Asian Journal of Electrical Sciences
ISSN: 2249-6297 (P); 2583-7923 (E)
Vol.14 No.2, 2025, pp.53-66

www.crijournals.org
DOI: https://doi.org/10.70112/ajes-2025.14.2.4291

Relational Database Normalization vs Denormalization: A Performance Perceptive

Wumi Ajayi[1], Kikelomo Okesola[2], Deogratias Ntezirayao[3], Francis Odo[4] and Alfred Udosen Akpan[5*]

[1&2]Department of Software Engineering, [5]Department of Computer Science, School of Computing, Babcock University, Ilishan-Remo, Nigeria

[3]Department of Computer Science, Adventist University of Central Africa (AUCA), Kigali, Rwanda

[4]Cisco Systems West Tower, Lagos, Nigeria

E-mail: ajayiw@babcock.edu.ng, okesolak@babcock.edu.ng, deogratias.nteziryayo@auca.ac.rw, fraodo@cisco.com

*Corresponding Author: udosena@babcock.edu.ng

(Received 16 August 2025; Revised 8 October 2025; Accepted 31 October 2025; Available online 8 December 2025)

Abstract **- Relational database architecture has a significant impact on how data are managed, retrieved, and stored, making it essential for effective data management. Two key design strategies that influence the structure of a relational database are normalization and denormalization. Normalization organizes data into structured tables to eliminate redundancy and ensure data integrity. Although this approach simplifies updates, it may lead to performance degradation due to complex queries and frequent join operations. In contrast, denormalization improves performance by reducing or eliminating join operations, at the cost of increased data redundancy and storage requirements. This paper investigates the impact of these design strategies on database performance, with a focus on improving query efficiency by minimizing the number of joins required for data retrieval. Using SQL Server as the chosen RDBMS and applying it to a School Grades Management System, this study demonstrates practical implementations of normalized and denormalized schemas through structured queries. Furthermore, by presenting performance benchmarks supported by indexing optimization strategies, this work aims to guide database designers in selecting an appropriate design strategy that achieves an optimal balance between data integrity and system performance.**

Keywords**: Relational Database Design, Normalization, Denormalization, Query Performance, SQL Server**

I. INTRODUCTION

Relational databases form the foundation of many information systems and are the most commonly used database type by businesses worldwide. Relational database normalization is considered one of the most effective approaches for creating large-scale and high-quality database systems [1]. To reduce data redundancy and improve data consistency by minimizing anomalies, normalization was introduced to organize data into "relations," or tables, based on predefined rules. Beyond improving data quality, it has also been argued that normalization enhances performance and maintainability [2].

However, in some cases, normalization alone is insufficient; therefore, developers adopt database denormalization to further improve database performance. Denormalization is a technique used in database design to reduce query execution time by introducing duplicate data into tables [3]. It is achieved by combining two or more normalized tables into a single table. As a result, join operations are reduced, and query execution becomes faster because fewer tables need to be accessed. However, if not managed properly, denormalization can lead to data redundancy and potential data inconsistency [4]. The decision between normalization and denormalization is one of the primary design choices that developers must make when defining a database structure. Both normalization and denormalization are techniques related to the organization and representation of data [5].

As organizations expand and increasingly automate their operations, the volume of data collected and processed grows significantly. Consequently, efficient database performance is essential to ensure fast data retrieval, seamless user experiences, and reliable backend operations [6]. Database performance optimization plays a critical role in ensuring smooth application delivery. Poorly designed queries, missing indexes, and capacity limitations can result in performance bottlenecks and system failures [7]. One widely used approach for improving database performance is denormalization. Denormalization involves adding redundant rows or columns to a normalized database schema to enhance read performance [8].

Conversely, normalization remains a fundamental aspect of data tuning, as it focuses on structuring the database to improve data consistency and minimize data duplication. By reducing redundancy, normalization helps improve query accuracy and reduce storage costs [9]. Database management systems play a critical role in data processing for decision-making, and SQL Server is a widely used relational database management system due to its scalability and feature-rich environment [10]. The primary objective of this study is to identify, analyze, and apply normalization and denormalization techniques to improve query efficiency in a school database, specifically for a student grades management system. Indexing is adopted as a complementary optimization strategy commonly used in relational databases.

II. LITERATURE REVIEW

A. Normalization

Applying the normalization process and understanding the rules defined by normal forms are essential when designing a relational database. Normalization is the process of decomposing a dataset into smaller entities with multiple attributes by examining their relationships in an organized and conflict-free manner [11]. Relational databases provide a reliable and scalable approach to storing and managing data, making them a fundamental component of modern software applications. A well-designed database schema improves application performance and scalability while also simplifying maintenance and enhancing overall efficiency [12]. The primary objective of effective database design in relational data models is to create an accurate representation of data, its relationships, and its constraints. To achieve this objective, database normalization-an established discipline since the publication of E. F. Codd's seminal work on normal forms in 1970-organizes data by eliminating anomalies, inconsistent dependencies, and redundancies [13]. The purpose of normalization is to structure data in a way that removes redundant information and ensures data integrity. In this study, normalization principles were applied to reduce data duplication by dividing data into smaller, conceptually related tables. This approach not only improved data consistency but also streamlined data management and reduced storage requirements [14]. In database systems, data normalization applies a set of formal rules to produce standardized and well-structured data. This process involves removing unstructured and unnecessary data and ensuring consistent data representation across all records and fields. Additionally, normalization helps prevent common database anomalies, including insertion, update, and deletion anomalies [15]. According to [16], multiple normal forms exist within the normalization process, each defined by specific rules and constraints. The most commonly used normal forms include:

1. *First Normal Form (1NF):* This is the initial stage of database normalization. A table is considered to be in 1NF if it satisfies the following conditions:
   a. All attribute values are atomic (indivisible).
   b. There are no repeating groups or arrays.

Table I shows an example of a table that violates 1NF, storing the information about students and their courses in the same table as shown below.

TABLE I A RELATION THAT VIOLATES THE 1NF [16]

Student_ID	Student_Name	Courses
1	Alice	Math, Physics, Chemistry
2	Bob	Biology, History
3	Carol	Math, History

To convert this table into 1NF, the "Courses" column is decomposed into separate rows for each course, as shown in Table II.

The objective of the first rule of normalization, known as First Normal Form (1NF), is to facilitate efficient data searching within a table [15].

TABLE II A NORMALIZED RELATION IN 1NF [16]

Student_ID	Student_Name	Courses
1	Alice	Math
1	Alice	Physics
1	Alice	Chemistry
2	Bob	Biology
2	Bob	History
3	Carol	Math
3	Carol	History

2. *Second Normal Form (2NF):* Building upon the framework of 1NF, Second Normal Form (2NF) introduces an additional constraint concerning the relationship between a table's primary key and its non-key attributes [16]. A table must satisfy the following conditions to achieve 2NF:
   a. The table must already be in 1NF.
   b. Every non-key attribute must be fully functionally dependent on the entire primary key. This means that non-key attributes should depend on the complete primary key, rather than on only a subset of it.

