Managing Technical Debt in Database Normalization

Mashel Albarak, Rami Bahsoon, Ipek Ozkaya, and Robert Nord

Abstract— Database normalization is one of the main principles for designing relational databases, which is the most popular database model, with the objective of improving data and system qualities, such as performance. Refactoring the database for normalization can be costly, if the benefits of the exercise are not justified. Developers often ignore the normalization process due to the time and expertise it requires, introducing technical debt into the system. Technical debt is a metaphor that describes trade-offs between short-term goals and applying optimal design and development practices. We consider database normalization debts are likely to be incurred for tables below the fourth normal form. To manage the debt, we propose a multi-attribute analysis framework that makes a novel use of the Portfolio Theory and the TOPSIS method (Technique for Order of Preference by Similarity to Ideal Solution) to rank the candidate tables for normalization to the fourth normal form. The ranking is based on the tables estimated impact on data quality, performance, maintainability, and cost. The techniques are evaluated using an industrial case study of a database-backed web application for human resource management. The results show that the debt-aware approach can provide an informed justification for the inclusion of critical tables to be normalized, while reducing the effort and cost of normalization.

Index Terms—Database Normalization, Multi-attribute analysis, Software Design, Technical Debt

1 Introduction

Information systems refactoring is integral to their development and evolution to respond to data growth, improving quality, and changes in users' requirements. The refactoring process faces many risks, such as budget overruns, schedule delays and increasing chances of failure. Databases are the core of information systems; evolving databases' schemas through refactoring is common practice for improving quality, meeting new requirements, among other structural and behavioral qualities.

Database normalization is one of the main principles for Relational Database Model design, invented by the Turing Award winner Ted Codd [1]. Normalization concept was developed to organize data in "relations" or tables following specific rules to minimize data redundancy, and consequently, improve data consistency by reducing anomalies. Normalization benefits go beyond data quality, and can further improve maintainability and performance [2], [3].

Despite the advances in database models and the emergent of new models where normalization is irrelevant (such as NoSQL), a recent study by DB-Engines Ranking [4], has shown that the Relational Database Model is the dominant model, where 139 of Database Management Systems (DBMS) leverage this model among 350 different DBMS. The study has evidenced that the relational model is still the most popular one , with a 75.2 % popularity score, and used by the top 4 DBMS, namely: Oracle, MySQL, MS SQL Server and PostgreSQL. The model is also used by other major DBMS such as IBM Db2 and SQLight.

Conventional approaches often suggest higher level of normalization for database tables to achieve structural and behavioral benefits in the design. However, practically, developers tend to overlook normalization process due to time and expertise it requires [5]. With the growth of data, normalizing the database becomes essential to treat deteriorations in data quality, performance and overall database design [2]. Conversely, database normalization and refactoring can be costly, if the benefits are not justified. For example, consider a database that consists of a large number of weakly normalized tables, developers may decide on normalizing the design to improve its quality and performance. Normalization may also require refactoring the applications using the database and other artifacts. Though some tables can be candidate for normalization, they might not have severe impact on quality to justify the cost of investing into normalization.

To address this issue, technical debt metaphor can be used as a tool to justify the value of database normalization, through capturing the likely value of normalization (e.g. quality improvements) relative to the cost and effort of embarking on this exercise. Technical debt is a metaphor coined to describe trade-offs between short term goals (e.g. Fast system release; savings in Person Months) and applying optimal design and development practices [6]. Technical debt has been studied extensively over the past years (see MTD workshop series and TechDebt Conference [7], EuroMicro track on technical debt [8], Dagstuhl seminar [9]). Despite the increasing volume of research on technical debt, there is no general agreement on technical debt definition. However, the common ground of existing definitions is that the debt can be attributed to poor and sub-optimal engineering decisions that may carry immediate benefits, but are not well geared for long-term benefits. Database

- *Mashel Albarak is with the School of Computer Science, University of Birmingham, UK and King Saud University, KSA. E-mail: mxa657@cs.bham.ac.uk*
- *Rami Bahsoon is with the School of Computer Science, University of Birmingham, UK. E-mail: r.bahsoon@cs.bham.ac.uk*
- *Ipek Ozkaya is with the Software Engineering Institute, Carnegie Mellon University, USA. E-mail: ozkaya@sei.cmu.edu*
- *Robert Nord is with the Software Engineering Institute, Carnegie Mellon University, USA. E-mail: rn@sei.cmu.edu*

normalization is among the core engineering decisions that are performed to meet both structural and behavioral requirements, such as performance, general data qualities and maintainability among the others. Sub-optimality, taking the form of weaker normal forms, can have direct negative consequences on meeting these requirements. Database normalization is essentially a design decision that can incur a technical debt, where the debt interest can be manifested into degradation in qualities and increased level of data inconsistency and duplication over the lifetime of the software system. If not managed, the consequence can resemble an accumulated interest on the debt that can grow with the growth of the data over time.

The majority of technical debt research have focused on code and architectural level debts [6], [10]. Technical debt linked to databases design was first attempted in the context of missing foreign keys in a database [11]. In [12] , [13] we were the first to explore a new context of technical debt which relates to database normalization issues. We have considered that potential normalization debts are likely to be incurred for tables below the fourth normal form, since fifth normal form is regarded as a theoretical form [14].

The underlying assumption of the normalization theory is that the database tables should be normalized to a hypothetical fifth normal form, to achieve benefits [14]. While this assumption holds in theory, practically it fails due to the required time and expertise. To address this issue, in [12], we proposed a prioritization method of the tables that should be normalized to the fourth normal form based on their likely impact on data quality and operations performance. For, operations performance, table's growth rate was considered as a critical factor to address performance impact accumulation in the future. We utilized the portfolio theory [15] to rank the tables based on their impact on performance under the risk of tables' growth.

The contribution of this research is a multi-attribute decision analysis approach for managing technical debt related to databases normalization. our proposed approach provides databases designers and decision makers with a systematic framework to identify the debt items, quantify likely impact of normalization debts, and to provide mitigation strategies to manage the debts through justified and more informed normalization that consider cost, qualities and debts. In this study, the proposed framework focuses on the prioritization aspect of managing technical debt, to rank the tables that are most affected by the weakly normalized design.

This study goes beyond our previous work [12] in the following:

- In addition to data quality and performance, we considered potential debt tables' impact on maintainability. We used tables' number of attributes as a measure of the tables' complexity [16].
- We extend the application of the portfolio analysis technique to cover data quality and maintainability, considering the diversification and mitigation of tables' growth rate risk across three quality dimensions.
- The cost of normalization is an important factor that we incorporated in the decision analysis process.
- Consequently, managing the debt should consider how the debt can relate to data quality, performance, maintainability, and cost of normalization not in isolation but also in conjunction. Therefore, we view normalization debt management as a multi-attribute decision problem, where developers need to prioritize tables that should be normalized based on their impact on the three considered qualities, in addition to the cost of normalization. The framework makes a novel use of the TOPSIS method (Technique for Order of Preference by Similarity to Ideal Solution) [17], to rank the debt tables that should be normalized to the fourth normal form.

The techniques are evaluated using an industrial case study of a database-backed web application for human resource management in a large company. The database consists of 97 tables of industrial scale, each filled with large amount of data. The results show that the debt-aware approach has provided an informed justification for the inclusion of critical tables to be normalized. Equally important, it reduced the effort and cost of normalizing by eliminating unnecessary normalization tasks. Our framework has the promise to replace ad-hoc and un-informed practices for normalizing databases, where debt and its impact can motivate better design, optimize for resources and justify database normalization decisions.

2 Background and Motivation

In this section, the key concept of our work is summarized. Normalization theory was introduced by Codd in 1970 [1], as a process of organizing the data in tables. The main goal of normalization is to minimize data redundancy, which is accomplished by splitting a table into several tables after analyzing the dependencies between the table's attributes. Advantages of normalization was discussed and proved in the literature [2], [14]. Examples of such advantages include: data quality improvement as it reduces redundancy; reducing update anomalies and facilitating maintenance. Fig 1 illustrates the normal forms hierarchy. Higher level of normal form indicates a better design [14], since higher levels eliminate more redundant data. The main condition to go higher in the hierarchy is

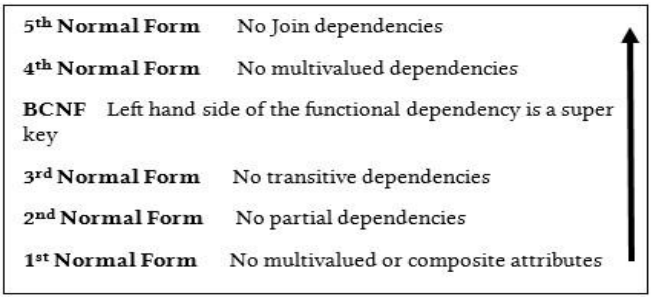

Fig 1 Database Normalization Hierarchy

based on the constraint between two sets of attributes in a table, which is referred to as dependency relationship.

2.1 Benefits of Database Normalization: Data Quality, Maintainability and Performance

Data quality enhancement is one of the main benefits of database normalization [2], [14]. The enhancement is linked to decreasing the amount of data redundancy as we move higher in the normalization hierarchy. By redundancy we mean, recording the same fact more than once in the same table. In poorly or un-normalized tables that suffer from data redundancy, there is always a possibility of updating only some occurrences of the data, which will affect the data consistency. Data quality is a crucial requirement in all information systems, as the success of any system relies on the reliability of the data retrieved from the system.

In addition to data quality, improving maintainability is another important benefit from database normalization [3], [2]. Weakly or un-normalized tables' design involve bigger number of attributes in each table compared to highly normalized tables, which will increase complexity in terms of retrieving data from those tables and implementing new rules on the tables [3]. Moreover, reducing the size of the table by normalization will facilitate the back-up and restore process.

