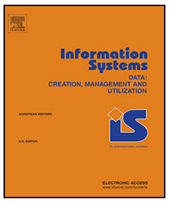
Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/is



The effects of database normalization on decision support system performance

Marin Fotache [a], Marius-Iulian Cluci [a,b], Toni Taipalus [c,*], George Talaba [a]

[a] *Department of Business Information Systems, Alexandru Ioan Cuza University of Iasi, Romania*
[b] *HVM SRL Iasi, Romania*
[c] *Faculty of Information Technology and Communication Sciences, Tampere University, Finland*

ARTICLE INFO

Keywords:
Database
Performance
Decision support system
Normalization
TPC-H benchmark

ABSTRACT

Relational database normalization strives to minimize update anomalies and data redundancy, often at the cost of performance. Transactional systems typically require a higher degree of normalization since data are updated more frequently than in read-intensive decision support systems. While these reasons for and effects of normalization can be considered common knowledge, there are hardly any empirical studies on the query performance implications of various degrees of normalization in decision support systems. That is, it seems that the magnitude of the effects of normalization is not widely understood, even though performance implications are of importance to managers and analysts utilizing decision analytics, and for end-user information needs to be timely satisfied. In this study, the effects of normalization on a decision support database were tested for three popular SQL/relational database servers. The results raise serious concerns about the conventional consensus on the performance gains incurred by the reduced number of table joins. Even for small-sized databases, the penalties due to the extra volume caused by redundancy associated with lower normal forms seem larger than the performance gains due to the reduced number of joins. These results have practical implications on which design principles should be followed for efficient decision support system databases.

1. Introduction

Database performance is paramount in decision support systems (DSS, often also called *online analytical processing*, or simply OLAP) due to its direct impact on the efficiency and effectiveness of decision-making processes. Decision support systems rely on timely access to large volumes of data for informing strategic decisions [1]. Database performance ensures that data retrieval operations are executed promptly, enabling end-users to analyze data without experiencing delays or disruptions. Additionally, optimal database performance facilitates the processing of complex analytical queries and the generation of real-time or near-real-time reports for addressing dynamic business requirements.

Traditionally, relational database design involves trade-offs between update anomalies and efficiency [2,3], generally referred to as database normalization. Even though the effects of database normalization on performance are known in theory, the magnitude of these effects has been scarcely studied in DSSs, if at all. Understanding the magnitude of the trade-offs in DSS database design benefits various information technology industries, as database design decisions are closely tied to the amount of hardware needed to store data and serve end-users, as well as how fast the queries of the end-users can be answered. In this study, we test the effects of database normalization on a DSS database performance and query completion success using the well-established TPC-H benchmark deployed on PostgreSQL, MySQL, and Microsoft SQL Server database management systems (DBMSs). As normalization affects several aspects in the analytics queries, such as the number of tables and joins, we also study the effects of query *constituents* (i.e., different syntactical and logical elements in the queries) with the following research questions (RQ):

- **RQ 1.1** How does the database's normal form affect successful query completion in a decision support system database?
- **RQ 1.2** Which constituents of queries explain differences in successful query completion in databases adhering to different normal forms?
- **RQ 2.1** How does the database's normal form affect query performance in a decision support system database?
- **RQ 2.2** Which constituents of queries explain differences in query execution times in databases adhering to different normal forms?

* Corresponding author.
E-mail address: toni.taipalus@tuni.fi (T. Taipalus).

https://doi.org/10.1016/j.is.2025.102636
Received 12 June 2024; Received in revised form 5 October 2025; Accepted 13 October 2025

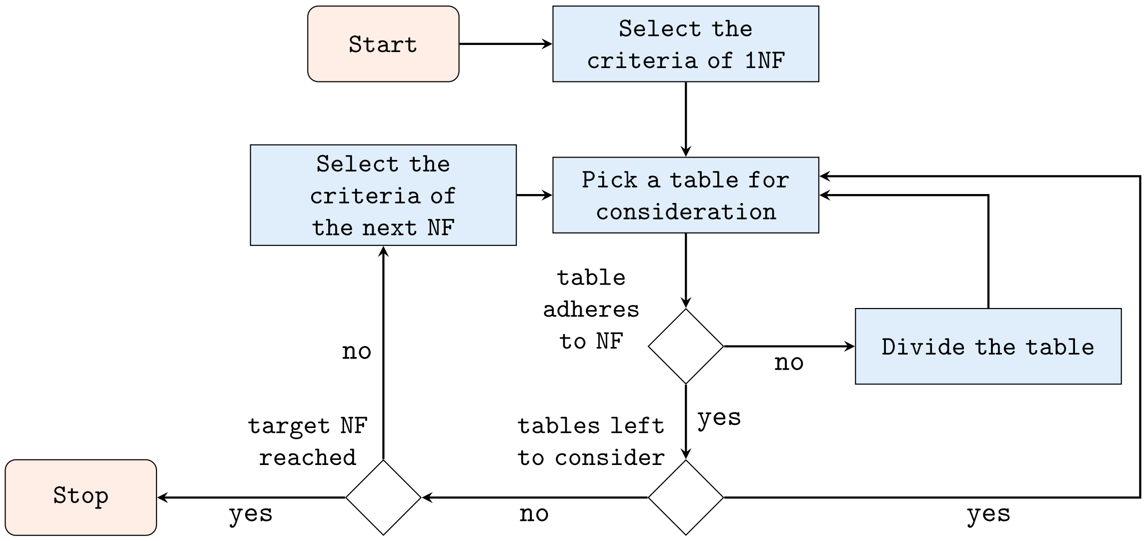


Fig. 1. An overview of the general process of relational database normalization through decomposition.

The results suggest that the performance gains associated with a reduced number of joins required in lower normal forms do not compensate for the penalties associated with retrieving data from larger tables specific to the lower normal forms. These results have practical and theoretical implications in decision support systems, enterprise resource planning data warehouses, operation cost planning, and logical database design, especially in read-intensive business domains.

The rest of the study is structured as follows. In the next section, we describe the theoretical background needed to understand the results of the study, namely normalization theory, database management systems, database performance, and benchmarking. In Section 3, we detail the benchmarking environment setup, hardware, and data analysis. Section 4 contains the results. In Section 5, we discuss the practical implications of the results and threats to validity. Section 6 concludes the study.

2. Theoretical background

2.1. Normalization theory

Database normalization theory was proposed along with the relational model [4,5], and is arguably well understood. Normalization through decomposition is an iterative process with formally defined steps called normal forms. As a set of guidelines for designing relational database schema [6,7], normalization is intended to minimize data redundancy, which consequently minimizes update anomalies and data inconsistencies [3,8]. However, normalization tends to penalize database querying, since some data that may have been retrievable from one table in a less normalized schema must be retrieved from several tables in schemas incorporating higher normal forms [3,8].

Normalization is an iterative relational database design process (Fig. 1). Each step in the iterative process is referred to as a normal form (NF), where each normal form incorporates more and more strict restrictions on the database structure to minimize data redundancy. Due to a lack of empirical knowledge, it is not widely understood up to which normal form databases should be normalized, but it is commonly accepted that the same normal form does not fit all databases [9]. Many normal forms rely on the concept of a functional dependency, which is a constraint between two sets of columns in a table that describes the relationship between the values of those attributes. More formally, in a relation (i.e., table) R, a functional dependency between two sets of attributes (i.e., columns) X and Y can be denoted as X → Y, where X and Y are subsets of the attributes of R. The functional dependency X → Y indicates that for every valid instance of the relation R, if two tuples (i.e., rows) have the same value for the attributes in set X, then they must also have the same value for the attributes in set Y. In other words, the value of Y is functionally dependent on the value of X [5].

Normal forms are often formally defined, yet in layman's terms, first normal form (1NF) forbids multiple values in a table cell, second normal form (2NF) requires that the columns in a table are functionally dependent on the whole set of columns which values differentiate the rows from each other, third normal form (3NF) forbids functional dependencies among non-key columns, and Boyce/Codd normal form (BCNF) forbids functional dependencies X → Y in cases where X is not a set of columns differentiating table rows from each other. Following the iterative process, there are other normal forms as well, both between [10] and after [11] the aforementioned normal forms. While normalization optimizes data modification, denormalization does the opposite — it optimizes data retrieval at the expense of data modification [12]. Sanders and Shin [13] provided a detailed review of denormalization, offering guidelines and a methodology for evaluating its effects through relational algebra. They highlighted the role of denormalization in enhancing query performance, aligned with application needs, yet they cautioned against its indiscriminate use without considering trade-offs in system performance. In a subsequent study, Shin and Sanders [14] explored various denormalization strategies, evaluating four prominent approaches across different scenarios, and developed a mathematical model for assessing the benefits of each pattern using cost–benefit analysis. They concluded that denormalization may offer positive effects on database performance – a view that normalization theory has supported since its inception.

2.2. Database performance assessment

Database performance is typically measured in throughput (i.e., how many concurrent clients the DBMS can serve) or in latency (i.e., how much time an operation requires). The former metric is typically used in transaction processing environments, in which different end-users compete for computing resources, and whose operations affect each other through, e.g., database locks and latches, and the latter metric is used in DSS databases [15]. The execution times of transactions in transaction processing are often measured in milliseconds, as the end-users are often business customers. In DSSs, the end-users are typically data analysts and managers creating insights from data. Their information requirements are answered by querying the database using statements written in SQL or some other query language. Depending on the requirements complexity and the database size, the statement execution often takes minutes, hours, or even more.

Lee [6] proposed a cost/benefit approach to measure the normalization effect on database performance with cost factors such as anomalies, storage requirements, and join operations in database queries. They emphasized the complexity of choosing the appropriate normal forms, proposing a systematic approach modeled as a decision tree. They hinted towards developing a computerized decision support system to enhance this methodology. Bock and Schrage [16] pointed out that denormalization requires rigorous system testing to prove the effects of denormalized tables on the processing efficiency, and that unanticipated *ad hoc* queries that use secondary data access paths may be adversely affected by denormalized table structures. Boscarioli et al. [17] explored the impact of normalization up to 3NF for data warehouse applications, but compared a normalized relational database to a NoSQL database, making the comparison problematic, as different data models and DBMS products are designed for different use-cases. Such performance comparisons have been questioned, as they are often reported inadequately to replicate, and do not consider the trade-offs of increased performance [15]. In summary, rigorously conducted empirical studies on database performance are scarce, and to the best of our knowledge, nonexistent in DSS databases.