For example, the unnormalized relation (shown above) is decomposed into two relations (shown below) by ensuring that each non-prime attribute is fully functionally dependent on the primary key, as illustrated in Table III.

TABLE III SECOND NORMAL FORM ILLUSTRATED [4]

SSN	PNUMBER	PNAME	HOURS
100	1000	Hadoop	50
220	1200	CRM	200
280	1000	Hadoop	40
300	1500	Java	100
120	1000	Hadoop	120

⇩ 2NF

PNUMBER	PNAME
1000	Hadoop
1200	CRM
1500	Java

SSN	PNUMBER	HOURS
100	1000	50
220	1200	200
280	1000	40
300	1500	100
120	1000	120

3. Third Normal Form (3NF): According to [16], the next stage in the database normalization process is Third Normal Form (3NF). A table must satisfy the following conditions to achieve 3NF:

a. The table must already be in 2NF.
b. No transitive dependencies should exist; that is, non-key attributes must not depend on other non-key attributes.

In other words, 3NF ensures that data are structured to prevent the redundant storage of non-key attribute information. For example, the unnormalized relation (shown above) is decomposed into two relations (shown below) by ensuring that no non-prime attribute is transitively dependent on the primary key.

TABLE IV THIRD NORMAL FORM ILLUSTRATED [15]

Student_ID	Student_Name	Subject_ID	Subject	Address
1DT1SENG01	Alex	15C511	SQL	Goa
1DT1SENG02	Barry	15C513	JAVA	Bengaluru
1DT1SENG03	Clair	15C512	C++	Delhi
1DT1SENG04	David	15C513	JAVA	Kochi

⇩ 3NF

Student_ID	Student_Name	Subject_ID	Address
1DT1SENG01	Alex	15C511	Goa
1DT1SENG02	Barry	15C513	Bengaluru
1DT1SENG03	Clair	15C512	Delhi
1DT1SENG04	David	15C513	Kochi

Subject_ID	Subject
15C511	SQL
15C513	JAVA
15C512	C++
15C513	JAVA

4. *Boyce–Codd Normal Form (BCNF):* According to [16], building upon the principles of 1NF, 2NF, and 3NF, Boyce–Codd Normal Form (BCNF) represents a higher level of normalization. To satisfy BCNF, a table must meet the following conditions:

a. The table must already be in 3NF.
b. Every determinant must be a superkey, where a superkey is any set of attributes that uniquely identifies a tuple.

In other words, BCNF ensures that all non-key attributes are fully functionally dependent on the entire primary key or on a superkey, thereby eliminating partial dependencies. BCNF is an enhanced version of 3NF and was introduced by Edgar F. Codd and Raymond F. Boyce to address certain anomalies that 3NF could not resolve [15]. As noted in [17], a lossless decomposition of a relational database schema into BCNF may not always exist and depends on the given set of functional dependencies. However, the authors argue that more efficient methods are required to achieve BCNF in practice, particularly through the use of automated design tools. According to [19], BCNF enables users to automatically generate schema transformation scripts. For example, as illustrated in Figure 5.

a. One student may register for multiple subjects.
b. A single subject may be taught by different professors.
c. For each subject, a professor is assigned to a student; however, the table does not satisfy BCNF.

TABLE V TABLE TO BE CONVERTED INTO BCNF [15]

Student_ID	Subject	Professor
1DT1SENG01	SQL	Prof. Mushra
1DT1SENG02	JAVA	Prof. Anand
1DT1SENG02	C++	Prof. Kanthi
1DT1SENG03	JAVA	Prof. Anand
1DT1SENG04	DBMS	Prof. Lokesh

a. In this case, *Student ID* and *Subject* together form the primary key, which means that the *Subject* attribute is a prime attribute.
b. However, there exists an additional functional dependency: *Professor* → *Subject*.
c. Although *Subject* is a prime attribute, *Professor* is a non-prime attribute; this violates the requirements of BCNF.
d. To satisfy BCNF, the table is decomposed into two separate tables. One table retains the existing *Student ID*, while a second table is created to include the *Professor ID* attribute.

TABLE VI SHOWS THE BCNF NORMALIZED TABLE

Student_ID	Subject	Professor_ID
1DT1SENG01	SQL	1DTPF01
1DT1SENG02	JAVA	1DTPF02
1DT1SENG02	C++	1DTPF03
1DT1SENG03	JAVA	1DTPF02
1DT1SENG04	DBMS	1DTPF04

TABLE VII NORMALIZED INTO BCNF [15]

Professor_ID	Professor	Subject
1DTPF01	Prof. Mushra	SQL
1DTPF02	Prof. Anand	JAVA
1DTPF03	Prof. Kanthi	C++
1DTPF02	Prof. Anand	JAVA
1DTPF04	Prof. Lokesh	DBMS

In the second table, the columns *Professor ID*, *Professor*, and *Subject* are included. This table now satisfies the requirements of BCNF.

5. *Fourth Normal Form (4NF):* According to [16], Fourth Normal Form (4NF) extends the concepts of 1NF, 2NF, 3NF, and BCNF. A table must satisfy the following conditions to achieve 4NF:

a. The table must already be in BCNF.
b. The table must not contain any non-trivial multi-valued dependencies among non-key attributes.In other words, 4NF ensures that no sets of non-key attributes exhibit multi-valued dependencies or are functionally dependent on the primary key. In Figure 8, the unnormalized relation (shown above) is decomposed into two relations (shown below) by ensuring that, for every non-trivial multi-valued dependency X ↠ Y, X is a superkey.

6. *Fifth Normal Form (5NF):* Fifth Normal Form (5NF), also known as Project-Join Normal Form (PJ/NF), is a higher level of database normalization that addresses join dependencies. By achieving 5NF, tables are structured to minimize the need for complex joins in queries. A table must satisfy the following conditions to attain 5NF:

a. The table must already be in 4NF.
b. The table must not rely on decomposing and joining multiple tables to access data; this condition is referred to as a join dependency. In other words, 5NF resolves situations where a relation cannot be reconstructed without joining multiple tables.

Thus, 5NF aims to eliminate the need to join tables to retrieve information [16]. For example, consider a library database that tracks information about books, authors, and publishers, as illustrated in Figures 9–12 below.

