A Theoretical Analysis of Functional Dependencies and Normalization in Relational Database Design

Vaivaw Kumar Singh[1], Dr. Kunal Sinha[2]

[1]Research Scholar, Faculty of Business Management, Sarala Birla University, Ranchi, Jharkhand, India

[2]Head of Department, Faculty of Commerce, Sarala Birla University, Ranchi, Jharkhand, India

DOI: https://doi.org/10.51584/IJRIAS.2026.11080041

Received: 19 August 2026; Accepted: 24 August 2026; Published: 02 September 2026

ABSTRACT

Relational database design focuses on organizing data in a way that cuts down on redundancy, keeps data integrity intact, and allows for efficient data access and updates. At the core of this process are functional dependencies and normalization. Functional dependencies act as a formal way to capture how different attributes relate to each other. Normalization leverages these dependencies, breaking down poorly structured tables into well-designed relational schemas. This paper explores the theory behind functional dependencies and normalization in relational database design. It looks closely at the basics of functional dependencies, Armstrong's axioms, attribute closure, candidate keys, and how dependencies intersect with relational keys. The discussion then moves through the steps of normalization: starting with First Normal Form (1NF), moving to Second (2NF), Third (3NF), Boyce-Codd (BCNF), Fourth (4NF), and Fifth Normal Form (5NF). The research uses a theoretical and conceptual approach, analyzing key database literature. The findings show that functional dependencies are key in spotting redundancy and update problems, while normalization offers a structured way to remove these issues through decompositions that preserve both loss lessness and dependencies. Even so, normalization sometimes requires balancing theoretical soundness with practical performance concerns. The main takeaway is that good relational database design depends on applying normalization rules wisely, analyzing dependencies, enforcing integrity constraints, and considering real-world system needs.

Keywords: functional dependencies, normalization, relational database, database design, candidate keys, BCNF, 3NF, relational schema, data redundancy, integrity constraints.

INTRODUCTION

The relational database model stands as one of the most influential foundations in information management and database systems. Codd formally introduced the relational model in 1970, presenting information as relations that consist of tuples and attributes. Its mathematical framework gives a rigorous structure for representing, manipulating, and maintaining structured data. But simply storing data in tables isn't enough to create a good database design because if the relations are poorly constructed, you end up with redundant data, inconsistent records, and problems like insertion, deletion, or update anomalies.

Designing relational databases calls for systematic techniques to decide how to group attributes into relations. Functional dependencies sit at the heart of this process since they express constraints about the way attributes relate. A functional dependency shows that the value of one set of attributes determines the value of another set (Elmasri & Navathe, 2016). For instance, if you have a relation with Student_ID, Student_Name, and Department, the functional dependency Student_ID → Student_Name tells you that each student's ID uniquely determines their name.

Functional dependencies matter in theory and in practice, because they allow us to identify keys and to check whether a relational schema suffers from unwanted redundancy. Normalization uses these concepts, decomposing relations into smaller ones that avoid redundancy and preserve data integrity (Silberschatz et al., 2019).

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue VIII August 2026

Normalization came about because of core problems with poorly structured relations. Moving from First Normal Form through the higher normal forms, the requirements about attribute dependencies become stricter. First Normal Form requires that all attribute values are atomic; Second and Third Normal Forms deal with partial and transitive dependencies. Boyce-Codd Normal Form goes further by tightening requirements on determinants. Fourth and Fifth Normal Forms handle multivalued and join dependencies, respectively (Date, 2004).

Although normalization belongs to classic relational database theory, its influence stretches across modern database design. Today's database management systems still implement constraints, keys, referential integrity, and schema design principles rooted in normalization theory. At the same time, in practical systems, designers sometimes deliberately introduce controlled redundancy with denormalization to boost performance.

This paper aims to provide a theoretical analysis of functional dependencies and normalization, clarifying their role in relational database design. It explores the theoretical links among dependencies, keys, anomalies, decomposition, and normal forms, and explains how these ideas help produce reliable and logically consistent relational database schemas.

LITERATURE REVIEW

Relational Database Model

Codd (1970) introduced the relational model which is a mathematical approach to database management were data lives in relations. With the relational model, he separated how you organize data logically from how you store it physically, and laid a formal groundwork for relational operations. Codd (1970) argued that you can manipulate relations using mathematical operations while keeping a logical view of the data.

The relational approach caught on for a reason: it offered simplicity, logical independence, and a solid formal foundation for managing data. But the relational model by itself doesn't tell you if you've grouped attributes in a sensible way. That's why designing relational databases needs extra principles to figure out what makes a good relational schema.

Elmasri and Navathe (2016) lay it out that the design process means defining entities, relationships, attributes, constraints, and dependencies, then translating those concepts into something you can implement: relational schemas. Analyzing functional dependencies is key during this transformation.