Despite its age, the TPC-H benchmark [18,19] remains popular in decision support system benchmarking [15,20]. The benchmark proposes a real-world-resembling database structure, a set of 22 archetypal decision support queries, a module to populate the schema with a desired amount of data, and instructions on how to measure the DBMS performance. The TPC-H benchmark database consists of eight tables with a business domain that stores the product sales in a generic company. While there are several transaction processing system benchmarks such as TPC-C [21], LUBM [22] and YCSB [23], there a fewer DSS benchmarks, TPC-H being one of them, along with benchmarks such as TPC-DS, CAB [24] and M2Bench [25].

This study uses PostgreSQL, SQL Server and MySQL for benchmarking. PostgreSQL is an open-source object-relational database management system (DBMS) especially known for its stability, extensibility, and compliance with the SQL Standard [26,27]. PostgreSQL supports various indexing methods, including B^+-trees and hash indices, offering means for efficient data retrieval. The extensibility of PostgreSQL is evident through the support for user-defined data types, functions, and operators, allowing users to customize their database environment [28]. SQL Server, on the other hand, is an enterprise-scale relational DBMS tightly connected to Microsoft's .NET environment. MySQL has been the most popular open-source DBMS. In September 2025, DB-Engines (https://db-engines.com/en/ranking) ranked MySQL, SQL Server and PostgreSQL as the 2nd, the 3rd and the 4th most popular DMBSs.

3. Research method

In this section, we describe the research setting: logical and physical database setup, benchmarked queries and variables, and data analysis. The research setting is reported in detail in a GitHub repository,[1] which contains scripts and instructions needed to reproduce the benchmark, as well as our raw data results.

3.1. Logical database setup

One crucial aspect for our research setting is to create the same database in different normal forms in order to compare them. The 3NF database schema (Fig. 2) is provided by the TPC-H benchmark. We designed the 2NF schema by denormalizing a copy of the original 3NF schema into tables that adhere to 2NF but not to 3NF (Fig. 3). Next, we created a copy of the 2NF schema and denormalized its structure until it did not adhere to 2NF, but adhered to 1NF (Fig. 4). These three schemas form the basis of the performance comparison tests.

[1] https://github.com/marinfotache/normal_forms_and_sql_query_performance.

In the 3NF database schema (Fig. 2), tables can be joined within three axes originating in table `lineitem`:

(a) `lineitem-partsupp-part` which might be called the *product description* axis
(b) `lineitem-partsupp-supplier-nation-region` which might be called the *product provenance* axis
(c) `lineitem-orders-customer-nation-region` which might be called the *product sales* axis

In both 2NF and 1NF schemas, table names suggest the source of the data. For example, in the 2NF schema, table `lineitem_partsupp_part` is the result of merging three tables from the 3NF schema, namely `lineitem`, `partsupp`, and `part`. As expected, due to denormalization some duplicates occur, such as with `l_partkey`, `ps_partkey` and `p_partkey` in table `lineitem_partsupp_part` of the 2NF schema. Table names containing underscores also indicate that they are merged from two or more tables of the original 3NF schema. In 1NF the TPC-H database schema contains three tables that correspond to the above three join axes.

Moreover, some queries may demand computing resources beyond those available in the current system, and they are usually aborted by the DBMS. Consequently, in this paper, we used an additional performance metric, i.e., query completion, which signals if the query was completed within the 30 min timeout or canceled by the DBMS.

The goal of the experiment in this paper was to examine query performance on three schema variants of the TPC-H database: 3NF, 2NF, and 1NF. Each database schema was tested with two database sizes, namely 0.1 GB and 1.0 GB of data spread among the tables. The dataset sizes are referred to as the *scale factors* (SF) as per TPC-H vocabulary. Overall, the tests were run as follows:

1. Create a schema for the 3NF schema.
2. Create and populate the original 3NF schema using the TPC-H's *DBGen* utility with a scale factor of 0.1 GB.
3. Generate 1000 SQL queries for the original 3NF schema.
4. Set a timeout of 30 min for each query execution. Queries exceeding the execution time of 30 min will be considered *aborted* due to timeout. Other queries might be canceled since they require resources exceeding those available in the physical setup described in Section 3.2.
5. Execute the 1000 queries and collect data on query completion (binary, i.e., *completed* or *canceled*) and query duration in seconds.
6. Create and populate the 2NF schema; 2NF tables were populated by extracting data from the 3NF schema tables.
7. Create logical equivalents of the initial 1000 queries for the 2NF schema.
8. Repeat steps 4. and 5. for the 2NF schema.
9. Create and populate the 1NF schema; 1NF tables were populated by extracting data from the 3NF schema tables.
10. Create logical equivalents of the initial 1000 queries for the 1NF schema.
11. Repeat steps 4. and 5. for the 1NF schema.
12. Delete the 1NF and 2NF schemas along with their data, and delete all the data from the 3NF schema. Repeat the steps from 3. to 11, but using a scale factor of 1.0 GB.

This 12-step scenario was executed for each of the three DBMSs, sharing the query sets, with some tweaks related to SQL syntax differences. A small number of PostgreSQL queries could not be conveniently converted into SQL Server's SQL dialect. For example, in SQL Server, `ORDER BY` items can appear in the select list only if `SELECT DISTINCT` is specified.

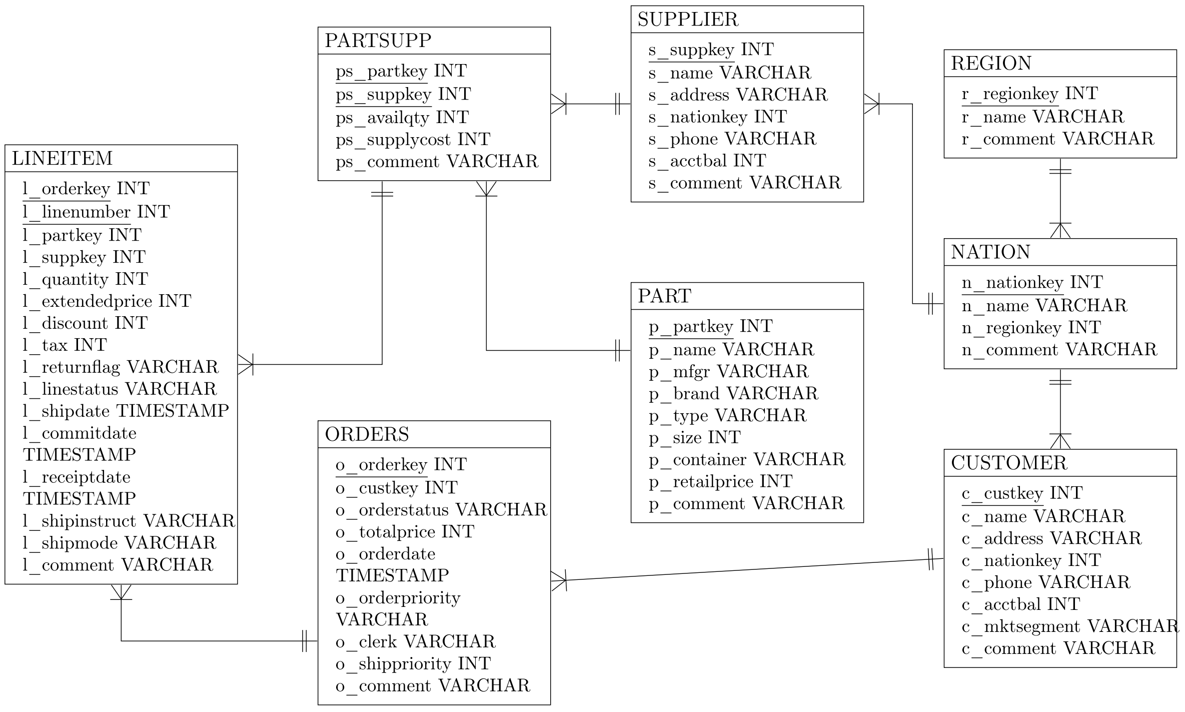


Fig. 2. The original TPC-H schema, which adheres to 3NF.

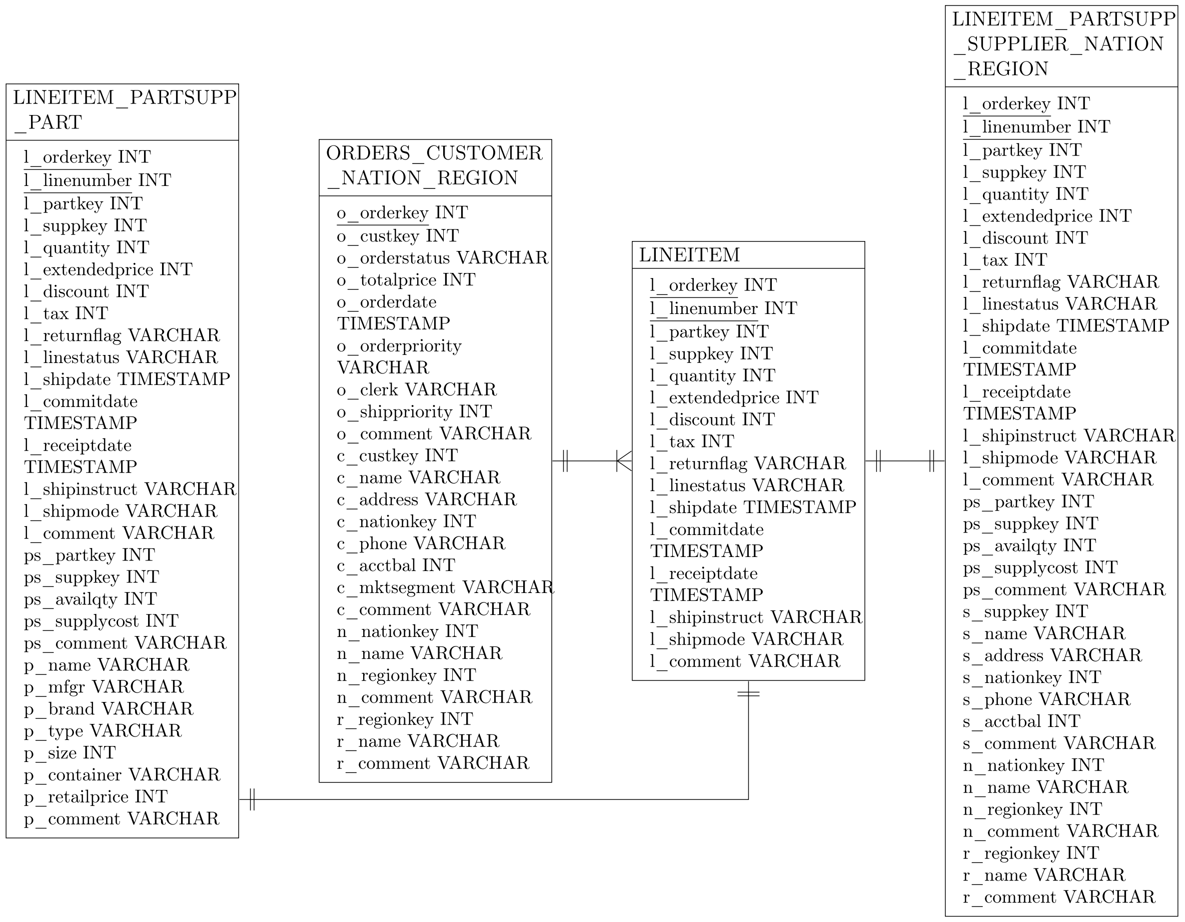


Fig. 3. TPC-H database schema in 2NF.