Benefits of normalization can also be observed on operations performance. Performance has always been a controversial subject when it comes to normalizing databases. Some may argue that normalizing the database involves decomposing a single table into more tables, henceforth; data retrieval can be less efficient since it requires joining more tables as opposed to retrieving the data from a single table. Indeed, de-normalization was discussed as the process of "down-grading" table's design to lower normal forms and have limited number of big tables to avoid joining tables when retrieving the required data [2]. Advocates of de-normalization argue that the Database Management System (DBMS) stores each table physically to a file that maintains the records contiguously, and therefore retrieving data from more than one table will require a lot of I/O. However, this argument might not be correct: even though there will be more tables after normalization, joining the tables will be faster and more efficient because the sets will be smaller and the queries will be less complicated compared to the de-normalized design [2]. Moreover, weakly or un-normalized tables will be stored in a large number of files as opposed to the normalized design due to big amount of data redundancy, and consequently, increased records size and increased I/O cost [2], [18]. Adding to this, not all DBMS store each table in a dedicated physical file. Therefore, several tables might be stored in a single file due to reduced table size after normalization, which means less I/O and improved performance, as shown in [2], [18]

Despite the controversy about normalization and performance, several arguments in favor of normalization is presented by C. J. Date [2], an expert who was involved with Codd in the relational database theory:

- Some of the de-normalization strategies to improve performance proposed in the literature are not "de-normalizing" since they do not increase data redundancy. In fact, as Date states, some of them are considered to be good normalized relational databases.
- There is no theoretical evidence that de-normalizing tables will improve performance. Therefore, it's application dependent and may work for some applications. Nevertheless, this does not imply that a highly normalized database will not perform better.

In this study, we view these deteriorations in data quality, maintainability and performance as debt impact caused by inadequate normalization of database tables. The impact can accumulate overtime with data growth and increased data duplication. Short-term savings from not normalizing the table can have long-term consequences, which may call for inevitable expensive maintenance, fixes and/or replacement of the database. Therefore, we motivate rethinking database normalization from the debt perspective linked to data quality, maintainability and performance issues.

2.2 Cost of Database Normalization

Benefits of normalization comes with certain costs. Normalizing the database is a relatively complex process due to expertise and resources required. Decomposing a single table for normalization may involve [5]:

- Database schema alterations: create the new normalized tables; Ensure that the data in new tables will be synchronized and finally updating all stored views, procedures and functions that access the original table before normalization to reflect the changes.
- Data Migration: a detailed and secured strategy should be planned to migrate the data from the old weakly or un-normalized table to the new decomposed tables.
- Modifications to the accessed application/s: introduce the new tables' meta-data and update the applications' methods and classes' source code that accessed the original table. Occasionally, normalization may not require refactoring of the application, if the application is well developed and decoupled from the database. Nevertheless, normalization is considered to be complex since it would require other complex tasks, as mentioned in the above points.

Moreover, testing the database and the applications before deployment can be expensive and time consuming. Therefore, in this study, we formulate the normalization problem as technical debt. We start from the intuitive assumptions that tables, which are weakly/not normalized to the deemed ideal form, can potentially carry debt. To manage the debt, we adhere to the logical practice in paying the debt with the highest negative impact on quality, with respect to the cost of normalization.

2.3 Why Multi-attribute Analysis

Multi-attribute analysis techniques help decision makers evaluate alternatives when conflicting objectives must be considered and balanced, and when outcomes are uncertain [17]. The process provides a convenient framework

for developing a quantitative debt impact assessment that results in a set of prioritized tables to be normalized. The prioritization is based on the debt impact on data quality, maintainability, performance and the effort cost of normalizing the table. As this analysis involves multiple conflicting attributes, multi-attribute analysis will help structure the problem and provide a systematic and informed guidance to normalize tables to improve quality cost effectively. In this study, we utilize the TOPSIS method [17] to rank the tables that should be normalized. This method involved several steps to find the best alternative that is closest to the ideal solution. Detailed explanation of the method is demonstrated in section 4.3.

2.4 Database refactoring and Schema Evolution

Database refactoring is defined as "simple change to a database schema that improves its design while retaining both its behavioral and informational semantics" [5]. While code refactoring area is well researched in the literature [19], Database refactoring received less attention [20]. As mentioned by Vial in [20], only 16 studies on "database refactoring" were found in ACM and IEEE digital libraries in 2015, Including reviews of "Refactoring Databases: Evolutionary Database Design" [5]. Vial reports lessons learned from refactoring a database for industrial logistics applications [20]. One of the key lessons learned is that some refactoring patterns might not yield the benefits they envisioned, specifically in the case of reducing the data duplication in the database. Therefore, he stated that the database might encounter some level of debt (in his case the debt resembles in data duplication) as long as the debt is known and documented. Additional studies between 2015 and 2019 on the topic are in line with Vial's conclusion [21], [22].

Schema evolution on the other hand, has been extensively studied over the past years [23], [24], [5]. Schema evolution is modifying the database schema to evolve the software system in meeting new requirements [5]. Researchers have attempted to analyze the schema evolution from earlier versions [23], and they developed tools to automate the evolution analysis process [24]. However, schema evolution literature has focused on limited tactics, such as adding or renaming a column, changing the data type, etc. Evolving the database schema for the objective of normalizing the database to create value and avoid technical debt has not been explored. We posit that normalization is a process that should create a value to ensure the system's sustainability and maintainability in the future.

3 Normalization Debt Definition

The common ground of existing definitions of technical debt is that the debt can be caused by poor and sub-optimal development decisions that may carry immediate benefits, but are not well geared for long-term benefits [10], [6]. Database normalization had been proved to minimize data duplication, and several studies have shown that it also improved performance as the table is further normalized to higher normal forms (see Fig 1) [18], [2]. Therefore, tables in the database that are below the fourth normal form can be subjected to debts as they potentially lag behind the optimal, where debt can be observed on data consistency, maintainability and performance degradation as the database grows [2]. To address this phenomenon of normalization and technical debt, in previous works [12], [13], we have considered fourth normal form as the target normal form for the following reasons:

- In practice, most database tables are in third (rarely achieve BCNF) normal form [14]. However, fourth normal form is considered as a better design since it is a higher level and more redundant data is eliminated [14].
- Fourth normal form criteria relies on multi-valued dependencies that are common to address in reality [25].
- While fifth normal form is higher in the normalization hierarchy, it is considered as a theoretical value, since it is based on join dependencies between attributes, which rarely arise in practice [14].

4 Proposed Framework to Prioritize Technical Debt in Database Normalization

To facilitate managing normalization debt in an explicit way, we propose a simple management framework. The framework is meant to help organize the activities and information needed to prioritize tables in the database that should be normalized to the fourth normal form. As depicted in Fig 2, the framework consists of three major phases: identification of potential debt tables (phase 1), estimating its likely impact and cost (phase 2) and using this

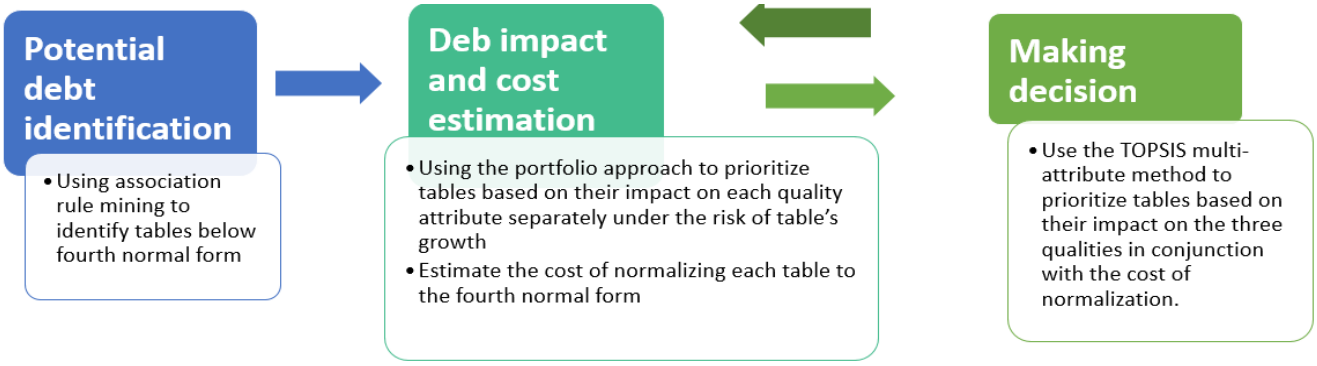

Fig 2 Normalization debt management framework

information to prioritize tables which are candidate for normalization (phase 3). As shown in the figure, phases 2 and 3 are performed periodically. After the decision has been made to normalize a specific table, the rest of the identified debt tables should be monitored as their impact on quality and the cost to normalize them may change as time passes.

4.1 Phase1: Potential Debt Identification

Given the previous definition of normalization potential debt tables, our objective is to identify the tables that are below the fourth normal form by determining the current normal form of each table in the database. Determining the normal form can be done through knowledge of the functional dependencies that holds in a table. A functional dependency is a constraint that determines the relationship between two sets of attributes in the same table [2]. Example of a functional dependency in a table that stores Staffs' information: *JobTitle→ Salary*. This dependency is interpreted as: all staff members having the same JobTitle, will also have the same Salary. Therefore, the Salary is functionally dependent on the JobTitle.

These dependencies are elicited from a good knowledge of the real world and the enterprise domain, which the database will serve. Experienced database developers and the availability of complete documentation can also be a source for extracting dependencies. Practically, none of these might be available, which makes dependency analysis a tough task. In previous work [13], we proposed a framework that involves several steps to determine the current normal for each table by mining the data stored in each table. The framework uses a data mining technique called association rule mining [26] to analyze the data and test candidate dependencies in each table. Detailed explanation of the framework is found in [13].

4.2 Phase 2: Debt Impact and Cost Estimation

4.2.1 Normalization Debt Impact

The second phase of the proposed framework involves estimating the impact of each potential debt table on data quality, maintainability and performance, where the estimation will be the one of the drivers of prioritizing debts to be paid in addition to the normalization cost.

Data Quality: In this study, the quality of the data is represented by its consistency. In weakly or un-normalized tables that store large amount of data redundancy, there is always a possibility of changing or updating some occurrences of the data leaving the same redundant data un-changed. Therefore, the risk of data inconsistency will increase. We view the risk of data inconsistency as an impact incurred by weakly normalized tables. This Impact can be quantified using the International Standardization Organization (ISO) metric [27], risk of data inconsistency. According to ISO, The risk of data inconsistency is proportional to the number of duplicate values in the table and it is reduced if the table further normalized to higher normal forms. This risk can be measured using the following formula:

$$X=A/B \quad (1)$$

With $\binom{n}{k}$ sets of k attributes for a table with n attributes (k=1,..n),

$A=\sum k\sum j \sum i D_{ijk}$, D_{ijk} = number of duplicate values found in set i of k attributes of table j.