TABLE VIII FOURTH NORMAL FORM ILLUSTRATED [4]

COURSE	INSTRUCTOR	BOOK
Database Management	Baesens	Database cookbook
Database Management	Lemahieu	Database cookbook
Database Management	Baesens	Database for dummies
Database Management	Lemahieu	Database for dummies

⇩ 4NF

COURSE	INSTRUCTOR
Database Management	Baesens
Database Management	Lemahieu

COURSE	BOOK
Database Management	Database cookbook
Database Management	Database for dummies

TABLE IX UNNORMALIZED BOOKS TABLE FOR 5NF

Book_ID	Title	Author_ID	Publisher_ID
1	“Book 1”	1	1
2	“Book 2”	2	2
3	“Book 3”	3	3

TABLE X UNNORMALIZED AUTHORS TABLE FOR 5NF

Author_ID	Author_Name
1	“Author A”
2	“Author B”
3	“Author C”

TABLE XI UNNORMALIZED PUBLISHERS TABLE FOR 5NF

Publisher_ID	Publisher_Name
1	“Publisher X”
2	“Publisher Y”
3	“Publisher Z”

TABLE XII TABLE BOOK INFORMATION NORMALIZED FOR 5NF [16]

Book_ID	Title	Author_Name	Publisher_Name
1	“Book 1”	“Author A”	“Publisher X”
2	“Book 2”	“Author B”	“Publisher Y”
3	“Book 3”	“Author C”	“Publisher Z”

TABLE XIII SUMMARY OF THE FIVE NORMAL FORMS RULES [20]

Normal Form	Rule
5th Normal Form	No join dependencies
4th Normal Form	No multivalued dependencies
BCNF	Left hand side of the fucntional dependency is a superkey
3rd Normal Form	No transitive dependencies
2nd Normal Form	No partial dependencies
1st Normal Form	No multivalued or composite attributes

To achieve 5NF, a new table can be created to eliminate the need for joins while storing all relevant information. In this 5NF structure, information from the *Publishers*, *Books*, and *Authors* tables has been merged into a single table called *Book Information*. Since all relevant data are now contained within one table, join operations are no longer required to retrieve detailed book information. Table XIII summarizes the five normal form rules discussed above. Table XIV concludes this section by summarizing the various normalization steps and the types of dependencies considered.

TABLE XIV OVERVIEW OF NORMALIZATION STEPS AND DEPENDENCY [4]

Normal Form	Type of dependency	Description
2NF	Full functional dependency	A functional dependency X ⟶ Y is a full functional dependency of any attribute type A from X means that the dependency does not hold anymore
3NF	Transitive funcional dependency	A functional dependency X ⟶ Y in a elation R is a transiive dependency if there is a set of atttribute ypes Z that is neither a canidate key nor a subset of any key of R, both X ⟶ Z and Z ⟶ Y hold.
BCNF	Trivial funcional dependency	A functional dependency X ⟶ Y is called trivial if Y is a subset of X
4NF	Multivariate funcional dependency	A dependence X ⟶ ⟶ Y is multi-valued if and only if each X value exactly deternines a set of Y values, indepedently of the other attribute types.

B. Denormalization

The first concept that comes to mind when discussing databases is a location where information is organized to facilitate efficient management, access, and manipulation. A database consists of multiple relations that store information about one or more entities, and relationships are established when data are distributed across several tables; this type of database is known as a relational database [21].

However, this structure can make it more challenging to retrieve useful information. Additionally, identifying the logical sequence of tables that must be joined to consolidate data from different parts of the database can be laborious, particularly for large databases that require numerous permutations of distinct data items. A more significant limitation is that many data exploration or analysis tasks require the database administrator to create denormalized tables in advance for end users who do not have direct access [22].

To facilitate data access and analysis, denormalization is the process of merging data from multiple sources or tables into a single table. Denormalization may involve combining tables, adding redundant data, or duplicating columns to improve efficiency and streamline analytical processes. Denormalization is particularly useful when fast data access and system performance optimization are required [5].

According to [14], by reducing the need for complex joins, denormalization techniques improve query performance. The author further noted that adding redundant data or duplicating specific columns can accelerate query execution in certain scenarios, thereby improving response time, reducing computational overhead, and minimizing the number of required tables joins. Furthermore, [21] confirmed that while database normalization is a well-studied topic, the literature on denormalization is limited; therefore, a related work section is not included.

C. Advantages and Disadvantages of Normalization

According to [2], the three main benefits of database normalization are data quality enhancement, maintainability, and performance. These are broadly consistent with the benefits presented by [5], which include flexible data organization, minimized data duplication, and facilitated processes for updating and maintaining data. As the normalization hierarchy advances, as shown in Figure 13, reducing data redundancy is associated with improved data quality. Enhanced maintainability is evident because, compared to highly normalized tables, weakly normalized or unnormalized tables contain more attributes per table, making data retrieval and the implementation of new rules more difficult. Finally, normalization has been shown to improve operational performance. However, [5] also highlighted certain drawbacks of normalization, including the need for additional data merging operations and a potential decrease in query performance.

D. Advantages and Disadvantages of Denormalization

According to [5], there are three advantages of denormalization, which include:

1. A denormalized data schema enables faster information retrieval.
2. A denormalized data schema reduces the number of data merging operations in queries, potentially improving system performance.
3. Denormalization streamlines query execution, since the data is already consolidated in a single collection and information retrieval does not require complex data merging operations.

However, [5] also identified four drawbacks of denormalization:

1. *Data duplication:* The same data may be repeated across multiple documents due to denormalization, leading to increased storage requirements.

2. *Larger collection sizes:* Denormalization can result in larger collection sizes, particularly for high-volume datasets, which may negatively affect system performance.
3. *Difficulty in updating and maintaining data:* Because data may be distributed across multiple documents, updating and maintaining a denormalized schema can be more challenging and labour-intensive.
4. *Possible decrease in query execution performance:* Denormalization can sometimes reduce query performance because data may be spread across various documents, requiring additional data merging operations to retrieve all relevant information.

A comparative analysis of the advantages and disadvantages of normalization and denormalization is presented in Table XV, as shown below:

TABLE XV NORMALIZATION AND DENORMALIZATION COMPARATIVE ANALYSIS TABLE FOR ADVANTAGES (INDICATED BY THE SIGN +) AND DISADVANTAGES (INDICATED BY THE SIGN -)

Aspect Comparison	Denormalized Data Schema	Normalized Data Schema
Flexible data organization		+
Minimization of data duplication		+
Facilitate the processes of updating and maintaining data.		+
Additional data merging operations		-
Possible decrease in query performance		-
Fast retrieval of information	+	
Reduces the number of data merging operations in queries	+	
Simplifies the process of executing queries	+	
Data duplication	-	
Increased collection sizes	-	
Difficulty in updating and maintaining data	-	
Potential performance reduction in query execution	-	

In light of this, this study demonstrates that when choosing between denormalized and normalized data schemas in MongoDB or other databases, it is essential to carefully consider the application's requirements and select the strategy that best meets the project's performance objectives while also addressing the needs of database administrators and developers.

E. Strategies for Improving the Database Performance

For effective data management and retrieval, database performance is critical, particularly in environments that use Structured Query Language (SQL) Server. Database performance directly affects user satisfaction, overall productivity, and application responsiveness, and supporting essential business processes requires a well-optimized database capable of handling large transaction volumes and providing fast query responses [10]. Since indexes are the most commonly used technique for accelerating query response, their creation is crucial. By implementing effective database optimization techniques, high performance of databases can be maintained. Indexing is one of the most important strategies for ensuring that relational databases operate at optimal levels. Indexing solutions are essential for addressing poor database performance and enhancing database performance optimization [22].

According to [21], an index is a data structure that speeds up data retrieval operations in a database table at the cost of additional writes and storage space required for index maintenance. By using indexes, data can be located quickly without scanning all rows of a table for each access. One or more columns from a database table can be used to create an index, enabling efficient access and fast random lookups. Furthermore, [23] confirmed that indexing is one of the most effective methods for improving query performance, as it reduces the time required to locate and retrieve data in a database. Although relational databases make extensive use of indexing, a comprehensive understanding of how different indexing techniques perform under various contexts and query types is still lacking.