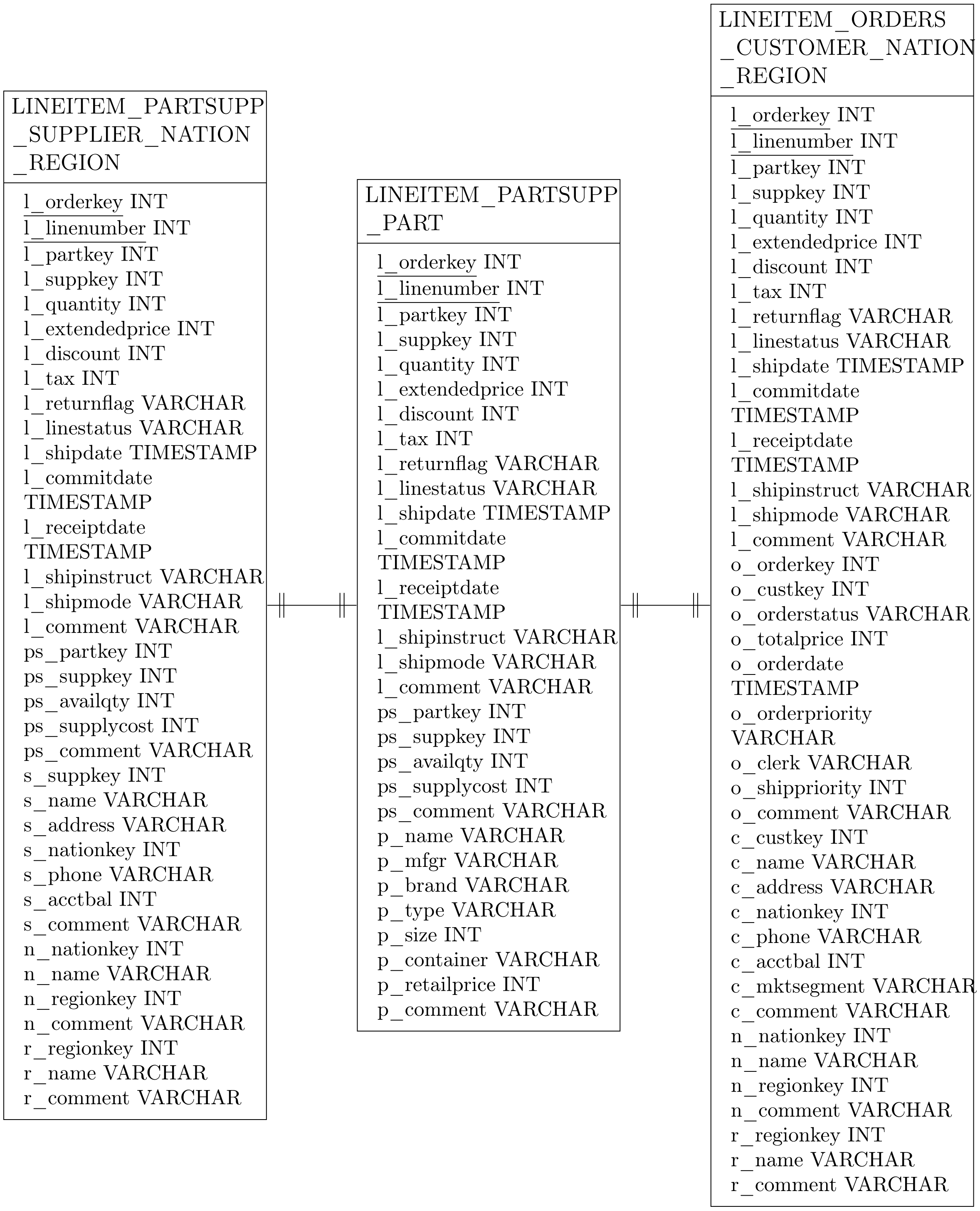


Fig. 4. TPC-H database schema in 1NF.

3.2. Physical database setup

The tests were run in an OpenStack cloud environment using a single-node server. The environment and hardware are detailed in Table 1. The physical database structures for all schemas were kept unoptimized, except for the indexes associated with the primary keys and the foreign keys. That is, as DSSs typically process *ad hoc* queries rather than queries embedded in the host language of the system, the physical structures of DSSs often lack supporting data structures such as indexes. To simulate the nature of decision support databases, and to not impose any advantage to some queries over others, we did not create any additional index in any of the schemas.

3.3. Benchmark queries

Although the TPC-H benchmark includes 22 fine-crafted variants of DSS queries, using them in this study was not possible, as their small number makes them less prone to statistical analysis. Instead, we computationally generated 1000 different SQL queries which were also typical analytical queries encompassing several tables, filters, tuple grouping, sorting, etc. These queries were initially designed for the

```
 1 -- Q2302110937000003
 2 SELECT RTRIM(t1.n_name)        AS RTRIM__t1__n_name,
 3     t1.n_comment               AS t1__n_comment,
 4     t1.n_nationkey             AS t1__n_nationkey,
 5     MIN(LTRIM(t2.n_comment))   AS MIN__LTRIM__t2__n_comment,
 6     MIN(t2.s_suppkey)          AS MIN__t2__s_suppkey
 7 FROM
 8  (SELECT * FROM nation) t1
 9   RIGHT JOIN
10  (SELECT * FROM supplier supplier2
11  INNER JOIN nation nation2 ON supplier2.s_nationkey = nation2.n_nationkey) t2
12   ON t1.n_nationkey = t2.n_nationkey
13 WHERE t2.n_name NOT BETWEEN 'EGYPT      ' AND 'MOROCCO    '
14  AND t1.n_regionkey NOT BETWEEN 0 AND 3
15  AND t2.n_comment LIKE '%lar, ironi'
16 GROUP BY RTRIM(t1.n_name),
17          t1.n_comment,
18          t1.n_nationkey
19 HAVING MIN(t2.s_suppkey) <= (
20  SELECT MIN(t2.s_suppkey) AS MIN__t2__s_suppkey
21  FROM
22   (SELECT * FROM nation) t1
23    RIGHT JOIN
24   (SELECT * FROM supplier supplier2
25   INNER JOIN nation nation2 ON supplier2.s_nationkey = nation2.n_nationkey) t2
26    ON t1.n_nationkey = t2.n_nationkey
27  WHERE t2.n_name NOT BETWEEN 'EGYPT      ' AND 'MOROCCO    '
28  AND t1.n_regionkey NOT BETWEEN 0 AND 3
29  AND t2.n_comment LIKE '%lar, ironi'
30  AND t1.n_comment <= 'ously. final express gifts cajole a'
31  AND t2.s_address BETWEEN 'bWwHUuQgVo689rHdr9S7tX2czhAeL3Lp4MU1m6W' AND 'tfZRUl9jXa j'
32  AND t2.n_regionkey <> 3
33  )
34  OR MIN(LTRIM(t2.n_comment)) >= (
35   SELECT MIN(LTRIM(t2.n_comment)) AS MIN__LTRIM__t2__n_comment
36   FROM
37    (SELECT * FROM nation) t1
38     RIGHT JOIN
39    (SELECT * FROM supplier supplier2
40    INNER JOIN nation nation2 ON supplier2.s_nationkey = nation2.n_nationkey) t2
41     ON t1.n_nationkey = t2.n_nationkey
42   WHERE t1.n_regionkey NOT BETWEEN 0 AND 3
43   AND t2.n_comment LIKE '%lar, ironi'
44   AND t2.s_name NOT BETWEEN 'Supplier#00001653  ' AND 'Supplier#00007455  '
45   AND t2.s_comment > 'deposits cajole slyly! ironic, silent accounts breach. carefu'
46  )
47 LIMIT 297;
```

Fig. 5. One of the 1000 queries run on the 3NF schema (PostgreSQL).

Table 1
Hardware and software used.

Component	Description
CPU	Xeon-Gold 6240 @ 2.6 GHz (8 vcores)
Memory	16 GB DDR4 @ 2933 MHz
Disk	RAID-5 NL-SAS HDDs (12K IOPS)
Environment	OpenStack cloud
Benchmark	TPC-H 3.0.0 with custom queries
OS	Ubuntu 20.04.3 LTS
DBMS 1	PostgreSQL 16.0
DBMS 2	SQL Server 2019
DBMS 3	MySQL 8.0.42

3NF schema, yet logical equivalents of the queries were created for the 2NF and 1NF schemas. Fig. 5 shows an example query (written in PostgreSQL's SQL dialect) of the initial 1000-query sets used in our tests, which will also serve to define the variables used in subsequent statistical tests and machine learning models.

The generated queries contain SQL features typical for analytics, such as scalar and non-scalar subqueries, GROUP BY, HAVING, and ORDER BY clauses with SKIP and OFFSET. Queries were generated randomly, limiting some parameters to control the query complexity, described in more detail in Section 4.

Expressions in the WHERE and HAVING clauses may contain operators such as BETWEEN, IN and LIKE, as well as classical comparison operators. Tuple groups created with the GROUP BY clause are based on primitive table attributes or scalar functions such as RTRIM().

