AUTOMATED MAINTENANCE SUPPORT FOR DATA-TIER SOFTWARE

A Dissertation

by

Ersin Ersoy

Submitted to the
Graduate School of Sciences and Engineering
In Partial Fulfillment of the Requirements for
the Degree of

Doctor of Philosophy

in the
Department of Computer Science

Özyeğin University
July 2022

AUTOMATED MAINTENANCE SUPPORT FOR DATA-TIER SOFTWARE

Approved by:

Assoc. Prof. Hasan Sözer, Advisor
Department of Computer Science
Özyeğin University

Assoc. Prof. Mehmet S. Aktaş
Department of Computer Science
Yıldız Teknik University

Assist. Prof. M. Furkan Kıraç
Department of Computer Science
Özyeğin University

Assoc. Prof. Kamer Kaya
Department of Computer Science
and Engineering
Sabancı University

Date Approved: 7 July 2022

Prof. O. Örsan Özener
Department of Industrial
Engineering
Özyeğin University

To my family

ABSTRACT

Data-tier software includes the data model and business logic of enterprise systems, and it is subject to long-term maintenance. Even though the user interface of these systems can be completely replaced, data-tier software usually evolves for decades. The number of domain experts with extensive knowledge about the overall software diminishes in time and applying extensions or changes becomes increasingly effort-consuming and error-prone for new developers. In this thesis, we introduce techniques and tools to provide automated maintenance support for data-tier software. These techniques and tools aim at reducing effort and the number of errors specifically for three challenging maintenance tasks: *i)* correct placement of a new object like a stored procedure in data-tier software, *ii)* evaluating the impact of changing database tables on software modules, and *iii)* evaluating the impact of table extensions on other tables of the same database. The first task is important because introducing a new object to data-tier software should not hamper its modular structure. This structure is defined by the allocation of objects among a set of schemas. Therefore, we introduce an approach and a tool to automatically predict the correct placement of new objects. We extract dependencies among various types of objects (database types, sequences, tables, procedures, functions, packages, and views) that are already placed in schemas. These dependencies are used for training an artificial neural network model, which is then used for prediction. Our industrial case studies show that our approach can reach an accuracy of 89%, whereas the baseline approach using coupling and cohesion metrics can reach 57.4% accuracy at most. There are already several techniques and tools for supporting the second task of analyzing the impact of changes in the data model on the source code. However, they fall short to analyze dynamically created SQL statements, queries on multiple tables, and other types

of statements that allow data manipulation in PL/SQL, which is a commonly used language for developing data-tier software. We introduce techniques and a tool to parse both the data model and the source code (i.e., PL/SQL functions and procedures) taking part in all the schemas of a given database. Then, a dependency model is created based on queries and manipulation of database tables. Unlike prior studies, our tool can analyze queries that are created dynamically and that involve multiple tables as well as PL/SQL-specific features. We use the derived dependency model to estimate effort for two different common refactoring types on real systems. We observe high consistency between the automated estimations and manual estimations. The third task is concerned with the impact of changing tables on other tables of the same database. There are only a few studies that focus on this concern. Moreover, these studies consider the impact of deletion and modification of columns in database tables only. To address this limitation, we introduce an approach and a tool for automatically detecting the impact of data model extensions on the data model itself. We employ Siamese networks to detect similarities among database tables and such, to learn implicit relations among them. Table similarities are used as the basis for identifying potential impact. We develop another tool as the baseline, which employs the cosine similarity metric to measure similarity among database tables. Results obtained with Siamese networks turned out to be better than the baseline, achieving the mean F1 score of 96.1%.

ÖZETÇE

Yazılımların veri katmanları, kurumsal sistemlerin veri modeli ile iş mantığını içermekte olup, bu katmanlar uzun süreli bakıma ihtiyaç duymaktadır. Sistemlerin kullanıcı arayüzü tamamen değiştirilse de, veri katmanı yazılımı genellikle uzun yıllar boyunca geliştirilmeye devam etmektedir. Yazılım hakkında kapsamlı bilgiye sahip olan alan uzmanlarının sayısı zamanla azalmaktadır. Bu durum yazılıma yeni özellikler eklemeyi veya mevcutta olan yazılımı değiştirmeyi özellikle yeterli tecrübeye sahip olmayan geliştiriciler için zorlaştırmakta ve daha fazla efor gerektiren bir çalışma haline gerektirmektedir. Bu durum aynı zamanda hata eğilimini de arttırmaktadır. Bu tezde, veri katmanı yazılımı için otomatik bakım desteği sağlamak üzere teknikler ve araçlar tanıtılmaktadır. Bu teknikler ve araçlar, özellikle üç zorlu bakım ihtiyacı için eforu ve hata sayısını azaltmayı amaçlamaktadır: *i)* veri katmanı yazılımında prosedür gibi yeni bir nesnenin doğru modül ile ilişkili olarak konumlandırılması, *ii)* veritabanı tablolarının değiştirilmesinin yazılım modülleri üzerindeki etkisinin belirlenmesi, ve *iii)* veritabanı tablolarının genişletilmesinin aynı veritabanındaki diğer tablolar üzerindeki etkisinin belirlenmesi. Yukarıda listelenen ilk bakım ihtiyacı önemlidir, çünkü veri katmanı yazılımına yeni bir nesne eklemek modüler yapıyı bozmamalıdır. Bu yapı, nesnelerin veritabanı şemaları arasında doğru konumlandırılmasıyla sağlanmaktadır. Bu bakım ihtiyacı ile ilgili olarak, yeni nesnelerin doğru yerleşimini otomatik olarak tahmin etmeyi amaçlayan bir yaklaşım ve araç sunmaktayız. Şemalar üzerinde yer alan çeşitli nesne türleri (veri tipleri, sekans, tablolar, prosedürler, fonksiyonlar, paketler ve görünümler) arasındaki bağımlılıkları çıkarıp, bu bağımlılıkları daha sonra doğru modülün tahmini için kullanılan bir yapay sinir ağı modelinin eğitimi için kullanmaktayız. Endüstriyel örnek vaka çalışmalarımız, yaklaşımımızın 89% düzeyine kadar bir doğruluğa ulaşabileceğini, buna karşın

birleştirme ve uyum metriklerini temel alan yaklaşımın en fazla 57.4% düzeyinde bir doğruluğa ulaşabildiğini göstermektedir. Veri modelindeki değişikliklerin kaynak kodu üzerindeki etkisini analiz etmeye yönelik ikinci bakım ihtiyacını desteklemek için halihazırda birkaç teknik ve araç bulunmaktadır. Ancak, bu teknik ve araçlar dinamik olarak oluşturulmuş SQL deyimlerini, birden çok tablo ile bağlantılı sorguları ve veri katmanı yazılımı geliştirmek için yaygın olarak kullanılan bir dil olan PL/SQL'de veri işlemeye izin veren diğer deyim türlerini analiz etmekte yetersiz kalmaktadırlar. Bu nedenle belirli bir veritabanının tüm şemalarında yer alan hem veri modelini hem de kaynak kodunu (PL/SQL fonksiyonları ve prosedürleri vb.) elde edip, ardından sorgulara ve veritabanı tablolarını güncelleyen komutlara dayalı bir bağımlılık modeli oluşturan bir araç sunmaktayız. Önceki çalışmalardan farklı olarak, aracımız dinamik olarak oluşturulan ve birden çok tablonun yanı sıra PL/SQL'e özgü özellikleri içeren sorguları analiz edebilmektedir. Gerçek bir sistem üzerinde iki farklı yeniden yapılandırma çalışması ile ilgili gerekli olan eforu tahmin etmek için araç tarafından oluşturulan bağımlılık modelini kullanmaktayız. Bağımlılık modeli kullanılarak hesaplanan otomatik tahminler ile alan uzmanları tarafından yapılan manuel tahminler arasında yüksek tutarlılık gözlemlenmektedir. Üçüncü bakım ihtiyacı, genişletilen tabloların aynı veritabanındaki diğer tablolar üzerindeki etkisiyle ilgilidir. Bu ihtiyaca odaklanan sadece birkaç çalışma bulunmaktadır. Ayrıca, bu çalışmalar yalnızca veritabanı tablolarındaki sütunların silinmesi ve değiştirilmesinin etkisini dikkate almaktadır. Bu çalışmada, veri tabanı tablolarının genişletilmesinin veri modelinde hali hazırda mevcut olan diğer tablolar üzerindeki olası etkisini otomatik olarak tespit eden bir yaklaşım ve araç sunulmaktadır. Veritabanı tablolarının benzerliklerini tespit etmek, aralarındaki örtük ilişkileri öğrenmek için Siyam ağlarını kullanmaktayız. Temel bir yöntem olarak, veritabanı tabloları arasındaki benzerliği ölçmek için kosinüs benzerlik metriğini kullanan başka bir araç daha sağlıyoruz. Siyam ağlarıyla 96.1% F1 skoruna ulaşılarak, temel yönteme göre daha iyi sonuç elde edildiği görülmektedir.

ACKNOWLEDGEMENTS

First of all, I would like to thank Assoc. Prof. Hasan Sözer for his support, guidance, and motivation throughout the thesis work and especially for helping me constantly. I also want to thank him for his contributions to my education during my graduate years.

Special thanks to Assist. Prof. M. Furkan Kıraç and Prof. O. Örsan Özener for their support, feedback, and guidance during the monitoring process of this thesis work.

I thank Assoc. Prof. Mehmet S. Aktaş, Assoc. Prof. Kamer Kaya, Assist. Prof. M. Furkan Kıraç, and Prof. O. Örsan Özener for participating in my thesis jury and giving me feedback.

I also want to thank my friends Dr. Demet Ayvaz, Assist. Prof. Selami Bağrıyanık, Hüseyin Doğanyılmaz, and Coşar Baykal for their support, motivation, and the unique friendship that we shared throughout these years.

This work is supported by The Scientific and Research Council of Turkey with grant number 120E488. I would like to thank Turkcell and Paycell for the great opportunity and support for pursuing my academic studies.

Finally, I should express my gratefulness to my family, especially to my wife Gönül Yılmaz Ersoy for her love, support, self sacrifice, and endless-trust in me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Glossary

ANN Artificial Neural Network. 15, 22, 24–27, 33, 35, 42

CMS Campaign Management System. 26, 29, 31–33, 35, 38, 42, 43, 89

CRM Customer Relationship Management. 4, 26, 29–31, 33, 35, 38, 42, 43, 62, 78, 89

DML Data Manipulation Language. 9, 15, 16, 53, 55, 56, 59, 66, 92

PL/SQL Procedural Language/Structured Query Language. 4–10, 12–15, 20–22, 26, 42, 46, 48, 50, 52–54, 62, 63, 65, 66, 68, 69, 86, 87

SN Siamese Network. 18, 19, 73, 78, 80, 85

SQL Structured Query Language. 5, 8, 11, 12, 23, 46, 50, 53–56, 59, 61, 66–69, 92

CHAPTER I

INTRODUCTION

There exist large-scale legacy software systems that were developed decades ago. Since change is continual, software systems continuously evolve [1]. As a result, legacy systems are still subject to maintenance efforts, which account for the highest fraction of costs in the software development life-cycle [2]. They often need to be modified and/or extended with new features. These changes are required for satisfying business needs and incorporating contemporary functionality or technology. Software developers must sustain a modular architecture design [3] for keeping the maintainability of the system high while applying the necessary alterations. Unfortunately, these alterations can hamper the cohesion of architectural modules and the initial modular structure, leading to *architecture erosion* [4]. Moreover, they can lead to unforeseen side-effects. Changes that are applied to a particular module can lead to unanticipated effects on other modules of the system. These effects can lead to inconsistencies that can be exposed to users as failures. These are general and long-studied problems that are faced during software maintenance [5]. The same problems are relevant for the maintenance of databases. Database schemas are also subject to modifications and extensions. Extensions must be applied in a way that they do not lead to architecture erosion. On the other hand, modifications on database elements should not be bug-inducing due to their impact on the other elements of the same database or the software modules that rely on the modified elements. In this thesis, we focus on two types of maintenance activities that are related to data-tier software: *i)* adding a new object (procedure, function, or package) to the database, and *ii)* modifying an existing table in the database, which can involve the deletion, modification, and extension of its columns. We address three challenges related to these two types of

alterations. The first challenge is related to the first type of alteration, where we need to locate the most suitable schema for the new object. The second challenge is related to the second type of alteration, where we need to evaluate the impact of table modifications on software modules. In particular, we need to pinpoint locations in the source code that are affected by these modifications. The last challenge is evaluating the impact of table extensions on other tables of the same database. In this thesis, we introduce methods and tools for addressing these challenges for data-tier software. These challenges are mainly addressed by domain experts in state-of-the-practice. These experts have the knowledge about the potential impact of changes and where to integrate new features. However, the number of such experts with extensive knowledge about the overall system diminishes in time. Furthermore, the high turnover rate makes it difficult nowadays to keep domain experts for all companies. This trend is also known as *great resignation* [6]. On the other hand, it is not possible for new developers to grasp the design of a large-scale system without getting help from experts. As a result, maintenance efforts together with the number of design flaws and errors introduced during maintenance tend to increase. We elaborate on three challenges that are addressed in this thesis to overcome this problem in the following.

A major cause of the first challenge regarding extensions of schemas with new objects is the lack of precise, up-to-date documentation of software architecture [7]. Initially created documentation is not always precise enough and/or it is usually not updated consistently with respect to the evolving system. Software architecture recovery [8] or reconstruction [9] techniques can be used to recover architectural documentation for a software system. These techniques mainly aim at clustering various types of objects regarding the system such as functions, source files, or classes in object-oriented programs. The clustering is performed based on various types of inter-dependencies [8] among them such as function calls [10, 11], shared data [12], common usage of database objects [13] and common usage scenarios [14]. The objective of clustering is to increase cohesion within clusters

while decreasing coupling among them [15]. Such a clustering provides valuable insight regarding the overall organization of the objects. However, it can not automatically provide an answer to queries about where to incorporate a new feature/object. One has to manually analyze and interpret the clustering results to identify where the new object fits. Moreover, unsupervised learning techniques like clustering techniques can reach a limited accuracy level [8]. Therefore, extending the system with a new object is still problematic, especially for large-scale software systems.

The second challenge is related to the impact of changes in the data model. Most of the previous studies have focused on the impact of changes in application software rather than data-tier software. In general, database refactoring [16, 17] has received less attention in the literature [18]. However, many of the large, heterogeneous, and highly complex legacy systems are data-intensive systems [19]. Co-evolution of database schemas and application source code is inevitable for these systems [20, 21]. Therefore, change impact analysis activities should also explicitly consider data-intensive systems and databases. Frequent evolution of database schemas and side effects caused by this evolution have been acknowledged [20, 22]. As such, there are several studies [22, 23, 24, 25, 26] aiming at analyzing the impact of changes in database models on the source code. However, they have limitations that we address in this thesis.

The third challenge is related to the impact of data model extensions. A recently published industrial case study [27] investigates problems that are encountered during the maintenance of relational database systems. Six problems are identified in that study in total. Two of these are specified as follows: *i)* difficulty in analyzing and visualizing dependencies among database elements, and *ii)* difficulty in evaluating the impact of changes applied on database models. There exist tools and methods [22, 23, 24, 25, 26] developed particularly for the latter. However, they focus on the impact of changes on the source code rather than the data model itself. Those studies [28] and tools [29] that consider the impact on the

data model, and evaluate the impact of deletion and modification of columns in database tables only. The impact of extensions of database tables on other tables has not been considered in the scope of change impact analysis.

We applied the so-called industry-as-laboratory approach [30] by using real systems from the telecommunication domain as case studies. In the following, we clarify the scope of our studies and provide motivation in this context. Then, we introduce our research questions. We conclude the chapter by summarizing our contributions and providing an overview regarding the organization of the thesis.