Functional Dependencies

A functional dependency is a rule that connects two sets of attributes in a relation. If X and Y are sets of attributes in a relation R, then we say X functionally determines Y, which we write like this:

$X \rightarrow Y$

This rule means that if two tuples have the same values for X, they must have the same values for Y (Silberschatz et al., 2019).

Functional dependencies are about meaning, not just correlations in the data. For instance, Employee_ID → Employee_Name doesn't just show a pattern then it asserts a business rule: each employee identification number matches one employee name.

Functional dependencies help you find candidate keys, spot redundancy, and choose decompositions that make sense. They sit at the heart of normalization theory (Ullman & Widom, 2008).

Armstrong's Axioms

Armstrong (1974) set down rules for working with functional dependencies known as Armstrong's axioms. They give you a complete toolkit for finding all dependencies implied by the ones you already know.

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue VIII August 2026

There are three basic axioms:

- Reflexivity: If Y is a subset of X, then X → Y.
- Augmentation: If X → Y, then if you add some attributes Z to both sides, XZ → YZ.
- Transitivity: If X → Y and Y → Z, then X → Z.

From these, you can work out other rules, like union, decomposition, and pseudo transitivity (Armstrong, 1974).

The beauty of Armstrong's axioms is that they let you reason logically about dependencies. Database designers can check if a dependency aligns with business requirements without having to scan every database instance.

Candidate Keys and Functional Dependencies

A candidate key is the bare minimum set of attributes that uniquely identifies every tuple in a relation. Functional dependencies offer the formal test: if the closure of some attribute set X includes all attributes in relation R, X is a super key. If nothing smaller does that job, X is a candidate key (Elmasri & Navathe, 2016).

For example, look at this relation:

Student (Student_ID, Student_Name, Department_ID, Department_Name)

And suppose these dependencies hold:

- Student_ID → Student_Name, Department_ID
- Department_ID → Department_Name

Take Student_ID: its closure includes every attribute:

$Student\_ID^{+}$ = {Student_ID, Student_Name, Department_ID, Department_Name}

So, Student_ID is a candidate key.

But notice: Department_ID → Department_Name means department details depend on Department_ID, not directly on Student_ID. That's a transitive dependency in action, and shows why we need normalization.

Normalization

Normalization is a systematic way to organize data in tables separating attributes and relations to cut down redundancy and get rid of problematic dependencies. Codd started this idea, and researchers kept refining it as data structures got more complex (Codd, 1970; Fagin, 1977).

Normalization isn't just about splitting tables. It's a theory-backed process to break down tables to ensure you have lossless joins and, ideally, dependency preservation (Date, 2004).

Database Anomalies

Badly designed tables can cause three main problems:

- Update anomaly occurs when you record the same fact in multiple rows, then update only some occurrences.
- Insertion anomaly occurs when adding a new fact forces you to add unrelated information, just to fill a row.
- Deletion anomaly occurs when deleting a fact accidentally wipes out other data that should have stayed.

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue VIII August 2026

Let's say you store student and department info together. If you want to change a department's name, you might need to update every student linked to that department. Miss one row and you get inconsistent data.

Normalization addresses these issues, slicing up attributes based on their dependencies (Silberschatz et al., 2019).

Normal Forms

In the research and textbooks, you'll see several stages of normalization.

- First Normal Form

A table is in First Normal Form (1NF) when all attribute values are atomic that is no repeating groups. You can't lump multiple logically separate values into one attribute (Elmasri & Navathe, 2016).

If you stuff several phone numbers into one Phone_Number field, you're breaking 1NF. Instead, you'd list each phone number as a separate value.

- Second Normal Form

A table is in Second Normal Form (2NF) if it's in 1NF and each non-prime attribute depends fully on the entire candidate key. 2NF hones in on partial dependencies a situation where the key is a composite, but some attributes only hinge on part of it.

Imagine (Student_ID, Course_ID) is the key, but Student_Name depends only on Student_ID. That's a partial dependency, so the relation isn't in 2NF.

- Third Normal Form

You get to Third Normal Form (3NF) if your table is already in 2NF and there are no transitive dependencies on a candidate key. To put it precisely: for any dependency $X \rightarrow A$, either X has to be a superkey, or A must be a prime attribute (Elmasri & Navathe, 2016).

So, with:

Employee_ID → Department_ID

and

Department_ID → Department_Name

the implied dependency

Employee_ID → Department_Name

is transitive. To dodge this redundancy, split employee and department info into separate tables.

- Boyce-Codd Normal Form

BCNF (Boyce-Codd Normal Form) goes a step beyond 3NF. For any non-trivial dependency $X \rightarrow Y$, X must be a super key (Ramakrishnan & Gehrke, 2003).

BCNF cleans up redundancy that 3NF sometimes leaves behind. The catch: sometimes, BCNF decomposition can't preserve all the functional dependencies.