III. METHODOLOGY

A study by [2], with a 75.2% popularity score, shows that the relational model remains the most widely used Database Management System (DBMS). The top four DBMSs are Oracle, MySQL, MS SQL Server, and PostgreSQL. The relational model and SQL query language have gained popularity and are widely adopted in corporate environments due to their ease of use [21]. It is noteworthy that, as this study focuses on relational databases, MS SQL Server, using the "Display Estimated Execution Plan" and "Include Client Statistics" options, was selected as the tool for analyzing the comparative performance between normalization and denormalization. A proposed Student Grades Management System from the Adventist University of Central Africa (AUCA), located in Kigali, Rwanda, was selected and adapted as a sample for this study. The database stores students' grades and is used to generate

transcripts. The main table in the database is *Grades*, which contains more than six hundred thousand records. The current sample includes 10 base relations. Note that a table is referred to as either *T_TABLE* or *TABLE* interchangeably, as one is a copy of the other. Figure 14 illustrates the database for the Grades Management System.

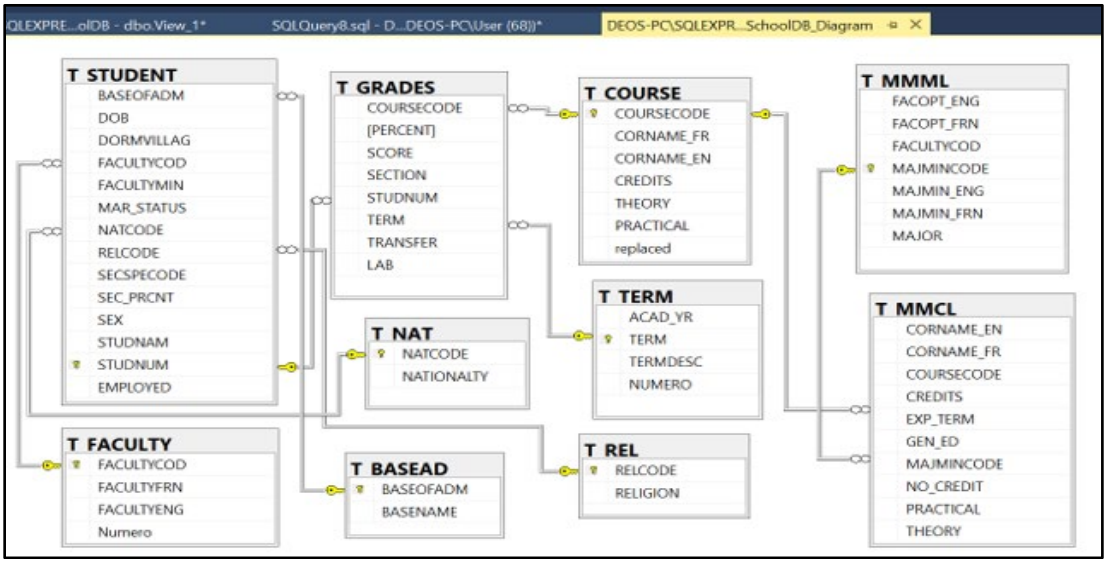

Fig.1 School Database Diagram - Grades Management System

According to [22], the two most commonly used types of database indexes are:

1. *Clustered indexes:* The primary key is used to organize the data in the table using clustered indexes, which are unique to each table. When the primary key is defined, the clustered index is automatically created.
2. *Non-clustered indexes:* In a non-clustered index structure, the index defines the logical order of the data even when the data is stored in an arbitrary sequence [25]. Non-clustered indexes are commonly used with JOIN, WHERE, and ORDER BY clauses to optimize column retrieval. These indexes are suitable for tables with frequently changing values. When the CREATE INDEX command is executed, Microsoft SQL Server automatically generates non-clustered indexes.

Figure 1 illustrates how SQL queries utilize indexes. In this study, non-clustered indexes will primarily be applied to one or a group of tables where performance improvement is required.

IV. DATA ANALYSIS AND DISCUSSION

From the normalized database sample, it is necessary to compare the performance of the normalization technique with that of denormalization. Figure 2 shows a query that retrieves the list of students from the Department of Information Technology who have completed their studies across the five normalized and joined tables. The query calculates the total number of credits and the cumulative mean. Its performance is first evaluated, then improved using indexes, and finally measured again to assess the effect of the added indexes.

```
SELECT TOP (100) PERCENT S.STUDNUM, S.STUDNAM, SUM(C.CREDITS) AS TotalCredits,
ROUND(SUM(C.CREDITS * G.SCORE) / SUM(C.CREDITS), 2) AS WeightedAverage FROM   dbo.STUDENT AS S INNER JOIN
                dbo.GRADES AS G ON S.STUDNUM = G.STUDNUM INNER JOIN
                dbo.COURSE AS C ON G.COURSECODE = C.COURSECODE INNER JOIN
                dbo.MMCL ON C.COURSECODE = dbo.MMCL.COURSECODE INNER JOIN
                dbo.MMML ON dbo.MMCL.MAJMINCODE = dbo.MMML.MAJMINCODE AND S.FACULTYCOD = dbo.MMML.FACULTYCOD
WHERE (G.SCORE >= 10) AND (dbo.MMCL.NO_CREDIT <> 1) AND (dbo.MMCL.GEN_ED <> 1 OR
                dbo.MMCL.GEN_ED = 1)
GROUP BY S.STUDNUM, S.STUDNAM, S.FACULTYCOD
HAVING (S.FACULTYCOD = 16) AND (SUM(C.CREDITS) >= 136)
ORDER BY S.STUDNUM
```

100 %

Results | Messages

	STUDNUM	STUDNAM	TotalCredits	WeightedAverage
1	202416	BU202416A	138	14.62
2	202556	BU202556A	136	14.57
3	203436	BU203436A	136	15.95
4	204236	BU204236A	136	15.8
5	204976	BU204976A	141	12.81
6	206676	BU206676A	137	13.82
7	207096	BU207096A	137	14.41
8	207196	BU207196A	139	12.63
9	207256	BU207256A	140	14.96
10	208916	BU208916A	139	13.63
11	209696	BU209696A	144	14
12	209716	BU209716A	140	13.37
13	210756	BU210756A	140	14.38
14	211056	BU211056A	144	14.03

Query executed successfully. DEOS-PC\SQLEXPRESS (16.0 RTM) DEOS-PC\User (167) SchoolDB 00:00:01 1,238 rows

Fig.2 A Query that Retrieves the List of Completed Students from IT Department Presented for Normalization Process Evaluation

The query in Figure 16 demonstrates the use of JOIN on five related tables, along with other SQL clauses such as WHERE, GROUP BY, HAVING, and ORDER BY. The query outputs the student matriculation number, name, the total number of credits completed, and the mean score over 20 for 1,238 graduates. Next, the execution time of the query is evaluated, as shown in Figure 3.