1.1 Thesis Scope and Motivation

The main goal of this thesis is to provide tool support for the maintenance of data-tier software. We apply our techniques and tools to real business applications in the context of industrial case studies. These business applications are owned by a technology company, Turkcell[1]. Turkcell is the largest telecom operator in Türkiye. As an integrated communications and technology services company, it offers its customers voice, data, TV services, and value-added individual and corporate services over mobile and fixed networks. Turkcell has more than 35 million customers. To serve these customers, there are hundreds of applications being developed and maintained by thousands of employees. The majority of these applications are developed as data-tier software by using the PL/SQL language [31]. These applications serve various business domains such as CRM (Customer Relationship Management) and Billing. In the scope of this thesis, we focus on three common challenges for the maintenance of data-tier software developed with the PL/SQL language as elaborated in the following.

i) Provide guidance in extending PL/SQL programs: Databases are just considered as data stores in many applications for which the module concept is associated with packages, components, or similar types of other entities that encapsulate various parts of the source code. This is not the case for PL/SQL programs, which

[1] `http://www.turkcell.com.tr`

are commonly used in the telecommunication, insurance, and banking domains. Hereby, business processes are mainly managed via stored procedures that are organized around a number of schemas in the database. Hence, schemas represent the gross-level modules of a PL/SQL program and its modular decomposition is defined by the distribution of objects among a set of schemas. Correct placement of new objects in these schemas is important for maintaining a modular structure. This turns out to be a challenge and it requires expertise when extending large-scale, legacy PL/SQL programs.

ii) Data model change impact analysis on PL/SQL programs: The majority of the existing studies so far have focused on the impact of changes applied in source code [32]. Data models also frequently change. Data-tier software is usually refactored to improve several quality attributes. In addition, tables are altered in relational databases to address emerging and changing business needs. These changes have an impact on the source code that relies on the corresponding tables. Unforeseen side effects can lead to failures. To prevent these side effects and/or to estimate the required refactoring effort beforehand, change impact analysis is necessary. It is both effort-consuming and error-prone to perform this analysis manually. There exist several techniques and tools [24, 33, 34, 35, 36, 37, 28, 38, 39] that are used for analyzing the impact of changes in the data model on the source code. They mainly analyze SQL statements in the source code to automatically identify database elements that are being accessed and such might have an impact in case they are modified. However, they fall short to analyze dynamically created SQL statements and other types of statements that allow data manipulation in PL/SQL programs.

iii) Data model extension impact analysis: Changes in the data model impact the data-tier software that relies on this model. On the other hand, changing a database table can also have an impact on other tables of the same database. There are only a few studies [28, 29] that focus on this aspect of the problem. Moreover, these studies consider the impact of deletion and modification of columns

in database tables only. The extension of tables has been neglected as a relevant type of change although it might also have an impact on the system. For instance, two tables must be kept consistent after extensions if one of them is responsible for saving the history of records for the other one. Ensuring this consistency is not trivial when there is a lack of explicit relations among such tables.

In the following section, we define our research questions aligned with these observations.

1.2 Research Questions

This thesis aims at addressing the challenges that are listed in the previous section. We define three main research questions that focus our research for this aim.

> ***RQ1: Is it possible to automatically predict the correct placement of new PL/SQL procedures in schemas?***
>
> ***RQ2: Is it possible to automatically detect the impact of changes in data model on PL/SQL programs?***
>
> ***RQ3: Is it possible to automatically detect the impact of data model extensions on the data model?***

Our first goal is to maintain the modular structure of a PL/SQL program while extending it and as such, to prevent architecture erosion. The modular structure is defined by the clustering of PL/SQL procedures in a set of schemas. A newly introduced procedure must be placed in one of these schemas. Domain experts can accurately identify the most appropriate schema for a new procedure. However, the required expertise is a scarce resource that is hardly available. Therefore, reliable tool support for automating this task would be very valuable. We defined *RQ1* based on this observation.

Our second goal is to prevent error-prone modifications on the data model, which might have unforeseen side effects on the source code that rely on this model. There are tools that can identify dependencies of source code elements

on database tables by analyzing queries hard-coded in the source code. However, PL/SQL programs employ advanced data manipulation mechanisms like dynamic query construction. Existing tools cannot identify dependencies formed via these mechanisms. *RQ2* is defined for investigating to what extent this limitation can be addressed.

Our third goal is to prevent error-prone modifications on the data model, which might have unforeseen side effects on the data model itself. This problem is less explored compared to the change impact analysis focusing on the side effects on source code. In particular, only the deletion and modification of table columns have been considered as relevant types of changes. The impact of the extension of tables on other tables is ignored. *RQ3* is defined to fulfill this gap.

1.3 Thesis Contributions and Overview

To answer *RQ1*, we introduced an approach and a tool to automatically predict the correct placement of new objects among a set of schemas in PL/SQL programs. We extracted dependencies among various types of objects (database types, sequences, tables, procedures, functions, packages, and views) that take place in PL/SQL programs. We represented these dependencies in the form of dependency matrices, where columns and rows stand for objects and values in cells indicate the existence of dependencies between the corresponding pairs of objects or lack thereof. Dependency matrices are constructed for different subsets of objects. For instance, one of these matrices captures dependencies on table and views only, whereas another one includes dependencies on procedure and functions as well. These dependency matrices were used as input for our tool, which employs artificial neural networks. We performed two case studies for two different real applications to evaluate the effectiveness of our approach. Results showed that our approach can reach an accuracy of 89%, whereas the baseline approach using coupling and cohesion metrics can reach 57.4% accuracy at most.

We developed a tool for addressing *RQ2*. The tool captures all the schemas of

a given database and the data model regarding each schema including its tables and views. It also parses the source code of all the objects (i.e., PL/SQL functions and procedures) taking part in each schema. Then, a dependency model is created based on the data model and an analysis of SQL queries in these objects. Unlike prior studies, our tool can analyze queries that are created dynamically and that involve multiple tables as well as PL/SQL-specific features. Finally, industrial case studies are performed to estimate effort for two different common refactoring types by using dependency analysis results. We observed high consistency between the automated estimations and manual estimations.

To address *RQ3*, we introduced an approach and a tool for automatically detecting the impact of data model extensions on the data model itself. We employed Siamese networks to detect similarities among database tables and such, to learn implicit relations among them. Table similarities are used as the basis for identifying potential impact. We developed another tool [40] as the baseline, which employs the cosine similarity metric to measure similarity among database tables. Results obtained with a Siamese network model turned out to be better than the baseline. It achieves the mean F1 score of 96.1%.

This thesis is organized as follows.

Chapter 2 provides background information on Oracle database system objects, PL/SQL programs, artificial neural networks, other classification techniques, and Siamese networks.

Chapter 3 focuses on the automated prediction of correct placement of new PL/SQL programs. It introduces an approach, a tool, and its evaluation. This chapter is a revised version of the work described in [41].

Chapter 4 introduces a tool that automatically provides impact analysis on PL/SQL programs in case the underlying data model changes. This chapter is a revised version of the work described in [42].

Chapter 5 explains an approach, a tool, and its evaluation for automatically analyzing the impact of data model extensions on the data model itself. This

chapter is a revised and extended version of the work described in [40].

Chapter 6 provides the conclusions. The evaluations and discussions for the particular contributions are provided in the corresponding chapters.

Appendix A provides the top ten accuracy values achieved with the approach that is described in Chapter 3.

Appendix B explains DML (Data manipulation language) types with a special focus on those used exclusively in PL/SQL programs. Analysis of these special cases is covered by the tool that is described in Chapter 4.

CHAPTER II

BACKGROUND

In this chapter, we first provide background information on Oracle[1] database objects. Then, we discuss PL/SQL programs with example code snippets. These programs directly work on an Oracle Database Management System and they are used for creating and/or using database objects. Finally, we provide background information about artificial neural networks, other classification techniques, and siamese networks.

2.1 Oracle Database Objects

There are several types of objects in an Oracle database. These are named as *tables, views, procedures, functions, packages, types, sequences*. Some of these objects (e.g., tables) hold data, some of them (e.g., views) just provide declarative definitions, whereas some of them (e.g., procedures) include business logic. Business logic is implemented using the PL/SQL language, which is introduced in the next subsection. All of these objects are organized in so-called *schemas*. They comprise a collection of objects and define the gross-level organization of the system. Therefore, we consider schemas as architectural modules in our context. Various types of objects and their brief descriptions are listed in the following.

- **Table:** unit of data storage.
- **Procedure, Function, Package:** units of business logic implemented in PL/SQL (described in the next section).
- **Type:** representation of real-world entities each defined as a set of attributes in primitive types such as *integer* and *varchar*.

[1] `http://www.oracle.com`

- **Sequence:** used for generation of numbers such as unique keys.
- **View:** a stored SQL query that collects data from one or more tables (See Listing 2.1 as an example).

All these objects are created and deleted via the `CREATE` and `DROP` SQL (Structured Query Language) commands. For instance a table can be created and deleted by executing the "`CREATE TABLE <table name> ...`" and "`DROP TABLE <table name>`" commands, respectively. Listing 2.1 shows an example code snippet used for creating a *view* object named *employees_view* (Line 1). Hereby, two tables named *employees* and *departments* are joined (Line 3). The resulting records are filtered for matching the *department_id* attribute (Line 4). Only 4 columns of the remaining records are included as listed in Line 2.

Listing 2.1: A sample *view* object.

```
1  CREATE VIEW employees_view AS
2  SELECT e.first_name, e.last_name, e.salary, d.↩
       department_name
3  FROM employees e, departments d
4  WHERE e.department_id = d.department_id
```

Objects can make use of each other, which creates dependencies among them. Special objects called *Oracle data dictionary objects* store information regarding these dependencies. We developed a custom SQL command to extract these dependencies as provided in Listing 2.2. Hereby, all objects are selected with dependencies among them (Lines 1-9), while system dependencies (Lines 10-11) and duplicates (Lines 12-13) are filtered out. For instance, we can see for the view object in Listing 2.1 that there are dependencies to two tables, namely, *employees* and *departments*. Such dependencies are automatically extracted with the code snippet shared in Listing 2.2.

Listing 2.2: An SQL query that is used for extracting dependencies among the Oracle database objects within a system.

```
1  SELECT a.object_id, a.object_type, b.owner ref_owner, b.↩
       object_type ref_type, b.object_name ref_name, b.↩
       object_id ref_id, b.status ref_status FROM SYS.↩
       ALL_OBJECTS a, SYS.ALL_OBJECTS b,
2  (SELECT object_id, referenced_object_id
3       FROM   (select object_id, referenced_object_id
4       FROM   PUBLIC_DEPENDENCY
5       WHERE  referenced_object_id <> object_id) pd
6       START WITH  object_id = iobject_id
7       CONNECT BY NOCYCLE PRIOR referenced_object_id =  ↩
          object_id) c
8  WHERE a.object_id = c.object_id
9  and   b.object_id = c.referenced_object_id
10 and   a.owner not in ('SYS', 'SYSTEM')
11 and   b.owner not in ('SYS', 'SYSTEM')
12 and   a.object_name <> 'DUAL'
13 and   b.object_type != 'SYNONYM';
```

2.2 PL/SQL Programs

PL/SQL (Procedural Language/Structured Query Language) is a 3rd generation language that puts together procedural language features with that of SQL such that declarative SQL statements can be intermixed with imperative code [43]. It is used for developing *procedures*, *functions* and *packages* in Oracle databases to implement the business logic of an enterprise application. All these objects can be called by any system which connects to the database.

The only difference between procedures and functions is that functions return a value, whereas procedures do not. For simplicity, we will refer to both functions

and procedures as procedures in the rest of this thesis. Procedures follow the basic PL/SQL block structure, which consists of 3 main parts as presented in Listing 2.3.

The declarative part (Lines 1-4) of the procedure block identifies variables that are used in the application logic. The executable part, starting with `BEGIN` and ending with `END` keywords (Lines 5-15), contains the business logic. This part is mandatory. An optional exception-handling part, starting with `EXCEPTION` (Lines 16-23), handles possible error conditions that may be raised in the executable part of the block.

Listing 2.3: A sample PL/SQL procedure [13].

```
1  PROCEDURE AddRatio(p_id IN NUMBER) IS
2   v_sales_num t_result.sales_num%TYPE;
3   v_total_num t_result.total_num%TYPE;
4   v_ratio_num NUMBER;
5  BEGIN
6   SELECT sales_num,total_num
7       INTO v_sales_num,v_total_num
8      FROM t_result WHERE result_id = p_id;
9   v_ratio_num := v_sales_num/v_total_num;
10  IF v_ratio_num > 25 THEN
11     INSERT INTO t_comm_tab
12        VALUES (p_id,v_ratio_num);
13  END IF;
14  COMMIT;
15 END
16 EXCEPTION
17  WHEN ZERO_DIVIDE THEN
18     INSERT INTO t_comp_tab
19     VALUES (p_id,-1);
20     COMMIT;
21  WHEN OTHERS THEN
22     ROLLBACK;
23 END;
```

Enterprise-level applications usually consist of thousands of procedures. The package concept was introduced as a collection of procedures to organize them. Oracle suggests the use of packages to group related procedures for increased modularity. The use of packages also improves the performance since the whole package can be loaded in memory at once. In fact, packages have also some additional features which are not possessed by procedures. However, we will not

further dive into these details.

Packages have two parts: *i)* specification, and *ii)* body. The specification part stores the interface of the package. It declares the variables, constants, exceptions, and procedures that can be called from outside of the package [43]. The package body contains the code that implements both the public and private procedures.

The package concept can be considered to be equivalent to a module in a PL/SQL program [13] since it is used for organizing procedures. In this thesis, we associate the schema concept with an architectural module. Schemas provide a higher level of abstraction just like the Java module system [44] introduced over packages. They might be aligned with the physical architecture in some database applications. However, they are also used for the logical decomposition of objects in an Oracle database. They can organize all the objects including both packages and standalone procedures. Schemas of the subject systems that we used in our thesis reflect a logical decomposition.

2.3 Artificial Neural Networks

An *artificial neural network (ANN)* is a mathematical model and information processing system that is inspired by biological nervous systems. It simulates specific behaviors of these systems. ANN is composed of *neurons* that are connected together with connection links. Information is processed by neurons and passed to the other neurons via these links, each of which is associated with a weight. Each neuron applies an activation function. This function basically collects the information transferred via its (weighted) incoming links, adds a bias, and then possibly transfers further information via outgoing links. There are several types of ANNs. In this work, we employed a *multilayer perceptron (MLP)* feed-forward DML that is trained with a back propagation algorithm [45, 46].

A typical MLP consists of at least 3 layers. For instance, Figure 1 depicts two DML models, one with 4 layers on the left-hand side and the other with 3 layers on the right-hand side. The first layer is called the input layer where external

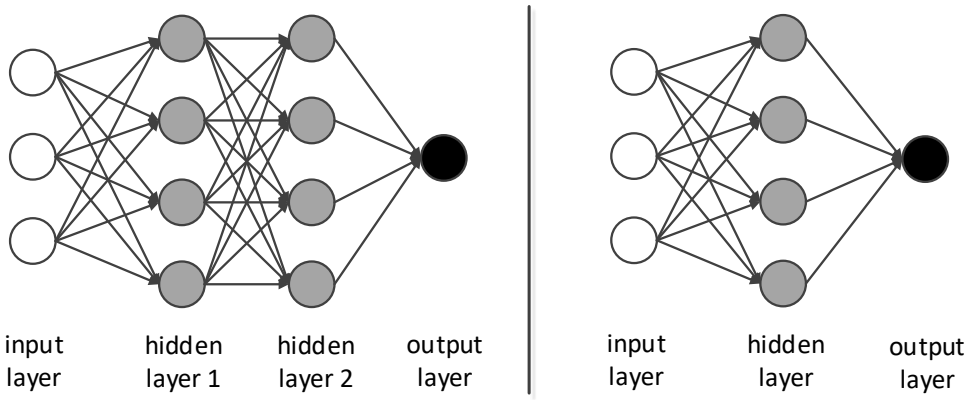


Figure 1: Sample neural network models with two hidden layers on the left hand side and one hidden layer on the right hand side.

information is fed. The last layer is called the output layer that provides the outcome. There are one or more layers in-between that are called the hidden layers. In each layer, there are a finite set of neurons. The number of neurons in the first and last layers is related to the size of the input and the desired output, respectively. On the other hand, the number of hidden layers and the number of neurons in each of these layers constitute a design parameter for an DML. Another design choice is regarding the activation function to be employed in neurons.

DML supports a type of supervised learning process. It must be trained before being used for a classification problem. Values to be assigned to the connection weights are determined during the training process. There exist various algorithms that can be used in this process. We employed the commonly used back propagation algorithm [45] for this purpose. This algorithm compares the output of the network with respect to the expected output for a given input as part of the training data. An error is calculated as a difference between the two and this error is propagated back through the connection links. Connection weights are updated to minimize the error [46].