The main driver for the query complexity is the number of subqueries declared in the FROM clause(s). Each set of subqueries included in FROM is called a *join path* and it retrieves data from any subset of tables in join axes (a)–(c) defined in Section 3.1. The query in Fig. 5 retrieves data from two join paths using two subqueries which materialize in *ad hoc* tables t1 (line 8) and t2 (line 11) for reference in subsequent sections. Table 2 describes the join paths of the query in Fig. 5 in all three schemas (1NF, 2NF, 3NF). For t1, even if the information may be retrieved from a single 3NF table (nation), in 2NF the same information must be extracted from a much larger

Table 2
Join paths for the three variations of the query shown in Fig. 5.

Normal form	Join path	Content of the derived table
3NF	t1	A single table.
3NF	t2	Two tables with an inner join.
2NF	t1	A single table incorporating four 3NF tables.
2NF	t2	A single table incorporating five 3NF tables.
1NF	t1	A single table incorporating five 3NF tables.
1NF	t2	A single table incorporating five 3NF tables.

table (`orders_customer_nation_region`), whereas in the 1NF schema, the table is even larger (`lineitem_orders_customer_nation_region`). In contrast, for `t2` a table join is required in the 3NF schema, but not in the 2NF or 1NF schemas.

The query generator can handle any number of join paths, but queries in this paper were limited to three join paths. Within a join path, tables may be joined using `INNER` or `RIGHT OUTER` joins. When two join paths can be joined (using `INNER` or any type of `OUTER` joins) by two or more tables, the linking table is chosen at random (in Fig. 5, `nation` is the linking table). For the query sets in this paper, the join paths may be joined only by the primary key of the linking table (lines 9 and 12). Subqueries may appear only in the `FROM` clause (for defining derived tables, such as `t1` and `t2`) or the `HAVING` clause, but neither in the `WHERE` nor in the `SELECT` clauses.

Each initial 1000-query set was generated for a given database scale factor and the query predicates included attribute values specific to that scale factor content. Within the initial query, queries are independent from one another. When converting the initial queries for execution into the 2NF schema, and then into the 3NF schema, all clauses are kept unchanged except for the `FROM` clause. All `FROM` clauses included in the main query and all its subqueries were updated so that the number of necessary joins was minimized.

3.4. Variables

Table 3 contains the list of variables used in all subsequent tests and machine learning models. Most variables are related to the query complexity and result size, i.e., the numbers of filtering predicates for tuples (in `WHERE`) and groups (in `HAVING`), the numbers of result columns and rows, etc.

The remaining variables in Table 3 refer to: the DBMS (`dbserver`); the database size (`scale_factor`); the normal form of the database (`normal_form`); a binary variable (`query_completion`) with two values, *canceled* if the query could not be completed within the 30-min timeout, and *completed* when the execution was successful; and the execution time in seconds of the completed queries (`duration_sec`). As Section 4 will show, due to the skewness of its distribution, in the ML scoring models variable `duration_sec` was transformed using the `log10` function.

Variables of interest such as the number of joins in the main `FROM` clause or the size of the processed tables were not included in the analysis since they are largely correlated with the normal form and the database size.

3.5. Data analysis

RQs 1.1 and 2.1 (i.e., the association between normalization and query completion and duration) were answered with statistical tests, while RQs 1.2 and 2.2 (i.e., the importance of different query constituents in different normal forms for query completion and duration) were assessed with machine learning models. An alpha level of $\alpha = .05$ was selected for all statistical tests.

For RQ 1.1, the statistical significance of the association between query completing successfully (i.e., within the 1,800 s timeout; a binary outcome) and the database normal form was assessed with Cochran's Q test, which is an extension of McNemar's test [29,30] for paired data. The null hypothesis assumes equal proportions for query completion success across all normal forms, while the alternative hypothesis suggests that at least one normal form differs.

For RQ 1.2, a series of machine learning models were built and tuned to predict the successful completion of the query within the 30 min timeout. The outcome variable, `query_completion`, is binary (*canceled* vs. *completed*). Predictors were all variables in Table 3 except for the query duration (`duration_sec`). The classification models were built using two of the most popular algorithms, random forest (RF) and extreme gradient boosting (XGBoost). Growing ensembles of classification or regression trees [31,32], RF [33] and XGBoost [34,35] have recorded good performance in both classification and regression problems [36]. Whereas RF performs better in variance reduction, XGBoost is better in bias reduction [37].

The importance of normal form among other predictors of query completion and the patterns of relationships between main predictors and the outcome were examined with techniques related to Interpretable Machine Learning [38–40]. Permutation-based variable importance methods measure how much the model performance changes when the effect of a selected predictor is removed through perturbations, i.e., permutations of the predictor's values [40]. The shape of the relationships between model predictors and the outcome was examined using techniques such as Partial Dependency Plots, and Accumulated Local Effects [38,40,41].

For RQ 2.1, the association between the normal form and the duration of the completed queries was assessed with Friedman's test [29] which is a non-parametric alternative to the one-way repeated measures ANOVA test. It estimates the statistical significance of differences between the distributions of three or more paired groups when the distribution of the outcome variable is not normal. In our case, the null hypothesis of Friedman's test assumes there are no significant differences in query duration among the three normal forms.

For RQ 2.2, another series of machine learning models were built and refined to predict the duration of completed queries. The initial outcome variable was query duration in seconds, but since it was highly skewed, it was transformed with the `log10` function, i.e., with common logarithm. The models were built using the same algorithms as in RQ 1.2, i.e., RF and XGBoost. The predictor's importance and the shape of the association between predictors and the outcome were examined using the same Interpretable Machine Learning techniques as for the classification models.

Similar to other machine learning algorithms, random forest and XGBoost have hyperparameters that cannot be learned directly from the data, but they need to be optimized [42]. In this paper, two RF hyperparameters were tuned in both classification and regression models: (1) the number of random attributes used at each node split (`mtry`), and (2) the minimum number of observations in a node as a requirement for further splitting (`min_n`). The number of tree parameters was fixed at 700 in all the RF models. Five hyper-parameters were tuned in XGBoost models: (1) the learning rate (`learn_rate`), (2) the minimum reduction in the loss function for proceeding to a further split (`loss_reduction`), (3) the maximum depth of the tree (`tree_depth`), (4) the size of the random samples (`sample_size`), (5) `min_n`, and (6) `mtry`. The later two hyperparameters are the same as in the RF models. The number of trees was fixed for all XGBoost models to 1000.

Random grid search was the preferred method used to find the best combination of hyperparameters [43]. Grids included 100 combinations of tuned hyperparameters in the RF-trained models and 300 combinations of tuned hyperparameters in the XGBoost-trained models. The performance of the classification models was assessed using the receiver operating characteristic area under the curve (ROC-AUC) metric [44]. The performance of scoring models was assessed with the root mean square error (RMSE) metric, but also the coefficient of determination (R2) was used to compare the models [44] since it is easier

Table 3
Variable names used in the analyses, and their descriptions.

Variable	Description
`SELECT_cols`	No. of columns in the result table
`SELECT_non_aggr_func`	No. of all non-aggregate functions appearing in the main SELECT clause
`SELECT_SUBSTR`	No. of SUBSTR functions in the main SELECT clause
`SELECT_date_func`	No. of date functions in the main SELECT clause
`SELECT_aggr_func`	No. of aggregate functions in the main SELECT clause
`FROM_join_paths`	No. of join paths in the main FROM clause
`WHERE_predicates`	No. of predicates in the WHERE clause
`WHERE_pkey_attribs`	No. of primary key attributes in WHERE clauses
`WHERE__between`	No. of BETWEEN operators in WHERE clauses
`WHERE__in`	No. of IN operators in WHERE clauses
`WHERE__like`	No. of LIKE operators in WHERE clauses
`WHERE_non_aggr_func`	No. of non-aggregate functions in the main WHERE clauses
`WHERE_func__date`	No. of date functions in the main WHERE clause
`GROUP_BY_cols`	No. of GROUP BY columns
`HAVING_non_scalar_subq`	No. of non-scalar subqueries in the HAVING clause
`HAVING_scalar_subq`	No. of scalar subqueries in the HAVING clause
`ORDER_BY_cols`	No. of ORDER BY columns
`limit`	No. of rows in the result table
`offset`	No. of rows skipped in the result table
`dbserver`	`mssqlserver` or `postgresql`
`scale_factor`	Database size (0.1 GB or 1.0 GB)
`normal_form`	Database normal form (1NF, 2NF or 3NF)
`query_completion`	Whether the query execution was completed during the 30 min timeout
`duration_sec`	Query execution times (in seconds) for each completed query

to interpret. To reduce overfitting, repeated (five times) five-fold cross-validation of the training set was combined with random grid search for hyperparameter tuning [44]. Initial datasets were split into the training and testing subsets, with a split ratio of 75/25. The stratified split for the classification models enforced the same distribution of the binary outcome in both the training and the testing subsets.

All exploratory data analysis, statistical tests, and machine learning modeling were performed in R [45]. The `tidyverse` ecosystem of R packages [46] served as the main tool for data preparation and exploratory data analysis. Most charts were generated by the `ggplot2` package, which is part of the `tidyverse`, with support from other packages, such as `ggforce` [47] and `corrplot` [48]. Cochran's Q test and the pairwise comparisons for RQ1.1 were performed with the package `rstatix` [49]. Package `ggstatsplot` [50] was utilized for the Friedman's test (RQ2.1).

The `tidymodels` ecosystem of packages [51] was employed for model building and tuning. The RF models were fitted with the `ranger` engine [52], whereas the engine used for building the XGBoost models was `xgboost` [53]. All the Interpretable Machine Learning techniques (Variable Importance, Partial Dependency Profiles, Individual Conditional Expectation, and Accumulated Local Effects Profiles) were deployed using the `DALEX` ecosystem [54,55].

4. Results

All data analysis techniques described in Section 3.5 were applied separately for PostgreSQL, SQL Server, and MySQL. In this paper, we were not concerned with the performance comparison between DBMSs, but whether the normalization results are consistent across DBMSs. Of the initial set of 1000 queries generated for each scale factor in PostgreSQL, only 983 could be adapted to perform identical tasks in SQL Server due to differences in SQL dialects. All of 1000 queries were adapted with minor adjustments to run in MySQL.

4.1. Higher normal form facilitates queries completing successfully

For RQ 1.1, the completion of queries in all DBMSs among scale factors and normal forms is summarized in Fig. 6. For all DBMSs and scale factors, the figure suggests that increasing the normal form is associated with better chances of query successful completion (within the 30 min timeout). For the tiny scale factor of 0.1 GB, in SQL Server the percentage of successful queries increased from 74% (1NF), to 77% (2NF), and to 89% (3NF). The trend was similar in MySQL: 70% (1NF), 71% (2NF), and 88% (3NF) and PostgreSQL: 72% (1NF), 75% (2NF), and 90% (3NF).

