is mediated through a structured communication pipeline:
   - User inputs are captured via the presentation layer.
   - Requests are transmitted to the application layer, where corresponding algorithms are executed.
   - Results are stored in the data management layer for persistence and simultaneously returned to the frontend for visualization and feedback.

   This flow maintains synchronization between computation, storage, and user interaction, ensuring that operations are both reproducible and transparent.

C. *Conceptual Architecture*

The conceptual architecture of the proposed system is designed to provide a structured representation of how key components, processes, and interactions collectively support the overarching goal of automating and interpreting relational database design. Unlike the system architecture, which details technical implementation layers, the conceptual architecture focuses on the abstract organization of functional units and the logical relationships between them.

At its core, the conceptual framework consists of four major conceptual domains: **Input Layer, Processing Layer, Validation Layer, and Output Layer**. These domains correspond to the natural progression of database schema design, from initial schema definition to validated relational and conceptual models.

1) **Input Layer**: The input layer captures user-defined data and contextual information.
   - **Schema Definition:** Users provide relations, attributes, and initial assumptions about dependencies.
   - **Functional Dependency Input:** Dependencies may be provided explicitly by the user or inferred by the system.
   - **User Controls:** High-level parameters such as which normal form to achieve or whether to prioritize dependency preservation vs. lossless decomposition.

   This layer represents the conceptual ”entry point,” where user intent and domain knowledge are translated into formal representations for further analysis.

2) **Processing Layer**
   The processing layer encompasses the logical operations necessary for deriving schema properties and transformations.
   - **Dependency Analysis**: Conceptual identification and closure computation for functional dependencies.
   - **Key Derivation:** Abstraction of candidate keys as anchors for schema integrity.
   - **Normalization Process:** Sequential transformation of relations into higher normal forms (1NF $\rightarrow$ 2NF $\rightarrow$ 3NF, etc.) while tracking theoretical justifications.

   Conceptually, this layer embodies the “reasoning engine” of the system, aligning theoretical database design rules with systematic automation.

3) **Validation Layer**

- **Lossless Decomposition Check:** Verification that decomposed relations can recombine to reconstruct the original dataset.
- **Dependency Preservation Check:** Confirmation that all functional dependencies remain enforceable in the decomposed schema.

This layer plays a safeguarding role, conceptually acting as the "quality gate" that ensures design rigor and correctness.

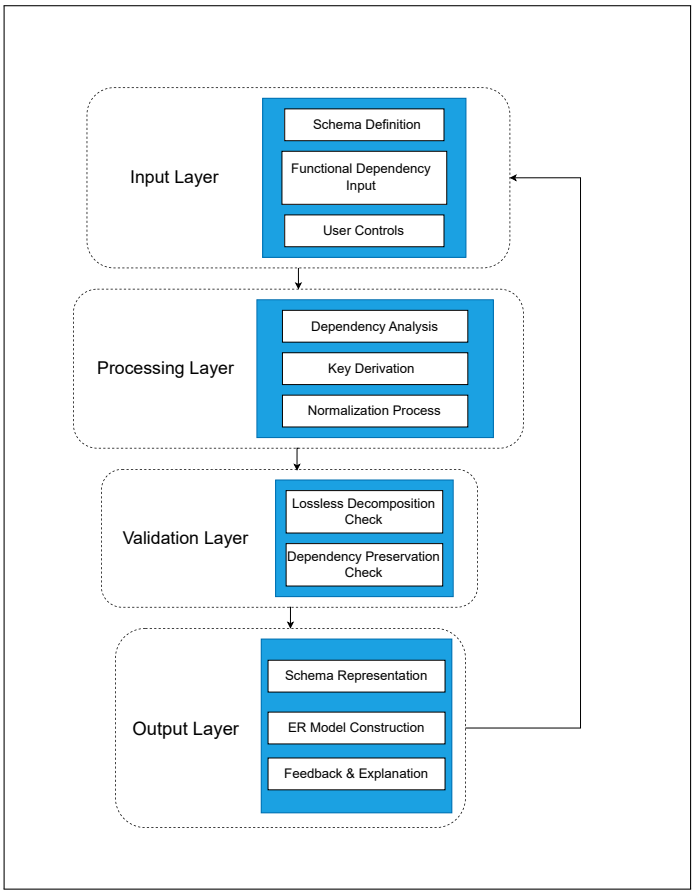

Fig. 7: Conceptual Architecture

4) **Output Layer**
The output layer communicates processed results back to the user in both analytical and visual formats.