2.4 Other Classification Techniques

Logistic regression [47, 48] is one of the most popular methods applied for machine learning classification. It is basically a predictive analysis used to describe relationships between a binary dependent variable and one or more independent variables. In a logistic model, the output is always binary such as pass/fail, 0/1, true/false while input independent variables can be either binary or continuous. Logistic regression has also extended versions to handle more than two outputs, ordinal dependent variables, and correlations among dependent variables.

k-Nearest Neighbors [49] is another classification method. Hereby, all the pairwise distances are computed for the items as part of the dataset first. Distance metrics such as Minkowski or Euclidean are used for quantifying such a distance [49]. Then classification is performed based on the distance of each item pair in the dataset. The k hyperparameter is the most important parameter which is set as usually less than 20 and 5 as default. This hyperparameter is defined to focus on the top k similar instances in the dataset.

Decision Tree [50] is one of the most commonly used predictive modeling tools used in statistics, data mining, and machine learning. The goal of a decision tree model is to predict the value of a target variable based on several input variables. Tree nodes are decision points consuming one of many input variables, while the leaf nodes represent the target variables. The type of the constructed tree depends on the type of target variables. The tree model is called a classification tree if the target variables take discrete values. Regression trees, on the other hand, have target variables, which can take continuous values.

Support vector machines (SVMs) [51, 52] are supervised learning models which are used for both classification and regression. The goal for SVMs is to find a hyperplane or set of hyperplanes, with the largest distance to the closest data points, namely the support vectors. By definition, SVMs are maximum-margin classifiers. SVMs can perform efficient non-linear classification by transforming their input space to a higher dimensional space by applying transformation functions.

Random forest [53] is an ensemble estimator which builds multiple tree models on subsets of a dataset. The final prediction is based on majority voting, where the mostly voted result is chosen. Each subset has the same size as the original dataset; however, samples are drawn with replacement.

2.5 Siamese Networks

Siamese Networks (SN) [54] employ artificial neural network architectures that contain two or more identical sub-networks (See Figure 2). These sub-networks have the same configuration with the same parameters and weights. Parameter updating is mirrored between them as well. Such an architecture is used for finding the similarity of inputs by comparing their feature vectors.

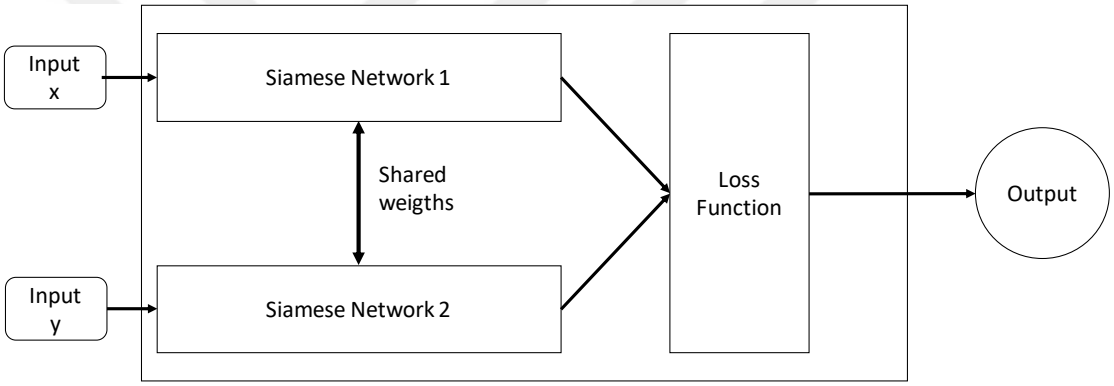


Figure 2: An abstract view of a SN architecture.

Traditional neural network models have two basic limitations. First, a neural network model learns to predict two or more classes for a given instance in the dataset. The model is trained with a set of instances for which the classes are defined. We have to update this model and retrain it in case a new class is added to the set of possible classes. Second, neural network models - especially deep neural networks - need a large volume of data for training. SN are not subject to these limitations. First, SN learn by using loss/similarity functions that make it possible to use an existing model without training it again in case of the introduction of a new class. We train an SN model to predict if any given two inputs are the same/similar or not. Hereby, loss functions measure the accuracy of prediction. New classes of data can be incorporated without training the model

from scratch. Mainly, there are two popular loss functions these are contrastive loss [55] and triplet loss [56] functions. It is possible to define custom loss functions as well. Second, SN models can provide accurate results without excessive training data [57]. These properties make SN a powerful tool for predicting similarities among database tables and as such supporting change impact analysis regarding table extensions. Applications of SN range from signature verification to image verification. However, they have not been employed for change impact analysis or in the context of software maintenance in general to the best of our knowledge. We summarize related studies in the following section.

In the following chapters, we explain our studies in detail and discuss the results.

CHAPTER III

GUIDANCE IN EXTENDING PL/SQL PROGRAMS

In this chapter, we propose a novel approach for automatically predicting the correct placement of a new object among existing architectural modules of PL/SQL programs. PL/SQL programs are composed of procedures, functions, views, and packages that are highly coupled with a database. Databases are just considered as data stores in many applications for which the module concept is associated with packages, components, or similar types of other entities that encapsulate various parts of the source code. This is not the case for PL/SQL programs, which are commonly used in the telecommunications domain. Hereby, business processes are mainly managed via stored procedures that are organized around a number of schemas in the database. Hence, schemas represent architectural modules in our context, whereas procedures, functions, views, and packages are objects of the system. We aim at capturing architectural knowledge regarding a PL/SQL program based on coupling among these objects. An artificial neural network tries to learn an abstract representation of a set of provided features in terms of its hidden layers and their interconnections. We consider such a network structure a good fit for capturing an abstraction of structural coupling to reason about the organization of objects in schemas. Therefore, we represent coupling among objects with artificial neural networks. We assume that the system has a stable architecture, where there are no architecture-level disruptive changes. In our context, this means that the addition and removal of schemas are unlikely. We train a network based on features extracted from the initial version of the source code that is assumed to represent the intended architecture. Then, given a new software object and the list of other software and/or database objects it uses, the network can predict the module (i.e., schema), where the object should be included. Thus, we adopt a

classification rather than a clustering approach to address the architecture erosion problem. The trained artificial neural network implicitly represents the high-level modular structure (i.e., architecture design) of the software system. To the best of our knowledge, this is the first study to utilize such a representation for supporting software maintenance.

We performed industrial case studies with two subject systems from the telecommunications domain. Both of these are PL/SQL programs each of which contains thousands of procedures and database tables. We trained artificial neural network models and evaluated the accuracy of predictions by using 10-fold cross-validation [58]. We also evaluated and compared the performance of other prominent classification techniques in their default configurations. These techniques include Logistic Regression, k-Nearest Neighbors, Decision Tree, Support Vector Machine, and Random Forest. Hereby, our goal is not to exhaustively explore the entire set of algorithm alternatives and their configurations. Our hypothesis is that an artificial neural network model would be a good fit for capturing structural coupling among software and database elements. We optimized our neural network model after observing that it also empirically outperforms the other alternatives in their default configurations. We experimented with various types of dependencies to train and optimize the number of hidden neurons and other hyperparameters of the network. Hereby, we considered varied subsets of dependencies among the database types, sequences, tables, procedures, functions, packages, and views. We showed that the accuracy of our approach reaches 86.7% and 89% for the two subject systems in our case studies. One can question the accuracy of our approach compared with respect to a simpler one, which uses a cohesion metric and object couplings only. Therefore, we also conceived a baseline approach, where an object is assumed to belong to the schema that includes most of the other interdependent objects. We observed that this approach reaches 55.5% and 57.4% accuracy levels for the same subject systems, respectively.

The remainder of this chapter is organized as follows. In the following section,

we present the overall approach and prediction model, which is illustrated in Section 3.2, in the context of the industrial case studies. We discuss the results in Section 3.2.5. Finally, we summarize the related studies and our contributions in Section 3.3.

3.1 Overall Approach

The overall approach, which involves 3 main steps and several sub-steps, is depicted in Figure 3. The 3 main steps are related to the preparation of a dataset and the ANN model (1), performing the prediction task for a new object (2), and extending the dataset for improving the model (3). Hereby, all schemas and objects of the selected enterprise application are collected from the database first (1.1). Then, dependencies among these objects are extracted (1.2). A dataset is prepared based on the extracted dependencies (1.3). This dataset is used for optimizing the design parameters of the ANN including the number of hidden layers, the number of neurons in each layer, and the activation function to be used by neurons (1.4). The ANN models for each hyperparameter candidate are trained using the dataset (1.4). The optimized ANN is used for performing classification and as such, predicting the schema that is relevant for a given new object (2). Once approved by a domain expert, the new object and the corresponding schema can be included in the dataset to further improve the performance of the ANN (3). We explain each step of the approach in more detail in the following.

3.1.1 Schema Object Collection

PL/SQL programs run on an Oracle database, which comprises various types of objects as described in Chapter 2. These objects are organized in a number of schemas (See Chapter 2), which we consider as the gross-level modular decomposition of the enterprise application. The first step of the approach involves the collection of these objects and schemas.

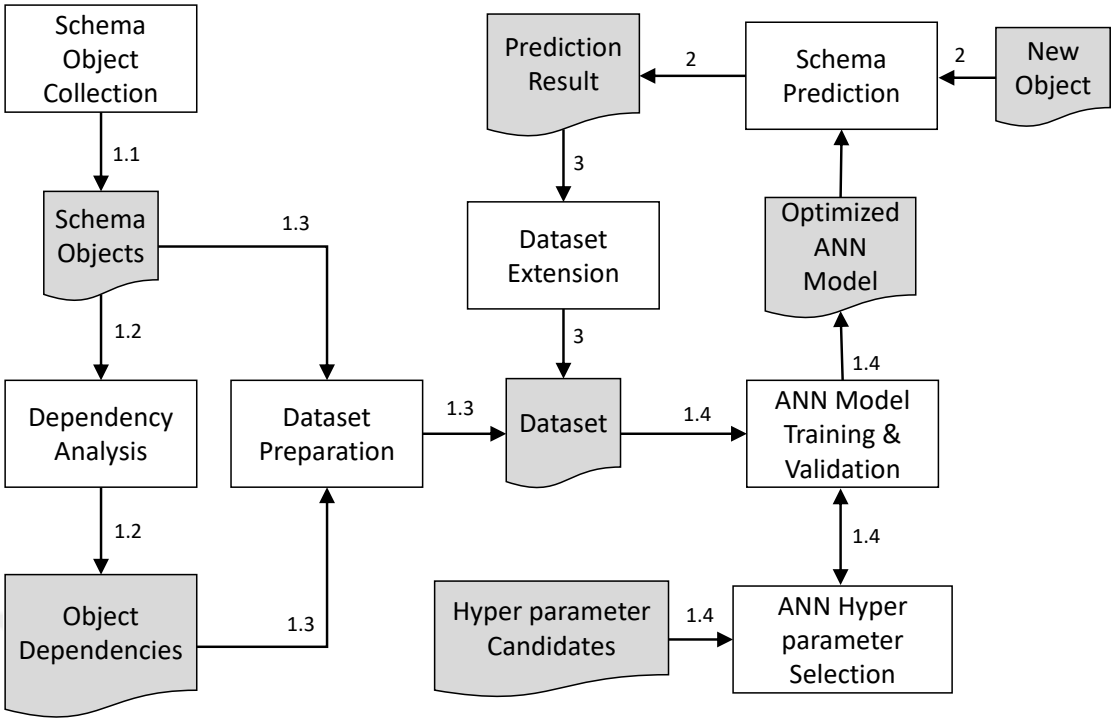


Figure 3: The overall process.

3.1.2 Dependency Analysis

In the second step, we use data dictionary views that are provided by Oracle databases to collect information regarding which objects depend on which other objects. Object dependencies are extracted with the SQL query presented in Listing 2.2. Dependencies on commonly used utility objects (stored as part of `SYS` and `SYSTEM` schemas) and libraries (e.g., Java standard library) are filtered out during this process.

3.1.3 Dataset Preparation

Dataset preparation involves two activities: data acquisition and data preprocessing. Data acquisition is responsible for collecting raw data from one or more information sources. In our approach, this activity is already realized by the *Dependency Analysis* step, where dependencies among objects are collected as raw data. However, this data cannot be directly used for training machine learning techniques such as ANNs. Therefore, preprocessing is required. In this step, the dependency information is captured in the form of a binary matrix. Hence, if

there are multiple dependencies among a pair of objects, only one of them is processed and the rest of them are ignored as duplicates. Table 1 presents a sample snippet from a prepared dataset. Hereby, rows of the table list the so-called *main objects*. These are the objects for which we perform classification. Features that are listed in columns represent the so-called *depended objects*. Main objects consist of views, packages, procedures, and functions. Depended objects additionally comprise types, sequences, and tables. In principle, any type of depended object can be used as a feature. For instance, *triggers* are not used in our subject systems due to the organization policy. Therefore, we could not include them in our case study. However, they can easily be incorporated as just another type of depended object as a feature. Our model is agnostic to the type of depended objects that are used as features.

In our case study, as described in detail in the next section, we prepared multiple datasets each of which is based on a varied subset of dependencies among the database types, sequences, tables, procedures, functions, packages, and views. Our goal was to experiment with these alternative datasets and evaluate/compare the accuracy achieved by each of them.

3.1.4 ANN Hyperparameter Selection

Optimizing the parameters of an ANN is another crucial step of the approach. In our study, we optimized 4 parameters: *i)* the number of hidden layers, *ii)* the number of neurons in each layer, *iii)* the activation function, and *iv)* alpha value for L2 regularization [59]. There exist certain rule-of-thumb methods for determining the values of these parameters. For instance, the number of hidden layers is usually set as 1 or 2 [60]. On the other hand, the number of neurons in a hidden layer is suggested to be 2/3 of the number of neurons in the input layer plus the number of neurons in the output layer [61]. In this study, we refrained ourselves from suggesting best practices based on our results only. The effectiveness of parameter values might be highly dependent on the features, which vary based on the size (number of main and depended objects) and the structure

(coupling among the objects) of the subject system. Therefore, parameters should be tuned based on a particular system.

There can be several candidate values for each parameter and one has to determine an optimal combination of these values. There exist various *model selection* methods such as *grid search*, *random search* and *manual search* to find these values [62, 63]. In our work, we adopted grid search for this purpose, where we tried every parameter combination exhaustively. The model was trained with the training set, whereas its performance was calculated on the validation set. We used L2 regularization to avoid overfitting and we applied 10-fold cross-validation to avoid any bias regarding the selection of training and validation sets as explained in the following.

3.1.5 ANN Model Training and Validation

There are several validation techniques that are used for ANN training such as *holdout*, *cross-validation* and *bootstrap*. We used the so-called *stratified k-fold cross-validation* to validate our model. k-fold cross validation basically splits dataset into k partitions and performs k tests by keeping one partition for test at a time, while the rest is used for training. Stratified k-fold cross-validation is a variation of k-fold cross-validation which tries to keep roughly the same proportions of class labels of each split. In this study, k is selected as 10; as such, we applied 10-fold cross-validation. This is one of the common validation techniques, which was also recommended in Kohavi's study [58].

3.1.6 Schema Prediction

At this point of the approach, ANN is ready for performing classification. A new object and its dependencies are provided to ANN as input. ANN classifies this input and predicts the schema, where the new object should belong.

3.1.7 Dataset Extension

Each new object added to the application can be considered as new data that can be used for improving the accuracy of ANN further. We envisioned an iterative process to benefit from expert opinions while using the approach. For this purpose, each schema extension by a new object should be presented to a domain expert for approval. We consider the decision of the domain expert as the ground-truth. This decision can either approve or disapprove the prediction result. In any case, it can be utilized as new labeled data to be included in the dataset. Then, the training process can be repeated with the extended dataset. However, we did not investigate the *Dataset Extension* step deeper to determine, for instance, when or how often re-training should occur. This step of the process is out of the scope of this work.

3.2 Industrial Case Studies

In this section, we present two industrial case studies in which we perform automatic schema classification for two different large-scale legacy applications implemented with the PL/SQL language. The first one is a system called *Customer Relationship Management (CRM)* and the second one is *Campaign Management System (CMS)*. These applications are developed and being maintained by Turkcell[1], which is the largest GSM operator in Turkey. We should note that CRM and CMS are maintained by different teams in the company. There is no overlap between these teams. CRM is maintained by 8 teams and there are currently more than 80 developers as part of these teams. On the other hand, CMS is maintained by 6 teams, which consist of more than 50 developers in total.