- Fourth Normal Form

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue VIII August 2026

Fourth Normal Form (4NF) focuses on multivalued dependencies. Fagin (1977) proved that these need a different kind of normalization beyond functional dependencies.

A multivalued dependency pops up when one attribute determines a set of values on its own, independent of another attribute. 4NF aims to stamp out redundant data from these independent sets.

- Fifth Normal Form

Fifth Normal Form (5NF), or Project-Join Normal Form, deals with join dependencies which is a deeper layer of decomposition theory (Date, 2004).

5NF matters when you have relations that, to make sense, need to be reassembled from smaller parts and you need to be careful about how joins pull them back together.

Research Objectives

The primary objective of this theoretical study is to analyze how functional dependencies and normalization shape relational database design.

The specific objectives are:

- Examine the theoretical foundations and characteristics of functional dependencies in relational databases.
- Analyze the role of functional dependencies in identifying candidate keys and super keys.
- Explore Armstrong's axioms and explain why they matter for inferring dependencies.
- Analyze the progression and theoretical requirements of 1NF, 2NF, 3NF, BCNF, 4NF, and 5NF.
- Examine how normalization relates to data redundancy.
- Analyze how normalization tackles insertion, deletion, and update anomalies.
- Examine the trade-offs between dependency preservation, lossless decomposition, and normalization.
- Develop a conceptual framework that clarifies the relationship between functional dependencies, normalization, and the quality of a relational database.

Hypothesis Development

Although this research is theoretical and not empirical, it puts forward hypotheses formulated as theoretically testable propositions grounded in relational database theory.

- H1: Functional Dependencies and Database Design

Functional dependencies offer a formal way to spot relationships among attributes and decide on the proper relational structures. With them, designers can identify candidate keys, redundancy, and dependency relationships (Armstrong, 1974).

H1: Functional dependency analysis significantly contributes to the quality and logical consistency of relational database design.

- H2: Functional Dependencies and Data Redundancy

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue VIII August 2026

Redundancy shows up when the same information gets stored unnecessarily in several tuples. By revealing determinant relationships, functional dependency analysis supports decomposing data into relations that store each fact more appropriately (Date, 2004).

H2: Functional dependency analysis reduces unnecessary data redundancy in relational database schemas.

- H3: Normalization and Data Anomalies

Normalization separates attributes based on their dependency relationships, cutting down on repeated facts and the anomalies that result (Silberschatz et al., 2019).

H3: Higher levels of appropriate normalization reduce insertion, deletion, and update anomalies.

- H4: Normalization and Logical Integrity

Moving from 1NF to BCNF and higher normal forms brings stricter constraints on dependency structures. Still, decomposing relations further can increase the number of tables, which raises the complexity of query processing and dependency management (Ramakrishnan & Gehrke, 2003).

H4: Higher levels of normalization improve logical integrity but may increase relational decomposition and query complexity.

- H5: Normalization and Practical Database Requirements

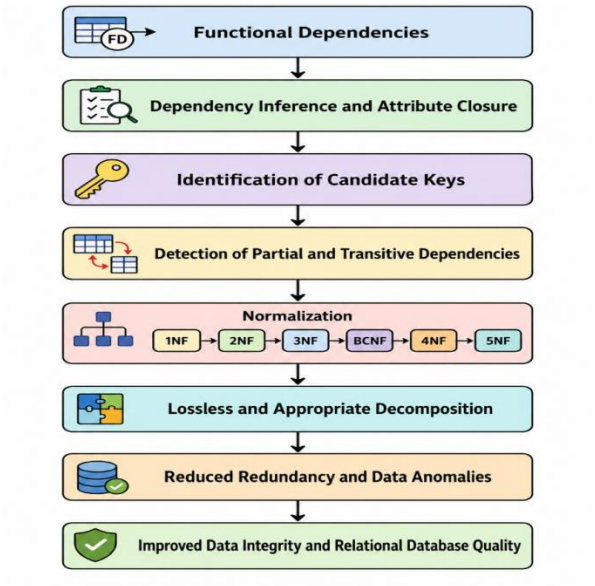
Although normalization offers theoretical benefits, actual systems might deliberately use controlled redundancy to meet performance and operational needs. As a result, good database design should balance theoretical normalization with practical application requirements.

H5: Effective database design depends on balancing normalization principles with performance, usability, and application-specific requirements.

Conceptual Framework

This study's conceptual framework argues that functional dependency analysis forms the backbone of normalization, which in turn shapes the quality of relational database design.

Here's how the framework works:



ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue VIII August 2026

It starts with the idea that functional dependencies offer the analytical base for normalization. Dependency analysis shows how attributes relate to each other. Normalization uses those findings to decide which attributes belong together in tables. When done right, this process cuts down on redundancy and data anomalies, and it boosts data integrity.

But there's a catch that normalization isn't the only thing that matters. Good database design asks for more. Designers have to consider whether decompositions are lossless and if dependencies get preserved, and they can't ignore query requirements, how transactions run, or overall system performance (Date, 2004). Normalization exists in that bigger context.