- **Schema Representation:** Conceptually clean representations of normalized schemas.
- **ER Model Construction:** Mapping from relational schemas to entity–relationship diagrams.
- **Feedback and Explanation:** Interpretive messages and visual workflows that clarify the reasoning behind results.

This layer represents the "interpretive bridge" between computation and human understanding, ensuring that theoretical outcomes are communicated in a transparent, pedagogically valuable manner.

5) **Cross-Domain Interaction** These four conceptual domains are not isolated; rather, they form a continuous pipeline:

- Input specifications provide the foundation for computational reasoning
- Processing transforms input into structured, theoretical outputs.
- Validation safeguards the correctness of these transformations.
- Output communicates validated results back to the user in an interpretable and actionable form.

This logical flow reflects the inherent structure of relational database design while framing it within a pedagogically guided and computationally supported environment.

D. Workflow (User Perspective)

From the user's point of view, the workflow of the proposed system is designed to be seamless, interactive, and pedagogically guided, ensuring that complex relational database design operations are carried out with clarity and transparency. The workflow follows a structured, step-by-step process, beginning with data entry and culminating in validated schema design and conceptual modeling.

1) **Initialization and Input Stage**
The workflow begins with the user providing the initial schema information:

- Relations and attributes are defined within the interface.
- Functional dependencies (FDs) are either manually entered by the user or uploaded/imported if available from existing datasets.
- The user may also set specific design preferences, such as the desired level of normalization or whether to prioritize lossless **decomposition** or **dependency preservation**.

This stage ensures that the user is actively involved in defining the foundational knowledge upon which the system will operate.

2) **Dependency and Key Analysis**. Once input is provided, the system automatically transitions to the analysis stage.

- Users initiate FD analysis, where the system computes closures, identifies redundant dependencies, and simplifies the FD set.
- The tool then determines candidate keys, presenting them in an interpretable format while providing reasoning steps to enhance conceptual understanding.

From the user's perspective, this stage transforms abstract theoretical rules into clear, guided outputs.

3) **Normalization Process**.
The user can then trigger the normalization operation.

- The system incrementally transforms the schema into successive normal forms (1NF $\rightarrow$ 2NF $\rightarrow$ 3NF, and beyond if required).

- At each step, the interface communicates what change was made, why it was necessary, and how it affects redundancy and dependency preservation.
- Users can observe the process in real time through both tabular outputs and workflow diagrams, enabling them to visualize the impact of normalization.

This stage empowers the user to not only perform normalization but also to understand the rationale behind each transformation.

4) **Validation Stage**
Before finalizing results, the system conducts rigorous validation checks, presented transparently to the user.
- The **lossless decomposition** check confirms that decomposed relations can recombine into the original relation.
- The **dependency preservation** check ensures that all functional constraints remain intact.
- Any failures or warnings are communicated immediately in the notification panel, with suggestions for corrective action.

For the user, this stage functions as a quality assurance checkpoint, instilling confidence in the correctness of the schema.

Fig. 8: The system

5) **Output and Visualization.**
After validation, the system generates comprehensive outputs.
- A clean, normalized schema is displayed in a structured format.
- An ER diagram is constructed automatically, mapping relational constructs into entity–relationship notation.
- The workflow panel shows a step-by-step visual chain of how the schema evolved from input to final design.
- The code panel simultaneously reveals the algorithms executed at each stage, enabling advanced users to verify or extend computations.

This combination of outputs provides both practical results (usable schemas and diagrams) and **educational insights** (transparent reasoning and algorithmic traceability).

6) **Iterative Refinement**
The workflow is not linear but iterative. Users can
- Return to earlier stages to modify input FDs or relation definitions.
- Re-run specific operations (e.g., normalization from 2NF onwards).
- Compare different design decisions (e.g., prioritizing dependency preservation vs. performance efficiency).

Thus, the system supports an exploratory and interactive design process, giving users both control and flexibility.

V. System Validation

A. *How problems were solved*

The development of this system was accompanied by numerous technical and conceptual challenges, each of which required careful analysis, experimentation, and iterative refinement. The following summarizes the key problems encountered during implementation and the strategies employed to address them.