[1] `http://www.turkcell.com.tr`

Our goal is to address the following research questions in our case study:

- **RQ1:** Which classification technique provides better accuracy when used with the default (hyper)parameters?
- **RQ2:** How accurately a new object can be placed in a schema by using artificial neural networks?
- **RQ3:** Which types of dependencies among the objects contribute to the accuracy of classification?
- **RQ4:** How does the accuracy of a trained artificial neural network compare to the accuracy of the baseline approach that maximizes schema cohesion measured based on object couplings?

3.2.1 Experimental Setup and Metrics

3.2.1.1 Experimental Setup

In order to train, optimize and validate the ANN models, we employed 15 machines in various configurations; one Linux machine with 32 core CPU and 196 GB RAM, two Windows Fsv2 series virtual machines from Microsoft Azure with 2 core CPU and 4 GB RAM, twelve Windows Fsv2 series virtual machines from Microsoft Azure with 4 core CPU and 8 GB RAM. It took approximately 504 hours (21 days) to complete the training, optimization, and validation for all datasets by using all these machines in parallel. Prediction of an object takes just a second on commodity hardware (with an Intel i5 10th generation processor and 8 GB memory) based on the trained and optimized ANN model.

3.2.1.2 The Baseline Approach and Evaluation Metrics

We evaluated our approach based on a set of metrics and compared the obtained results with respect to those obtained with a baseline approach. Our baseline approach is based on the modularity principle [3]. An ideal design according to this principle would place objects in schemas such that the number of interdependencies among objects within each schema is maximized (maximum cohesion) and the

number of interdependencies among objects located in different schemas is minimized (minimum coupling). Assuming that these are the only criteria, neglecting domain knowledge and the underlying business logic, one would place a new object in a schema that contains the most number of objects that are interdependent with the new object.

We used several metrics in our evaluation as they are currently implemented as part of the Scikit learn library[2][3]. In the following, we introduce a set of definitions followed by the formulas regarding these metrics.

$\mathbf{y_i}$:The true label

$\hat{\mathbf{y}}_\mathbf{i}$:The predicted label

$\mathbf{n}$:The number of all samples that are enumerated from 0 to $n-1$

$\mathbf{n_k}$:The number of samples that belong to a particular class k

$\mathbf{c}$:The number of all classes that are enumerated from 0 to $c-1$

$\mathbf{TP_k}$:The number of samples in class k that are predicted as k

$\mathbf{FP_k}$:The number of samples not in class k that are predicted as k

$\mathbf{FN_k}$:The number of samples in class k that are not predicted as k

In our context, each sample is an object and each class represents a schema, where this object should be placed. Labeling or classification refers to the placement of objects (i.e., samples) in schemas (i.e., classes).

$$Accuracy = \frac{1}{n}\sum_{i=0}^{n-1} 1 \times (\hat{y}_i = y_i) \tag{1}$$

$$Macro\ Precision(P_m) = \frac{\sum_{k=0}^{c-1} \frac{TP_k}{TP_k+FP_k}}{c} \tag{2}$$

$$Macro\ Recall(R_m) = \frac{\sum_{k=0}^{c-1} \frac{TP_k}{TP_k+FN_k}}{c} \tag{3}$$

[2]https://scikit-learn.org/stable/modules/model_evaluation.html\#accuracy-score

[3]https://scikit-learn.org/stable/modules/model_evaluation.html\#multiclass-and-multilabel-classification

$$Macro\ F1(F1_m) = \frac{2 * P_m * R_m}{P_m + R_m} \tag{4}$$

$$Weighted\ Precision(P_w) = \frac{\sum_{k=0}^{c-1}(\frac{TP_k}{TP_k+FP_k} * n_k)}{n} \tag{5}$$

$$Weighted\ Recall(R_w) = \frac{\sum_{k=0}^{c-1}(\frac{TP_k}{TP_k+FN_k} * n_k)}{n} \tag{6}$$

$$Weighted\ F1(F1_w) = \frac{2 * P_w * R_w}{P_w + R_w} \tag{7}$$

3.2.2 Dataset Preparation

One of our research questions (*RQ3*) is related to the impact of the consideration of various types of dependencies on the accuracy of classification. Therefore, we prepared multiple datasets each of which is based on a varied subset of dependencies. Objects that do not have any dependency on other objects are not included in the dataset. We prepared 4 datasets for CRM system and 3 datasets for CMS system. Dataset types are listed in the following. Note that TV dataset is the same as T dataset for CMS system due to the lack of dependencies on views in this system. Therefore, TV dataset does not exist for CMS.

- **T**: includes dependencies to tables only
- **TV**: includes dependencies to tables and views only
- **TVPF**: includes dependencies to tables, views, procedures and functions
- **ALL**: includes dependencies to all the other object types (table, procedure, function, package, type, view, sequence)

Table 1 presents a sample snippet from a prepared dataset. Hereby, each row is associated with one of the objects, whereas each column except the last one represents one of the depended objects. The value of each cell is assigned as 1 if the corresponding main object uses the corresponding depended object, or 0, otherwise. The last column lists the schema, where the main object is located.

Table 1: A sample snippet from a prepared dataset.

Object	**F1**	**F2**	**F3**	**F4**	**F6**	**...**	**Fn-1**	**Fn**	**Target**
Function 1	0	0	0	0	0	...	0	0	Schema 6
Function 2	0	1	0	0	1	...	0	0	Schema 3
Function 3	0	0	1	0	0	...	0	0	Schema 1
Function 4	0	0	1	0	1	...	0	0	Schema 1
Procedure 1	0	0	0	0	0	...	0	0	Schema 3
Procedure 2	0	0	0	0	0	...	0	0	Schema 6
Procedure 3	0	0	0	0	0	...	0	0	Schema 6
Procedure 4	0	0	0	0	0	...	0	0	Schema 2
View 1	0	0	0	0	0	...	0	0	Schema 4
View 2	0	0	0	0	0	...	0	0	Schema 6
View 3	0	0	0	0	0	...	0	0	Schema 6
View 4	0	0	0	0	0	...	0	0	Schema 6
Package 1	0	0	0	0	0	...	0	0	Schema 6
Package 2	0	0	0	0	0	...	0	1	Schema 6
Package 3	0	0	0	0	0	...	0	0	Schema 5
Package 4	0	0	0	0	0	...	0	0	Schema 5

3.2.2.1 Dataset Preparation for CRM

CRM is composed of 6 schemas. There are 1355 objects in these schemas that have dependencies on other objects. The number of each type of object that takes place in CRM is listed in Table 2. We refer to these objects as *main objects* of the application in the rest of this work. Table 3 lists the distribution of these objects to the 6 schemas of the application, labeled as CRM-A, CRM-B, CRM-C, CRM-D, CRM-E, and CRM-F. These objects may depend on various types of objects including table, procedure, function, package, type, view, and sequence. We refer to these objects as *depended objects*. These objects can be among the main objects as well as those that are external to CRM.

Table 2: The number of each type of main object that takes place in CRM.

Object type	**Number of objects**
View	394
Package	443
Procedure	242
Function	276

We collected 3172 depended objects in total. Table 4 lists the distribution of

Table 3: The number of main objects in each schema of CRM.

Schema	**Number of objects**
Schema CRM-A	685
Schema CRM-B	400
Schema CRM-C	96
Schema CRM-D	68
Schema CRM-E	60
Schema CRM-F	46

these objects to 19 different schemas. Six of these schemas are marked (*) as they are part of CRM, whereas the others are external to the application. Note that Table 4 lists only those schemas, which include 20 or more depended objects.

Table 4: The number of depended objects in various schemas (Schemas 1, 2, 4, 5, 6 and 8 correspond to the schemas of CRM labeled as CRM-A, CRM-B, CRM-C, CRM-D, CRM-E and CRM-F, respectively).

Schema	**Number of objects**	**Schema**	**Number of objects**
*Schema 1	730	Schema 11	36
*Schema 2	616	Schema 12	34
Schema 3	292	Schema 13	31
*Schema 4	284	Schema 14	30
*Schema 5	278	Schema 15	30
*Schema 6	196	Schema 16	25
Schema 7	77	Schema 17	24
*Schema 8	62	Schema 18	270
Schema 9	41	Schema 19	22
Schema 10	41		

The number of objects and features of CRM for each dataset is provided in Table 5.

3.2.2.2 Dataset Preparation for CMS

CMS is composed of 4 schemas. There are 1061 objects in these schemas that have dependencies on other objects. The number of each type of main objects that takes place in CMS is listed in Table 6. Table 7 lists the distribution of these objects to the 4 schemas, labeled as CMS-A, CMS-B, CMS-C, CMS-D.

We collected 1563 depended objects in total. Table 8 lists the distribution of

Table 5: The number of objects and features for each dataset of CRM.

Dataset	Number of main objects	Number of features (Depended objects)
T	1230	1435
TV	1238	1566
TVPF	1238	1656
ALL	1355	3172

Table 6: The number of each type of main object that takes place in CMS.

Object type	Number of objects
View	175
Package	369
Procedure	85
Function	432

Table 7: The number of main objects in each schema of CMS.

Schema	Number of objects
Schema CMS-A	631
Schema CMS-B	306
Schema CMS-C	87
Schema CMS-D	37

these objects to 8 different schemas. 4 of these schemas are marked (*) as they are part of CMS, whereas the others are external. Note that Table 8 lists only those schemas, which include 20 or more depended objects.

Table 8: The number of depended objects in various schemas (Schemas 1, 3 , 6 and 7 correspond to the schemas of CMS labeled as CMS-A, CMS-B, CMS-C, CMS-D, respectively.).

Schema	Number of objects	Schema	Number of objects
*Schema 1	842	Schema 5	33
Schema 2	86	*Schema 6	120
*Schema 3	347	*Schema 7	60
Schema 4	42	Schema 8	21

The number of objects and features of CMS for each dataset is provided in Table 9.

Table 9: The number of objects and features for each dataset of CMS.

Dataset	Number of main objects	Number of features (Depended objects)
T	815	599
TVPF	819	671
ALL	1061	1563

3.2.3 Comparison of Classification Techniques

Our approach is based on MLP ANNs. However, there are various classification techniques as outlined in Chapter 2. An exhaustive exploration of the entire set of algorithm alternatives and their configurations is out-of-scope of this work. Nevertheless, we wanted to evaluate the performance of MLP and compare it with respect to other classification techniques in their default configurations as emphasized in *RQ1*. Hereby, we used all the classification techniques with their default (hyper)parameters and applied 10-fold cross-validation on the whole dataset. Figure 4 and Figure 5 depict box plots regarding the achieved accuracy levels for CRM and CMS, respectively. Table 10 and Table 11 present macro and weighted scores (precision, recall and F1) besides the accuracy scores. We can see that MLP consistently outperforms the others in Figure 4 and Figure 5. Not only its median value is the highest but also it leads to lower variance with respect to others. In addition, we can see in Table 10 and Table 11 that all the scores regarding MLP except macro precision are better than those obtained with the other classification techniques. We moved forward with the ANN approach after empirical results confirmed our expectations, where it outperforms the other alternatives in their default configurations. We optimized the parameters of the MLP based on the datasets to further improve its accuracy as described in the following subsections.

3.2.4 Neural Network Building and Model Selection

Scikit learn [64] library is used in this study for model building and training. In particular, the *MLPClassifier* and *GridSearchCV* modules from this library are used for building our ANN model and applying grid search.

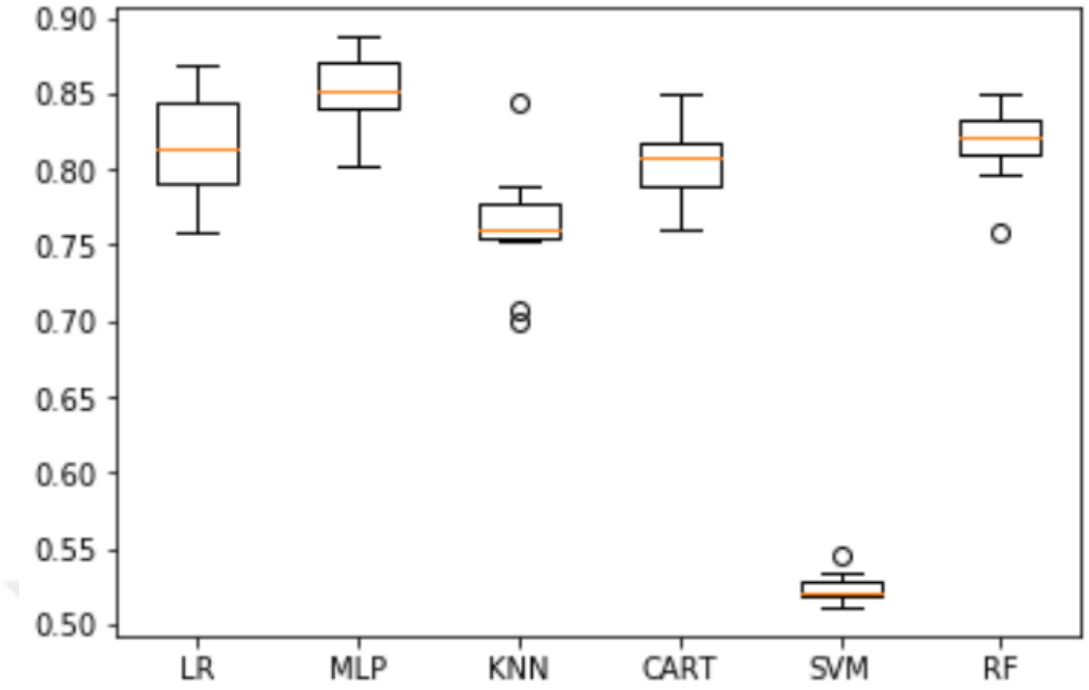


Figure 4: Comparison of performance among classification techniques on CRM application ALL dataset (LR: Logistic Regression, MLP: Multilayer Percepton ANN, KNN: k-Nearest Neighbor, CART: Classification And Regression Tree, SVM: Support Vector Machine, RF: Random Forest)

We optimized 4 parameters in our study, as listed in Section 3.1. Table 12 lists these parameters together with the candidate values we considered for each. We applied the grid search method to find the combination of selections out of these values that give the best accuracy for our training dataset. In particular, we applied grid search with stratified 10-fold cross-validation for all the datasets on both subject systems. In this process, we used Adam optimizer, which is a variation of stochastic gradient-based optimizer. Hereby, the maximum iteration is set to 500. That is, the optimizer runs until convergence or the number of iterations reaches 500. Softmax is used as the output layer activation function in the last layer.

The best hyperparameter value combinations found for each dataset are listed in Table 13.

We can see that the activation function is Sigmoid for all datasets except CMS-TVPF, for which the best function is Tanh. The number of hidden layers

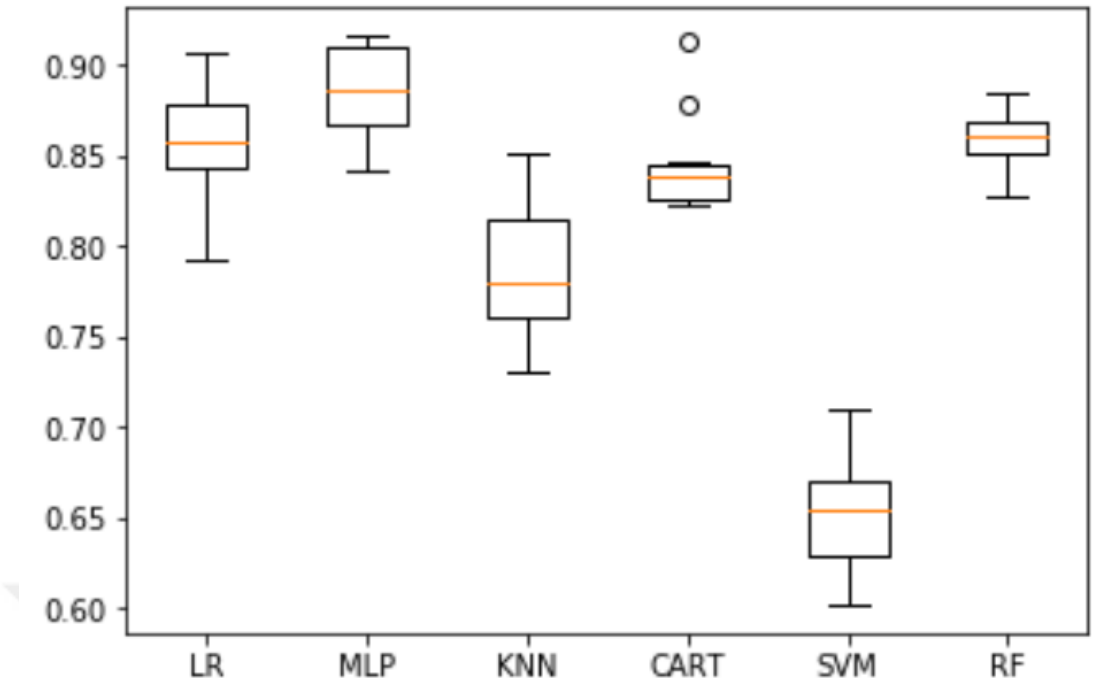


Figure 5: Comparison of performance among classification techniques on CMS application ALL dataset (LR: Logistic Regression, MLP: Multilayer Percepton ANN, KNN: k-Nearest Neighbor, CART: Classification And Regression Tree, SVM: Support Vector Machine, RF: Random Forest)

is selected as 2 for all the datasets. The third parameter (Alpha) is 0.0001 for all datasets except TVPF, for which the parameter is set as 0.001 and 0.0005 for CMS and CRM, respectively. There exist a variation regarding the last parameter, which is the number of neurons taking place in the first and the second hidden layer of the ANN.