By increasing the scale factor to 1 GB, the percentages of successful queries decreased for all normal forms and DBMSs, since the queries processed larger amounts of data which requires longer execution times (sometimes over the 30 min timeout), or more computing resources than those provided by the current system. Nevertheless, among normal forms, the success rate increased from 60% (1NF), to 62% (2NF), and to 78% (3NF) in SQL Server, from 59% (1NF), to 60% (2NF), and to 78% (3NF) in MySQL, and from 56% (1NF), to 59% (2NF) and to 81% (3NF) in PostgreSQL. Migrating the database from 2FN to 3NF appeared particularly important for query completion.

Cochran's Q test determined that, for all DBMSs, there was a statistically significant difference in the proportion of successful queries among the normal forms: $\chi^2(2) = 477$, $p < .001$ for SQL Server, $\chi^2(2) = 540$, $p < .001$ for MySQL, and $\chi^2(2) = 679$, $p < .001$ for PostgreSQL. There is enough statistical evidence to state that the normal form is associated with successful query completion, for both scale factors, given the 30-min timeout set for the query completion. In all three DBMSs, all pairwise comparisons (i.e., 1NF-2NF, 1NF-3NF, and 2NF-3NF) were statistically significant ($p < .001$), except for the 1NF-2NF pair in MySQL.

4.2. Normal form is ranked as the second most important predictor for queries completing successfully

For RQ 1.2, the importance of the query constituents resulting from database normalization in query completion was assessed with a series of machine learning classification models using two popular algorithms, random forest (RF), and extreme gradient boosting (XGBoost), as described in Section 3.5. We fitted and refined separate models for each database server, to check if the results of (de)normalization are similar.

The best models for RF and XGBoost were selected among 5 cross-validation folds based on their performance, as assessed with the ROC-AUC metric. For all three DBMSs, the best RF performance was recorded for models with three randomly selected features at each split (`mtry`) and four minimum observations required to split a node further (`min_n`). SQL Server and PostgreSQL shared the best combination of hyperparameters for the XGBoost classification models: 9 features at each split (`mtry`); 11 minimum observations required to split a

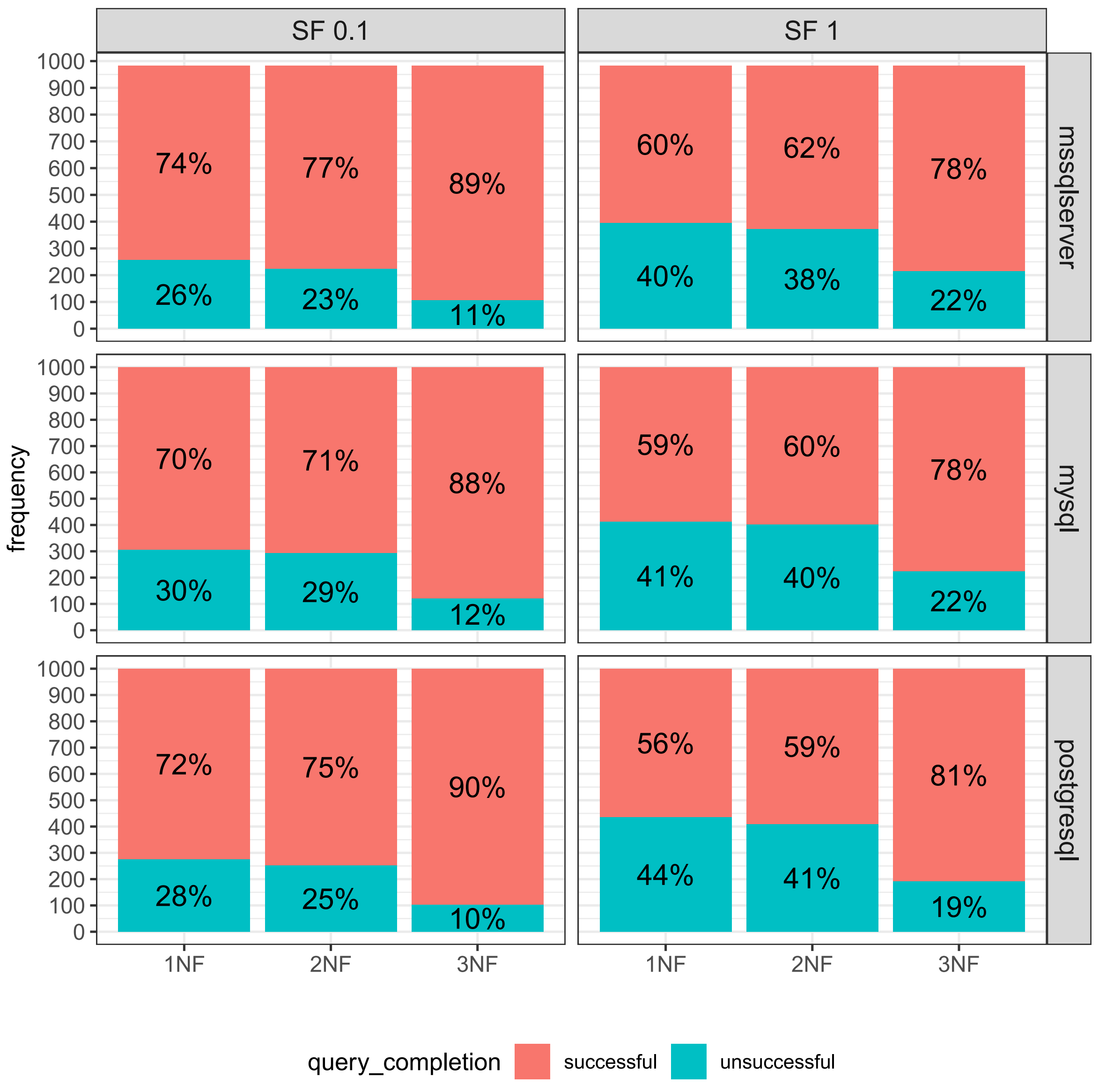


Fig. 6. Query completion among DBMSs, scale factors, and normal forms.

node further (min_n); maximum 10 levels of depth for each tree in the boosting ensemble (tree_depth); a learning rate of 0.086 (learn_rate); a minimum amount of loss reduction required to make a further partition on a leaf node of 0.00000423 (loss_reduction); a sample size of 0.928 (sample_size). For MySQL, the best average ROC-AUC across the cross-validation folds was recorded for mtry = 19, min_n = 8, tree_depth = 13, learn_rate = 0.067, loss_reduction = 1.55, and sample_size = 0.866.

All selected models seem reliable since they recorded very good performance on new data, i.e., the test set, with RF models slightly outperforming their XGB counterparts. The ROC-AUC for the selected RF models was 0.945 for SQL Server, 0.947 for MySQL and 0.943 for PostgreSQL, whereas the selected XGBoost models recorded the ROC-AUC of 0.933 for SQL Server, 0.930 for MySQL and 0.935 for PostgreSQL. Also, in terms of accuracy, the RF final models performed better than the XGBoost selected models: 0.902 vs. 0.883 for SQL Server, 0.891 vs. 0.870 for MySQL, and 0.885 vs. 0.881 for PostgreSQL.

The results in Fig. 7 show that, for all three DBMSs, the number of join paths (FROM_join_paths) is the most important predictor for queries completing successfully, followed by the normal form. The importance of the remaining predictors is notably smaller. This was surprising, especially for the scale factor, since we expected the database size to play a greater role in the query completion. Admittedly, the variation of the database size was small, since the tests covered only the 0.1 GB and 1.0 GB scale factors. The variable importance plots were drawn only for the selected RF models since they outperformed XGBoost models.

For the four most important predictors (as ranked by the selected RF classification models) associated with the probability of a query being completed, Fig. 8 presents two popular plots for the model-level interpretation, i.e., Partial Dependency Profiles (PDP) and Accumulated Local Effects (ALE) Profiles. They complement each other and, when converging, the results are more reliable. Each of the profiles shows how an increase in the predictor's value (displayed on the x-axis) is associated with an increase or decrease in the probability (displayed on the y-axis) of the query being completed. The shape of the plots is similar for all four predictors and three DBMSs.

Increasing the number of join paths (from 1 to 2, and then to 3) is associated with a steep probability decrease in the query being completed within the 30 min timeout. By contrast, when increasing the normal form, the probability increases; the probability increase is steeper when moving from 2NF to 3NF than when the database normal form increases from 1NF to 2NF. Increasing the database size from 0.1 GB to 1.0 GB decreases the odds that the query would be completed,

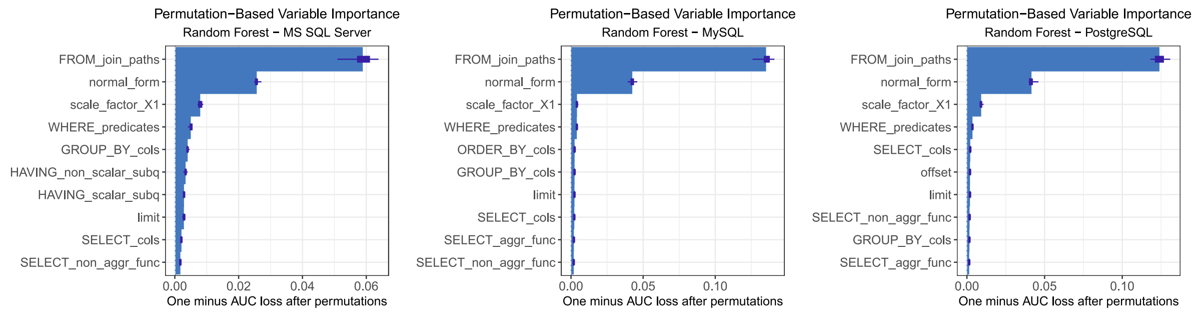


Fig. 7. Ten most important predictors for queries completing successfully in the random forest classification models.