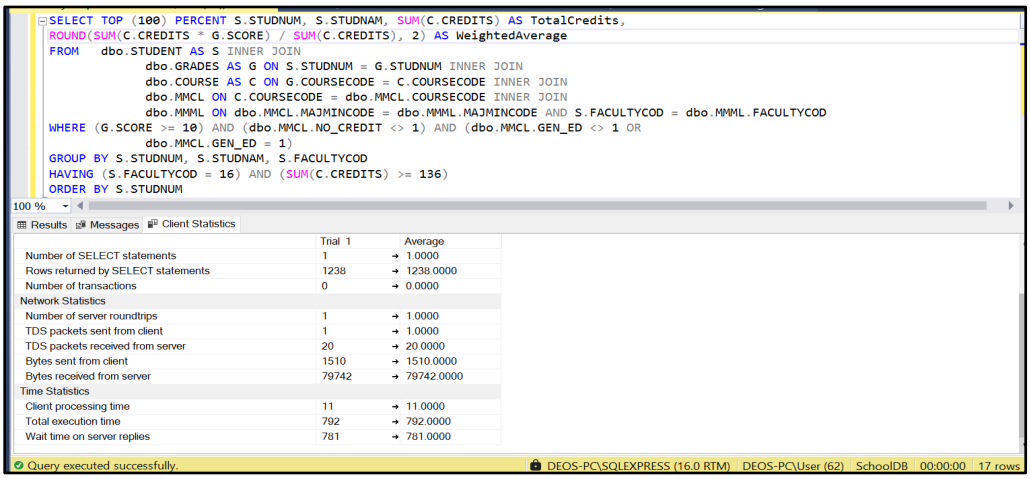

Fig.3 Total Execution Time for the Query that Justify the Normalization Process – Not Optimized

Figure 3 presents various statistical information on the query performance; however, the primary focus of this study is the second-to-last line, “Total execution time”: 792 milliseconds. The next step involves adding an index to each of the five tables, as shown in Figure 4.

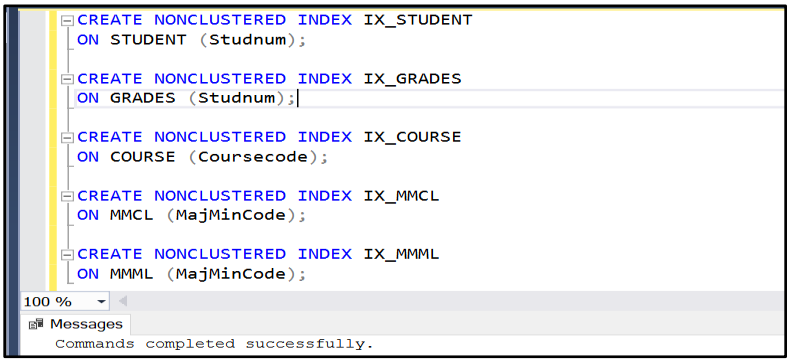

Fig.4 Indexing for Optimizing Normalized Tables

The same query, consisting of five joined tables and optimized with indexes, was finally evaluated to measure the performance of normalization, as shown in Figure 19. As shown in Figure 5, which presents the performance of the optimized query, the total execution time is 393 milliseconds. This represents a positive improvement of 399 milliseconds (50.4%) compared to the result shown in Figure 3 (792 milliseconds) for the non-optimized query.

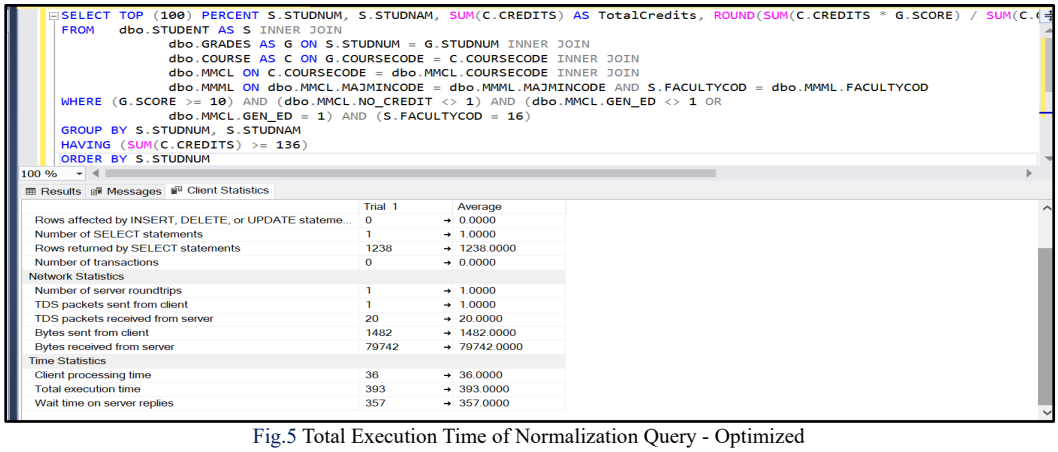

Fig.5 Total Execution Time of Normalization Query - Optimized

This demonstrates the significant impact of indexes, as they contributed to the reduction in query execution time. The denormalization demonstration begins by combining the top five tables into a single table. This process requires merging tables with many-to-many or one-to-one relationships. As shown in Figure 6, the view *V_StudentGradesSummary* illustrates the relationships: the table *GRADES* contains the foreign key from *STUDENT* to *COURSE*, the table *MMCL* (which contains courses by program) includes the foreign key from *COURSE* to *MMML* (which contains the list of programs), and finally, *MMML* also has a foreign key connecting it to *STUDENT*. The view *V_StudentGradesSummary*, representing the five tables, contains 604,538 rows. These tables need to be combined and transformed into a single table to produce the same results as the initial example: the list of completed students in the Information Technology program.