In the following subsection, we present and discuss the obtained results. We shared the source code developed for creating the ANN model, optimizing the hyperparameters of the model via grid search, training, and validating the model. It is published on GitHub as a public repository[4]. The repository also includes all the datasets used for training and validating the model as well as the obtained results.

[4]`https://github.com/ersinersoy/annplsqlclassification`

Table 10: Comparison of performance among classification techniques by using macro and weighted scores (accuracy, precision, recall and F1) on CRM application ALL dataset (LR: Logistic Regression, MLP: Multilayer Percepton ANN, KNN: k-Nearest Neighbor, CART: Classification And Regression Tree, SVM: Support Vector Machine, RF: Random Forest)

Technique	**Accuracy**	$\mathbf{P_m}$	$\mathbf{R_m}$	$\mathbf{F1_m}$	$\mathbf{P_w}$	$\mathbf{R_w}$	$\mathbf{F1_w}$
LR	81.5	**79.5**	54.5	60.3	81.3	81.5	79.0
MLP	**85.4**	77.1	**65.3**	**68.4**	**85.1**	**85.4**	**84.3**
KNN	76.5	63.9	45.9	48.8	75.8	76.5	73.7
CART	80.6	67.9	58.1	61.1	79.6	80.6	79.4
SVM	52.4	18.5	17.7	13.3	43.7	52.4	38
RF	82.0	77.8	58.6	63.6	81.9	82.0	80.3

Table 11: Comparison of performance among classification techniques by using macro and weighted scores (accuracy, precision, recall and F1) on CMS application ALL dataset (LR: Logistic Regression, MLP: Multilayer Percepton ANN, KNN: k-Nearest Neighbor, CART: Classification And Regression Tree, SVM: Support Vector Machine, RF: Random Forest)

Technique	**Accuracy**	$\mathbf{P_m}$	$\mathbf{R_m}$	$\mathbf{F1_m}$	$\mathbf{P_w}$	$\mathbf{R_w}$	$\mathbf{F1_w}$
LR	86.6	**88.7**	67.3	72.9	87.6	86.6	85.6
MLP	**88.6**	83.2	**75.5**	**77.8**	**88.7**	**88.6**	**88.2**
KNN	77.9	68.5	59.6	61.3	77.8	77.9	77.2
CART	85.1	79.3	69.1	71.7	85.4	85.1	84.6
SVM	64.7	40.7	29.5	26.7	66.2	64.7	54.4
RF	85.5	79.2	70.1	72.6	85.9	85.5	85.1

3.2.5 Results and Discussions

Top 4 accuracy values achieved with the datasets for *CRM* application *CRM-T*, *CRM-TV*, *CRM-TVPF* and *CRM-ALL* are listed in Table 14, Table 15, Table 16 and Table 17, respectively (See the Appendix for the extended list with top 10 results). There are 4 parts (Hyperparameters, Accuracy, Macro Scores, Weighted Scores) in the tables for each dataset. The first part shows the best hyperparameters for the top 4 results. The second part shows the accuracy results for each of these settings. Since our datasets are imbalanced, we also calculated precision (Equations 2 and 5), recall (Equations 3 and 6) and F1 (Equations 4 and 7)

Table 12: Candidate values for hyperparameters.

Parameter	**Candidates**	**Details**
Activation function	*Sigmoid* or *tanh*	f(x) = 1 / (1 + exp(-x)) or f(x) = tanh(x)
Number of hidden layers	1 or 2	*Not Applicable*
Alpha value for L2 regularization	0.0001, [0.0005 - 0.002]	Step size = 0.0005
Number of neurons each hidden layer	50, 100, [250 - 3000]	Step size = 250

Table 13: The best hyperparameters values found with grid search based on accuracy.

Dataset	**Activation function**	**Number of hidden layers**	**Alpha**	**Number of neurons**
CRM-T	Sigmoid	2	0.0001	500 and 1750
CRM-TV	Sigmoid	2	0.0001	750 and 2750
CRM-TVPF	Sigmoid	2	0.001	50 and 2000
CRM-ALL	Sigmoid	2	0.0001	500 and 750
CMS-T	Sigmoid	2	0.001	50 and 750
CMS-TVPF	Tanh	2	0.0005	1500 and 150
CMS-ALL	Sigmoid	2	0.0001	1000 and 500

scores besides the accuracy (Equation 1) score. Note that precision, recall, and F1 score are calculated in two different ways [65], namely with the *macro* approach and the *weighted* approach. The third and fourth parts list these scores, respectively. The macro approach does not take data imbalance into account, whereas the weighted approach does so.

F1 score is a metric that balances precision and recall. In our context, there is no cost difference between false-positive and false-negative cases. That is, precision and recall metrics have the same level of importance. Therefore, we consider F1 score to be a relevant metric for evaluating our approach even in the case of imbalanced datasets. Results show that weighted scores of precision, recall, and F1 are quite close to the accuracy scores. We can see that the highest accuracy and macro F1 values are obtained with the *CRM-ALL* dataset as 86.7% and 72%, respectively. The lowest accuracy and macro F1 values are 83.4% and 52.8%,

respectively, which are obtained with the *CRM-T* dataset, including only the table dependencies. These values increase to 83.8% and 55.1%, when view dependencies are considered as well. Consideration of procedure dependencies further improves the values to 84% and 58.7% .

Top 4 accuracy values achieved with the datasets for *CMS* application *CMS-T*, *CMS-TVPF* and *CMS-ALL* are depicted in Table 18, Table 19 and Table 20, respectively (See the Appendix for the extended list with top 10 results). We can see that the highest accuracy and macro F1 values are obtained with the *CMS-ALL* dataset as 89% and 80%, respectively. The lowest accuracy and macro F1 values are 87% and 68.3%, respectively, which are obtained with the *CMS-T* dataset, including only the table dependencies. These values increase to 87.4% and 70.4%, when procedure dependencies are considered as well.

Table 14: Top 4 accuracy values obtained with the CRM-T dataset in the *CRM* case study, and the corresponding ANN hyperparameters

	Top 1	**Top 2**	**Top 3**	**Top 4**
Hyperparameters				
Hidden Layers & # of Neurons	(500, 1750)	(750, 250)	(750, 50)	(2000, 150)
Alpha Value	0.0001	0.0005	0.002	0.0005
Activation Function	Sigmoid	Sigmoid	Sigmoid	Sigmoid
Accuracy	**83.4**	83.0	82.8	82.7
Macro Scores				
Precision	61.6	61.2	64.6	70.7
Recall	50.6	51.6	50.0	54.1
F1	52.8	53.5	53.1	**57.6**
Weighted Scores				
Precision	80.6	80.6	80.8	82.7
Recall	83.4	83.0	82.8	82.7
F1	81.0	81.0	80.7	**81.2**

We also collected results regarding the accuracy of the baseline approach, where a schema for a new object is determined as the one that contains most of the depended objects. Results for *CRM* application are listed in Table 21 for various types of depended objects considered. We can see that the accuracy is 44.9%

Table 15: Top 4 accuracy values obtained with the CRM-TV dataset in the *CRM* case study, and the corresponding ANN hyperparameters

	Top 1	**Top 2**	**Top 3**	**Top 4**
Hyperparameters				
Hidden Layers & # of Neurons	(750, 2750)	(750, 750)	(500, 1000)	(750, 3000)
Alpha Value	0.0001	0.0005	0.0001	0.001
Activation Function	Sigmoid	Sigmoid	Sigmoid	Sigmoid
Accuracy	**83.8**	83.4	83.2	83.2
Macro Scores				
Precision	64.0	60.6	61.3	59.9
Recall	53.0	51.7	53.8	52.6
F1	55.1	**53.3**	**55.3**	54.0
Weighted Scores				
Precision	81.5	81.0	81.5	80.4
Recall	83.8	83.4	83.2	83.2
F1	81.5	81.2	**81.6**	81.1

Table 16: Top 4 accuracy values obtained with the CRM-TVPF dataset in the *CRM* case study, and the corresponding ANN hyperparameters

	Top 1	**Top 2**	**Top 3**	**Top 4**
Hyperparameters				
Hidden Layers & # of Neurons	(50, 2000)	(200, 2250)	(100, 1000)	(150, 3000)
Alpha Value	0.001	0.0005	0.0005	0.001
Activation Function	Sigmoid	Sigmoid	Sigmoid	Sigmoid
Accuracy	**84.0**	83.8	83.7	83.7
Macro Scores				
Precision	64.9	64.8	62.2	63.5
Recall	56.7	55.3	55.2	54.8
F1	**58.7**	57.6	56.9	56.8
Weighted Scores				
Precision	82.6	82.1	81.9	82.2
Recall	84.0	83.8	83.7	83.7
F1	**82.7**	82.3	82.2	82.2

Table 17: Top 4 accuracy values obtained with the CRM-ALL dataset in the *CRM* case study, and the corresponding ANN hyperparameters

	Top 1	**Top 2**	**Top 3**	**Top 4**
Hyperparameters				
Hidden Layers & # of Neurons	(500, 750)	(750, 1250)	(750, 200)	(500, 250)
Alpha Value	0.0001	0.001	0.001	0.0001
Activation Function	Sigmoid	Sigmoid	Sigmoid	Sigmoid
Accuracy	**86.7**	86.2	86.2	86.1
Macro Scores				
Precision	82.4	80.0	80.4	79.2
Recall	67.2	66.2	64.2	67.2
F1	**72.0**	70.4	68.9	70.6
Weighted Scores				
Precision	86.6	86.0	85.4	85.7
Recall	86.7	86.2	86.2	86.1
F1	**85.8**	85.2	84.7	85.1

Table 18: Top 4 accuracy values obtained with the CMS-T dataset in the *CMS* case study, and the corresponding ANN hyperparameters

	Top 1	**Top 2**	**Top 3**	**Top 4**
Hyperparameters				
Hidden Layers & # of Neurons	(50, 750)	(250, 2250)	(3000,2000)	(50, 200)
Alpha Value	0.0001	0.0001	0.0001	0.0001
Activation Function	Sigmoid	Sigmoid	Sigmoid	Sigmoid
Accuracy	**87.0**	87.0	87.0	86.9
Macro Scores				
Precision	76.3	75.3	73.1	76.8
Recall	65.3	64.4	62.5	65.9
F1	68.3	67.2	65.2	**68.9**
Weighted Scores				
Precision	86.0	85.8	84.7	86.3
Recall	87.0	87.0	87.0	86.9
F1	85.9	85.7	85.2	**86.0**

Table 19: Top 4 accuracy values obtained with the CMS-TVPF dataset in the *CMS* case study, and the corresponding ANN hyperparameters

	Top 1	**Top 2**	**Top 3**	**Top 4**
Hyperparameters				
Hidden Layers & # of Neurons	(1500, 150)	(1750, 150)	(1500, 150)	(1750, 250)
Alpha Value	0.0005	0.0001	0.0001	0.002
Activation Function	Tanh	Tanh	Tanh	Tanh
Accuracy	**87.4**	87.3	87.2	87.1
Macro Scores				
Precision	77.2	77.6	75.6	77.2
Recall	67.8	67.2	66.5	67.9
F1	**70.4**	70.2	68.9	70.0
Weighted Scores				
Precision	87.0	86.9	86.5	87.1
Recall	87.4	87.3	87.2	87.1
F1	**86.7**	86.6	86.3	86.5

Table 20: Top 4 accuracy values obtained with the CMS-ALL dataset in the *CMS* case study, and the corresponding ANN hyperparameters

	Top 1	**Top 2**	**Top 3**	**Top 4**
Hyperparameters				
Hidden Layers & # of Neurons	(1000, 500)	(1750,1000)	(500, 1500)	(1500, 150)
Alpha Value	0.0001	0.0001	0.001	0.0001
Activation Function	Sigmoid	Sigmoid	Tanh	Sigmoid
Accuracy	**89.0**	88.7	88.7	88.5
Macro Scores				
Precision	86.5	83.8	80.5	84.1
Recall	77.8	77.2	78.4	76.7
F1	**80.0**	78.5	78.1	77.8
Weighted Scores				
Precision	89.7	89.1	88.9	88.9
Recall	89.0	88.7	88.7	88.5
F1	**88.7**	88.4	88.5	88.3

when table dependencies are considered only. The accuracy increases only up to 55.5% when all types of depended objects are used. Results for *CMS* application are listed in Table 22 for various types of depended objects considered. We can see that the accuracy is 56.9% when table dependencies are considered only. The accuracy increases only up to 57.4% when all types of depended objects are used.

We can answer our research questions based on these results as the following. Regarding *RQ1*, we observed that ANN consistently outperformed the other classification techniques with their default configurations. Regarding *RQ2*, we observed that the accuracy of ANN to determine a schema for a given new object is above 86%. We also observed that the macro F1 score of ANN on determining a schema for a given new object is above 72%. The more types of dependencies are considered, the higher becomes both the accuracy and F1 score for both subject systems, CRM and CMS. Therefore, we conclude for *RQ3* that including more dependencies improves both accuracy and F1 score regardless of the types of dependencies. Since PL/SQL programs are highly coupled with databases, an accuracy level of 83.4%, 87% can be achieved even if only table dependencies are used for model building and training for CRM and CMS, respectively. This is not the case for macro F1 scores. The consideration of other types of dependencies increased macro F1 scores significantly in our case study. Finally, the accuracy of the baseline approach turned out to be much lower with respect to ours (*RQ4*).

Table 21: Accuracy levels obtained with the baseline approach for T(table), TV(table,view), TVPF(table,view,procedure,function) and ALL of CRM application (1355 in total).

Object type	**Same schema objects**	**Other schema objects**	**No dependency**	**Accuracy**
CRM-T	608	622	125	44.9%
CRM-TV	658	580	117	48.6%
CRM-TVPF	662	576	117	48.9%
CRM-All	752	603	NA	55.5%

Table 22: Accuracy levels obtained with the baseline approach for T (table), TVPF (table,view,procedure,function) and ALL of CMS application (1061 in total).

Object type	**Same schema objects**	**Other schema objects**	**No dependency**	**Accuracy**
CMS-T	604	211	246	56.9%
CMS-TVPF	606	213	242	57.1%
CMS-All	609	452	NA	57.4%

3.2.6 Threats to Validity and Limitations

In our study, we wanted to perform industrial case studies on real systems. However, it is very challenging to access such systems and take support from experts. Both CRM and CMS systems that we used in our study are from the telecommunication domain and they are developed within the same company. Although these systems are maintained by different teams, our evaluation is subject to an *external validity* threat. Additional case studies can be performed to mitigate this threat. We already found out regarding *RQ3* that including more dependencies improves both accuracy and F1 scores regardless of the types of dependencies. We observed consistent results regarding the other research questions though. Another *external validity* threat that limits the generalizability of our approach is our assumption regarding the stability of the architecture. We assume that the system has a stable architecture, where there are no disruptive changes at the architecture level. In our context, this means that the addition and removal of schemas are unlikely.

There exist an *internal validity* threat due to the data imbalance problem. For instance, 60% of all the objects belong to schema CMS-A in our dataset regarding CMS. Therefore, one can easily achieve 60% accuracy by just predicting this schema for all the objects. However, we believe that the presented results are not affected by this issue for two reasons. First, we used 10-fold cross-validation with stratified k-fold, instead of regular k-fold. Stratified k-fold preserves the percentage of samples for each class. That is, the ratios of objects belonging to each schema in the evaluation set are equal to their ratios in the overall set.