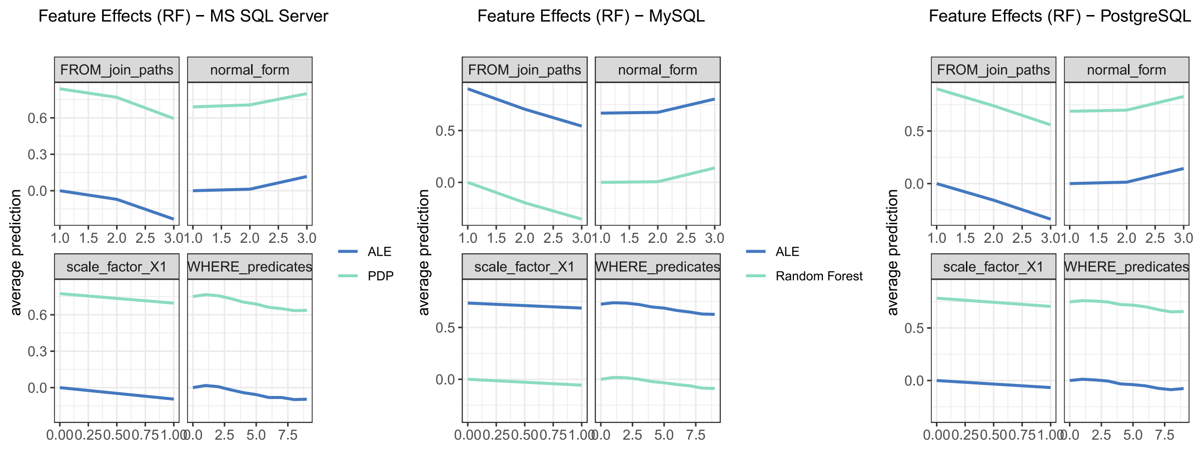


Fig. 8. Feature effects of the selected random forest classification model for the most important four predictors of successful query execution; probability of execution to be successful is on the y axis; green lines represent partial dependency profiles and blue lines represent accumulated local effects profiles. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

but the decrease is smaller than in the case of the number of join paths. Finally, Fig. 8 suggests that a larger number of predicates included in the main WHERE clause (this number varied between 0 and 9) slightly decreased the odds of the query being completed.

4.3. Higher normal form facilitates faster query execution time

For analyzing RQ 2.1, we removed queries that were not completed in all three normal forms and all DBMSs. This resulted, for each DBMS, in 598 queries for the scale factor of 0.1 GB and 436 queries for the scale factor of 1.0 GB. The statistical significance of the association between query duration and database normal form was assessed with Friedman's test, as duration was not normally distributed.

Results in Fig. 9, as provided by the `ggstatsplot` package, revealed statistically significant relationships between the normal and the query duration for all servers when the queries of both scale factors were tested together: $\chi^2(2) = 635$, $p < .001$ for SQL Server, $\chi^2(2) = 254$, $p < .001$ for MySQL, and $\chi^2(2) = 559$, $p < .001$ for PostgreSQL. The effect size, as measured by Kendall's W, was found to be 0.31 (CI95% [0.28, 1.00]) for SQL Server, 0.12 (CI95% [0.10, 1.00]) for MySQL, and 0.27 (CI95% [0.23, 1.00]) for PostgreSQL. For a TPC-H database size under 1 GB, the intensity of the relationship between query duration and the database normal form seems to be moderate in SQL Server and PostgreSQL, but weaker in MySQL.

Durbin-Conover-corrected pairwise comparisons were conducted as *post hoc* tests, revealing that, for each DBMS, all pairwise comparisons between the normal forms in both scale factors were statistically significant ($p < .001$).

4.4. Normal form is ranked as the most important predictor for query execution time

As with RQ 2.1, for RQ 2.2 we removed queries that were not completed in any of the three database schemas. For each of the three DBMSS, a series of scoring models (based also on RF and XGBoost) were built and tuned to predict $\log_{10}$ of query duration (we opted to transform duration with the logarithm because of its skewed distribution) in relation to all variables in Table 3 except for `query_completion`. The best RF and XGBoost models were identified by tuning the same hyper-parameters as in the classification models, selecting at random (by the method of random search) 100 combinations of hyper-parameters for the RF models and 300 combinations for the XGB models; model performance was assessed across five cross-validation folds with the root mean square error (RMSE) metric.

For both SQL Server and MySQL, the lowest RMSE in RF models was recorded for `mtry` = 9 and `min_n` = 4 (for the full names of hyper-parameters, see the classification models in Section 4.2). For PostgreSQL, the best RF models had `mtry` = 4 and `min_n` = 5.

In the XGBoost models, all three DBMSs shared the best combinations of hyper-parameters: `mtry` = 4, `min_n` = 5, `tree_depth` = 6, `learn_rate` = 0.0309, `loss_reduction` = 2.00e−10, and `sample_size` = 0.944.

While RMSE was the metric used to identify the best model, the coefficient of determination (R^2) is easier to interpret, as a percentage of the outcome variability explained by the model. On the test set, the selected XGBoost models recorded RMSE = 0.625 and R^2 = 0.701 for SQL Server, RMSE = 0.700 and R^2 = 0.674 for MySQL, and RMSE = 0.701 and R^2 = 0.654) for PostgreSQL, outperforming their RF counterparts, where RMSE = 0.701 and R^2 = 0.635 for SQL Server, RMSE = 0.725 and R^2 = 0.658 for MySQL, and RMSE = 0.739 and R^2 = 0.636) for PostgreSQL.

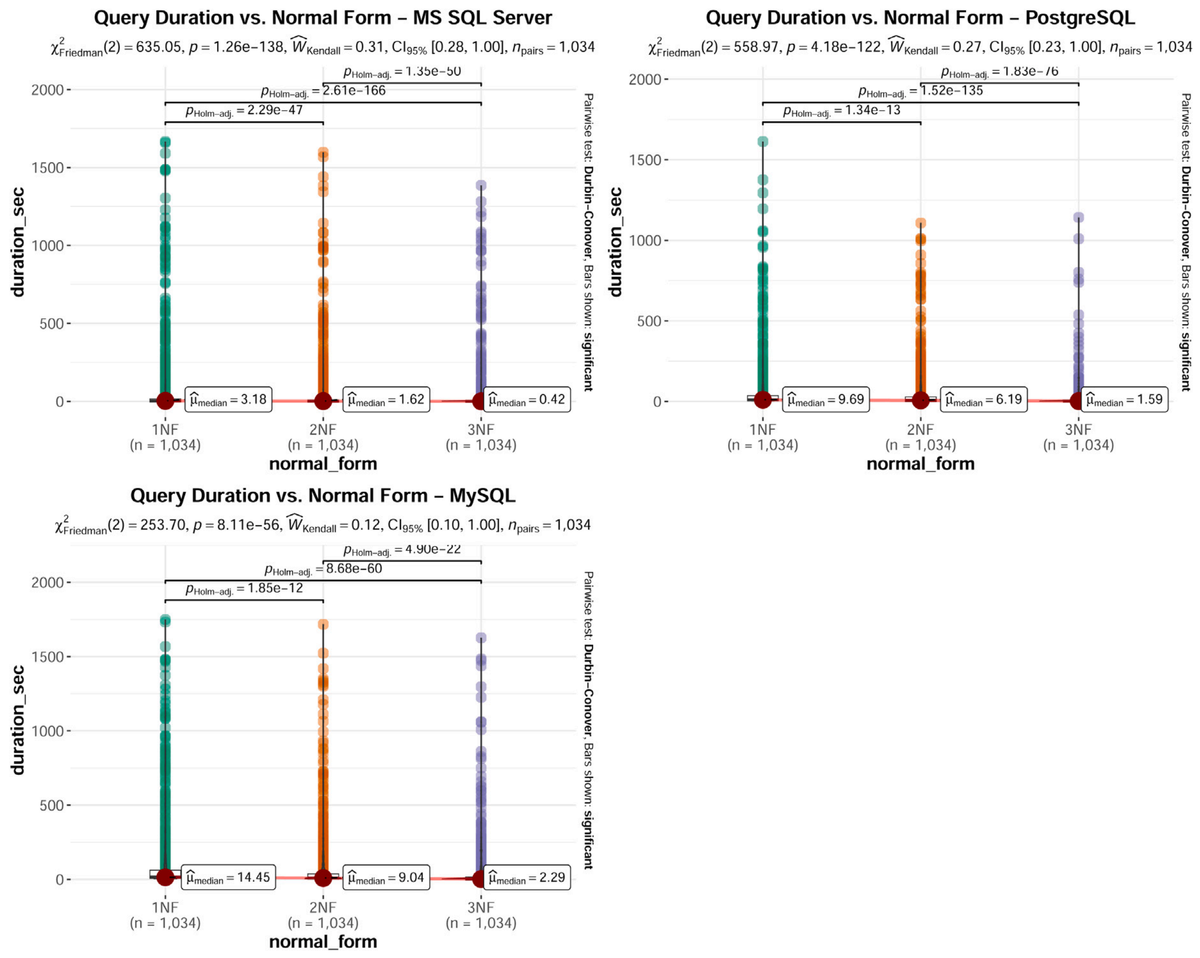


Fig. 9. Association between the query duration and the database normal form (both scale factors).

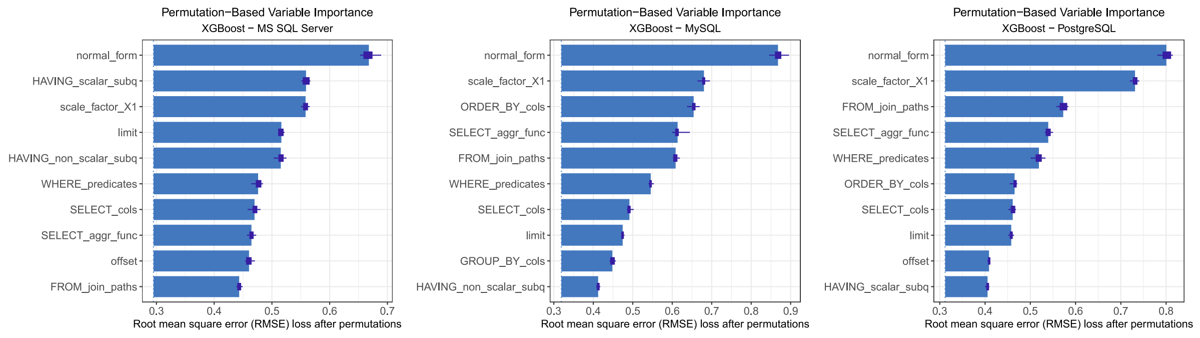


Fig. 10. Ten most important predictors for query execution time in the best (selected) XGBoost scoring models.