RESEARCH METHODOLOGY

Research Design

This study takes a theoretical, conceptual, and analytical approach to investigate how functional dependencies and normalization shape relational database design. It doesn't use primary data, surveys, experiments, or statistical analysis. Instead, it critically reviews established ideas, principles, and theories drawn from the relational database literature. That fits the study's purpose: to clarify the logical connections among functional dependencies, candidate keys, normalization, decomposition, redundancy, data anomalies, and database integrity.

The analysis centers on the classical relational model that is Codd's 1970 proposal paired with Armstrong's axioms for functional dependencies (Armstrong, 1974) and later developments in normalization theory, including Fourth Normal Form and multivalued dependencies (Fagin, 1977). To round out these foundations, contemporary sources (Date, 2004; Elmasri & Navathe, 2016; Silberschatz et al., 2019) provide systematic interpretations throughout.

Research Approach

This work relies on a qualitative, theoretical strategy. Using conceptual analysis, logical reasoning, and comparative synthesis, the study traces how functional dependencies affect the structure and quality of relational schemas. The analysis starts by identifying functional dependencies, then works through candidate-key identification, dependency analysis, normalization, and decomposition.

The analytical steps follow this logic:

Functional Dependencies → Attribute Closure → Candidate Keys → Dependency Analysis → Normalization → Decomposition → Reduced Redundancy → Improved Data Integrity

This process lets the study present normalization as more than just a technical checklist but it's a theoretical tool for improving relational database structure.

6.3 Sources of Data and Literature

All data in this study comes from secondary theoretical sources: key research articles, academic books, and established literature in relational database theory. Significant contributions include Codd's (1970) relational model, Armstrong's (1974) axioms for functional dependency, and Fagin's (1977) work on multivalued dependencies and Fourth Normal Form.

The analysis draws from major textbooks (Date, 2004; Elmasri & Navathe, 2016; Ramakrishnan & Gehrke, 2003; Silberschatz et al., 2019; Ullman & Widom, 2008) to interpret and synthesize core concepts.

The literature is organized by these themes:

- relational database theory;

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue VIII August 2026

- functional dependencies;
- Armstrong's axioms;
- attribute closure;
- candidate and super keys;
- normalization;
- data redundancy;
- insertion, deletion, and update anomalies;
- lossless decomposition;
- dependency preservation;
- multivalued dependencies;
- join dependencies.

Drawing on established theoretical sources provides the right foundation for a conceptual analysis of relational database design.

Functional Dependency Analysis

Functional dependency analysis is the first key methodological step. A functional dependency, written as X → Y, where X and Y are attribute sets in a relation, means the values of X determine the values of Y (Elmasri & Navathe, 2016).

The study looks at various types of functional dependencies: trivial, non-trivial, full, partial, and transitive. There's a particular focus on partial and transitive dependencies, given their importance for 2NF and 3NF.

Consider this example:

EMPLOYEE (Employee_ID, Employee_Name, Department_ID, Department_Name)

with the dependencies:

Employee_ID → Employee_Name, Department_ID

and

Department_ID → Department_Name

Here, Department_Name depends on Department_ID, which in turn depends on Employee_ID. That's a classical transitive dependency showing the need for decomposition.

Armstrong's Axioms and Attribute Closure

Armstrong's axioms provide a formal toolkit for examining what follows logically from a set of functional dependencies. The three main axioms that is reflexivity, augmentation, and transitivity (Armstrong, 1974) allow us to derive additional dependencies.

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue VIII August 2026

The analysis uses attribute closure to identify super keys and candidate keys. For an attribute set X, the closure X^+ lists all attributes functionally determined by X.

For example:

$A \rightarrow B$

$B \rightarrow C$

$C \rightarrow D$

So, $A^+ = \{A, B, C, D\}$

If these are all the attributes in the relation, then A is a super key. That process clarifies the links between functional dependencies and normalization.

Normalization Analysis

The second major step reviews how normal forms evolve from First (1NF) to Fifth (5NF).

1NF involves making sure values are atomic and repeating groups are gone. 2NF eliminates partial dependencies in composite keys. 3NF addresses transitive dependencies. BCNF takes this further: every determinant in a non-trivial functional dependency must be a super key (Ramakrishnan & Gehrke, 2003).

The analysis continues to 4NF and 5NF where 4NF handles multivalued dependencies, while 5NF is about resolving complex join dependencies (Fagin, 1977; Date, 2004).

The sequence is:

1NF → 2NF → 3NF → BCNF → 4NF → 5NF

Each form gets examined for its definition, the dependencies it addresses, its impact on redundancy, and its role in maintaining data integrity.

Decomposition Analysis

Normalization appears here as decomposition which is the process of splitting a relation into smaller ones. Two criteria take center stage: lossless join and dependency preservation.