$B= \sum j\ m_j*n_j/T$ (j=1..T) , T= number of tables, m_j= number of rows of table j, n_j= number of columns of table j.

For X, lower is better.

Maintainability: Maintainability refers to the ease of which a software product can be altered after delivery in order to correct defects, add or improve features or cope with future changes in the environment [28]. In databases, the number of attributes is proposed as a metric to measure the complexity of each table [29], [16]. Weakly or un-normalized tables consist of a big number of attributes. Too many attributes indicate a lack of cohesion and storing data for multiple entities in one table [5]. In [30] the authors showed that tables with big number of attributes are more vulnerable to changes as the system evolves. They also argued that a good design may involve tables with smaller number of attributes, which is accomplished by normalizing those tables to higher normal forms through decomposition. In this study, the complexity of each potential debt table is measured by the number of its attributes. This complexity is decreased by normalizing the table to fourth normal form as it will involve decomposing the table to two or more tables and reduce the number of attributes.

Performance: Operations performed on the database include update; insert; delete and data retrieval operations. Each of these operations incur a cost on the number of disk pages read or written, which is referred to as the Input/Output "I/O" cost [31]. The impact of normalization debt can be observed through the I/O cost incurred by the operations performed on the potential debt table in its current normal form. Due to the huge amount of data duplication in tables below fourth normal form, tables will require more pages to be stored in, which will affect performance of the operations executed on those tables. Meaning, the more data stored in the table, the more disk pages it requires and henceforth, more time to go through the pages to execute the operation. Therefore, normalization is the sensible solution that will improve performance, as shown in [2], [18].

Unlike data quality and maintainability, a metric to quantify the I/O cost of all the operations performed on debt tables is not available. In previous work [12], we proposed the following model to estimate the average I/O costs for operations executed on each debt table:

$$\text{I\O cost} = \sum_{x=1}^{n_u} C_x^u \lambda_x^u + \sum_{x=1}^{n_i} C_x^i \lambda_x^i + \sum_{x=1}^{n_d} C_x^d \lambda_x^d + \sum_{x=1}^{n_s} C_x^s \lambda_x^s \quad (2)$$

Where nu, ni , nd , ns are the number of update, insert, delete and select operations on the potential debt table R respectively. λ_x^u , λ_x^i , λ_x^d , λ_x^s represent the execution rates of the xth update, xth insert, xth delete and xth select respectively. Finally, the I\O costs of the xth update, xth insert, xth delete and xth select are represented by C_x^u, C_x^i, C_x^d , C_x^s respectively.

Debt Impact Accumulation and Tables' Growth:

In technical debt area, the debt interest is a crucial factor for managing debts. In general, the interest of technical

debt is the cost paid overtime by not resolving the debt [10]. Researchers have coined interest with implication on qualities [32]. The analogy is applicable to the case of normalization debt as striving for the ideal fourth normal form will reduce impact on data quality, maintainability and performance. We view that all tables do incur interest overtime. However, interest varies between tables in how much and how fast they accumulate interest, which our work uses to identify tables that are candidate for normalization. Tables' growth rate is a crucial factor that will cause the impact of the debt on the three qualities to accumulate faster. For performance, I/O cost of the operations changes based on the tables' growth rate. If the table is likely to grow faster than other tables, the I/O cost for the operations executed on that table will accumulate faster than others. This due to the fact increasing table size implies more disk pages to store the table and therefore, more I/O cost. Similarly, risk of data inconsistency increases as the table is growing. Regarding maintainability, Existing metric has looked at the number of attributes of each table as a measure for complexity. However, discussing normalization debts can make this measure limited as the debt can intuitively linked to not only number of attributes but also the growth rate and the population size of the table.

Therefore, tables' growth rate is a crucial measure to prioritize tables needed to be normalized. If the table is not likely to grow or its growing rate is less than other tables, a strategic decision would be to keep the debt and defer its payment. Table growth rate can be elicited from the database monitoring system. The growth rate of a table can be viewed as analogous to interest risk or interest probability. Interest probability captures the uncertainty of interest growth in the future [10]. Debt tables which experience high growth rate in data can be deemed to have higher interest rate. Consequently, these tables are likely to accumulate interest faster.

4.2.2 Prioritizing Debt Tables Using Portfolio Theory:

We use portfolio theory to prioritize tables that need to be normalized considering their impact on the three qualities, taking into consideration the likely growth rate of the table size and henceforth, the risk of interest accumulation.

Modern Portfolio Theory (MPT) was developed by the Nobel Prize winner Markowits [15]. The aim of this theory is to develop a systematic procedure to support decision making process of selecting capital of a portfolio consisting of various investment assets. The assets may include stocks, bonds, real estate, and other financial products on the market that can produce a return through investment. The objective of the portfolio theory is to select the combination of assets using a formal mathematical procedure that can maximize the return while minimizing the risk associated with every asset. Portfolio management involves determining the types that should be invested or divested and how much should be invested in each asset. This process draws on similarity with the normalization debt management process, where developers can make decisions about prioritizing investments in normalization, based on which technical debt items should be paid, ignored, or can further wait. With the involvement of uncertainty, assets expected return and variance of the return are used to evaluate the portfolio performance.

To fit in portfolio management, each debt table below the fourth normal form is treated as an asset. For each table, we need to determine whether it is better to normalize that table to the fourth normal form (pay the debt) or keep the table in it is current normal form (defer the payment). To decide on this, we need to determine what the expected return of each debt table is. In the case of normalization debt, the expected return of the debt table resembles the estimated quality impact of the table. Tables with the lowest estimated impact are deemed to carry higher expected return. In other words, If the estimated quality impact of table A is less than estimated quality impact of table B, then table A expected return would be higher than B; B will then has a higher priority for normalization due to high quality impact. We balance the expected return with the risk. In portfolio management, this risk is represented by the variance of the return. For the debt tables, this risk is represented by the tables' growth rate. Tables with the highest growth rate are considered to be risky assets, their likely interest and so the debt will grow faster than other tables of low growth rate.

In order to apply the portfolio theory to normalization debt, few considerations need to be taken into account:

- The expected return of the debt table is equal to 1÷ quality impact.
- The risk of each table is equal to the table growth rate for each debt table. The growth rate can be elicited from the database management system by monitoring the table's growth.
- We set the correlation between the debt tables to zero for several reasons: First, the quality impact represented by I/O costs, risk of data inconsistency and number of attributes for the debt tables are independent. Meaning, the I/O cost of the operations executed on a debt table has no effect on the I/O cost of the operations executed on another debt table. The same reasoning apply for the risk of data inconsistency and the number of attributes for each table. Moreover, the growth rate for each table, which affects the quality impact accumulation, is unique and independent from each other. Lastly, each debt table design is independent from other debt tables, as the decision to keep the debt or normalize the table have no effect on the design and the data of the other debt tables.

Taking into account these considerations, we can apply the portfolio theory, where the database developer is investing in tables' normalization. The database developer needs to build a diversify portfolio of multiple debt tables. Multiple debt tables in the database represent the assets. For each asset i, it has its own risk Ri and quality impact Qi. Based on these values the developer then can prioritize tables to be normalized. The expected return of debt tables' portfolio Ep, built by prioritizing debt tables from the database of m debt tables can be calculated as in the following equation:

$$Ep = \sum_{i=1}^{m} w_i \frac{1}{Q_i} \quad (3)$$

With one constraint represented in the following equation:

$$\sum_{i=1}^{m} w_i = 1 \qquad (4)$$

Where w_i represents the resulted weight of each debt table. This weight will resemble the priority of each table for normalization as explained in the process steps.

The risk of table growth rate for debt table i is represented by Ri. The global risk of the portfolio Rp is calculated as the following:

$$R_p = \sqrt{\sum_{i=1}^{m} w_i^2 R_i^2} \qquad (5)$$

Process Steps: The following steps will be executed three times (except for the first step, which will be executed once) to prioritize tables, based on their impact on performance, data quality and maintainability, separately for each quality attribute.

1. Determine the potential debt tables' growth rate from the database monitoring system. This step will simplify the method to examine only tables with high growth rate.
2. Consider only tables of high growth rate to measure their impact on the three quality attributes, using the metrics discussed in section 4.2.1.
3. Calculate the values of the portfolio variables (expected return for each table = 1÷ quality impact, risk= table's growth rate)
4. Run the model on the available data to produce the optimal portfolio of the debt tables. The portfolio model will provide the highest weights to those tables with low quality impact and low table growth rate. Therefore, debt table that has the lowest weight implies the highest priority table that should be normalized.

4.2.3 Estimating Normalization Debt cost

The second phase of the proposed framework also involves estimating the cost of normalizing each debt table to the fourth normal form. Normalizing and splitting a single table in the database requires several alterations on the database schema, applications using the database and migrating data to new decomposed tables [5]. Some of these tasks can be automated depending on the technologies used in the project. However, regardless of the size of the system, refactoring the database for normalization can be very complex and time consuming. Therefore, to structure the process of estimating normalization cost, first, the main tasks required to normalize each table to the fourth normal form should be defined. Then decompose each main task to sub-tasks and estimate the time required for each sub-task. The following simple model can be used to estimate the total effort cost required to normalize each table:

Normalization cost for each table (in hours unit) = $\sum_{i=1}^{m} \sum_{x=1}^{n}$ (Estimated hours needed to complete the xth sub-task) (6)

Where n is the total number of sub-tasks and m is the total number of the main tasks for each table. The tasks and time required to perform the refactoring tasks can be estimated from historical effort data and domain experts. In this study we will rely on experts' estimation for the time required for each task. All the major refactoring tasks that are required to normalize the tables in our case study application were determined by the technical team. The tasks were further refined into sub-tasks to ease and concretize costing. The task and subtasks were as follows:

1. Task: Split the table and create the new decomposed tables into the 4th normal form. Sub-tasks include: write scripts to create new tables and backup table data.
2. Task: Update ORM. Sub-tasks include: create new classes, update ORM models and update application services and domain layers.
3. Task: Migrate data to the new tables. Sub-tasks include: write scripts to migrate data to new decomposed tables and run those scripts.
4. Task: Refactor the application code. Sub-tasks include: refactor web helper classes, view models and class controllers.
5. Task: Test the database and application. Sub-tasks include: run automation test, end to end testing of the application, and sanity and smoke tests.
6. Task: Integrate and apply changes in the production database. Sub-tasks include: backup database, stop the application and update the new files and performance testing.