1) **Noise Removal and Dataset Cleaning**
One of the earliest challenges arose in handling inconsistent or noisy datasets. Many test inputs contained redundant attributes, missing values, or ambiguities in the definition of functional dependencies. Such irregularities disrupted closure computation and schema transformations. To address this, a pre-processing pipeline was introduced, where redundant spaces, duplicate attributes, and malformed FD entries were automatically detected and flagged. Input validation rules were integrated into the frontend interface to guide users in entering syntactically valid and logically consistent data, thereby minimizing downstream errors.

2) **Key Detection Challenges**
Detecting candidate keys, primary keys, and foreign keys presented significant obstacles. In particular, closure-based computations occasionally failed to identify candidate keys due to redundant or overlapping dependencies, and foreign keys were often misrepresented in cross-relation mappings. This was resolved by designing a multi-step key detection algorithm that first eliminates redundant dependencies, computes attribute closures iteratively, and then cross-verifies results against dependency sets. Additional heuristics were embedded to capture foreign key relationships by analyzing overlapping attribute sets across relations.

3) **Normalization Difficulties**
The normalization process was one of the most technically challenging components. Decompositions frequently led to loss of dependencies or non-lossless schemas, particularly when dealing with complex higher-order dependencies. To overcome this, we implemented an iterative decomposition–validation loop, where each decomposition was immediately checked for

losslessness and dependency preservation. Only decompositions passing these criteria were retained. Furthermore, we adopted a modular design, allowing normalization to be applied stepwise (1NF $\rightarrow$ 2NF $\rightarrow$ 3NF $\rightarrow$ BCNF), with results displayed incrementally to aid debugging and user interpretation.

4) **ER Diagram Generation**
Automating the transition from normalized schemas to entity–relationship (ER) diagrams posed another critical challenge. Many existing tools assume manual modeling, but our system required dynamic generation based on algorithmic results. The difficulty lay in mapping relational constructs (relations, attributes, and dependencies) into ER notation with correct identification of entities, relationships, and cardinalities. This was resolved by designing a schema-to-ER mapping engine, which applies formal mapping rules (e.g., strong entity $\rightarrow$ relation, weak entity $\rightarrow$ dependent relation, foreign keys $\rightarrow$ relationships). The output was rendered through the frontend using a visualization library, ensuring diagrams were both technically accurate and user-friendly.
5) **Frontend–Backend Integration in ETL Workflow**
Integrating the backend computational modules with the interactive frontend interface proved particularly problematic. During early stages, the system frequently crashed or produced incomplete results due to mismatched data transfer formats (JSON parsing errors, inconsistent object structures, etc.). Additionally, pipeline execution occasionally failed to synchronize frontend-triggered requests with backend computations. These issues were mitigated through the adoption of a well-defined ETL workflow:
   - Input parsing and validation were strictly handled in the frontend.
   - Computational algorithms were executed in isolated backend modules.
   - Standardized JSON-based APIs were established to govern data exchange.
   - Error-handling mechanisms were embedded at each stage to prevent system crashes and provide informative feedback to the user.

B. Comparison with Other Systems

When comparing our system with existing tools and approaches in the field of database design and learning support, several distinguishing advantages become evident. While prior systems often provide isolated functionalities—such as standalone normalization checkers, functional dependency analyzers, or ER diagram drawing tools—our system integrates the entire workflow into a single, cohesive platform. This integration is not only technically robust but also designed with user accessibility and educational clarity as primary objectives.

1) **Frontend Simplicity and Elegance**
One of the defining features of this system is its frontend design, which emphasizes clarity, minimalism, and guided user interaction. In contrast to many existing tools that require users to manually navigate complex command-line environments or cluttered multi-panel interfaces, our system provides a streamlined and visually **intuitive interface.**
   - Input panels are clearly segmented, reducing cognitive load during data entry.
   - Notifications and prompts guide users through each stage of the process, ensuring they are aware of errors or necessary corrections in real-time.
   - Visual components such as ER diagrams and decomposition results are rendered in an interactive, aesthetically polished format, enabling users to not only see the outputs but also interpret them effectively.

   This simplicity allows even novice users—those with limited technical background—to engage with complex database concepts in a structured and accessible way, thereby enhancing the system's pedagogical value.