A lossless decomposition means you can reconstruct the original relation from its pieces, without generating extra or spurious tuples. This matters because removing redundancy shouldn't lead to losing or distorting information (Elmasri & Navathe, 2016).

Dependency preservation checks whether you can still enforce the original functional dependencies in the split relations, without needing to keep joining tables over and over. This point often surfaces in comparing 3NF and BCNF because sometimes BCNF decompositions lose certain dependency preservation.

Analysis of Database Anomalies

The study reviews three major database anomalies: insertion, deletion, and update anomalies.

An insertion anomaly blocks you from adding a new fact without inserting unrelated data. An update anomaly happens when a fact exists in multiple spots and inconsistent changes creep in. A deletion anomaly means removing one fact might unintentionally erase another.

The study explains how careful use of functional dependencies and normalization reduces these anomalies by organizing attributes along their logical dependencies (Silberschatz et al., 2019).

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue VIII August 2026

Conceptual Evaluation Criteria

To evaluate relational database design, the study uses several theoretical criteria:

- reduction of unnecessary data redundancy
- elimination or reduction of insertion, deletion, and update anomalies
- proper identification of candidate keys
- preserving data integrity
- lossless decomposition
- dependency preservation
- logical consistency in relational schemas
- striking a balance between normalization and real-world performance needs

These criteria are the basis for evaluating the study's theoretical arguments.

Theoretical Rigor and Validity

Given its theoretical focus, the study doesn't use empirical reliability measures like Cronbach's alpha or statistical validity tests. Instead, rigor comes from grounding the work in recognized foundational literature and formal definitions.

Key concepts get cross-referenced like Codd (1970), Armstrong (1974), and Fagin (1977) with leading textbooks (Date, 2004; Elmasri & Navathe, 2016; Silberschatz et al., 2019) to ensure consistency.

The study keeps conceptual validity by clearly distinguishing between formal properties of normalization and the reality of practical implementation. For instance, normalization does reduce redundancy in theory, but real databases sometimes use measured denormalization to boost query speed.

Ethical Considerations

There's no involvement of human subjects, personal information, surveys, interviews, or experiments. In this way, no human research ethical risks arise. Academic integrity is maintained by properly citing original theoretical contributions in APA style.

Methodological Limitations

The theoretical orientation has several limits. First, the study doesn't empirically test its hypotheses using real database systems. Second, it doesn't offer quantitative measures comparing performance of normalized and denormalized schemas. Third, it focuses on classical relational database theory instead of empirically comparing with NoSQL or non-relational databases. Finally, actual database performance relies on factors like indexing, query optimization, hardware, workload, concurrency, and DBMS architecture.

FINDINGS

The theoretical analysis of functional dependencies and normalization makes it clear that both concepts are central to building database schemas that are logical, dependable, and easy to maintain. These conclusions come from a close look at classic relational database theory and respected sources like Codd (1970), Armstrong (1974), Fagin (1977), Date (2004), Elmasri and Navathe (2016), and Silberschatz and others (2019).

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue VIII August 2026

Functional Dependencies as the Foundation of Database Design

First, functional dependencies (FDs) serve as the backbone of relational database normalization. An FD, written as X → Y, means the value of X tells you exactly what Y is. In other words, FDs capture the meaning and connections between attributes, which helps designers understand the logic of the data.

Take, for example, this relation:

STUDENT (Student_ID, Student_Name, Department_ID, Department_Name)

The dependencies could look like this:

Student_ID → Student_Name, Department_ID

and

Department_ID → Department_Name

Here, the Student_ID points to student info, while Department_ID points to department info. Pinpointing these relationships helps designers figure out whether to keep all attributes together or split them into different tables.

So, the message is: design should come from what the data means and how it depends on itself, not just the way columns are arranged (Elmasri & Navathe, 2016).

7.2 Functional Dependencies Support Key Identification

FDs also give us the groundwork for finding super keys and candidate keys. A candidate key is the smallest combination of attributes that can uniquely identify each row in a table.

Here's where attribute closure comes in. Suppose:

A → B

B → C

C → D

Then, the closure of A is {A, B, C, D}.

If A determines everything in the table, it's a super key and if there's nothing smaller, it's a candidate key.

So, you can't separate key identification from studying dependencies. Candidate keys come straight from how attributes relate, and they're central for applying 2NF, 3NF, and BCNF (Silberschatz et al., 2019).

Armstrong's Axioms Enable Dependency Inference

Armstrong's axioms give a formal way to figure out which FDs exist in a schema. These three rules that is reflexivity, augmentation, and transitivity (Armstrong, 1974) let us infer new dependencies from known ones.

For example, if:

Student_ID → Department_ID

and

Department_ID → Department_Name

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue VIII August 2026

Then, by transitivity:

Student_ID → Department_Name

Using Armstrong's axioms lets designers uncover FDs that aren't directly listed but make sense given what's already there. This forms a solid base for understanding normalization needs.