4.3 Phase 3: Making Decisions

We view normalization debt management as a multi-attribute decision making process, where developers need to make a decision about which table(s) to normalize to the fourth normal form, to improve quality at while minimizing effort cost, and which should be kept in its current normal form because it is likely to have the least impact on quality and most expensive to carry out. The final phase of the proposed framework incorporates information obtained from the previous phase 2 to prioritize tables. The proposed approach uses the TOPSIS method [17] to rank the debt tables to be normalized. TOPSIS is a mainly a utility-based method that compares each alternative directly based on data in the evaluation matrix and the weight of each quality attribute. The fundamental idea of TOPSIS is that the best solution is the one, which has the shortest distance to the positive ideal solution and the farthest distance from the negative ideal solution. This method has several advantages such as [33]:

- The method results represent scalar values that account for the best and worst alternatives simultaneously.
- The ability to visualize the performance measures of all alternatives on attributes.
- Human choice is represented by a sound logic.

The process of TOPSIS method is carried out as the following:

Step1: Construct an evaluation matrix: The matrix consists of m tables and n criteria, with the intersection of each alternative and criteria given as X_{ij}, we therefore have a matrix $(X_{ij})_{m*n}$. The criteria in our study would be risk of data inconsistency, maintainability, performance and cost.

Step 2: Calculate the normalized matrix: the

normalized matrix is represented by $R=(r_{ij})_{m*n}$. Since the criteria are all measured in different units, this step will normalize all the values in the matrix using the following formula to make the criteria comparable:

$$r_{ij} = \frac{x_{ij}}{\sqrt{\sum_{i=i}^{m} x_{ij}^2}} \quad (7)$$

i=1,2…m , j=1,2…n

Step 3: Calculate the weighted normalized matrix

$$t_{ij} = r_{ij} \times w_j \quad (8)$$

i=1,2…m , j=1,2…n

w_j is the weight criteria j. The weight of the criteria reflects the importance of this specific criterion compared to the other criteria. The summation of all weights must not exceed 1. In this study, we assumed that all four criteria are equally important. Therefore, the weight of each criterion would be 0.25. Nevertheless, the analyst can adjust these weights based on importance and/or relevance.

Step 4: Determine the positive ideal solution A+ and the negative ideal solution A- for each criterion as the following:

$$A+ = \{<\max(t_{ij} \mid i=1,2,\ldots m) \mid j \in J+>, <\min(t_{ij} \mid i=1,2,\ldots m) \mid j \in J-> \} \equiv \{t_{bj} \mid j=1,2,\ldots..n\} \quad (9)$$

$$A- = \{<\min(t_{ij} \mid i=1,2,\ldots m) \mid j \in J+>, <\max(t_{ij} \mid i=1,2,\ldots m) \mid j \in J-> \} \equiv \{t_{wj} \mid j=1,2,\ldots..n\} \quad (10)$$

Where

J+= {1,2,….,n | j} associated with the criterion having a positive impact.

J-= {1,2,….,n | j} associated with the criterion having a negative impact.

In this study, The positive ideal solution for each criterion would be the minimum value for that specific criterion, and the negative ideal solution is the maximum value. As explained in section 4.2.2, table with lowest weight indicates the table of the highest impact on performance, data quality and maintainability and therefore, has the highest priority for normalization. Similarly, the positive ideal solution in the cost criterion would be the lowest cost.

Step 5: Calculate the distance between each table and the best condition d_{ib} , and the distance between each table and the worst condition d_{iw} as the following:

$$d_{ib} = \sqrt{\sum_{j=1}^{n} (t_{ij} - t_{bj})^2} \qquad i=1,2,3...m \quad (11)$$

$$d_{iw} = \sqrt{\sum_{j=1}^{n} (t_{ij} - t_{wj})^2} \qquad i=1,2,3\ldots m \quad (12)$$

Step 6: Calculate the relative closeness to the ideal solution C_i as the following:

$$C_i = \frac{d_{iw}}{d_{ib} + d_{iw}} \quad (13)$$

Where $0 \leq C_i \leq 1$

Step 7: Rank the preference order: The best table will be the table with C_i closest to 1.

5 Case Study

In order to control the results of the evaluation effort, we conducted the case study following the DESMET methodology [34], for guiding the evaluation of software engineering methods. DESMET sees that the first decision to make when undertaking a case study is to determine what the study aims to investigate and verify, in other words, to define the goals of the case study. In the context of evaluating the normalization debt prioritization framework, our goal is to demonstrate the validity of the following hypothesis:

Database normalization debt prioritization framework provides a systematic method to improve the quality of decisions made when refactoring relational databases for the purpose of normalization.

To test the validity of the stated hypothesis, we define specific questions under which we will evaluate the results of the industrial case study:

1. Does the prioritization framework provide systematic and effective method to guide the prioritization of technical debt in database normalization?

   To answer this question, we conduct a case study by systematically executing the phases of the framework, as described in section 4. The effectiveness of the proposed framework is observed through the overall ease of use, the systematic process, and the difference in effort cost between the conventional approach and the debt aware approach for database normalization.
2. Does the debt quality impact estimation provide insights to identify tables that are most affected by the debt?

   Insights from quality impact estimation are derived by measuring the impact on data quality, performance and maintainability of the identified deb tables, using the metrics described in section 4.2.1.
3. Is the portfolio analysis technique effective in prioritizing debt tables with high growth rate in the future?

   We apply the portfolio model as described in section 4.2.2 on the data collected from quality impact estimations for tables of high growth rates. To evaluate the effectiveness of the technique, we simulate future scenario and enlarge the debt tables according to their growth rates. Then, we re-measure the quality impact after tables' enlargement.
4. Does the TOPSIS method provide useful guidance to make informed normalization decisions?
   This question is answered through observation of the application of the TOPSIS method on the collected data, and see to what extent does it provide guidance to improve the quality of normalization

decisions.

5.1 Subject Application

The subject application was selected by the first author, who served as the company contact as well as well as the main researcher conducting the case study. The project is a database-backed web application for human resource management in a large company of 855 employees. The application is used to record employees' information and provide different services for the human resource department such as, attendance management, payroll management and employees' requests management. The application consists of a relational database with 97 tables and the program code was written in C# based on the Object Relational Mapping framework (ORM). ORM framework is a programming technique converting data between incompatible type systems in object-oriented programming languages [35]. This technique is gaining popularity among developers because it simplifies code generation. This is due to the fact that ORM abstracts the access to the database, and allows methods calls to be transparently translated to SQL queries by the framework. The application was developed with the following infrastructure: ASP.net and Microsoft SQL Server (To manage the database).

5.2 Case Study Process

There were 1 lead architect and 2 developers who audited and maintained the application. The case study was executed by the principal researcher PR (first author), with the cooperation of the developers. The PR played the role of the analyst. We simulated the phases of the framework using the data provided by the application and the technical team. We have conducted the study for nearly 8 months, where the PR met with the developers on weekly basis, each meeting lasting for an average of 2 to 3 hours. The meetings were used for obtaining data about the application and execute the phases of the proposed framework. The information used to conduct the case study came from a variety of sources, such as regular meetings with the developers, inspecting the database monitoring systems, and artifacts including the database management system, the code of the application, supported by a review of available literatures. The case study was conducted with the following general steps (details in the following subsections):

1. We obtained the data stored in the database tables and applied the framework described in [13], to identify potential debt tables below fourth normal form.
2. Utilizing the database monitoring system, the developers extracted the growth rates of the identified potential debt tables. We considered the top five tables of the highest growth rates for the analysis in this study.
3. For each of the five tables, we measured the impact on data quality using the risk of data inconsistency metric, performance using the model in equation (2) and maintainability using the number of attributes.
4. We applied the portfolio model on the collected data to produce optimum weights for the tables.
5. We met with the developers to estimate the effort cost of normalizing each of the five tables to the fourth normal form.
6. We applied the TOPSIS method detailed in section 4.3 on the collected data (debt tables' weights and normalization cost), to rank the debt tables for normalization.
7. We simulated future scenario and enlarged the debt tables according to their growth rates. We accomplished this by using the Redgate data generation tool [36] to populate each table with the required amount of data. The choice of using this tool was based on the development team's suggestion. After data generation, we re-measured the impact of each table on data quality, performance and maintainability and obtained the results.
8. With the cooperation of the developers, we normalized the suggested table and refactor the application code accordingly to measure improvements in the three quality attributes (data quality, performance and maintainability).
9. We met with the development team to discuss the outcomes and gather their feedback.

5.3 Case Study Execution

In this section, we will demonstrate the details and the results of the process steps presented in the previous section 5.2.

5.3.1 Framework Phase 1: Potential Debt Identification

We applied the steps of the framework explained in [13] to the tables in the database to identify tables below the fourth normal form. A database dictionary that describes the tables and the attributes of the database was not available. While most of the attributes were self-described by their names, such as Employee_ID and Salary, some of the attributes were described by the technical team. For each of the available tables we were allowed to obtain the stored data and load it to the RapidMiner software [10] to execute the mining algorithm and extract functional dependencies that holds in the table.

Following the steps of the framework detailed in [13] we were able to identify 17 potential debt tables below the fourth normal form. The following Table 1 demonstrates some of the identified tables and their normal forms:

Table 1 Potential debt tables below fourth normal form

Table name	Normal form
AttendanceRecords	BCNF
EmployeeDayAttendance Logs	2nd normal form
AuditTrials	2nd normal form
Leaves	1st normal form
LeaveBalances	2nd normal form

5.3.2 Framework Phase 2: Debt Impact and Cost Estimation

Estimating the Debt Quality Impact:

1. As explained in section 4.2.2 the first step is to determine the tables' growth rate. Tables of higher growth

rates are considered to be risky and their impact on quality will accumulate faster than other tables of lower growth rates. To collect information about the tables' growth, the lead developer generated a report from the database monitoring system that showed tables' growth rates in the past month. Note that we have access to the database monitoring system with a one month log of operations and tables information. The following Table 2 demonstrates the top 5 highest growth rate tables and their growth rates. As for the rest of the tables, most of them did not grow during the past month, and the remaining few tables had a substantially low growth rate, which indicates that the impact of those tables on quality attributes won't accumulate in the future. Therefore, we analyzed only those five risky tables (presented in Table 2), which are the best candidates for normalization due to their high growth rates, which will accumulate the impact on quality attributes faster than the other tables. Nevertheless, the analysis techniques presented in this study, can be applied to any number of tables.