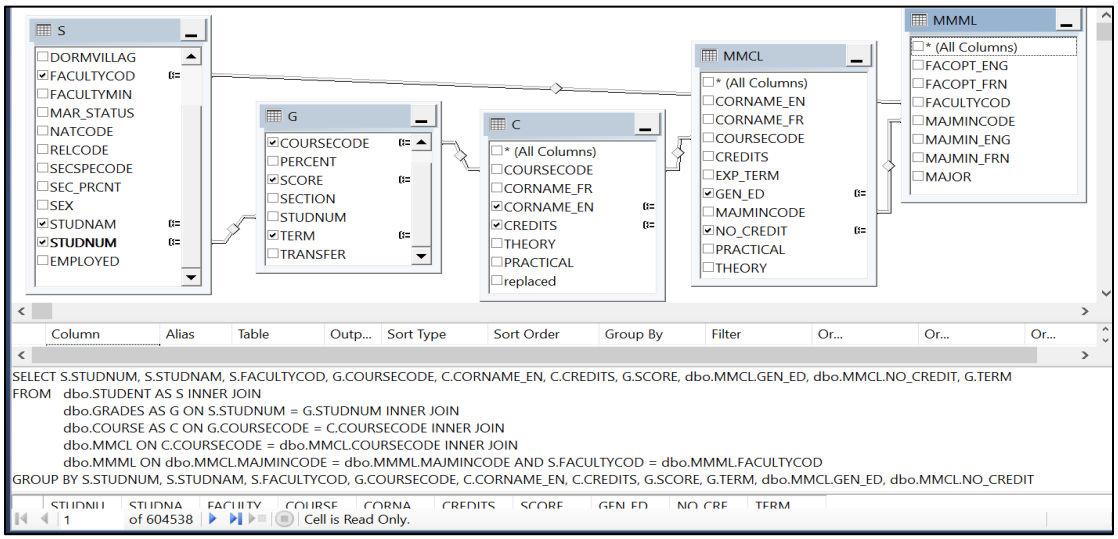

Fig.6 V_StudentGradesSummary View

The next step, shown in Figure 7, presents the definition and design of the table *StudentGradesReport1*, which will receive information from the five tables described above. The table above will capture the essential information required to generate the requested report, in this case, the list of students in the Information Technology department who have completed their studies.

```
SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[StudentGradesReport1](
    [COURSECODE] [nvarchar](9) NULL,
    [CORNAME_EN] [nvarchar](58) NULL,
    [CREDITS] [float] NULL,
    [SCORE] [float] NULL,
    [GEN_ED] [bit] NOT NULL,
    [NO_CREDIT] [bit] NOT NULL,
    [STUDNUM] [float] NULL,
    [STUDNAM] [nvarchar](50) NULL,
    [FACULTYCOD] [float] NULL,
    [FACULTYENG] [nvarchar](35) NULL,
    [TERM] [nvarchar](6) NULL
) ON [PRIMARY]
GO
```

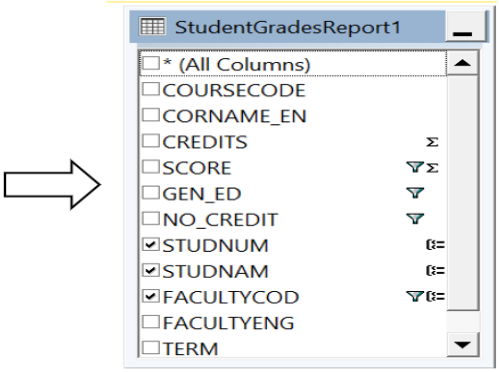

Fig.7 The Definition and Design of the New Denormalized Table

The next step, shown in Figure 8, presents the query using the INSERT INTO statement to import the relevant information-including some redundancy-into the newly created denormalized table. The output confirms that all 604,538 records were successfully added to the table from the view.

```
INSERT INTO StudentGradesReport1 (COURSECODE,CORNAME_EN, CREDITS,SCORE, GEN_ED, NO_CREDIT, TERM, FACULTYCOD, STUDNUM,STUDNAM,FACULTYENG)
select COURSECODE,CORNAME_EN, CREDITS,SCORE, GEN_ED, NO_CREDIT, TERM, FACULTYCOD, STUDNUM,STUDNAM,FACULTYENG FROM V_StudentGradesSummary
```

100 %

Messages

```
(604538 rows affected)

Completion time: 2025-01-18T22:01:23.2600065+01:00
```

Fig.8 The Query to Insert Data from V_StudentGradesSummary into StudentGradesReport1

In the following exercise, shown in Figure 9, the query is similar to the one in Figure 2. The first query (Figure 2) retrieves information from five joined and normalized tables, while the query below retrieves the same information from a denormalized table. Both queries produce identical results. The process involves four steps: first, evaluating the performance of the query on the denormalized table; second, improving it by adding indexes; third, re-evaluating the execution time; and finally, comparing the queries from both approaches-normalization and denormalization-to assess the performance of the SELECT operation between the two techniques.

```
SELECT DISTINCT TOP (100) PERCENT STUDNUM, STUDNAM, SUM(CREDITS) AS TotalCredits,
ROUND(SUM(CREDITS * SCORE) / SUM(CREDITS), 2) AS WeightedAverage FROM   dbo.StudentGradesReport1
WHERE (SCORE >= 10) AND (NO_CREDIT <> 1) AND (GEN_ED <> 1 OR GEN_ED = 1)
GROUP BY STUDNUM, STUDNAM, FACULTYCOD
HAVING (FACULTYCOD = 16) AND (SUM(CREDITS) >= 136)
ORDER BY STUDNUM
```

100 %

Results Messages

	STUDNUM	STUDNAM	TotalCredits	WeightedAverage
1	202416	BU202416A	138	14.62
2	202556	BU202556A	136	14.57
3	203436	BU203436A	136	15.95
4	204236	BU204236A	136	15.8
5	204976	BU204976A	141	12.81
6	206676	BU206676A	137	13.82
7	207096	BU207096A	137	14.41
8	207196	BU207196A	139	12.63
9	207256	BU207256A	140	14.96
10	208916	BU208916A	139	13.63
11	209696	BU209696A	144	14
12	209716	BU209716A	140	13.37
13	210756	BU210756A	140	14.38
14	211056	BU211056A	144	14.03

Query executed successfully. DEOS-PC\SQLEXPRESS (16.0 RTM) DEOS-PC\User (156) SchoolDB 00:00:00 1,238 rows

Fig.9 A Query that Retrieves the List of Completed Students from IT Department Presented for Denormalization Process Evaluation

Figure 10 shows the evaluation results for the query that retrieves the list of students in the Department of Information Management who have completed their studies. This query has not yet been optimized with indexes.

```
SELECT DISTINCT TOP (100) PERCENT STUDNUM, STUDNAM, SUM(CREDITS) AS TotalCredits, ROUND(SUM(CREDITS * SCORE) / SUM(CREDI
FROM   dbo.StudentGradesReport1
WHERE (SCORE >= 10) AND (NO_CREDIT <> 1) AND (GEN_ED <> 1 OR
             GEN_ED = 1)
GROUP BY STUDNUM, STUDNAM, FACULTYCOD
HAVING (FACULTYCOD = 16) AND (SUM(CREDITS) >= 136)
ORDER BY STUDNUM
```

100 %

Results Messages Client Statistics

	Trial 1	Average
Client Execution Time	07:45:57	
Query Profile Statistics		
Number of INSERT, DELETE and UPDATE statements	0	0.0000
Rows affected by INSERT, DELETE, or UPDATE stateme...	0	0.0000
Number of SELECT statements	1	1.0000
Rows returned by SELECT statements	1238	1238.0000
Number of transactions	0	0.0000
Network Statistics		
Number of server roundtrips	1	1.0000
TDS packets sent from client	1	1.0000
TDS packets received from server	20	20.0000
Bytes sent from client	782	782.0000
Bytes received from server	80981	80981.0000
Time Statistics		
Client processing time	15	15.0000
Total execution time	409	409.0000
Wait time on server replies	394	394.0000

Query executed successfully. DEOS-PC\SQLEXPRESS (16.0 RTM) DEOS-PC\User (159) SchoolDB 00:00:00 17 rows

Fig.10 Total Execution Time for the Query that Justify the Denormalization Process – Non-Optimized

Similar to Figure 3, Figure 10 presents various statistical information generated using the Display Estimated Execution Plan and Include Client Statistics SQL query commands. Among this information, the primary focus is the "Total execution time": 409 milliseconds. This demonstrates that the query for the denormalization process is faster than the query for the normalization process, which had a total execution time of 792 milliseconds. The improvement is 383 milliseconds (48.4%). In Figure 25, an index was applied to the query to further verify its performance.