Since XGBoost models recorded better performances for all three servers, subsequent analyses do not consider the RF models, even if their results largely converge (e.g., in ranking the normal form as the most important predictor). As Fig. 10 shows, XGBoost found the normal form as the most important predictor for explaining the variability of query duration in all three DBMSs. As for the other predictors' importance, there are considerable differences that could be explained by the server's query engine features.

Fig. 11 shows the Partial Dependency Profiles and the Accumulated Local Effects Profiles of the two most important predictors (as assessed by the XGBoost models) associated with the $\log_{10}$ of the query duration for all three DBMSs. Interpretation is similar to Fig. 8, but here, on the y-axis, the predicted value of the outcome ($\log_{10}$ of the query duration) is represented.

The shape of PDP and ALE profiles is similar for each of the four predictors on all servers. When increasing the normal form of the database, the duration of the query decreases; the effect is greater when moving the database schema from 2NF to 3NF.

5. Discussion

5.1. Practical implications

The key findings of this study were: *(i)* higher normal forms (2NF, 3NF) improve the likelihood of queries completing successfully within a 30 minute timeout, *(ii)* the number of join paths is the top predictor for successful query completion, followed by database normal form, and *(iii)* higher normal forms facilitate faster query execution times,

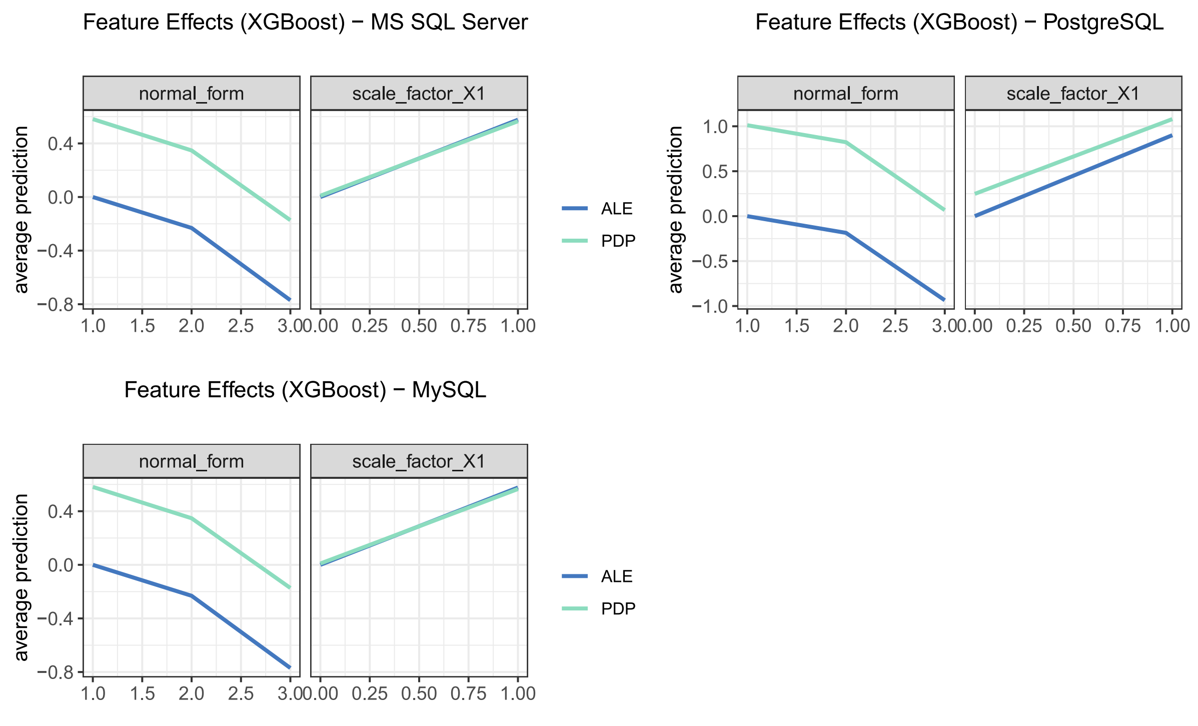


Fig. 11. Feature effects of the selected XGBoost scoring model for the two most important predictors of query duration.

with significant differences across the three tested normal forms. Additionally, both random forest and XGBoost models were shown reliable in this study, with XGBoost showing slightly better performance in predicting query execution times. Also, results were consistent among the chosen database servers.

The results suggest that higher normal forms can significantly reduce execution times. This can be applied in DSS database design towards more efficient data retrieval and processing, which is arguably important for industries relying on large-scale databases. Additionally, by reducing query execution times and increasing query completion success, operational costs related to database management can be lowered without investing more in hardware.

The effects of database normalization should not be considered in isolation, i.e., not only in regards to query completion and query execution time. As normalization is about reducing data redundancy as well as update anomalies, it generally was expected that stricter normal forms decrease the database size. In the case of the TPC-H database, for the scale factor of 1.0 GB, the size of the dataset in the 2NF schema is 75% of the 1NF database, whereas the size of the dataset in the 3NF schema is only 15% of the 2NF database. This highlights the magnitude of the effects of normalization on database size, as all three databases contain the same data.

These results have practical applications in computing education. The results arguably show the importance of teaching database normalization and its practical benefits, especially for DSS databases. Educators can design exercises that allow students to experience first-hand the performance improvements associated with higher normal forms, reinforcing theoretical concepts with practical application.

This study provides a relatively transparently-reported performance comparison with a purposefully limited scope for evaluating the impact of database normalization on query performance and query completion success. This can serve as a benchmark for future research in the field. The machine learning models suggested that the number of join paths and normalization level were the most important predictors for query execution times and successful query completion. Researchers can build on these findings to explore other factors influencing database efficiency in different business domains, with benchmarks other than TPC-H, and with different database architectures and configurations, using the provided methodology and findings as a baseline.

Previous studies have generally supported the notion that higher normal forms improve data integrity and reduce redundancy. However, the specific impact on query performance has been less frequently quantified. This study provides concrete metrics (e.g., query completion rates, execution times) that extend earlier findings. Furthermore, the use of machine learning models to predict query performance is, to the best of our knowledge, a relatively novel approach. The incorporation of random forest and XGBoost models offers a new perspective and demonstrates the applicability of these methods in database performance research.

5.2. Limitations and threats to validity

There are several limitations to our study. Except for the primary keys and foreign keys indexes, none of the databases were optimized in terms of physical or logical database structures, nor were SQL Server, MySQL, and PostgreSQL, or TPC-H, configured besides their default configurations. While none of these choices arguably reflect real-world DSS scenarios, many decision support system queries (as opposed to transactional systems) do not rely on physical or logical structure optimizations, as they are often *ad hoc* queries. However, the aim of the study was not DBMS-DBMS performance comparison.

The timeout of 30 min (i.e., 1800 s) used in this study is arbitrary. Arguably, data analysis reports which take longer than 30 min are acceptable. However, we expect that the chosen scale factors mitigate this, as the datasets used in this study are relatively small when compared to enterprise data warehouses.

One limitation in scope is that this study focuses on the business domain of warehouses, uses merely data retrieval statements, and merely one, although relatively large, set of queries. Therefore, it is unclear how extensively the results of this study generalize to other domains or queries. Additionally, without similar prior studies, it is challenging to compare our results to what others have observed, which in turn calls for similar future studies with different settings to understand the effects of normalization in wider contexts.

6. Conclusion

Decision support systems are important tools in modern business environments, providing support for decision-making processes across various industries. These systems integrate data with sophisticated analytical models to facilitate informed and timely decisions. The significance of DSS lies in their ability to enhance the quality and speed of decision-making. As decision support systems often handle large amounts of data, generating insights from this data through querying is understandably computationally slow. Despite the importance of decision support systems in general, as well as the need for timely insights, scientific research concerning the DSS performance has been relatively scarce. In this study, we analyzed how logical database design through normalization affects long-running queries completing successfully and query execution time. Rather unintuitively, higher normal forms showed both better query completion success as well as query execution times, explained by several predictors in the queries such as the paths used to join tables, database size, and which predicates the queries contained. These results have practical applications in logical design of DSS databases, as the results imply that not only higher normal forms eliminate data redundancy, they also speed up queries and facilitate higher query completion success in long-running queries.

CRediT authorship contribution statement

Marin Fotache: Writing – review & editing, Writing – original draft, Supervision, Project administration, Methodology, Formal analysis, Conceptualization. **Marius-Iulian Cluci:** Writing – review & editing, Software, Methodology, Formal analysis, Data curation, Conceptualization. **Toni Taipalus:** Writing – review & editing. **George Talaba:** Writing – review & editing, Software, Formal analysis, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This research was supported by the Digital and Sustainability Transitions in Society (DigiSus) research platform.

Data availability

Manuscript contains a link to the data.