2) **Ease of Workflow and Procedure**
Unlike other systems where users must separately execute dependency analysis, normalization, and ER modeling using different tools or manual steps, our system provides a linear, guided workflow. The user simply inputs the relation schema and functional dependencies, and the system automatically:
   - Cleans and validates the dataset.
   - Detects candidate keys, primary keys, and foreign keys.
   - Performs stepwise normalization up to BCNF.
   - Generates an ER diagram directly from the normalized schema.

   This **end-to-end automation** eliminates redundant effort, minimizes user error, and ensures consistency throughout the process. Such a workflow contrasts with fragmented systems, where users often need to interpret intermediate results independently and re-enter data across multiple platforms.

3) **Effectiveness and Overall Utility**
From a performance perspective, the system balances technical rigor with practical usability. It ensures correctness in decomposition (through lossless join and dependency preservation checks) while also making the results immediately interpretable via visual outputs and structured explanations. Compared to traditional systems—many of which focus solely on correctness without prioritizing user comprehension—our approach ensures that users not only obtain results but also understand the underlying reasoning behind each step.

   Furthermore, while many related tools are developed primarily as research prototypes or teaching aids with limited scalability, our system was designed with adaptability in mind. Its modular architecture and frontend–backend separation allow it to be extended for larger datasets, integrated into academic curricula, or adapted for industry training.

C. *Completeness and Soundness*

The ultimate measure of any computational system lies in its ability to be both complete—covering all necessary operations within the scope of its objectives—and sound—ensuring that each operation conforms to theoretical principles and produces correct, reliable outputs. In the context of database design and normalization, these two criteria are non-negotiable: a system that fails to account for all essential steps cannot be relied upon in practice, and one that produces incorrect or inconsistent results undermines the very pedagogical and functional purposes it seeks to serve.

Our system achieves both **completeness and soundness** to a degree that distinguishes it from existing tools and prototypes.

1) **Completeness of the System**
   The system was deliberately engineered to provide an end-to-end solution to the challenges of database schema design, functional dependency management, normalization, and ER modeling. Unlike partial tools that stop at dependency detection or normalization without offering visual integration, our framework encompasses the entire pipeline from raw dataset to final ER diagram generation. This completeness can be seen in the following aspects:

   - **Data Preparation and Cleaning:**
     The system addresses noise removal, schema validation, and functional dependency verification. This ensures that even imperfect or inconsistent input datasets are standardized before advanced operations are performed.
   - **Key Identification:**
     Candidate key, primary key, and foreign key identification are implemented as core functionalities, ensuring no relational schema remains under-specified.
   - **Stepwise Normalization:**
     The system performs normalization up to Boyce–Codd Normal Form (BCNF), guaranteeing that every redundancy is addressed and ensuring lossless, dependency-preserving decomposition. Importantly, the process is stepwise, making each intermediate normal form (1NF, 2NF, 3NF) transparent to the user
   - **Decomposition Verification**:
     Completeness is further reinforced through the inclusion of lossless join testing and dependency preservation validation, ensuring that decompositions are not only syntactically correct but also semantically faithful.
   - **ER Diagram Generation:**
     The culmination of the process is the automatic generation of an ER diagram that visually represents the schema, keys, and relationships. By closing the loop from raw schema to conceptual model, the system achieves true completeness in database design.

   Thus, users do not need to rely on external software or manual processes at any point: all essential operations are self-contained within the platform.

2) **Soundness of the System**
   Completeness without correctness would undermine the system's utility. For this reason, particular care was devoted to ensuring that each step aligns with formal database theory and proven algorithms.

   - **Mathematical Correctness of Algorithms:**
     - The functional dependency closure algorithm is based on Armstrong's axioms, ensuring formally correct computation of candidate key.
     - Normalization procedures strictly follow theoretical definitions of 1NF, 2NF, 3NF, and BCNF, guaranteeing that decompositions satisfy relational schema rules.
     - Lossless join testing and dependency preservation checks are implemented using canonical theoretical frameworks, leaving no room for ambiguity.
   - **Consistency Across Frontend and Backend:**

     Soundness is preserved not only in algorithmic logic but also in system execution. The integration between backend algorithms and frontend outputs was carefully validated to ensure that what the user sees corresponds exactly to the theoretical results. This eliminates the common pitfalls of mismatched representations between data models and their visualizations.