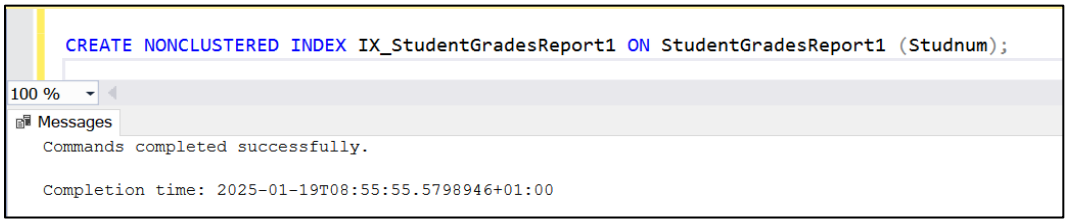

Fig.11 Indexing for Optimizing Denormalized Table

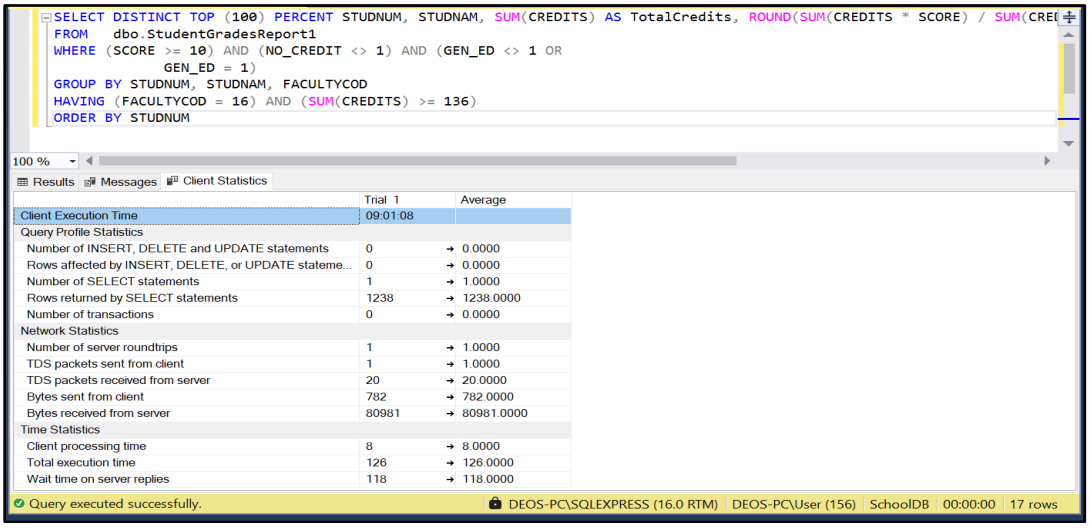

Fig.12 Total Execution Time for the Query that Justify the Denormalization Process –Optimized

The query in Figure 12 improves upon the query in Figure 10, which had a total execution time of 409 milliseconds. It retrieves information from a single denormalized table and is optimized using the index shown in Figure 11. The result shows that the total execution time of this optimized query is 126 milliseconds. The difference between the non-optimized and optimized query performance is 283 milliseconds (69.2%), demonstrating the importance of using indexes. From the previous discussions of this data analysis, we were comparing performance of non-optimized and optimized queries. Also, there was a comparison of the performance of optimized queries only, between normalization and denormalization based on their results as shown on figures 19 and 26. The total execution time for the normalization process is 393 milliseconds whereas the one for denormalization is 126 milliseconds. The obtained improvement is of 267 milliseconds (67.9%). Table II below shows the summary of query performance analysis results.

TABLE XVI SUMMARY OF QUERY PERFORMANCE ANALYSIS RESULTS

Estimated Total Execution Time - Query speed in milliseconds / Figure N°			
Technique Applied	**Before indexing (Query not optimized)**	**After indexing (Query Optimized)**	**Improvement (Effect of indexing)**
Normalization	792 ms (Fig.17)	393 ms (Fig.19)	399 ms (50.4%)
Denormalization	409 ms (Fig.24)	126 ms (Fig.26)	283 ms (69.2%)
Improvement (Effect of Indexing)	383 ms (48.4%)	267 ms (67.9%)	-

Based on the foregoing, we may conclude that denormalization performs better than normalization in terms of query retrieval (SELECT). However, what about other basic SQL operations such as INSERT, DELETE, or UPDATE? This question is important because the primary goal of normalization is to reduce data redundancy and eliminate anomalies during INSERT, DELETE, and UPDATE operations. On the other hand, one may ask whether denormalization can also perform these operations efficiently. The answer is no, because denormalization is designed with intentional redundancies. Although denormalization improves data reading performance

through SELECT operations, it is less efficient for data writing and may introduce errors due to its inherent design. However, this does not imply that denormalization is always slower for INSERT, DELETE, or UPDATE; each operation requires specific evaluation.

Firstly, for the INSERT operation, the following query was executed to add one row to the denormalized table: INSERT INTO StudentGradesReport1 VALUES ('ACCT 2000', 'AI in Business','3', NULL, 0, 0, NULL, NULL, NULL, NULL, NULL); and its total execution time was 21 milliseconds; while the same operation performed over the normalized table COURSE with this query codes: INSERT INTO COURSE VALUES ('INSY 2001','Intro. a l Intelligence Artificielle', 'Introduction to AI',3,30,15,0); only the total of execution time was 12 milliseconds.

Secondly, for the DELETE operation, the query executed on the denormalized table was: DELETE FROM StudentGradesReport1 WHERE COURSECODE= 'ACCT 2000'; and its total execution time was 161 milliseconds; while the same operation performed over the normalized table COURSE with this query codes: DELETE FROM COURSE WHERE COURSECODE = 'INSY 2001'; the total of execution time generates 17 milliseconds.

Finally, for Update operation, the following change of the course code was applied to the denormalized table and the following query was been written: UPDATE StudentGradesReport1 SET coursecode = 'ACCT 2001' WHERE coursecode = 'ACCT 2000'; and its total execution time was 230 milliseconds; while the same operation performed over the normalized table COURSE with this query codes: UPDATE COURSE SET coursecode= 'INSY 2002' WHERE coursecode = 'INSY 2001'; only the total of execution time gives 67 milliseconds. From these findings, it is evident that for the three basic SQL data-writing operations (INSERT, DELETE, and UPDATE), normalization not only prevents anomalies but also executes faster than denormalization, as its execution times were consistently lower.