Table 2 fastest growing tables and their growth rates

Table name	Growth rate
AttendanceRecords	1.04
EmployeeDayAttendance Logs	0.87
AuditTrials	0.86
Leaves	0.52
LeaveBalances	0.68

2. For each of the previous tables:
   - For data quality impact measurement, we calculate the risk of data inconsistency using the ISO metric [27] , a script was written in SQL language for each debt table that sums the duplicate values in all possible combinations of columns. Then, the result was divided by the number of columns multiplied by the number of rows. This process was repeatedly executed for each of the previous tables to calculate their risk of data inconsistency.
   - For maintainability impact measurement, the number of attributes for each of the previous tables was retrieved by the SQL server.
   - Finally, to measure performance impact, we extracted operations executed on those tables and their execution rates in the last month. Since the application was developed in an ORM framework, there were no SQL language queries written in the code. The ORM framework would translate methods calls to SQL language queries and execute them on the database. Therefore, operations were extracted from the database monitoring system. The database monitoring system has a feature of displaying the operations that were executed on the database in last month and their execution rates. So, a report was generated displaying those operations and how many times they were executed in the last month. Most of the extracted operations were long and complex. All of the operations were analyzed so that only operations that were executed on the tables of high growth rates were considered. To Determine the I/O costs of each operation executed on the debt table, SET STATISTICS IO feature in SQL server was used [37]. This feature display information about the amount of disk activity generated by SQL statements. We re-executed the queries with this feature to display the I/O costs. Some of the queries consists of 1 or more variables, we replaced those variables with actual data from the table that were selected randomly. Then, the queries were re-executed using the SET STATISTICS IO feature and the average I/O cost was recorded. A list was constructed with each debt table name, I/O cost of the operations executed on this table and the execution rate for each operation. After that, using the proposed model explained in section 4.2.1, we calculated the I/O cost for each of the previous tables.
3. We calculated the values of the portfolio variables for each quality attribute (expected return for each table = 1÷ quality impact, risk= table's growth rate reported in Table 2) the quality impact values were the results obtained from the previous step.
4. We have implemented a program to aid the execution of the portfolio model for each quality attribute and calculate the optimum weight for each table. The program applies a non- linear optimization method called Generalized Reduction Gradient (GRG) [38] to equation (3) as the fitness function and equation (4) as the constraint. We ran the portfolio model on the available data three times separately for each quality attribute to produce weights for each debt table. The resulted weights for each table impact in data quality, maintainability and performance are demonstrated in Table 3, Table 4 and Table 5 respectively. The portfolio model will provide the highest weights to tables with low quality impact and low growth rate. Therefore, table that has the lowest weight in each quality attribute implies the highest priority table that should normalized due to high impact on quality.

Table 3 Debt tables' information and weights of data quality impact

Table name	Risk of data inconsistency	Expected return	Weight
AttendanceRecords	2.5	0.4	0.0581
EmployeeDayAttendance Logs	0.236	4.237	0.7359
AuditTrials	1.31	0.763	0.1341
Leaves	5.55	0.180	0.0523
LeaveBalances	11.34	0.088	0.0196

From the results of Table 3, we can determine that table LeaveBalances has the highest priority to normalize to fourth normal form to improve data quality, since it got the lowest weight. Although this table has a growth rate less than table AttendanceRecords, the risk of data inconsistency of this table is the highest among the tables and it will accumulate faster than other than other tables.

It is also observable from the results in Table 3 that the risk of data inconsistency is independent from the normal form of the debt table. As shown, table AttendanceRecords has a higher risk of data inconsistency than table EmployeeDayAttendanceLogs and table AuditTrials, even though the latter tables are in a weaker normal form, which denotes that the normal form alone is not sufficient to make the right decision on which table to normalize.

Table 4 Debt tables information and weights of maintainability impact

Table name	Number of attributes	Expected return	Weight
AttendanceRecords	6	0.1667	0.1824
EmployeeDayAttendance Logs	8	0.125	0.1635
AuditTrials	10	0.1	0.1323
Leaves	7	0.1429	0.3127
LeaveBalances	8	0.125	0.2092

Based on the results from Table 4, table AuditTrails has the highest priority to normalize to decrease complexity. Even though this table has a growth rate lower than AttendeanceRecords and EmplyeeDayAttendanceLogs, the number of AuditTrails attributes is the highest and the complexity of this table will increase faster with growing data to be stored in this table. It is also observable that table EmployeeDayAttendanceLogs is the second prioritized table to normalize to the fourth normal form, even with number of attributes similar to table LeaveBalances, however, the growth rate of table EmployeeDayAttendanceLogs is higher which will increase the complexity of the table faster.

Table 5 Debt tables' information and weights of performance impact

Table name	Average I/O cost	Expected return	Weight
AttendanceRecords	62822956	1.59	0.000073
EmployeeDayAttendance Logs	27897228	3.58	0.000197
AuditTrials	23020	4344.05	0.242
Leaves	15184	6585.88	0.606
LeaveBalances	46240	2162.63	0.152

Table 5 indicates that table AttendanceRecords has the highest priority to normalize to the fourth normal form to improve performance. This is due to the fact that the I/O cost incurred by operations executed on this table is the highest among tables, meaning the cost of operations performed on that table and the execution rate of those operations on a monthly basis are high, in addition to the highest growth rate of table AttendandanceRecords, which will accelerate I\O cost accumulation in the future and affect performance. Table EmployeeDayAttendanceLogs has the second priority for normalization. As seen, the difference of the weights between table AttendanceRecords and table EmployeeDayAttendanceLogs is relatively small, which indicates that both tables are semi-equally important to normalize, considering time and budget constraints.

Estimating the Cost of Normalization:

To estimate the cost of normalizing each table to the fourth normal form, we met with technical team and determine all the major refactoring tasks that are required to normalize the tables. The tasks and sub tasks were as mentioned in section 4.2.3. The cost of each candidate table was analyzed in accordance to those tasks. As shown, the tasks were further refined into sub-tasks to ease and concretize costing. For each sub-task under consideration, we solicited from the team three estimates for the time required for each subtask: best, most likely and worst case. The team used previous experience to arrive on the best and worst cases estimates. For the most likely case, the team suggested taking the average of the best and worst. In our calculation, we considered the most likely time estimate for each subtask and calculated the total time required to normalize each table, using equation (6). The results are shown in Table 6. It is worth noting the estimates are in the context of the team's experience and capabilities to perform the said tasks calculated in hours (i.e. they are relative estimates).

Table 6 Normalization cost for each table

Table name	Average cost to normalize to fourth normal form in hours
AttendanceRecords	268
EmployeeDayAttendance Logs	235
AuditTrials	252
Leaves	524
LeaveBalances	525

5.3.3 Framework Phase 3: Making Decisions Using the TOPSIS Method

So far, we have estimated the impact of the debt tables on three quality attributes. We have also estimated the cost of normalizing each table to the fourth normal form. The next step is to incorporate all this information to prioritize tables that need to be normalized. The proposed approach uses the TOPSIS method detailed in section 4.3. We implemented a program to aid in calculations of TOPSIS following the TOPSIS method equations explained in section 4.3.

Step 1: **Construct the evaluation matrix:** Table 7 demonstrates the evaluation matrix that consists of the debt tables, the criteria, the tables' weights from the portfolio analysis and the tables' normalization cost.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2020.3001339, IEEE Transactions on Software Engineering

Table 7 Evaluation Matrix

Table/Criteria	Performance	Data Quality	Maintainability	Cost/ hours
AttendanceRecords	0.000073	0.0581	0.1824	268
EmployeeDayAttendanceLogs	0.000197	0.7359	0.1635	235
Audit Trials	0.242	0.1341	0.1323	252
Leaves	0.606	0.0523	0.3127	524
LeaveBalances	0.152	0.0196	0.2092	525

The numeric calculations of steps 2 to 6 are depicted in the Appendix. The calculations followed the principles and equations corresponding to each step as described in section 4.3.

Fig 3 plots the distances calculation results of step 5. The vertical axis corresponds to the distance between each table and the worst condition (D_{iw}), while the horizontal axis presents the distance from the best condition (D_{ib}). Looking at the figure, we can observe that tables that fall above the diagonal are the most promising tables to normalize since they have the shortest distance from the best condition and longest distance from the worst condition. As seen, table AttendanceRecords is potentially the best table to normalize to improve the three qualities at the minimum cost possible. Although table EmployeeDayAttendanceLogs is possibly the cheapest to normalize, this table is the farther from the best condition, as opposed to table LeaveBalances, which is closer to the best condition despite having the highest cost of normalization. This can be justified through the distances between each table and the best table to normalize to improve each criterion. For example, table EmployessDayAttendaceLogs has the lowest priority when it comes to improve data quality, and the distance between EmployeeDayAttendencLogs and the best solution to improve data quality is equal to 0.238 (calculated as the difference between: EmployeeAttendenceLogs value in data quality in the weighted normalized matrix, and the best solution to improve data quality, respectively depicted in Table 2A and Table 3A of the Appendix). On the other hand, table LeaveBalances has the lowest priority in cost (since it is the most expensive table to normalize), however, the distance between LeaveBalances and the best solution in cost criterion is equal to 0.0842 (calculated as the difference between: LeaveBalances value in cost in the weighted normalized matrix, and the best solution for cost, respectively depicted in Table 2A and Table 3A of the Appendix). Since the latter distance is shorter, this implies that table LeaveBalances has a bigger impact on cost than table EmployeeDayAttendenceLog's impact on data quality. Since all the three criteria are assumed to be equally important, and taking into account all the distances, table EmployeeDayAttendanceLogs is the farther from the best condition and closer to the worst condition.

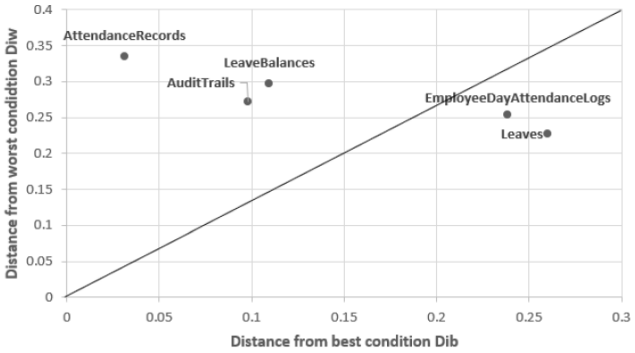

Fig 3 Debt tables distance from best and worst condition

Step 7: rank the preference order. Results of the final step and tables' ranking are shown in the following Table 8:

Table 8 Debt tables' ranks:

Table/Criteria	rank
AttendanceRecords	1
EmployeeDayAttendanceLogs	4
Audit Trials	2
Leaves	5
LeaveBalances	3

Results from Table 8 is consistent with the analysis described for the chart in Fig 3. As seen in the table, Attendance records is ranked first among the other tables that should be normalized to improve performance, data quality and maintainability at minimum cost.