References

[1] S. Liu, A.H. Duffy, R.I. Whitfield, I.M. Boyle, Integration of decision support systems to improve decision support performance, Knowl. Inf. Syst. 22 (2010) 261–286.
[2] W. Kent, Choices in practical data design, in: Proceedings of the 8th International Conference on Very Large Data Bases, VLDB'82, 1982, pp. 165–180.
[3] W. Kent, A simple guide to five normal forms in relational database theory, Commun. ACM 26 (2) (1983) 120–125, http://dx.doi.org/10.1145/358024.358054.
[4] E.F. Codd, A relational model of data for large shared data banks, Commun. ACM 13 (6) (1970) 377–387, http://dx.doi.org/10.1145/362384.362685.
[5] E.F. Codd, Further normalization of the data base relational model, Data Base Syst. 6 (1972) 33–64.
[6] H. Lee, Justifying database normalization: a cost/benefit model, Inf. Process. Manag. 31 (1) (1995) 59–67, http://dx.doi.org/10.1016/0306-4573(94)E0011-P.
[7] M. Fotache, Why normalization failed to become the ultimate guide for database designers? 2006, http://dx.doi.org/10.2139/ssrn.905060, SSRN, Available at SSRN: https://ssrn.com/abstract=905060.
[8] H.C. Smith, Database design: composing fully normalized tables from a rigorous dependency diagram, Commun. ACM 28 (8) (1985) 826–838, http://dx.doi.org/10.1145/4021.4024.
[9] S. Kolahi, L. Libkin, An information-theoretic analysis of worst-case redundancy in database design, ACM Trans. Database Syst. 35 (1) (2008) http://dx.doi.org/10.1145/1670243.1670248.
[10] C. Zaniolo, A new normal form for the design of relational database schemata, ACM Trans. Database Syst. 7 (3) (1982) 489–499, http://dx.doi.org/10.1145/319732.319749.
[11] H. Darwen, C.J. Date, R. Fagin, A normal form for preventing redundant tuples in relational databases, in: Proceedings of the 15th International Conference on Database Theory, ICDT '12, Association for Computing Machinery, New York, NY, USA, 2012, pp. 114–126, http://dx.doi.org/10.1145/2274576.2274589.
[12] K. England, G. Powell, Logical database design for performance, in: Microsoft SQL Server 2005 Performance Optimization and Tuning Handbook, Digital Press, 2007, pp. 19–64, http://dx.doi.org/10.1016/B978-155558319-4/50003-5.
[13] G.L. Sanders, S. Shin, Denormalization effects on performance of RDBMS, in: Proceedings of the 34th Annual Hawaii International Conference on System Sciences, 2001, pp. 1–9, http://dx.doi.org/10.1109/HICSS.2001.926306.
[14] S.K. Shin, G.L. Sanders, Denormalization strategies for data retrieval from data warehouses, Decis. Support Syst. 42 (1) (2006) 267–282.
[15] T. Taipalus, Database management system performance comparisons: A systematic literature review, J. Syst. Softw. 208 (111872) (2024) http://dx.doi.org/10.1016/j.jss.2023.111872.
[16] D.B. Bock, J.F. Schrage, Denormalization guidelines for base and transaction tables, SIGCSE Bull. 34 (4) (2002) 129–133, http://dx.doi.org/10.1145/820127.820184.
[17] C. Boscarioli, L. Torres, G.R. Krüger, M.S. Oyamada, Evaluating the impact of data modeling on OLAP applications using relacional and columnar DBMS, in: 2018 XLIV Latin American Computer Conference, CLEI, 2018, pp. 464–471, http://dx.doi.org/10.1109/CLEI.2018.00062.
[18] M. Poess, C. Floyd, New TPC benchmarks for decision support and web commerce, SIGMOD Rec. 29 (4) (2000) 64–71, http://dx.doi.org/10.1145/369275.369291.
[19] Transaction Processing Performance Council, TPC benchmark H (decision support) standard specification revision 3.0.1, 2022, https://www.tpc.org/TPC_Documents_Current_Versions/pdf/TPC-H_v3.0.1.pdf.
[20] M. Dreseler, M. Boissier, T. Rabl, M. Uflacker, Quantifying TPC-H choke points and their optimizations, Proc. the VLDB Endow. 13 (8) (2020) 1206–1220, http://dx.doi.org/10.14778/3389133.3389138.
[21] T.P.P. Council, TPC benchmark c standard specification revision 5.2, 2006, http://www.tpc.org/tpcc/spec/tpcc_current.pdf.
[22] Y. Guo, Z. Pan, J. Heflin, LUBM: A benchmark for OWL knowledge base systems, J. Web Semant. 3 (2) (2005) 158–182, http://dx.doi.org/10.1016/j.websem.2005.06.005, URL https://www.sciencedirect.com/science/article/pii/S1570826805000132.
[23] B.F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, R. Sears, Benchmarking cloud serving systems with YCSB, in: Proceedings of the 1st ACM Symposium on Cloud Computing, SoCC '10, Association for Computing Machinery, New York, NY, USA, 2010, pp. 143–154, http://dx.doi.org/10.1145/1807128.1807152.
[24] A. van Renen, V. Leis, Cloud analytics benchmark, Proc. VLDB Endow. 16 (6) (2023) 1413–1425, http://dx.doi.org/10.14778/3583140.3583156.
[25] B. Kim, K. Koo, U. Enkhbat, S. Kim, J. Kim, B. Moon, M2Bench: a database benchmark for multi-model analytic workloads, Proc. the VLDB Endow. 16 (4) (2022) 747–759.
[26] ISO/IEC, ISO/IEC 9075-1:2016, "SQL - Part 1: Framework", 2016, URL https://www.iso.org/standard/63555.html.
[27] ISO/IEC, ISO/IEC 9075-2:2016, "SQL - Part 2: Foundation", 2016, URL https://www.iso.org/standard/63556.html.
[28] A. Bond, D. Johnson, G. Kopczynski, H.R. Taheri, Architecture and performance characteristics of a PostgreSQL implementation of the TPC-E and TPC-V workloads, in: Revised Selected Papers of the 5th TPC Technology Conference on Performance Characterization and Benchmarking - Volume 8391, Springer-Verlag, Berlin, Heidelberg, 2013, pp. 77–92, http://dx.doi.org/10.1007/978-3-319-04936-6_6.
[29] J. Kloke, J.W. McKean, Nonparametric Statistical Methods Using R, CRC Press, Boca Raton, Florida, USA, 2015.
[30] M. Aslam, Cochran's Q test for analyzing categorical data under uncertainty, J. Big Data 10 (147) (2023) http://dx.doi.org/10.1186/s40537-023-00823-3.
[31] L. Breiman, J. Friedman, R. Olshen, C. Stone, Classification and Regression Trees (1st ed.), Hall/CRC, Wadsworth, New York, 1984, http://dx.doi.org/10.1201/9781315139470.
[32] W. Loh, Fifty years of classification and regression trees, Int. Stat. Rev. 82 (3) (2014) 329–348, http://dx.doi.org/10.1111/insr.12016.
[33] L. Breiman, Random forests, Mach. Learn. 45 (1) (2001) 5–32, http://dx.doi.org/10.1023/A:1010933404324.
[34] J. Friedman, T. Hastie, R. Tibshirani, Additive logistic regression: a statistical view of boosting (With discussion and a rejoinder by the authors), Ann. Statist. 28 (2) (2000) 337–407, http://dx.doi.org/10.1214/aos/1016218223.

[35] T. Chen, C. Guestrin, XGBoost: A scalable tree boosting system, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16, Association for Computing Machinery, New York, NY, USA, 2016, pp. 785–794, http://dx.doi.org/10.1145/2939672.2939785.

[36] B. Candice, C. Anna, M.-M. Gonzalo, A comparative analysis of gradient boosting algorithms, Artif. Intell. Rev. 54 (3) (2021) 1937–1967, http://dx.doi.org/10.1007/s10462-020-09896-5.

[37] B. Efron, T. Hastie, Computer Age Statistical Inference: Algorithms, Evidence, and Data Science, Cambridge University Press, 2016.

[38] R. Guidotti, A. Monreale, S. Ruggieri, F. Turini, F. Giannotti, D. Pedreschi, A survey of methods for explaining black box models, ACM Comput. Surv. 51 (5) (2018) http://dx.doi.org/10.1145/3236009.

[39] M. Du, N. Liu, X. Hu, Techniques for interpretable machine learning, Commun. ACM 63 (1) (2019) 68–77, http://dx.doi.org/10.1145/3359786.

[40] P. Biecek, T. Burzykowski, Explanatory Model Analysis: Explore, Explain, and Examine Predictive Models (1st ed.), Hall/CRC, Chapman and Hall/CRC, 2021, http://dx.doi.org/10.1201/9780429027192.

[41] R. Dwivedi, D. Dave, H. Naik, S. Singhal, R. Omer, P. Patel, B. Qian, Z. Wen, T. Shah, G. Morgan, R. Ranjan, Explainable AI (XAI): Core ideas, techniques, and solutions, ACM Comput. Surv. 55 (9) (2023) http://dx.doi.org/10.1145/3561048.

[42] P. Probst, A.-L. Boulesteix, B. Bischl, Tunability: Importance of hyperparameters of machine learning algorithms, J. Mach. Learn. Res. 20 (53) (2019) 1–32, URL http://jmlr.org/papers/v20/18-444.html.

[43] M. Kuhn, K. Johnson, Feature Engineering and Selection: a Practical Approach for Predictive Models, CRC Press, Boca Raton, Florida, USA, 2019.

[44] M. Kuhn, K. Johnson, Applied Predictive Modeling, Springer, New York, USA, 2013.

[45] R Core Team, R: A language and environment for statistical computing, 2024, https://www.R-project.org/.

[46] H. Wickham, M. Averick, J. Bryan, W. Chang, L.D. McGowan, R. François, G. Grolemund, A. Hayes, L. Henry, J. Hester, M. Kuhn, T.L. Pedersen, E. Miller, S.M. Bache, K. Müller, J. Ooms, D. Robinson, D.P. Seidel, V. Spinu, K. Takahashi, D. Vaughan, C. Wilke, K. Woo, H. Yutani, Welcome to the tidyverse, J. Open Sour. Softw. 4 (43) (2019) 1686, http://dx.doi.org/10.21105/joss.01686.

[47] T.L. Pedersen, Ggforce: Accelerating 'ggplot2', 2024, R package version 0.4.2, https://CRAN.R-project.org/package=ggforce.

[48] T. Wei, V. Simko, R package 'corrplot': Visualization of a correlation matrix, 2021, URL https://github.com/taiyun/corrplot, (Version 0.92).

[49] A. Kassambara, Rstatix: Pipe-friendly framework for basic statistical tests, 2023, R package version 0.7.2, https://CRAN.R-project.org/package=rstatix.

[50] I. Patil, Visualizations with statistical details: The 'ggstatsplot' approach, J. Open Sour. Softw. 6 (61) (2021) 3167, http://dx.doi.org/10.21105/joss.03167.

[51] M. Kuhn, J. Silge, Tidy Modeling with R, O'Reilly, Sebastopol, California, USA, 2022.

[52] M.N. Wright, A. Ziegler, Ranger: A fast implementation of random forests for high dimensional data in C++ and R, J. Stat. Softw. 77 (1) (2017) 1–17, http://dx.doi.org/10.18637/jss.v077.i01.

[53] T. Chen, T. He, M. Benesty, V. Khotilovich, Y. Tang, H. Cho, K. Chen, R. Mitchell, I. Cano, T. Zhou, M. Li, J. Xie, M. Lin, Y. Geng, Y. Li, J. Yuan, Xgboost: Extreme gradient boosting, 2024, URL https://CRAN.R-project.org/package=xgboost, R package version 1.7.7.1.

[54] P. Biecek, DALEX: Explainers for complex predictive models in R, J. Mach. Learn. Res. 19 (84) (2018) 1–5, URL http://jmlr.org/papers/v19/18-416.html.

[55] P. Biecek, H. Baniecki, Ingredients: Effects and importances of model ingredients, 2023, URL https://CRAN.R-project.org/package=ingredients, R package version 2.3.0.