Second, our accuracy and F1 score values are much higher than 60%. Even the accuracy of the baseline approach is higher.

Another *internal validity* threat is related to the formation of the training data in our experiments. There are two basic categories of objects in our dataset: *i)* those for which we perform classification (called as main objects) and *ii)* those that are only used as features (called as depended objects). However, there can also be interdependencies among the main objects, in which case, some of the objects in the test set can take place among features as depended objects in the training set. This is not a realistic scenario for a new object to be classified. There should not be other objects depending on this object yet and as such, it should take place among the features in the training dataset. Although such cases are possible as a threat to validity, they are very rare in practice. Depended objects consist of types, sequences, and tables in addition to views, packages, procedures, and functions. Overall, these objects significantly outnumber the main objects. Even the ratio of all the main objects (not only the tested ones) in features is less than 5%. Moreover, rows and features in the dataset are anonymous and there are no explicit relations among each other.

We employ a supervised learning approach. Therefore, our approach relies on some assumptions regarding the input data, which constitute a threat to *construct validity*. First, the collected dependencies that are used as input must reflect the intended architecture. We assumed that the initial version of the source code reflects this. Domain experts are expected to confirm the validity of the input data before it is used for training. Second, we assume a stable design at the architectural level. If a new schema is added, the number of objects placed under this schema will be expected to reach a certain size before it can be classified. Objects of the new schema can be included in the dataset and the model can be re-trained only after sufficient data is available. However, it should not be very common to extend the set of schemas. Both subject systems in our case study are legacy applications that have been used for about 20 years and they have not been

subject to such an extension. Finally, our approach characterizes an object based on its dependencies. Features of an object are defined based on its coupling with other objects. Hence, an object that has no dependencies, which is an unusual case, cannot be classified since it will not be possible to characterize the object. That means we have no information regarding the object for classification.

3.2.7 Usefulness of the Approach

Misplacement of newly developed objects in schemas causes architectural erosion, which in turn increases maintenance costs. Moreover, objects that take place in the same schema can access each other without authorization. Therefore, the correct placement of objects in schemas is important in terms of both maintainability and security. A correct placement requires expertise and novice developers have to ask the opinion of experts before making a decision. However, there are only a limited number of experts, who are usually busy. As a result, novice developers might not be able to find an available expert for a long time, a couple of days in some cases. Then, they move forward not to lose any time further and make their own decision based on the coupling of the newly developed object with other objects. Note that this is our baseline approach, which does not lead to very accurate results. Experts can pinpoint the misplacement of objects in schemas during code reviews or periodic application reviews. These have to be fixed later on and they cause additional maintenance effort. Each schema correction also requires re-deployment of the application and re-definition of authorization requests. Therefore, even if the classification accuracy is not 100%, any improvement of accuracy with respect to the baseline approach is very valuable for reducing the maintenance cost. Our case study was focused on the performance of the approach. We did not collect data regarding user experience and the amount of effort reduction. Since any object misplacement leads to rework and additional maintenance cost, we expect the amount of effort reduction to be proportional to the improvement of classification accuracy with respect to the baseline approach.

3.2.8 Extending the Approach for other Types of Systems

We proposed an approach for supporting the maintenance of PL/SQL programs. However, we do not consider our approach to be language-specific. It can be extended for other types of database applications and data access tiers of software systems. Such systems are very common and usually subject to long-term maintenance, especially in enterprise applications. Other than PL/SQL in Oracle database, there also exist other database technologies with different languages such as Transact-SQL (T-SQL) [66]. T-SQL is used for developing objects like stored procedures on Microsoft and Sybase databases. On the other hand, database applications that are developed with general-purpose programming languages like Java can make use of embedded SQL statements as well. Hereby, communication with the database is usually performed via the use of a CLI (Call Level Interface) such as JDBC (Java Database Connectivity) [67]. The corresponding parts of the source code can be parsed to extract dependencies on database objects. These extracted dependencies can be used as features for our approach. Therefore, our approach can be extended to be used for other types of database applications or at least parts of other systems that are used as a database abstraction layer.

3.3 Related Work and Our Contributions

Maintenance of legacy software systems is an acknowledged problem in the literature. Therefore, there exist many studies proposed to address the related subproblems [9]. A majority of these studies aim at reverse engineering documentation by analyzing the source code [68, 8] or runtime system behavior [69, 70]. This documentation mainly depicts high-level modular decomposition of the system [71, 72]. As such, it serves as a map for supporting maintenance activities. Alternatively, design rule spaces [73] have been used to provide an architecture insight based on structural or evolutionary relationships extracted from source code as well as pinpointing potential bugs in the form of modularity violations. In this work, our goal is not to reverse engineer documentation or detect bugs/violations. Instead,

we aim at automatically answering a particular query about where a new object must be placed among the existing modules.

There also exist various search-based software refactoring techniques proposed in the literature. Ouni *et al.* [74] proposed an approach called *LibFinder* to prevent missed reuse opportunities and provide decision support to find useful third-party libraries. Mkaouer *et al.* [75] prioritizes refactoring opportunities based on uncertainties related to code smell correction. Another work in this area is proposed by Kessentini *et al.* [76]. In their work, they described a search-based detection method that aims to detect model changes as a sequence of refactorings. Non-dominated Sorting Genetic Algorithm (NSGA-II) [77] has been used for generating such a sequence based on a list of base models with their subsequent refactorings and a model to be refactored [78]. Mansoor *et al.* [79] take good and bad design examples as input to detect code smells by using a variant of multi-objective genetic programming [80] that follows the same evolutionary process of NSGA-II. Although these search-based software refactoring techniques are related to our work, none of them utilize artificial neural networks and adopt a classification approach to particularly address the problem of identifying extension points for the placement of new objects in the system.

Some tools and techniques can be combined for supporting software architecture analysis. In that respect, ARCADE [81] (Architecture Recovery, Change, And Decay Evaluator) might be the most comprehensive framework in current state-of-the-art. It is composed of 5 subsystems designed for supporting 5 different analysis tasks. These tasks are (1) recovering software architecture by clustering software elements, (2) detection of architectural decay by identifying instances of a set of architectural smells [82], (3) measuring the amount of change and modularization quality by utilizing a set of metrics, (4) visualizing the clustered elements, and (5) predicting potential issues by machine learning based on the correlation of past issues and architectural smells. None of these tasks address the problem of identifying the placement of a new object in the existing architecture. Detection

of architectural smells can possibly identify a misplaced object, but only after the fact.

One can notice that most of the techniques and tools for supporting software maintenance are developed for C/C++ or Java programs. Although some of them are language independent [15, 72], they take input models that are extracted from either procedural or object-oriented programs. There exist only a few studies that focus on the maintenance of PL/SQL programs. These studies aim at reverse engineering documentation in the form of business rules [83], data flow graphs [83] and module views of the architecture design by either clustering [13, 4] or graph partitioning [84] techniques.

One of these tools is Metamorphosis [85], which parses and analyzes PL/SQL code to generate high-level representations in the form of data flow graphs for supporting the maintenance of PL/SQL systems. In general, there are not many studies focusing on the maintenance of database applications, despite the common use and importance of databases, especially in enterprise systems. In one of these studies, Gardikiotis et al. [86] introduce a tool that performs a data flow analysis to analyze the impact of changes in database schemas. However, none of these approaches adopt a classification approach to determine the placement of PL/SQL objects among candidate schemas of an enterprise application.

Artificial neural networks have been previously used in the software engineering domain for automated fault localization [87, 88] and software cost estimation [89, 90, 91]. To the best of our knowledge, they have not been used for learning/representing the structural dependencies within a software system and as such automatically answer queries for supporting software maintenance. A module architecture advisor [92] was previously proposed that uses a neural network for categorizing software units to recommend organizational changes in a software system. Hereby, the set of names used by a software unit as well as the names of places where the unit is used are considered as features characterizing

that unit. A similarity function is defined based on these features. Adding a common feature increases similarity; adding a distinctive feature decreases it. There are also weights associated with each feature that determines its significance. Categorization is performed with the K nearest neighbor algorithm. A neural network is used for learning from recommendations labeled by architects and adjusting feature weights accordingly. Although a classification approach and a neural network model are used in this study, it is not based on structural coupling of software elements with database elements. It does not also use a neural network for implicitly representing an abstraction of this information. Rather than being a general hidden-layer network, the network is designed specifically for modeling the similarity function to optimize its parameters and as such improving the similarity judgment. The evaluation of the approach is also based on a relatively small-scale system that comprises 64 procedures implemented in C language.

The contributions of this study are threefold. First, we introduce a novel approach for representing software/database objects and their interdependencies in terms of an artificial neural network formalism. Second, we introduce methods for optimizing the number of hidden neurons and other hyperparameters of the network automatically. Third, we report on industrial case studies to evaluate our approach with real and large-scale legacy systems.

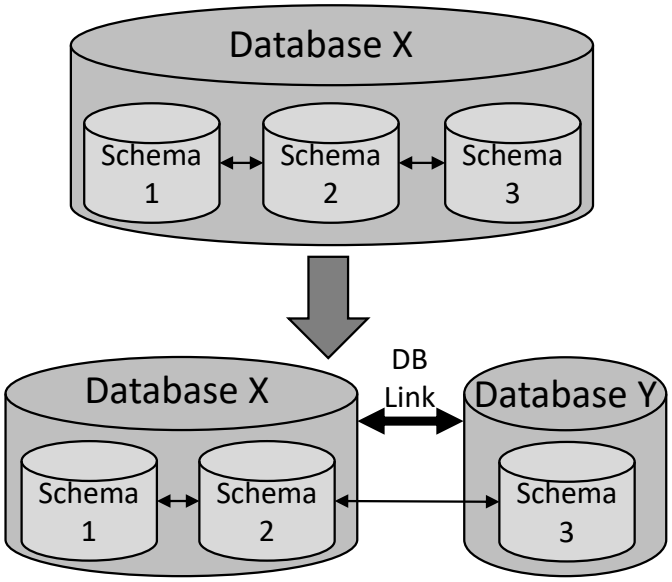
CHAPTER IV

DATA MODEL CHANGE IMPACT ANALYSIS

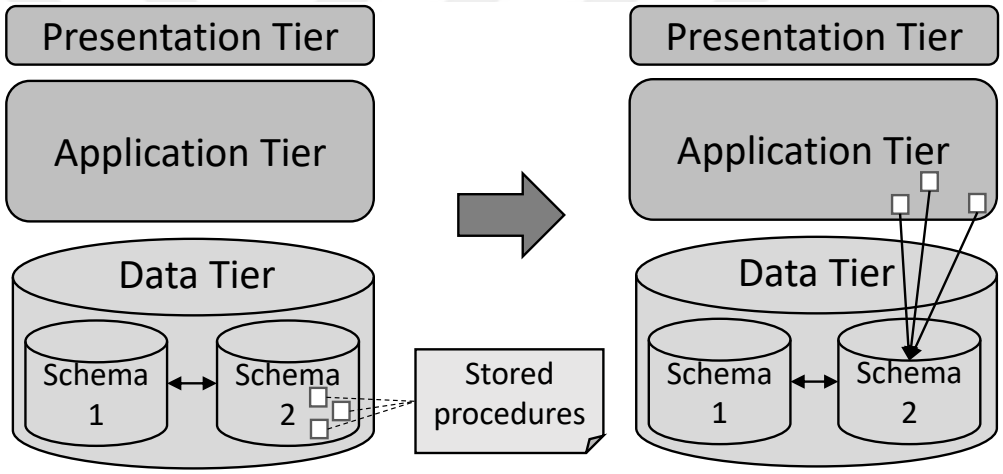
In this chapter, we present a tool that can automatically extract data model dependencies on PL/SQL programs. The tool captures all the schemas of a given database and the data model regarding each schema including its tables and views. It also parses the source code of all the objects (i.e., PL/SQL functions and procedures) in each schema. Then, a dependency model is created based on the data model and an analysis of SQL queries in these objects. Unlike prior studies, our tool can analyze queries that are created dynamically and that involve multiple tables as well as PL/SQL-specific features.

The approach is implemented for Oracle Database[1]. We evaluate its accuracy with an industrial PL/SQL program from the telecommunications domain. The accuracy of the tool is reviewed and approved by domain experts. As a second verification method, we apply industrial case studies to utilize the tool for effort estimation. Estimated efforts are related to two types of common refactoring needs depicted in Figure 6. The industrial case studies are applied by using the dependency model which is the output of the tool. First, some of the modules might need to be migrated to a separate database. In many cases, a complete schema has to be migrated (See Figure 6a). This type of refactoring is especially needed for migrating monolith applications to a microservices architecture [93, 94, 95, 96]. This migration can also be driven by quality factors such as security, reliability, and maintainability [97, 98], and the isolated schema can still be accessed by the other schemas via Database Link (DB Link) [99] after migration. Second, some of the modules in the data-tier might need to be migrated to the application tier (Figure 6b). Business logic that is outside the generic operation must be removed

[1] `https://www.oracle.com/database/`



(a) Migrating a schema to another database.



(b) Migrating business logic from Data-Tier to Application Tier.

Figure 6: Two types of refactoring commonly needed for Data-Tier software.

from the stored procedures and pushed up into the application tier as a design principle [100]. We calculate the estimated effort for refactoring 10 different schemas by using the dependency model which is the output of the tool. We also collect estimations of domain experts regarding the refactoring efforts for the same set of schemas. We observe high consistency between the automated estimations and manual estimations in the case studies.

The remainder of this chapter is organized as follows. We explain our approach

and its implementation in the following section. In Section 4.2, we present industrial case studies conducted for an empirical evaluation of our approach. Finally, we summarize the related studies and our contributions in Section 4.3.

4.1 Overall Approach

The overall approach, which involves four steps, is depicted in Figure 7. In the first step, all schemas and their objects (stored procedures and functions) are collected. The data model (tables, views, and their columns) is extracted as well. In the second step, PL/SQL statements are extracted. In the third step, dependencies are derived based on these statements and the data model. Finally, effort estimation is performed to apply industrial case studies as an extension of the tool.

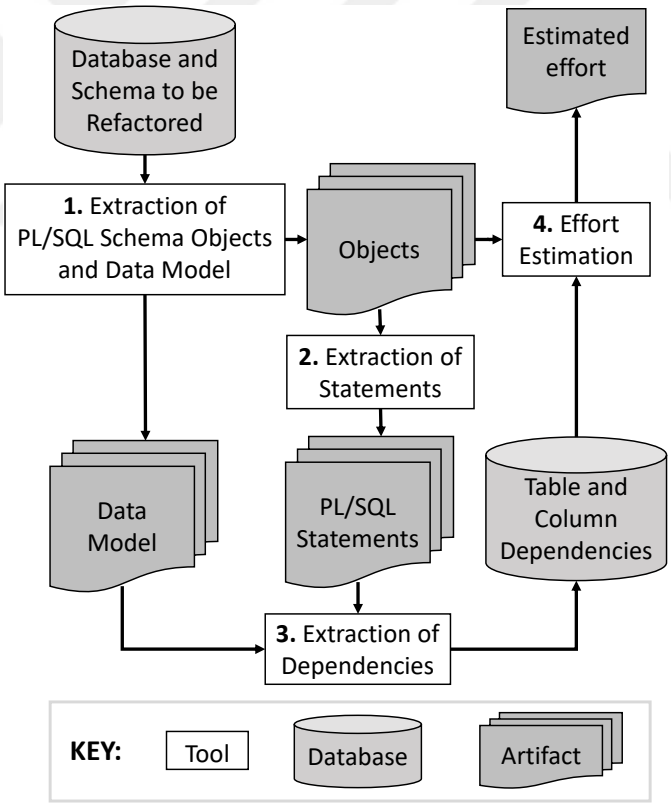


Figure 7: The overall process.

We implemented our approach for Oracle Database[2]. We explain each step of

[2]The binaries of our tool and its source code together with build and usage instructions are shared on a public GitHub repository at `https://github.com/ersinersoy/`

the approach, including the implementation details in the following.

4.1.1 Extraction of PL/SQL Schema Objects and Data Model

Schema term in Oracle database is equivalent to Database term in MSSQL [101]. Therefore, the existence of more than one database on an Oracle DBMS is possible with the concept of schema. Schemas comprise a collection of objects and they define the gross-level organization of the system. All database objects are organized in schemas. Oracle database has data dictionary objects [29] to keep meta-data regarding the database. One of these objects is DBA_OBJECTS, which keeps the information about all the objects in DBMS. Here, we used this data dictionary object to collect all procedures and functions (both hereinafter referred to as procedures) in schemas. Likewise, tables and views (as well as their columns) used in these objects are collected by using relevant data dictionary objects of Oracle DBMS. Source codes regarding all the procedures are extracted using the built-in GET_DDL function of DBMS_METADATA package provided by Oracle.