Sensitivity Analysis:

The purpose of the sensitivity analysis is to determine how sensitive the analysis to the developers cost estimates and the range they provided, which is inherently suffers from uncertainty. Sensitivity analysis allows the developers to explore changes (incremental for our experiment) in the cost estimates to understand how they affect the ordering of the tables. To do this, we ran the TOPSIS calculations several times with different cost values for each table at a time. To perform the sensitivity analysis, we have applied an increment of + 10% at each calculation, moving from the most likely estimate to the worst case one for cost. The following Table 9 demonstrates the results of analysis.

Table 9 sensitivity analysis of the cost estimations:

Table name	Rank	cost	Worst cost percentage increase	Rank after increase
AttendanceRecords	1	268	33.21%	1
EmployeeDayAttendanceLogs	4	235	34.04%	4
Audit Trials	2	252	32.14%	3
Leaves	5	524	30.73%	5
LeaveBalances	3	525	35.81%	3

The Worst cost percentage increase of Table 9 depicts the percentage increase on costs to reach the worst case estimates. As an example, Attendance records needs 33.21% increase in cost to reach its worst case estimate. The last 10% covers the worst cost percentage increase. Results of the previous table are plotted in Fig 4. As seen, all the tables' ranking were stable within the worst case estimate increase in the cost, except for table AuditTrails which rank changed from second to third table to normalize with 10.8% increase in its normalization cost and table LeaveBalances changes from third to second in ranking. However, this change does not affect the ranking of the first prioritized table (i.e. AttendanceRecords) that should be normalized to fourth normal form.

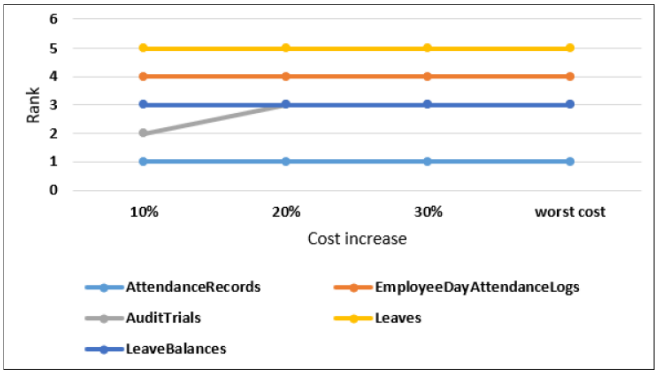

Fig 4 Stability of tables' ranking after cost increase

5.3.4 Future scenario simulation and debt table normalization

As discussed, we have performed this case study by simulating the phases of the proposed framework using real data from the Human resource application. To evaluate the results, we simulated future scenario and executed the following:

- We used data generation tool (Redgate tool [36]), suggested by the developers to populate the tables with the required amount data to nearly emulate the likely growth rate.
- We re- measured the tables' impact on data quality, performance and maintainability in the same manner as we did when we executed the phases of the framework. Results are shown in Table 10.

Table 10 Debt tables' impact on quality attributes after tables' growth

Table Name	Risk of data inconsistency	Performance (I/O Cost)	Maintainability
AttendanceRecords	3.63	122290310	6
EmployeeDayAttendanceLogs	0.425	66765276	8
Audit Trials	1.853	40092	10
Leaves	6.91	29924	7
LeaveBalances	12.28	86364	8

- We normalized table AttendenceRecords to the fourth normal form. The normalization included several tasks: decompose the table to three tables (AttendanceRecords, DeviceMode and DeviceProcess), write scripts to migrate data to new tables, create indexes and refactor the application code to reflect the changes.
- Finally, we measured the improvement after table's normalization in the three quality attributes as the following:
  For data quality: we measured the risk of data inconsistency in all three tables using the ISO metric in equation (1).
  For performance: we extracted the new SQL code of the operations that used to retrieve data from AttendanceRecords. The new operations will now retrieve data from the new three tables after decomposition. We obtained the total I/O costs of the operations executed on the new three tables. Then, Assuming the execution rate of each operation is the same as before normalization, we used equation (2) explained in section 4.2.1 to get the total average I/O costs of the operations executed on the three tables.
  For maintainability: number of attributes of table AttendanceRecords was easily obtained from the SQL server.
  Results of quality impact improvements are shown in Table 11

Table 11 table AttendanceRecords impact on quality attributes after normalization

Quality attribute		Value
Data quality	Before normalization	3.63
	After normalization	1.137
	% Rate of impact reduction	- 68.68%
Performance (I/O cost)	Before normalization	122290310
	After normalization	87751829
	% Rate of impact reduction	-28.24%
Maintainability	Before normalization	6
	After normalization	5
	% Rate of impact reduction	-16.7%

5.4 Analysis and Discussion

In this section, we present a critical evaluation of our framework by assessing the framework against the evaluation questions defined in section 5

1.Does the prioritization framework provide systematic and effective method to guide the prioritization of technical debt in database normalization?

The first aspect we discuss is to what extent the framework facilitates the normalization decision making process. The ease to use is a key aspect that has motivated the development of the framework. Several aspects of the framework contribute to making it easy to use. For

instance, the use of tools (database monitoring tool to elicit required information, and programs to execute some of the steps in each phase of the method), facilitates the application of the method without previous experience in database normalization. The framework is based on intuitive concepts organized in a structured manner, which contributes to its repeatability. Furthermore, users of the framework do not need to understand the details of the decision theory behind TOPSIS or Portfolio analysis technique. On the other hand, users must have a comprehensive understanding of organization resources and knowledge about the refactoring tasks to estimate the cost of normalization. The guidance provided by us during the case study process to execute each step and obtain required information seems to be beneficial and easy to apply in the context of the human resource web application.

Concerning the systematic process of the framework, we have observed that the framework covers all major steps necessary to conduct a disciplined selection process. The framework phases provide a clear, structured process to guide the acquisition and analysis of relevant information. This information assists the developers in making informed decisions, and therefore, choosing a table to normalize to improve quality at minimum cost. The added value the framework we propose has given to the decision process is that it provides guidance on how to measure the debt impact of individual tables to facilitate the prioritization process. Concerning normalization cost estimation, our work is consistent with current practices in considering the effort and person hours per task. These task's cost were estimated using expert s' judgments. We shall note that paper is not claiming contribution on effort estimation for database refactoring, In addition to experts' judgement, future work may consider developing specialized parametric models for the cost/effort of database refactoring, which is outside the scope of our current contribution.

Though our work provides a list of prioritized tables that need to be normalized based on the debt information, the developers may decide to further revisit the list to eliminate tables that are not worth the consideration. The decision to eliminate a table from the list is highly dependent on the expertise of the developers, their familiarity with a similar case, and whether the improvements are not worth the consideration. Nevertheless, we have taken a conservative approach by using the available resources to decide on the tables to normalize among the prioritized list. The following Table 12 shows that following the conventional approach, which encourage normalizing all the tables to fourth normal form is more costly; time consuming and ad-hoc than the debt-aware approach for four scenarios: The first scenario prioritizes debt-tables based on their impact on data quality, where the portfolio approach results from Table3 suggests to only normalize one table, table LeaveBalances, based on the biggest amount of data duplication it holds and its growth rate compared to other tables. The second scenario considers the performance impact of each debt table, and the likely accumulation of this impact in the future. By utilizing the portfolio approach, results from Table 5 suggest two tables for normalization, AttendanceRecords and EmployessDayAttendanceLogs. To minimize complexity, the third scenario suggests normalizing table AuditTrails based on the results from Table 4. Finally, to improve all three quality attributes and taking into account the effort cost of normalizing each table, table AttendanceRecords is suggested to be normalized.

Table 12 Difference in effort between conventional approach and the debt-aware approach of normalization

Approach	Number of tables to normalize
Conventional Approach	17
Debt-aware approach to improve data quality	1
Debt-aware approach to improve performance	2
Debt-aware approach to improve maintainability	1
Debt-aware approach to improve three qualities	1

Moreover, depending on available resources, developers can include more tables to normalize and justify their decisions based on the debt tables' impact on data quality, performance, maintainability and cost.

Our approach is built on the premise that the identified root causes of the debt have to do with inadequate normalization, and its impact is observed on qualities such as performance, maintainability and data quality. It is imperative that the architect may consider other architectural tactics and fixes to rectify these problems. Though the tactics and fixes may provide immediate benefits in some contexts, they won't eliminate the root causes of the debt and the debt will continue to be dormant. The architect may need to outweigh the long-term benefits and costs of these fixes against that of normalization. Otherwise, these fixes can be regarded as taking a debt on existing debt.

2. Does the debt quality impact estimation provide insights to identify tables that are most affected by the debt?

Our interest in debt impact is centered on the fact that it is suitable for identifying tables that are candidate for normalization based on their data quality, performance and maintainability impact, regardless of the normal form of the table. The notion of debt impact is particularly appropriate to understand to what extent a potential debt table may hurt the quality compared to other potential debt tables, and therefore facilitate the decision-making process regarding which table to normalize.

We have started the case study by identifying potential debt tables (i.e. tables below the fourth normal form) in the database. After the identification process, it can be suggested to normalize the weakest normal form tables to improve the design and the systems' quality. The suggestion will be based on the fact that in the normalization theory, the higher the normal form of the table the better design will be. Which is true "design-wise", however when the notion of quality impact is introduced and after measuring the data quality of the debt tables, evidence has shown that weakly normalized tables may have better data quality than higher normal form tables as observed from Table 3. This is due to the fact that the data quality metric present

from ISO, measures the risk of data inconsistency based on the amount of duplicated data stored in the table. Some weakly normalized tables store less amount of data and thus have less impact on data quality. This empirical evidence reinforces our assumption that the prioritization process is independent from the table normal form. Similarly, when measuring the performance impact, our proposed model is based on I/O cost of operations executed on the table which is determined by the table size and the execution rate of the operation, regardless of the table normal form.