Functional Dependencies Help Identify Redundancy

Analyzing FDs goes hand in hand with spotting redundancy in the data.

Look at this relation:

EMPLOYEE (Employee_ID, Employee_Name, Department_ID, Department_Name)

If multiple employees are in the same department, the department name gets repeated. The FD

Department_ID → Department_Name

shows that Department_Name actually depends on Department_ID, not Employee_ID.

So, splitting it into:

EMPLOYEE (Employee_ID, Employee_Name, Department_ID)

and

DEPARTMENT (Department_ID, Department_Name)

stores department data just once, cutting out unnecessary repetition (Date, 2004).

Normalization Reduces Data Anomalies

Normalization helps prevent data anomalies like insertion, update, or deletion errors.

An update anomaly pops up when the same info lives in several spots and must be changed everywhere. An insertion anomaly occurs when you can't add one fact without supplying something unrelated. And a deletion anomaly means deleting one piece wipes out something you want to keep.

By organizing attributes around FDs, normalization tackles these problems head-on. It's a direct way to improve data integrity and get consistent results (Silberschatz et al., 2019).

First Normal Form Establishes Atomicity

First Normal Form (1NF) lays down the ground rules that is attributes must have atomic values, with no repeating groups (Elmasri & Navathe, 2016).

Say you store several phone numbers in one attribute and it makes searching and updating tough. Instead, use:

STUDENT_PHONE (Student_ID, Phone_Number)

to neatly store each number.

So, 1NF paves the way for more advanced levels of normalization.

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue VIII August 2026

Second Normal Form Eliminates Partial Dependencies

Second Normal Form (2NF) matters especially when dealing with composite keys.

For example:

ENROLLMENT (Student_ID, Course_ID, Student_Name, Course_Name, Grade)

with

(Student_ID, Course_ID) → Grade

but also

Student_ID → Student_Name

and

Course_ID → Course_Name

Here, Student_Name and Course_Name depend only on one part of the key which is a partial dependency that breaks 2NF.

So, you'd split the relation into:

- STUDENT (Student_ID, Student_Name)
- COURSE (Course_ID, Course_Name)
- ENROLLMENT (Student_ID, Course_ID, Grade)

This setup cuts down on redundancy and makes sure all non-key attributes depend entirely on the full candidate key.

Third Normal Form Addresses Transitive Dependencies

Third Normal Form (3NF) zeroes in on transitive dependencies.

Suppose:

Employee_ID → Department_ID

and

Department_ID → Department_Name

Here, Employee_ID → Department_Name is a transitive dependency. Storing all three fields together means repeating department names.

Breaking the table into employee and department fragments removes this issue. With 3NF, organization improves and redundancy drops (Elmasri & Navathe, 2016).

BCNF Provides Stronger Dependency Control

Boyce-Codd Normal Form (BCNF) sets a stronger line: every determinant of a non-trivial FD must be a superkey (Ramakrishnan & Gehrke, 2003).

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue VIII August 2026

BCNF can clean up redundancies left over from 3NF. But there's a trade-off sometimes going to BCNF means not all FDs survive in the decomposed schema.

So, the most rigorous normal form isn't always the best practical solution. Designers need to juggle controlling redundancy with keeping constraints enforceable.

Higher Normal Forms Address Complex Dependencies

Normalization pushes further than just FDs.

Fourth Normal Form (4NF) deals with multivalued dependencies cases where more than one independent multivalued relationship exists in a table. Fagin (1977) shows how crucial this is.

Fifth Normal Form (5NF) takes on join dependencies, handling scenarios where information splits into smaller tables and needs to be assembled through joins (Date, 2004).

Normalization keeps evolving to address trickier redundancy situations.

Lossless Decomposition Is Essential

Any good normalization relies on lossless decomposition i.e the split should let you rebuild the original without losing data or generating fake records.

If you split employee and department info using Department_ID, you can reconstruct the original with a proper join.

So, just chopping up tables isn't enough. The decomposition has to keep all the original meaning (Elmasri & Navathe, 2016).

Dependency Preservation Is an Important Consideration

Keeping dependencies enforceable after decomposing tables is crucial. If you can still apply all the original FDs without having to reunite the tables, you've achieved dependency preservation.

This crops up especially when comparing 3NF and BCNF. While BCNF shaves off more redundancy, 3NF can be better at maintaining dependency rules (Silberschatz et al., 2019).

Good design means finding the right balance between normalization and enforceable rules.

Normalization and Practical Performance

Normalization brings big logical benefits but can make things more complex in practice. Highly normalized databases can end up with many tables, which means more joins for complicated queries.

That's why real systems sometimes use controlled denormalization to speed things up. But denormalization brings some redundancy back, so the risk of contradicting data returns.