4.1.2 Extraction of Statements

We used an open source parser generator, ANTLR (ANother Tool for Language Recognition) [102] to generate a PL/SQL parser using the PL/SQL language grammar [103] made available for ANTLR v4 [104]. The generated parser creates an abstract syntax tree (AST) of a given PL/SQL program. This AST can be traversed via listener methods [104]. There are 1510 listener methods generated for the grammar that we utilized. We reviewed these and implemented custom listeners to capture *DML (Data Manipulation Language)*, *variable declaration*, *assignment*, *Cursor* and *Execute Immediate* statements. The last two are required for handling dynamically created SQL statements.

`db-refactoring-analyzer`.

4.1.3 Extraction of Dependencies

Table and column dependencies are extracted based on the data model and various types of PL/SQL statements taking part in the refactored objects. A pseudo-code regarding the main dependency extraction process is listed in Algorithm 1.

Algorithm 1 Main dependency extraction process.

```
1: STMT ← all PL/SQL statements in a given procedure
2: TM ← all dependent tables
3: TCM ← all columns of dependent tables
4: VM ← ∅
5: R ← ∅
6: for each stmt in STMT do
7:     switch stmt.Op do
8:         case Variable Declaration for a variable, v
9:             VM ← VM ∪ {{v,null}}
10:            if stmt has a default value, x then
11:                VM ← VM\{{v,null}} ∪ {{v,x}}
12:            end if
13:        case Assignment of a value x to a variable v
14:            VM ← VM ∪ {{v,x}}
15:        case Insert or Select or Update or Delete or Merge
16:            Dep(stmt,TM,TCM,R)
17:        case Dynamic Cursor or Dynamic SQL
18:            DynDep(stmt,TM,TCM,VM,R)
19: end for
20: Return R
```

Algorithm 1 starts with the initialization of a set of collections (Lines 1-5). The set of PL/SQL statements taking part in a given procedure are collected in $STMT$ (Line 1). Then, Oracle Metadata dictionary is utilized to collect table dependencies (Line 2). Note that TM includes only a subset of dependent tables; those that can be retrieved via the Oracle Metadata (See Table 29). This set is extended in the rest of the analysis process, while processing various types of SQL statements, including those that are dynamically created. TCM includes all the columns of tables included in TM (Line 3). VM is created for keeping the set of variables defined in the procedure and values assigned for these variables (Line 4). Finally, R (Line 5) is used for recording the set of detected dependencies as part of the database. This is the main output of the process (Line 20). In fact, all

the information collected during the analysis process (schemas, objects, identified dependencies, etc.) is stored in a database. R is a 4-tuple, {c,t,op,E} that includes the depended column (c), the associated table (t), type of DML operation (op) and the set of exceptional cases (E) regarding the dependency (See Table 28).

Each statement of the procedure is processed one by one (Lines 6-19) according to the type of statement. VM is updated for statements that include variable declarations and assignments (Lines 8-14). Regular DML statements are processed with the *Dep* function as listed in Algorithm 2 (Lines 15-16). *DynDep* function as listed in Algorithm 3 is used for processing statements that include dynamic creation of Cursor and SQL structures (Lines 17-18).

Algorithm 2 Dependency extraction for static DML statements.

```
1: function DEP(stmt,TM,TCM,R)
2:     T_a ← parse table aliases in stmt
3:     C_s ← find columns of stmt using TCM
4:     M ← mapping of TM and T_a
5:     T_sall ← parse tables used for Select All in stmt
6:     for each t in T_sall do
7:         for each c ∈ C_s defined in t do
8:             R ← R ∪ {{c,t,Select,∅}}
9:         end for
10:    end for
11:    for each c ∈ C_s do
12:        T_c ← find tables associated with c using M
13:        t_c ← an element of T_c
14:        if stmt.Op = Merge and |T_c| > 1 then
15:            R ← R ∪ {{c,t_c,Merge,{ETM}}
16:        else
17:            R ← R ∪ {{c,t_c,stmt.Op,∅}}
18:        end if
19:    end for
20: end function
```

The *Dep* function defined in Algorithm 2 first obtains the table names/aliases (more than one table name can refer to the same table) and columns associated with the processed statement (Lines 2-3). It creates a mapping between the tables in TM and the aliases found (Line 4). It also separately collects tables that take part in a *Select All* operation, if any (Line 5). A dependency is added to the

results for each column associated with the operation (Lines 6-10). The remaining columns associated with the statement are processed (Lines 11-19) based on table mappings (Line 12) and the DML operation. In case the operation is *Merge* and the column is associated with more than one table, then the dependency is recorded together with an ETM (See Table 28) exception (Line 15). Otherwise, the set of dependencies is extended without any exception (Line 17).

DynDep function as listed in Algorithm 3 first records the set of exceptional cases (See Table 28) associated with the statement, if any (Lines 2-11). Then, the dynamically created SQL statement is constructed depending on the type of creation used in the procedure (Lines 13-21). The set of tables is obtained for the constructed SQL (Line 23). TM and TCM are extended with these tables and the associated columns, which are then provided to the *Dep* function together with the constructed statement if there are no exceptional cases detected (Lines 25-31). A dependency is added to R together with the set of detected exceptions, otherwise (Line 34). The set of dependencies recorded is utilized. Table 23 presents a sample snippet from recorded dependencies.

4.1.4 Effort Estimation

The last step of the approach is effort estimation for the case studies. The calculation of this estimation is performed for two different refactoring types (See Figure 6). The first refactoring type is migrating a schema to another Oracle database (a) and it mainly involves three steps: *i)* creating a database link [99], *ii)* granting the necessary permissions to this database link, and *iii)* modifying the source code to replace direct table accesses with the use of this link. The second refactoring type is migrating business logic from the database tier to the application tier (b). This process requires the implementation of the migrated business logic at the application tier, including the implementation of a data model at that tier using, for instance, POJO [105] classes. We devised a set of formulas that can be used for calculating effort estimation for these two types of architectural refactoring based on the dependency analysis performed in the previous steps of

Algorithm 3 Dependency extraction for dynamically created DML statements

```
1: function DynDep(stmt,TM,TCM,VM,R)
2:     E ← ∅
3:     if Multiple variable assignments exist then
4:         E ← E ∪ {EMA}
5:     end if
6:     if Multiple variables exist then
7:         E ← E ∪ {ECA}
8:     end if
9:     if String concatenation exists then
10:        E ← E ∪ {ESC}
11:    end if
12:
13:    switch DynamicSQLType of stmt do
14:        case Direct SQL
15:            sql_d ← get SQL from stmt
16:        case SQL as a Variable
17:            sql_d ← get SQL from VM
18:        case SQL as a Dynamic Variable
19:            var_SQL ← get SQL from VM
20:            Execute var_SQL on the database
21:            sql_d ← Result of var_SQL execution
22:
23:    T ← parse tables in sql_d
24:
25:    if EMA ∉ E and ECA ∉ E and ESC ∉ E then
26:        for each t in T do
27:            if t ∉ TM then
28:                TM ← TM ∪ {t}
29:                TCM ← TCM∪ columns of t
30:            end if
31:        end for
32:        Dep(sql_d,TM,TCM,R)
33:    else
34:        R ← R ∪ {{c,t,sql_d.Op,E}}
35:    end if
36: end function
```

Table 23: A sample snippet from dependency results.

Proc Name	**Schema**	**Operation**	**Line**	**Table Schema**	**Table Name**	**Column Name**	**Is Exception**	**Multiple Assign**	**Complex Assign**	**String Concat**	**Table Not Catched**
Proc 1	Schema 1	INSERT	10	Schema 1	TAB 1	COL 1	0	0	0	0	0
Proc 1	Schema 1	INSERT	10	Schema 1	TAB 1	COL 2	0	0	0	0	0
Proc 2	Schema 1	SELECT	20	Schema 1	TAB 1	*	0	0	0	0	0
Proc 3	Schema 1	SELECT	20	Schema 2	TAB 2	COL 3	0	0	0	0	0
Proc 3	Schema 1	UPDATE	21	Schema 2	TAB 2	COL 3	0	0	0	0	0
Proc 3	Schema 1	DELETE	22	Schema 3	TAB 3	COL 4	0	0	0	0	0
Proc 3	Schema 1	MERGE	30	Schema 3	TAB 3	COL 5	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...	...	...	...
Proc n	Schema 1	MERGE	40	NA	NA	NA	1	0	0	0	1
Proc n	Schema 2	EXECUTE IMMEDIATE	100	Schema 4	TAB 4	COL 5	0	0	0	0	0
Proc n	Schema 2	EXECUTE IMMEDIATE	150	NA	NA	NA	1	1	0	0	0
Proc n	Schema 3	CURSOR	200	Schema n	TAB n	COL n	0	0	0	0	0

the approach. These formulas are listed in Equations 1-5. Table 24 provides the definitions of the set of terms used in these formulas.

$$DML_i = DMLt_i * DMLc_i + Sall_i \tag{8}$$

$$DSQL_i = 2 * DSQLt_i * DSQLc_i \tag{9}$$

$$EX_i = 3 * (EMA_i + ECA_i + ESC_i + ETM_i) \tag{10}$$

$$E_a = \frac{\sum_{i=0}^{n} DMLdst_i + Sallds_i + 2 * DSQLdst_i + EX_i}{BC_a} \tag{11}$$

$$E_b = \frac{\sum_{i=0}^{n} loc_i/LC + DML_i + DSQL_i + EX_i}{BC_b} \tag{12}$$

Equation 8 is used for calculating the data model complexity by multiplying the number of distinct tables ($DMLt$) and the number of distinct columns ($DMLc$) that are used in DML statements. The number of distinct tables that are used in Select All columns statement ($Sall$) is added as an additional contributing factor. Equation 9 is used for calculating the impact of the use of dynamic SQL. Hereby, the number of distinct tables ($DSQLt$) is multiplied by the number of distinct columns ($DSQLc$) that are used in dynamic SQL and dynamic Cursor statements. There is also a multiplying factor, set as 2, to reflect the amplified effort consumed for refactoring dynamically created statements. Equation 10 is used for calculating the impact of exceptional cases (See Table 28). Hereby, a multiplying factor of 3 is used after the number of such cases is calculated. Equation 8 and Equation 9 is only used in Equation 12, which is the main formula used for effort estimation regarding the refactoring type (b). Equation 10 is used for both types. Equation 11 is used for calculating effort estimation for refactoring type (a) by summing up the number of distinct tables on different schemas in DML ($DMLdst$) and the number of distinct tables on different schemas these are used in Select All columns statement ($Sallds$) and the number of distinct tables

Table 24: Terms that are used in Equations 1-5 (DT: Distinct tables, DTds: Distinct tables in different schemas, DC: Distinct columns).

Term	**Description**
n	# of Procedures in schema
loc	# of lines of code
Sallds	# of DTds that Select all columns statement is used
DMLdst	# of DTds in Select, Insert, Delete, Update and Merge
DSQLdst	# of DTds in dynamic SQL and dynamic Cursor
Sall	# of DT that Select all columns statement is used
DMLt	# of DT in Select, Insert, Delete, Update and Merge
DSQLt	# of DT in dynamic SQL and dynamic Cursor
DMLc	# of DC in Select, Insert, Delete, Update and Merge
DSQLc	# of DC in dynamic SQL and dynamic Cursor
EMA	# of multiple assignment exception
ECA	# of complex assignment exception
ESC	# of string Concatenation exception
ETM	# of table missing exception
$\mathbf{BC_a}$	Balance Constant for case a
$\mathbf{BC_b}$	Balance Constant for case b
LC	LOC Constant for case b

on different schemas in dynamic SQL statements ($DSQLdst$) with a multiplying factor of 2 and Ex_i (See Equation 10) for all procedures in the schema divided by a *Balance Constant* (BC_a). Equation 12 is used for calculating the effort estimation for refactoring type (b) by summing up the total lines of code divided by a constant (LC), DML_i (Equation 8), $DSQL_i$ (Equation 9) and Ex_i (Equation 10) for all procedures in the schema divided by a *Balance Constant* (BC_b). In the following, we present an empirical evaluation performed to test the accuracy of these formulas in effort estimation tasks.

4.2 Industrial Case Studies

In this section, we present industrial case studies for evaluating our approach.

We aim at answering the following research questions:

- **RQ1.** How accurate is our approach in capturing object dependencies?
- **RQ2.** How accurate is our approach in effort estimation?
- **RQ3.** How is the runtime performance of our approach?

In the following, we explain our experimental setup. Then, we present and discuss the results. We conclude the section with a discussion of limitations and validity threats regarding our evaluation.

Table 25: 10 schemas used as experimental objects.

Schema	**Total # of Procedures**	**# of Procedures with no Dependencies**	**# of Procedures with Parsing Errors**
Schema 1	163	48	0
Schema 2	149	25	2
Schema 3	340	38	5
Schema 4	171	18	0
Schema 5	177	92	1
Schema 6	325	35	15
Schema 7	316	140	81
Schema 8	157	50	12
Schema 9	203	19	12
Schema 10	163	31	5

4.2.1 Experimental Setup

We conducted our case studies on a large-scale legacy Oracle database. The database facilitates *Customer Relationship Management (CRM)* and *Billing* business applications. Business logic is implemented with the PL/SQL language on the database. They are developed and being maintained by more than 100 developers in a telecommunication operator company. There are 478 schemas, 3,073 tables, 42,396 columns, and 4,462 procedures in the database. We selected 10 schemas as experimental objects. These are the schemas that include the highest number of procedures. They are listed in Table 25. The first column enumerates the schemas. The second column shows the number of procedures for each schema. The third column is the # of procedures that do not depend on any table or view in the schema. The last column shows the number of those procedures that could not be processed by our tool due to either code encryption or errors raised by the parser.

We calculated effort estimations for 10 schemas as listed in Table 25. BC_a, BC_b and LC parameters in equations are set as 16, 40 and 10, respectively, These values are determined based on the feedback obtained from the domain experts. We compared the estimated effort and measured effort in terms of the relative error [106] measure. Our results are presented in the following.

4.2.2 Results and Discussions

88,351 dependencies are found in total. 1,175 of these dependencies are subject to exceptional cases, which are discussed in the following subsection. We could not find dependencies for 1,199 procedures since there is no dependency on tables for these procedures. Also, 192 procedures could not be parsed because 142 of these were encrypted [107] and 50 of the procedures were subject to parser errors. Dependencies that are found by the tool are reviewed and approved by domain experts. A sample snippet of dependency results is shown in Table 23. Therefore, we can conclude regarding *RQ1* that our tool can effectively capture dependencies

if they take place in reachable source code that conforms to the ANTLR PL/SQL grammar.

The calculated effort estimations regarding the two types of refactoring for 10 schemas are listed in Table 26 and Table 27. The first column enumerates the schemas in both tables. The second column provides the effort estimation of our tool in man-days. The third column lists the average of estimations provided by domain experts. We assumed each day is 8 hours long. The fourth column presents the relative error. As an answer for *RQ2*, the overall average accuracy turns out to be 87% based on the results obtained for both refactoring types. The lowest accuracy (60%) and the highest relative error (0.40) are associated with Schema 2 and Schema 10 for the refactoring type (a). Figure 8 depicts the distribution of the relative error for all the estimations. We can see that the variance in relative errors is less regarding effort estimations for refactoring type (b). This would be expected since the amount of effort for type (a) ranges between 5 and 68 man-days, while it ranges between 70 and 310 man-days for type (b). The ratio increases as the absolute value decreases. The median relative errors are close to each other though; they are calculated as 0.15 and 0.10 for types (a) and (b), respectively.

We were able to reach data regarding a refactoring task that was completed 3 years ago in the company. The task involved the migration of procedures in a schema to the application tier (i.e., refactoring type b). The refactored schema resembles (before its refactoring) schema 4 in terms of the number of procedures, their lines of code, and the number of procedures with no dependencies. We learned that the task was completed in 40 days by 3 engineers, i.e., 120 man-days. Tool estimation and average estimations of experts for the corresponding schema and refactoring type are 135 and 117 man-days, respectively. These correspond to accuracy values 88% and 98%, respectively. Although we have data regarding only one instance for such a refactoring task that actually took place, it shows that expert estimations are very accurate and the dependency model which is the output of the tool is accurate for use in effort estimations. This is valuable since

there are usually only a few experts, who are usually busy. Our tool can help automatically and quickly provide estimations.