Our maintainability metric does not solely rely on the number of attributes in a table, but also considers the growth rate of the table size. Our contribution is not aimed at designing a maintainability metric for databases per se; however, our approach is flexible enough to make use of other metrics for maintainability, if available. These metrics can be parametric; open-ended based on experts' judgment and/or back of the envelope calculation for a given domain; or can utilize more sophisticated methods, such as learning from cross-companies/projects databases to estimate likely maintainability effort for a given case. The measures can incorporate various maintainability concerns and dimensions depending on the context.

3. Is the portfolio analysis technique effective in prioritizing debt tables with high growth rate in the future?

As with most database applications, the human resource web application grows in the amount of data stored in the database. This growth is a significant factor to make a better decision regarding debt payment. Tables with high growth rate will accelerate the impact of the debt and its accumulation on the three qualities faster than the other tables of lower growth rates. Therefore, if the table is not likely to grow or its growing rate is less than other tables, a strategic decision would be to keep the debt and defer its payment. The portfolio analysis provides the developers with an objective tool to assess and rethink their normalization decisions using debt impact and the risk of table growth in the future.

The exercise of applying the portfolio analysis to the human resource web application proved to be valid based on the data reported after simulating future scenario and enlarging each table under analysis according to their growth rate. After data growth, we re-measured the debt tables' impact on the three quality attributes. Results are shown in Table 10.

Portfolio analysis prioritization, applied (before data growth), is depicted in Tables 3, 4, and 5. Table 10 Shows the ranking of these tables after data growth. The results show consistency with the Portfolio prioritization results. For example, Table 10 Shows LeaveBalances to have the highest risk of data inconsistency after data growth. The same table (i.e LeaveBalances) had the highest priority to be normalized to the fourth normal form, as suggested by the portfolio analysis in Table 3, where LeaveBalances had the lowest weight. Similarly, for performance, operations executed on table AttendanceRecords appears to have the highest I/O cost in as shown in Table 10, which is consistent with the results of the portfolio prioritization results in Table 5, where AttendanceRecords had the lowest weight that implies highest priority to normalize. Since the number of attributes has not changed, maintainability tends to be sensitive to the growth of data stored in the table, i.e. rows.

The results indicate the fitness of the proposed technique towards improving the quality considering future growth. Additional benefit of using the portfolio approach is that its input is from specific data drawn from the quality measurement and the database monitoring tool that elicited tables' growth rates, which eliminates biased and ad-hoc decisions regarding which table to normalize.

4. Does the TOPSIS method provide useful guidance to make informed normalization decisions?

Improving the quality of decisions made for database normalization is the main objective of this research. In this context, the objective of the TOPSIS method is to organize relevant information gathered through the prioritization framework. Utilizing the TOPSIS method to prioritize the debt tables can benefit the developers in two aspects. First, it gives the developers the flexibility to include or exclude the quality attributes that are necessary for given valuation context. Moreover, based on available resources and the systems conditions, developers are able to weigh the level of importance for each quality attribute.

In the case study, information about debt impact on data quality, performance, maintainability and cost of database normalization, have been estimated and gathered during the framework phase 2. Developers need a structured, flexible and simple method to organize all information to make the best decisions. The TOPSIS method represented tool that can be utilized at any time when new information becomes available or when a specific quality dimension required to be improved over the other, which may change as the system evolves. The TOPSIS method has given explicit reasoning of how trade-offs is achieved to deal with factors affecting debt tables' prioritization.

To further evaluate our proposed approach, we have worked with the development team to normalize table AttendanceRecords to the fourth normal form and refactor the application code as described in Section 5.3.4, After normalization, we re-measured the impact of the new decomposed tables on data quality, performance and maintainability as explained in section 5.3.4. The results are shown in Table 11.

As shown in Table 11, after normalization to the fourth normal form, the impact in data quality is reduced by 68.68%. Performance impact is decreased by 28.24% and maintainability impact is decreased by 16.7%. The results provide evidence of the negative impact that was attributed to a normal form weaker than the fourth normal form.

In summary, developers' feedback on the framework has been positive. The approach was an eye opening and has encouraged the team to reconsider their current practices: In particular, the developers often avoid, on any expenses, restructuring the database to tune performance, due to the lack of guidance on which table to refactor among the available ones. Our approach has led the developers to recognize that refactoring the database for normalization is essential to treat the root cause of the problem

and improve quality; they appreciate the structured approach to prioritize the tables that should be normalized to improve the quality at the minimum cost possible. Moreover, combining the systematic debt analysis with measuring the debt impact of the table on the three qualities based on data from the database and the monitoring system, has provided the developers with previously uncovered knowledge for informing the refactoring exercise. The approach was deemed to be more informative and objective regarding database normalization for mitigating the debts over ad hoc normalization and refactoring.

5.5 Threats to Validity

Our evaluation has used an industrial case study. Case studies are difficult to generalize its results [39], due to its specificity to the case/domain. In this study, the presented project is a database application, developed and maintained using specific technologies and infrastructure, which may have different cost pattern and management style than other projects. For example, managing trade-offs between qualities affected by the debt can be more complex in an integrated database project (i.e. a database that serves multiple applications [40]). Moreover, the evaluation has used the case study developers for feedback on the overall effectiveness of the framework.

Therefore, our findings reflect experience in just one particular case, the goal being transferability, not generalizability. Meaning, the specific findings of this study may be applicable to other projects, and the general lessons learned should be instructive to those applying and studying this approach in any situation. Further analysis may need to look at applying the method on several applications of industrial scale and/or to report on the applicability of the method using more than one independent team. Both routes are non-trivial and require a careful empirical/field study that goes beyond the scope of this paper. Nevertheless, this is subject to future work.

6 Related Work

Ward Cunningham was the first to introduce this metaphor in 1992 on the code level, as a trade-off between short term business goals (e.g. shipping the application early) and long term goals of applying the best coding practices [41]. The majority of researches have focused on code and architectural level debt [10],[6]. In [11], Weber et al. discussed a specific type of database schema design debt in missing foreign keys in relational databases. To illustrate the concept, the authors examined this type of debt in an electronic medical record system called OSCAR. The authors proposed an iterative process to manage foreign keys debt in different deployment sites. Our work is different as it is driven by normalization rules, which are fundamentally different in their requirements and treatment. Tables in databases have specific requirements and the likely impact of normalization can be observed on different quality metrics that are database specific.

As our work is close to debt prioritization, authors in [32] utilized prioritization to manage code level debts in software design. The authors estimated the impact of God classes on software maintainability and correctness. They prioritized classes that should be refactored based on their impact on those qualities. Portfolio Theory was proposed by researchers to manage technical debt [42]. In [42] the authors viewed each debt item as an asset, and they utilized Portfolio Theory to construct a portfolio of debt items that should be kept based on debt principal, which they defined as the effort required to remove the debt, and debt interest which is the extra work needed if the debt is not removed. Portfolio Theory was also proposed to manage requirements compliance debt in [43]. The authors viewed compliance management as an investment activity that needs decisions to be made about the right compliance goals under uncertainty. They identified an optimal portfolio of obstacles that needed to be resolved, and address value-driven requirements based on their economics and risks.

Since the process of normalizing a database can be very complex and costly, researchers have proposed algorithms to automate or facilitate the normalization process [44], [45], [46]. Their aim was to produce up to third normal form or BCNF tables automatically. However, the studies looked at the database schema in isolation from applications using the database. It is important to consider the applications to better estimate the cost of normalization taking into account refactoring and data migration tasks. Since this process can be very costly, this study aims to provide a method to prioritize tables to be normalized to improve the design and avoid negative consequences.

7 Conclusion and Future Work

We have explored the concept of technical debt in database design that relates to database normalization. Normalizing a database is acknowledged to be an essential process to improve data quality, maintainability and performance. Conventional approaches for normalization is driven by the acclaimed structural and behavioral benefits that higher normal forms are likely to provide, without explicit to value and debt information. The problem relies on the fact that developers tend to overlook this process for several reasons, such as, saving effort cost, lack of expertise or meeting the deadline. This can imply a debt that need to be managed carefully to avoid negative consequences in the future. Conversely, designers tend to embark on database normalization without clear justification for the effort and debt avoidance.

We reported on a framework to manage normalization debt in database design. A table below the fourth normal form is viewed as a potential debt item. Though we considered tables below the fourth normal form as potential debt items, in practice most databases lag behind this level [14]. Among the reasons for avoiding normalization, database refactoring is often acknowledged to be a tedious and expensive exercise which developers avoids on any expense due to its unpredictable outcome [5]. Additionally, it is impractical for developers to use the fourth normal form as the only criteria to drive the normalization exercise. To overcome these problems, we proposed a framework to prioritize tables and their candidacy for normalization. The framework utilizes the Portfolio theory and the

TOPSIS method to rank the tables based on their estimated impact on data quality, performance, maintainability, in addition to the cost of normalization.

The framework was applied to an industrial case study of a database-backed web application for human resource management. The database consists of 97 tables, exhibiting complex dependencies and populated with large amount of data. 17 out of the 97 tables were considered as potential debt tables. Out of the 17 tables, 5 exhibited a high growth rate and were found to have the highest risk of impact accumulation on data and system quality. Our framework ranked these tables based on the data and system quality impact and their corresponding cost for normalization. Depending on availability of resources and personnel, developers may normalize the 5 tables, with the highest priority given to tables with the highest impact and cost. The results show that rethinking conventional database normalization from the debt angle, by not only relying on the normal form to re-design the database, can provide more systematic guidance to justify normalization decisions and improve quality at minimum cost. The proposed framework focused on the prioritization aspect of managing the debt. The evaluation of the overall technical debt management framework may require an extensive empirical and/or field study, where the management of debt items may require monitoring longer period of time for debt and impact accumulation, which can be subjected to future work.

Though our framework has been described and applied to a standalone database, it can be further extended to serve other industrial large-scale systems, such as integrated databases. However, the formulation can be more complex as the prioritization needs to consider the various quality requirements per application in the integrated schema, manage their trade-offs and conflicts when consolidating the final set of requirements to be considered in the analysis, which is subject to future work due to its specificity in treatments.