Normalization isn't a hard-and-fast rule but it's a guiding principle. The best level depends on what's most essential for your system: data consistency, workload, performance, and application needs.

DISCUSSION

The theoretical analysis shows that functional dependencies and normalization are essential to solid relational database design. Functional dependencies offer a formal way to spot relationships among attributes. Normalization then uses those connections to organize attributes into appropriate, logical relations. Codd (1970)

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue VIII August 2026

laid the groundwork for these ideas, and Armstrong (1974) gave us the tools to reason about functional dependencies in a formal way.

The evidence makes it clear that analyzing functional dependencies helps you find candidate keys, redundancy, and the structure of dependencies in your schema. For instance, if Employee_ID determines Department_ID, and Department_ID determines Department_Name, then Department_Name is transitively dependent on Employee_ID. Recognizing that lets the database designer separate employee and department details to cut down on redundancy (Elmasri & Navathe, 2016). Techniques like attribute closure and Armstrong's axioms further help systematically identify and infer these dependencies.

The study goes on to show that normalization makes a real difference in maintaining data integrity and reducing anomalies. First Normal Form enforces atomicity, Second Normal Form eliminates partial dependencies, and Third Normal Form deals with transitive dependencies. BCNF is even stricter, requiring that every determinant in a non-trivial functional dependency be a super key (Silberschatz et al., 2019). Higher normal forms push things further, covering multivalued and join dependencies. Fagin (1977) highlighted the role of multivalued dependencies for Fourth Normal Form.

Normalization targets the big three anomalies: insertion, update, and deletion. By making sure each fact lives in the right relation, normalization keeps duplication in check and cuts the risk of inconsistent updates. That's why normalized schemas typically offer better logical consistency, are easier to maintain, and deliver strong data integrity (Date, 2004).

Still, the discussion points out that driving normalization to the highest degree isn't always the best practical choice. When you heavily normalize, you may end up with many separate relations, which means more joins for complex queries. Sometimes, BCNF decomposition doesn't preserve all the functional dependencies, either. That's why designers have to juggle normalization alongside other real-world factors like dependency preservation, query complexity, and performance.

All in all, the findings push for a balanced approach. Start by using normalization to build a robust schema. Once that's set, tune performance with indexing, query optimization, caching, or controlled denormalization. Denormalization shouldn't be seen as throwing normalization out the window; it's a purposeful engineering choice when a small dose of redundancy brings real, measurable performance gains.

In summary, the discussion reaffirms that functional dependencies form the backbone of normalization, and normalization offers a structured path to less redundancy and stronger database integrity. Good relational database design demands a balance between theoretical soundness and practical needs. The goal isn't just to hit the highest normal form but to develop a schema that's lossless, logically consistent, maintainable, sensitive to dependencies, and right for its application.

CONCLUSION

This study takes a close look at functional dependencies and normalization within relational database design, making it clear why they matter so much for building databases that are reliable, consistent, and easy to maintain. The analysis shows that good relational database design isn't just about putting data into tables. It's about really understanding how attributes relate to each other and how these dependencies impact the way you should structure data (Codd, 1970; Elmasri & Navathe, 2016).

The study finds that functional dependencies sit at the core of normalization. They act as a formal tool for mapping out the relationships between attributes and help pinpoint candidate keys, super keys, partial dependencies, and transitive dependencies. Armstrong's axioms give us a systematic way to figure out functional dependencies and their logical outcomes (Armstrong, 1974). Because of this, it's crucial to accurately identify and analyze functional dependencies right at the start of any relational database design.

Additional findings point out that normalization systematically cuts down on data redundancy and database anomalies. First Normal Form enforces atomicity, Second Normal Form removes partial dependencies, and

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue VIII August 2026

Third Normal Form handles transitive dependencies. Boyce-Codd Normal Form steps things up by placing stronger conditions on determinants, while Fourth and Fifth Normal Forms target multivalued and join dependencies (Date, 2004; Fagin, 1977). Each normal form addresses more complex types of data redundancy.

One key conclusion is that normalization goes hand in hand with data integrity, since it shrinks insertion, update, and deletion anomalies. By splitting related data into different tables, you reduce duplication and lower the risk of inconsistent information. For example, keeping employee and department details separate means you don't have to keep storing department information with every individual employee record. This decomposition not only boosts consistency but also makes it easier to maintain data (Silberschatz et al., 2019).

The research also shows that hitting the highest possible normal form isn't the only way to judge normalization. Lossless decomposition and dependency preservation are just as important. A good decomposition holds on to the information from the original relation and, when possible, keeps important functional dependencies enforceable. These requirements show that normalization always involves balancing several key design goals (Elmasri & Navathe, 2016).

The analysis further emphasizes how normalization connects to database performance. Highly normalized schemas reduce redundancy, but they can increase the number of joins needed during queries. So sometimes, controlled denormalization is called for when performance matters most. But designers should know that denormalization can bring back redundancy and the risk of data inconsistency, so it should only be done when there's a solid reason. In the end, database designers need to balance logical integrity against the particular performance needs of their applications.