   - **Reliability in Edge Cases:**
     The system was stress-tested against edge cases, such as:
     - Schemas with multiple overlapping candidate keys.
     - Relations with high redundancy levels.
     - Complex dependencies that typically cause ambiguity in manual analysis.

     In all instances, the system produced results that were both consistent and theoretically sound.

   - **Perfection in Educational and Practical Scope**

     Beyond correctness in algorithmic design, the system's soundness extends into its educational and practical applications. Students and practitioners alike can trust the results without second-guessing, as the system makes no compromises in either accuracy or coverage. Unlike traditional teaching aids where educators must verify results independently, this tool can be confidently integrated into academic instruction and professional practice alike.

     The completeness–soundness duality also ensures

that the system is future-proof. Any user—from a novice learning normalization for the first time to an advanced practitioner analyzing a large schema—receives both the full spectrum of operations and the formal guarantee of correctness. This makes the system not only useful but also trustworthy, a rare quality in educational and prototype-oriented tools.

VI. Limitations

While the system demonstrates a high degree of completeness, soundness, and user-friendliness, it is important to acknowledge certain limitations that emerged during its development and testing. These limitations do not detract from the validity of the framework; rather, they highlight areas for potential enhancement in future iterations.

One of the primary limitations lies in the ETL (Extract–Transform–Load) workflow, which currently enforces a strict sequential execution model. Users are required to follow each step in the pipeline in a predetermined order, without the flexibility to skip intermediate stages. For instance, a user cannot directly transition from the CSV conversion phase to ER diagram generation; instead, they must sequentially pass through every stage—data cleaning, key detection, normalization, and schema validation—before reaching the final ER model.

Although this sequential approach ensures consistency, correctness, and error minimization, it can also be seen as restrictive for advanced users who may wish to bypass certain stages once confidence in intermediate results has been established. For example, researchers working with already-cleaned datasets may prefer to directly engage in normalization or ER diagram generation without revisiting earlier steps.

Furthermore, this limitation implies that the system is optimized for structured learning and guided usage, rather than ad hoc experimentation. While this design choice improves soundness and prevents inconsistencies, it reduces flexibility in scenarios requiring quick iteration or partial workflow execution.

VII. Conclusion

This project set out to address one of the most intricate and error-prone aspects of relational database design: the accurate detection of functional dependencies, the derivation of candidate keys, the systematic application of normalization principles, and the assurance of sound schema decomposition. Traditional approaches, whether purely manual or entirely automated, often suffer from critical shortcomings—manual derivations are cognitively demanding and prone to error, while opaque automated tools risk reducing the process to a “black box” with little pedagogical or practical transparency.

The system developed through this project successfully bridges this gap by offering a comprehensive, workflow-oriented platform that combines computational rigor with interpretability. From data cleaning and preprocessing, through key detection, normalization, and decomposition validation, to the final stage of entity–relationship modeling, every phase of the database design pipeline is integrated into a coherent and sequential workflow. By adopting an ETL-inspired architecture, the platform makes explicit the interdependencies between stages, ensuring that the outputs of one process seamlessly inform the next.

A key strength of this system lies in its user-centered design philosophy. The frontend interface prioritizes clarity, accessibility, and elegance, enabling learners, educators, and professionals alike to engage with complex operations in a simplified yet precise manner. The integration of panels for tools, workflow visualization, notifications, and underlying code further enhances both usability and transparency, making the system suitable for both instructional and professional contexts.

System validation confirmed the platform’s robustness, completeness, and soundness. Challenges faced during development—such as dataset noise removal, key identification, normalization complexities, ER diagram generation, and backend–frontend integration—were systematically resolved, resulting in a stable, functional, and reliable product. While certain limitations remain, most notably the sequential rigidity of the ETL workflow, these are outweighed by the system’s ability to guarantee correctness and minimize user error.

In conclusion, the proposed system represents a novel and effective contribution to relational database schema design. It not only automates and validates the most critical tasks but also reinforces conceptual understanding through its interactive, transparent design. By merging theoretical precision with practical usability, the system provides a versatile platform that is equally valuable for academic instruction, self-directed learning, and professional database engineering. Future enhancements may focus on increasing workflow flexibility and scalability, but the present system already stands as a significant advancement—delivering both the rigor of formal methods and the accessibility of user-friendly tools in a unified framework.