V. CONCLUSION AND RECOMMENDATIONS

This study addresses the comparison between normalization and denormalization with respect to a single factor: performance. The trade-off between the two methods is that normalization is more efficient for data writing operations (INSERT, DELETE, and UPDATE), whereas denormalization performs better for data reading (SELECT) operations. Making a choice between them can be challenging. It is often preferable to create two separate tables: one for storing normalized data, and a view for data retrieval when needed. If there are frequent requests for the same information, the database may be managed in two logical parts. The first part consists of an independent physical denormalized table for data that rarely changes, used solely for retrieval. The second part retains separate normalized tables, primarily used for dynamic retrieval operations. If database partitioning is difficult to implement or its necessity is uncertain, it is better to avoid denormalization. Furthermore, given the ambiguity in choosing the appropriate technique, particularly when applying the method introduced by Raymond F. Boyce and Edgar F. Codd over 50 years ago, it is recommended that database designers, as well as other database users or platform developers, explore new approaches. Specifically, AI-driven or AI-assisted techniques for database normalization and optimization [24] may offer more powerful and effective solutions than traditional Codd methods.

Declaration of Conflicting Interests
The authors declare no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding
The authors received no financial support for the research, authorship, and/or publication of this article.

Use of Artificial Intelligence (AI)-Assisted Technology for Manuscript Preparation
The authors confirm that no AI-assisted technologies were used in the preparation or writing of the manuscript, and no images were altered using AI.

ORCID
Wumi Ajayi http://orcid.org/0000-0003-3362-4082
Kikelomo Okesola http://orcid.org/0000-0003-0944-1497
Deogratias Ntezirayao http://orcid.org/0009-0002-9222-9904
Francis Odo http://orcid.org/0009-0003-6301-4426
Alfred Udosen Akpan http://orcid.org/0000-0003-1454-0254

REFERENCES

[1] B. Alathari, "The Comparative Analysis for Data Normalization Using User Interface Normal Form," *Int. J. Recent Trends Eng. Res.*, vol. 4, no. 3, pp. 59–65, 2018, doi: 10.23883/ijrter.2018.4097.qlzhe.

[2] M. Albarak, R. Bahsoon, I. Ozkaya, and R. Nord, "Managing Technical Debt in Database Normalization," *IEEE Trans. Softw. Eng.*, vol. 48, no. 3, pp. 755–772, 2022, doi: 10.1109/TSE.2020.3001339.

[3] R. Angles, M. Arenas-Salinas, R. García, and B. Ingram, "An optimized relational database for querying structural patterns in proteins," *Database*, vol. 2024, no. 00, 2023, doi: 10.1093/database/baad093.

[4] B. Lemahieu, Seppe Broucke, *Principles of Database Management*, 2018, doi: 10.1017/9781316888773.

[5] A. N. Abyzov, "Balancing Data Normalization and Denormalization in Sports Competition Management Platforms: A Comparative Analysis," *Cyberleninka*, 2023.

[6] Y. Jani, "Optimizing Database Performance for Large-Scale Enterprise Applications," Oct. 2022, 2024, doi: 10.13140/RG.2.2.14180.59521.

[7] PhoenixNAP, "MySQL Performance Optimization - Day 5," pp. 1–16, 2021.

[8] F. Zhou and Y. Gao, "Performance Study on Normalization and Denormalization in MES System Databases," in *2024 Int. Conf. Intell. Comput. Data Mining (ICDM)*, IEEE, Sep. 2024, pp. 90–94, doi: 10.1109/ICDM63232.2024.10762243.

[9] Y. Zhang et al., "A fragmentation-aware redundancy elimination scheme for inline backup systems," *Future Gener. Comput. Syst.*, vol. 156, pp. 53–63, Jul. 2024, doi: 10.1016/j.future.2024.03.004.

[10] K. K. Tirupati, S. P. Singh, S. Nadukuru, S. Jain, and R. Agarwal, "Improving Database Performance with SQL Server Optimization Techniques," *Modern Dyn.: Math. Progressions*, vol. 1, no. 2, pp. 450–494, Aug. 2024, doi: 10.36676/mdmp. v1.i2.32.

[11] E. Akadal and M. H. Satman, "A Novel Automatic Relational Database Normalization Method," *Acta Informatica Pragensia*, vol. 11, no. 3, pp. 293–308, 2022, doi: 10.18267/j.aip.193.

[12] R. K. Rajendran and T. M. Priya, "Designing an Efficient and Scalable Relational Database Schema: Principles of Design for Data Modeling," in *The Software Principles of Design for Data Modeling*, 2023, pp. 168–176, doi: 10.4018/978-1-6684-9809-5.ch013.

[13] B. Alathari, "The Comparative Analysis for Data Normalization Using User Interface Normal Form," *Int. J. Recent Trends Eng. Res.*, vol. 4, no. 3, pp. 59–65, 2018, doi: 10.23883/ijrter.2018.4097.qlzhe.

[14] V. B. Ramu, "Optimizing Database Performance: Strategies for Efficient Query Execution and Resource Utilization," *Int. J. Comput. Trends Technol.*, vol. 71, no. 7, pp. 15–21, 2023, doi: 10.14445/22312803/ijctt-v71i7p103.

[15] N. Singh, "Normalization in DBMS (Normal Forms)," May 2023, *ResearchGate*.

[16] N. Amato, "Mastering Database Normalization: A Comprehensive Exploration of Normal Forms," *ResearchGate*, 2023.

[17] H. Köhler, "Finding Faithful Boyce–Codd Normal Form Decompositions," in *Proc. 2nd Int. Conf. Algorithmic Aspects Inf. Manag.*, Lecture Notes in Computer Science, Jun. 2006, doi: 10.1007/11775096_11.

[18] M. Fischer, P. Roessler, P. Sieben, J. Adamcic, C. Kirchherr, T. Sträubig, Y. Kaminsky, and F. Naumann, "BCNF* – From Normalized- to Star-Schemas and Back Again," in *Companion of the 2023 Int. Conf. on Management of Data (SIGMOD '23)*, New York, NY, USA: ACM, 2023, pp. 103–106, doi: 10.1145/3555041.3589712.

[19] T. Akbar, "Normalization in Database and Its Uses in Cloud Computing," *ResearchGate*, 2022.

[20] S. Shah, "A Systematic Method for On-The-Fly Denormalization of Relational Databases," pp. 1–10, 2020.

[21] I. C. Saidu, M. Yusuf, F. C. Nemariyi, and A. C. George, "Indexing Techniques and Structured Queries for Relational Databases Management Systems," *J. Niger. Soc. Phys. Sci.*, vol. 6, no. 4, 2024, doi: 10.46481/jnsps.2024.2155.

[22] M. V. Chikkamannur, A. A., and Praveena, "Indexing Strategies for Performance Optimization of Relational Databases," *Int. Res. J. Eng. Technol.*, no. May, pp. 3801–3805, 2021.

[23] A. Anchlia, "Enhancing Query Performance Through Relational Database Indexing," *Int. J. Comput. Trends Technol.*, vol. 72, no. 8, pp. 130–133, 2024, doi: 10.14445/22312803/IJCTT-V72I8P119.

[24] H. Gadde, "AI-Assisted Decision-Making in Database Normalization and Optimization," *Int. J. Mach. Learn. Res. Cyber Secur. Artif. Intell.*, vol. 11, no. 01, pp. 230–259, 2020.