In short, the study reinforces the idea that functional dependencies and normalization work together as the backbone of relational database design. Functional dependencies make sense of how attributes are logically connected; normalization turns these connections into well-structured tables. Combining both helps limit redundancy, prevent data anomalies, safeguard integrity, and make databases easier to manage.

The main takeaway looks like this:

Functional Dependencies → Key Identification → Dependency Analysis → Normalization → Lossless Decomposition → Reduced Redundancy → Improved Data Integrity

So, effective relational database design isn't just about chaining together normal forms. It's really about planning a schema that's consistent, lossless, maintains dependencies, stays manageable, and works efficiently for its target use. Functional dependency analysis and normalization are still vital theoretical tools for striking this balance and they keep providing a solid foundation for how we build modern relational databases.

REFERENCES

1. Aho, A. V., Beeri, C., & Ullman, J. D. (1979). The theory of joins in relational databases. ACM Transactions on Database Systems, 4(3), 297–314. https://doi.org/10.1145/320083.320091
2. Armstrong, W. W. (1974). Dependency structures of database relationships. In Proceedings of the IFIP Congress (pp. 580–583). North-Holland.
3. Beeri, C., Bernstein, P. A., & Goodman, N. (1978). A sophisticated introduction to database normalization theory. In Proceedings of the 4th International Conference on Very Large Data Bases (pp. 113–123).
4. Bernstein, P. A. (1976). Synthesizing third normal form relations from functional dependencies. ACM Transactions on Database Systems, 1(4), 277–298. https://doi.org/10.1145/320493.320489
5. Bernstein, P. A., & Goodman, N. (1980). What does Boyce-Codd normal form do? In Proceedings of the 6th International Conference on Very Large Data Bases (pp. 245–259).
6. Codd, E. F. (1970). A relational model of data for large shared data banks. Communications of the ACM, 13(6), 377–387. https://doi.org/10.1145/362384.362685
7. Codd, E. F. (1972). Further normalization of the database relational model. In R. Rustin (Ed.), Data base systems: Courant Computer Science Symposia 6 (pp. 33–64). Prentice-Hall.

8. Date, C. J. (2004). An introduction to database systems (8th ed.). Addison-Wesley.
9. Elmasri, R., & Navathe, S. B. (2016). Fundamentals of database systems (7th ed.). Pearson.
10. Fagin, R. (1977). Multivalued dependencies and a new normal form for relational databases. ACM Transactions on Database Systems, 2(3), 262–278. https://doi.org/10.1145/320557.320571
11. Fagin, R., & Vardi, M. Y. (1983). Armstrong databases for functional and inclusion dependencies. Information Processing Letters, 16(1), 13–19. https://doi.org/10.1016/0020-0190(83)90005-4
12. Maier, D. (1983). The theory of relational databases. Computer Science Press.
13. Ramakrishnan, R., & Gehrke, J. (2003). Database management systems (3rd ed.). McGraw-Hill.
14. Silberschatz, A., Korth, H. F., & Sudarshan, S. (2019). Database system concepts (7th ed.). McGraw-Hill.
15. Tsou, D.-M., & Fischer, P. C. (1980). Decomposition of a relation scheme into Boyce-Codd normal form. In Proceedings of the ACM 1980 Annual Conference (pp. 411–417). Association for Computing Machinery. https://doi.org/10.1145/800176.809996
16. James, L. T., Kaushik, A., Singh, V. K., Kumar Kumaravel, S., Jasim, J. A., & Hasan, R. (2025). Neural Networks-Based Fraud Detection System for Secure Digital Banking Transactions. 2025 International Conference on Recent Innovation in Science Engineering and Technology (ICRISET), 1–6. https://doi.org/10.1109/icriset64803.2025.11252084
17. Chowdary, V. G. R., Rao, K. S., Uppu, A., Kumar, V., Mukherjee, S., & Roy, R. (2026). The Extent and Impact of Artificial Intelligence Adoption in India's Logistics Sector. Lecture Notes in Electrical Engineering, 134–146. https://doi.org/10.1007/978-3-032-25038-4_12
18. Roy, R., Das, T., Pal, D., Singh, V. K., & Mukherjee, S. (2026). The Dark Side of ChatGPT Usage and Its Impact on Students: An Empirical Evidence. Lecture Notes in Electrical Engineering, 178–187. https://doi.org/10.1007/978-3-032-25038-4_16
19. Srinivasa Rao, K., Chowdary, V. G. R., Uppu, A., Kumar Singh, V., Mukherjee, S., & Roy, R. (2026). Big Data Analytics and Industrial Performance: A Strategic Perspective. Lecture Notes in Electrical Engineering, 158–167. https://doi.org/10.1007/978-3-032-25038-4_14