ACKNOWLEDGMENTS

For Mashel Albarak:
This research is supported by the Deanship of Scientific Research, King Saud University through the initiative of DSR Graduate Students Research Support.
For Robert Nord, and Ipek Ozkaya:
This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.
References herein to any specific commercial product, process, or service by trade name, trade mark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by Carnegie Mellon University or its Software Engineering Institute.
DM19-1256.

REFERENCES

[1] E. F. Codd, "A relational model of data for large shared data banks," *Communications of the ACM*, vol. 13, no. 6, pp. 377–387, Jun. 1970.

[2] C. Date, *Database Design and Relational Theory: Normal Forms and All That Jazz*. O'Reilly Media, Inc., 2012.

[3] D. G. Jha, *Computer Concepts and Management Information Systems*. PHI Learning Pvt. Ltd., 2013.

[4] "DB-Engines Ranking - popularity ranking of database management systems." [Online]. Available: http://db-engines.com/en/ranking. [Accessed: 16-Aug-2016].

[5] S. W. Ambler and P. J. Sadalage, *Refactoring databases: Evolutionary database design*. Pearson Education, 2006.

[6] Z. Li, P. Avgeriou, and P. Liang, "A systematic mapping study on technical debt and its management," *Journal of Systems and Software*, vol. 101, pp. 193–220, Mar. 2015.

[7] "TechDebt 2018 International Conference on Technical Debt - TechDebt 2018." [Online]. Available: https://2018.techdebtconf.org/track/TechDebt-2018-papers#Previous-Editions. [Accessed: 03-Jun-2018].

[8] "Euromicro DSD/SEAA 2018 | SEAA 2018." [Online]. Available: http://dsd-seaa2018.fit.cvut.cz/seaa/index.php?sec=sessions_seated#page_header. [Accessed: 22-Mar-2018].

[9] S. D.-L.-Z. für I. G. Wadern 66687, "Schloss Dagstuhl : Seminar Homepage." [Online]. Available: http://www.dagstuhl.de/en/program/calendar/semhp/?semnr=16162. [Accessed: 07-Aug-2017].

[10] C. Fernández-Sánchez, J. Garbajosa, A. Yagüe, and J. Perez, "Identification and analysis of the elements required to manage technical debt by means of a systematic mapping study," *Journal of Systems and Software*, vol. 124, pp. 22–38, Feb. 2017.

[11] J. H. Weber, A. Cleve, L. Meurice, and F. J. B. Ruiz, "Managing Technical Debt in Database Schemas of Critical Software," 2014, pp. 43–46.

[12] M. Albarak and R. Bahsoon, "Prioritzing Technical Debt in Database Normalization Using Portfolio Theory and Data Quality Metrics," in *Proceedings of the International Conference on Technical Debt*, Gothenburg, Sweden, 2018.

[13] M. Albarak, M. Alrazgan, and R. Bahsoon, "Identifying Technical Debt in Database Normalization Using Association Rule Mining," in *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, Prague, 2018, pp. 437–441.

[14] R. Elmasri and S. B. Navathe, "Database systems: models, languages, design, and application programming," 2011.

[15] H. Markowitz, "Portfolio selection," *The journal of finance*, vol. 7, no. 1, pp. 77–91, 1952.

[16] M. Piattini, C. Calero, and M. Genero, "Table oriented metrics for relational databases," *Software Quality Journal*, vol. 9, no. 2, pp. 79–97, 2001.

[17] C.-L. Hwang and K. Yoon, "Methods for multiple attribute decision making," in *Multiple attribute decision making*, Springer, 1981, pp. 58–191.

[18] S. Marion, A. Kagan, and H. Shimura, "Performance criteria for relational databases in different normal forms," vol. 34, no. 1, pp. 31–42, 1996.

[19] I. M. Keshta, "Software Refactoring Approaches: A Survey," *INTERNATIONAL JOURNAL OF ADVANCED COMPUTER*

SCIENCE AND APPLICATIONS, vol. 8, no. 11, pp. 542–547, 2017.

[20] G. Vial, "Database refactoring: Lessons from the trenches," *IEEE Software*, vol. 32, no. 6, pp. 71–79, 2015.

[21] K. Hamaji and Y. Nakamoto, "Toward a Database Refactoring Support Tool," in *2016 Fourth International Symposium on Computing and Networking (CANDAR)*, 2016, pp. 443–446.

[22] P. Khumnin and T. Senivongse, "SQL antipatterns detection and database refactoring process," in *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2017 18th IEEE/ACIS International Conference on*, 2017, pp. 199–205.

[23] A. Cleve, M. Gobert, L. Meurice, J. Maes, and J. Weber, "Understanding database schema evolution: A case study," *Science of Computer Programming*, vol. 97, pp. 113–121, Jan. 2015.

[24] L. Meurice and A. Cleve, "Dahlia: A visual analyzer of database schema evolution," in *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on*, 2014, pp. 464–468.

[25] M. S. Wu, "The practical need for fourth normal form," in *ACM SIGCSE Bulletin*, 1992, vol. 24, pp. 19–23.

[26] G. Piatetsky-Shapiro, "Discovery, analysis, and presentation of strong rules," *Knowledge discovery in databases*, pp. 229–238, 1991.

[27] "ISO/IEC 25024:2015 - Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- Measurement of data quality," *ISO*. [Online]. Available: http://www.iso.org/iso/catalogue_detail.htm?csnumber=35749.

[28] ISO/IEC, "International Standard-ISO/IEC 14764 IEEE Std 14764-2006 Software Engineering; Software Life Cycle Processes &; Maintenance," 2006.

[29] C. Calero and M. Piattini, "Metrics for databases: a way to assure the quality," in *Information and database quality*, Springer, 2002, pp. 57–83.

[30] G. Papastefanatos, P. Vassiliadis, A. Simitsis, and Y. Vassiliou, "Metrics for the prediction of evolution impact in etl ecosystems: A case study," *Journal on Data Semantics*, vol. 1, no. 2, pp. 75–97, 2012.

[31] V. E. Ferraggine, J. H. Doorn, and L. C. Rivero, *Handbook of Research on Innovations in Database Technologies and Applications: Current and Future Trends*. Information Science Reference Hershey, PA, 2009.

[32] N. Zazworka, C. Seaman, and F. Shull, "Prioritizing design debt investment opportunities," in *Proceedings of the 2nd Workshop on Managing Technical Debt*, 2011, pp. 39–42.

[33] H.-S. Shih, H.-J. Shyur, and E. S. Lee, "An extension of TOPSIS for group decision making," *Mathematical and Computer Modelling*, vol. 45, no. 7–8, pp. 801–813, 2007.

[34] B. Kitchenham, S. Linkman, and D. Law, "DESMET: a methodology for evaluating software engineering methods and tools," *Computing Control Engineering Journal*, vol. 8, no. 3, pp. 120–126, Jun. 1997.

[35] "What is Object/Relational Mapping? - Hibernate ORM." [Online]. Available: http://hibernate.org/orm/what-is-an-orm/. [Accessed: 04-Jul-2018].

[36] "SQL Data Generator - Data Generator For MS SQL Server Databases." [Online]. Available: https://www.red-gate.com/products/sql-development/sql-data-generator/. [Accessed: 12-May-2019].

[37] CarlRabeler, "SET STATISTICS IO (Transact-SQL)." [Online]. Available: https://docs.microsoft.com/en-us/sql/t-sql/statements/set-statistics-io-transact-sql. [Accessed: 05-Jul-2018].

[38] L. S. Lasdon, A. D. Waren, A. Jain, and M. Ratner, "Design and Testing of a Generalized Reduced Gradient Code for Nonlinear Programming," *ACM Trans. Math. Softw.*, vol. 4, no. 1, pp. 34–50, Mar. 1978.

[39] R. K. Yin, *Case study research Design and methods*, Third. Sage publications, 2003.

[40] M. Fowler, "IntegrationDatabase," *martinfowler.com*. [Online]. Available: https://martinfowler.com/bliki/IntegrationDatabase.html.

[41] W. Cunningham, "The WyCash Portfolio Management System," in *Addendum to the Proceedings on Object-oriented Programming Systems, Languages, and Applications (Addendum)*, New York, NY, USA, 1992, pp. 29–30.

[42] Y. Guo and C. Seaman, "A portfolio approach to technical debt management," in *Proceedings of the 2nd Workshop on Managing Technical Debt*, 2011, pp. 31–34.

[43] B. Ojameruaye and R. Bahsoon, "Systematic elaboration of compliance requirements using compliance debt and portfolio theory," in *International Working Conference on Requirements Engineering: Foundation for Software Quality*, 2014, pp. 152–167.

[44] M. Demba, "Algorithm for Relational Database Normalization Up to 3NF," *International Journal of Database Management Systems*, vol. 5, no. 3, pp. 39–51, Jun. 2013.

[45] Y. . Dongare, P. . Dhabe, and S. . Deshmukh, "RDBNorma: - A semi-automated tool for relational database schema normalization up to third normal form," *International Journal of Database Management Systems*, vol. 3, no. 1, pp. 133–154, Feb. 2011.

[46] J. Diederich and J. Milton, "New methods and fast algorithms for database normalization," *ACM Transactions on Database Systems (TODS)*, vol. 13, no. 3, pp. 339–365, 1988.

Mashel Albarak is with the School of Computer Science at the University of Birmingham, UK, and King Saud University, KSA. Her interests are in managing technical debt in databases and information systems.

Rami Bahsoon is an academic at the School of Computer Science, University of Birmingham, UK. His research is in software architecture, self-adaptive and managed architectures, economics-driven software engineering and technical debt management. He co-edited four books on Software Architecture, including Economics-Driven Software Architecture. He holds a PhD in Software Engineering from University College London and was MBA Fellow at London Business School. He is a fellow of the Royal Society of Arts and Associate Editor of IEEE Software.

Ipek Ozkaya is a technical director at the Carnegie Mellon University Software Engineering Institute, where she develops methods and practices for software architectures, agile development, and managing technical debt in complex systems. She coauthored a book on Managing Technical Debt: Reducing Friction in Software Development (2019). She received a PhD in Computational Design from CMU. Ozkaya is a senior member of IEEE and the 2019—2021 editor-in-chief of IEEE Software magazine.

Robert Nord is a principal researcher at the Carnegie Mellon University Software Engineering Institute, where he develops methods and practices for agile at scale, software architecture, and managing technical debt. He is coauthor of Managing Technical Debt: Reducing Friction in Software Development (2019). He received a PhD in computer science from CMU and is a distinguished member of the ACM.