Table 26: Tool estimations and the mean of expert estimations (in man-days) regarding the refactoring type (a) Migrating a schema to another database.

Schema	Tool Estimation	Expert Estimation	Relative Error
Schema 1	46	45	0.02
Schema 2	3	5	0.40
Schema 3	53	51	0.04
Schema 4	12	14	0.14
Schema 5	11	13	0.15
Schema 6	73	68	0.07
Schema 7	35	35	0.00
Schema 8	14	18	0.22
Schema 9	40	34	0.18
Schema 10	3	5	0.40

We measured the time spent for the analysis process to answer *RQ3*. The entire analysis of the database dependencies and effort estimations took approximately 6 hours. This analysis is performed with commodity hardware (with an Intel i5 10th generation processor and 8 GB memory) on a real large-scale legacy system. Therefore, we conclude that performance of the tool is acceptable for obtaining estimations on refactoring tasks that can take up to hundreds of man-days long. In the following, we discuss the limitations of our approach as well as the validity threats regarding our evaluation.

4.2.3 Threats to Validity and Limitations

Our evaluation relies on expert opinion. To mitigate *conclusion* and *construct validity* threats, we consulted multiple experts and take the mean of their estimations as the ground-truth. Also, there exist several parameters that we tune for calculating estimations. The corresponding formulas and the parameter settings

Table 27: Tool estimations and the mean of expert estimations (in man-days) regarding the refactoring type (b) Migrating business logic from data-tier to application tier.

Schema	Tool Estimation	Expert Estimation	Relative Error
Schema 1	198	200	0.01
Schema 2	62	70	0.11
Schema 3	325	310	0.05
Schema 4	135	117	0.01
Schema 5	129	130	0.11
Schema 6	278	250	0.29
Schema 7	129	100	0.08
Schema 8	115	125	0.04
Schema 9	302	290	0.15
Schema 10	86	75	0.10

can be further improved to increase the accuracy of estimations. Parameters facilitate flexibility to adjust the estimation for different contexts and to perform what-if analysis.

We performed the case studies on the same data-tier application. Although we have applied our approach to 10 schemas, they are part of the same database. Therefore, our evaluation is subject to an *external validity* threat [108]. More case studies can be conducted in different contexts to increase the generalizability of the results.

One can question the way that measurements are performed, concerning *internal validity* threats. In our case study, we used real artifacts without any instrumentation. We cannot share our experimental objects due to confidentiality. However, we share the implementation of our approach. Hence, one can apply our approach and test the validity of results on any PL/SQL program deployed on an Oracle DBMS.

We considered inter-procedure dependencies to be out-of-scope in this work

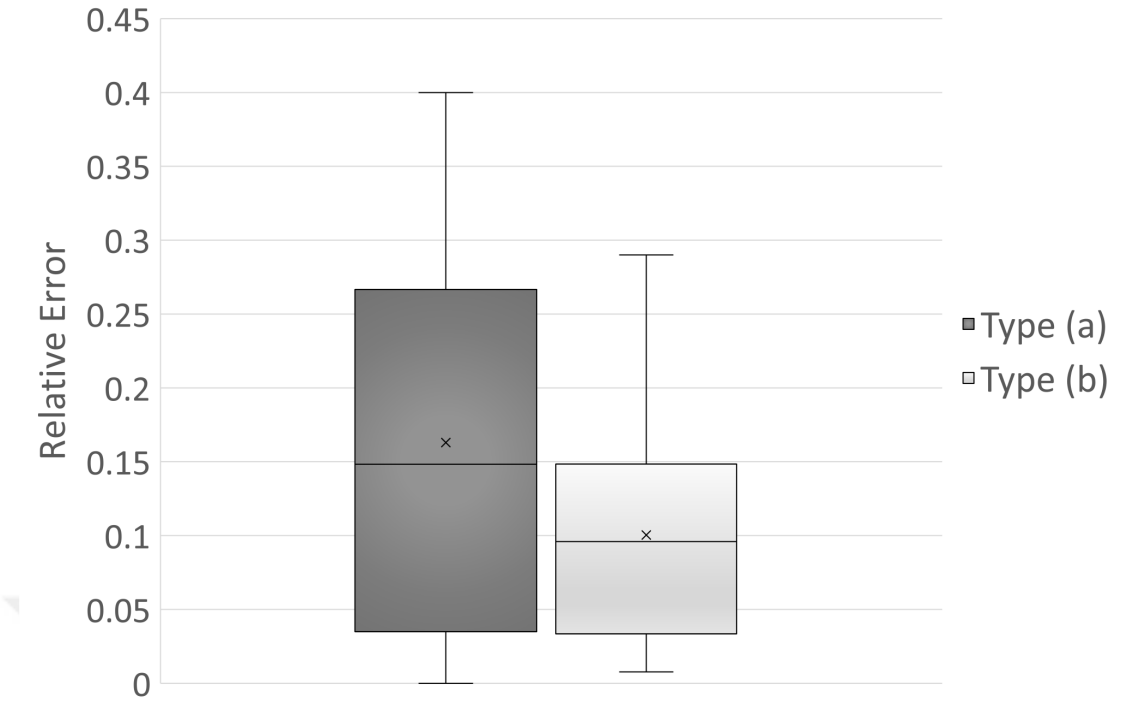


Figure 8: The relative error in estimations regarding the two refactoring types (a) and (b) considered for 10 schemas.

for two reasons. First, inter-procedure dependencies are rare in PL/SQL programs, where procedures are mainly coupled indirectly due to their database operations [13]. Second, procedures that are directly coupled with other procedures are unlikely to be subject to the types of refactoring we consider in this work. For instance, a procedure, which is called by other procedures in the database would not be migrated to another database or the application tier.

There are four exceptional cases, for which our tool lacks precision for detecting dependencies. These cases are listed and described with example code snippets in Table 28. Hereby, the first three represent special cases of dynamic SQL statement creation, where *i)* multiple values are assigned to a variable, which is then used for executing an SQL statement, *ii)* multiple variables are used for the creation of an SQL statement, and *iii)* multiple strings are concatenated for the creation of an SQL statement. The last exceptional case is related to the *Merge* DML. This case occurs when two tables with the same column names are merged together. These exceptional cases can be studied in further detail to increase the precision

of our analysis.

4.3 Related Work and Our Contributions

Database refactoring [16, 17] has received less attention in the literature [18]. However, many of the large, heterogeneous, and highly complex legacy systems are data-intensive systems [19]. Co-evolution of database schemas and application source code is inevitable for these systems [20, 21]. Therefore, change impact analysis and effort estimation activities should also explicitly consider data-intensive systems and databases. Although there have been studies spanning 3 decades in this domain [22], these studies do not address the types of changes that we consider in this work. In the following, we summarize those studies that focus on supporting data-intensive system evolution.

Technical debt [109] related to databases has been pointed out [110] and database normalization efforts have been studied to manage this debt [111]. A framework has been proposed [111] for identifying the debt regarding databases and quantifying the impact thereof. However, the types of refactoring we consider in this work are different than those applied for database normalization. In addition, we introduce a tool that can quantify the impact of refactoring by automated analysis of the data model and the source code that is coupled with this model.

Some of the previous studies [23, 24] rely on Object-Relational Mapping (ORM) tools such as Hibernate [25, 26] to perform dependency and change impact analysis. ORM facilitates a direct mapping between the database entities and objects. Therefore, the data model can be managed directly by manipulating object states instead of executing SQL queries. We consider the use of such technologies out-of-scope in our context, where we analyze module dependencies based on predefined SQL statements in legacy data-intensive systems. In the Ph.D. thesis of Cleve [112], program analysis and transformation techniques have been proposed for supporting the refactoring of data-tier software. There are also two types of refactoring considered in the thesis, which are different than those we consider

in this work. The first one is database platform migration, where a DBMS is completely replaced with another one. This modification is handled by generating wrappers that act as an adapter [113] between the new DBMS and the legacy application tier. The second type of refactoring is database schema refactoring, where the organization of tables and their relations are modified. Hereby, the types of modifications are similar to those that might be applied for database normalization [111], where for instance, a table is split into two tables. Embedded SQL statements in COBOL programs are analyzed to reveal the impact of these modifications on the source code. However, dynamic analysis is used for analyzing dynamically created queries. That is, execution traces at run-time are collected and analyzed to be able to identify and inspect the impact on these types of queries. Although dynamic analysis provides high precision, it is only limited to the run-time context and the execution scenarios, which are not complete. In our work, we focus on PL/SQL programs and we rely only on static analysis to capture dependencies regarding both statically and dynamically created SQL queries. The result of this analysis constitutes the main input for effort estimation.

There have been several other studies similar to that of Cleve [112] for performing dependency and impact analysis regarding modifications at the data-tier. They all analyze SQL statements to perform such analysis; however, they do not address the analysis of various types of statements that can be processed by our tool. Table 29 summarizes and compares the capabilities of the current techniques. The first column lists the types of SQL statements that lead to dependencies to be handled. Descriptions regarding these statement types are listed in Table 30. The second column shows the capabilities of our tool. The third column lists the capabilities of *Oracle Metadata*. This is actually an *undocumented* data dictionary view that takes part in Oracle DBMS. The view is DBA_DEPENDENCY_COLUMNS [114] and it is used for identifying column-level dependencies by the practitioners. We refer to this view as *Oracle Metadata*, which provides a useful baseline for dependency analysis as the state-of-the-practice.

However, it falls short in identifying dependencies on dynamically created Cursor structures and SQL statements.

There are studies in the literature for handling such statements as listed in Table 29. However, they present cases with statements that use single tables only. Moreover, there is no study that can handle SQL statement types such as *Single/Multi Table Dynamic SQL with SQL Result*, *Cursor with String* and *Cursor with Variable* to the best of our knowledge. These are mainly the types of queries, where the output of an SQL query is used for constructing another SQL query. These queries must be recursively analyzed for complete and precise dependency analysis.

In Table 29, a ✓and X signs represent the existence of the capability and lack thereof, respectively. Some cells are marked as *Not Applicable* (NA) since some of the SQL statements use the Cursor construct, which is specific to PL/SQL. Finally, some cells are marked as ? since there is no information or evidence provided for the corresponding capability. Our tool can address all the SQL statement types listed in Table 29, which can not be addressed altogether even when all the other existing tools are used in combination.

Table 28: Examples of Exceptional Cases.

Exception Case	**Description**	**Example**
Multiple Assignment (EMA)	There are more than one assignment to the same variable.	v_Mult_Assgn :='SELECT NAME FROM CUSTOMER'; v_Mult_Assgn :='SELECT NAME FROM NEW_CUSTOMER';
Complex Assignment (ECA)	Assignment done by using multible variables or strings.	v_Comp_Assgn := 'SELECT C.NAME FROM ' || v_Table || v_Where;
String Concatenation (ESC)	Assignment done by using string concatenations.	v_Conc :='SELECT ' || 'NAME ' || 'FROM CUSTOMER';
Table Missing (ETM)	Table name cannot be determined in a Merge DML, where columns of different tables being used with the same name.	MERGE INTO TABLE_PARAM E USING DUMMY_PARAM H ON (e.NID = h.NID) WHEN MATCHED THEN UPDATE SET e.NID = h.NID WHEN NOT MATCHED THEN INSERT (NID, TSQL) VALUES (h.NID, h.TSQL);

Table 29: Comparison with other tools/studies that perform table and column level dependency analysis. Descriptions regarding the listed types of SQL statements are provided in Table 30.

Type of SQL statement	**Our Tool**	**Oracle Metadata**	**[24, 33, 34, 35, 36]**	**[37]**	**[28]**	**[38]**	**[39]**
ST DML Statement	✓	✓	✓	✓	✓	✓	✓
MT DML Statement	✓	✓	?	?	?	?	✓
Subquery	✓	✓	?	?	?	?	✓
Explicit Cursor	✓	✓	NA	?	NA	NA	✓
Cursor with String	✓	X	NA	X	NA	NA	X
Cursor with Variable	✓	X	NA	X	NA	NA	X
ST D-SQL with String	✓	X	✓	X	✓	✓	X
ST D-SQL with Variable	✓	X	✓	X	✓	✓	X
ST D-SQL with SQL Result	✓	X	X	X	X	X	X
MT D-SQL with String	✓	X	?	X	?	?	X
MT D-SQL with Variable	✓	X	?	X	?	?	X
MT D-SQL with SQL Result	✓	X	X	X	X	X	X
Programming Language	PL/SQL	SQL, PL/SQL	Java	PL/SQL	C#	COBOL	SQL

Table 30: Types of SQL statements with a description for each (DML: Data Manipulation Language, ST: Single Table, MT: Multi Table, D-SQL: Dynamic SQL).

SQL Statement Types	**Description**
ST DML Statement	A simple DML that uses a single table.
MT DML Statement	A DML that uses more than one table.
Subquery	A query that is embedded in a DML
Explicit Cursor	A Cursor is a pointer to a memory area that stores the results regarding an SQL query. An explicit cursor is defined within a DECLARE section.
Cursor with String	A Cursor initialized with a String that contains a query.
Cursor with Variable	A Cursor initialized with a variable that contains a query.
ST D-SQL with String	A Dynamic SQL statement is parsed and executed at runtime.
ST D-SQL with Variable	A Dynamic SQL statement instantiated with a variable.
ST D-SQL with SQL Result	A Dynamic SQL statement instantiated with the results obtained from another SQL query
MT D-SQL with String	The same as *ST D-SQL with String* except the use of multiple tables.
MT D-SQL with Variable	The same as *ST D-SQL with Variable* except the use of multiple tables.
MT D-SQL with SQL Result	The same as *ST D-SQL with SQL Result* except the use of multiple tables.

CHAPTER V

DATA MODEL EXTENSION IMPACT ANALYSIS

In this chapter, we propose a novel approach by using Siamese Networks (SN) and a tool for modeling implicit relations among database tables and automatically provide an evaluation regarding the impact of changes applied on these tables. That tool identified potentially affected tables by calculating their distance with the extended table based on unique column names that are commonly taking place in them. We train a model that learns implicit relations among tables. The training data involves a sample set of table pairs with their unique column names. These pairs are labeled by domain experts as related or not. The trained model predicts the existence of relations and as such a potential change impact between any given pair of tables.

We conduct an industrial case study to evaluate the accuracy of our approach and compare it with respect to our recently proposed technique [40], which is considered as the baseline. Our subject system is a real system from the telecommunications domain, including 120 tables. Results show that our predictions are more accurate than those of the baseline technique. The model is both efficient and effective, even in the presence of limited training data. It achieves the average F1 score of 96.1% in prediction. In addition, training and validation of the model take less than a minute on commodity hardware (with an Intel i5 10th generation processor and 8 GB memory). We also develop a tool and observe that our approach is helpful for developers. It guides novice developers to perform accurate impact analysis without getting support from domain experts. Although the overall effort depends on specific changes, impact analysis does not take much time for a domain expert in general. However, domain experts are rarely available due to their scarcity and high-intensity workload. Novice developers might have to wait

for days until they can start getting any support. Our tool reduces this waiting time or eliminates it, if possible.

The remainder of this chapter is organized as follows. We explain our approach and its implementation in the following section. In Section 5.2, we present an industrial case study. Finally, summarize the related studies and our contributions in Section 5.3.

5.1 Overall Approach

The overall approach, which involves three process groups and several steps, is depicted in Figure 9. The first process group (1) is related to the preparation of the dataset and optimization of the SN model. The second process group (2) involves the distance calculation for all pairs of tables based on the optimized and trained model. Finally, the last process group (3) is applied for retrieving the list of tables that can be potentially affected, based on a given table that is to be extended in the data model and the distance calculations performed in the previous step. We implemented our approach for Oracle Database Management Systems[1] and used Keras [115] for SN implementation. We explain the 3 process groups and their steps, including the implementation details in the following subsections.

5.1.1 Dataset Preparation and Model Construction

Dataset preparation is performed in two sub-steps. The first sub-step involves the labeling of data by the domain experts. Rather than labeling each pair of tables separately, we asked them to identify groups of tables such that an extension of one of the tables in a group might have an impact on the other tables in the same group. That is, pairs of tables within a group are deemed to be related. The second sub-step involves the derivation of features from all the tables. This sub-step is performed by using the same method used in our previous study [40]. Algorithm 4 lists a pseudo-code that is used for dataset preparation. Hereby, we

[1] `https://www.oracle.com/database/`

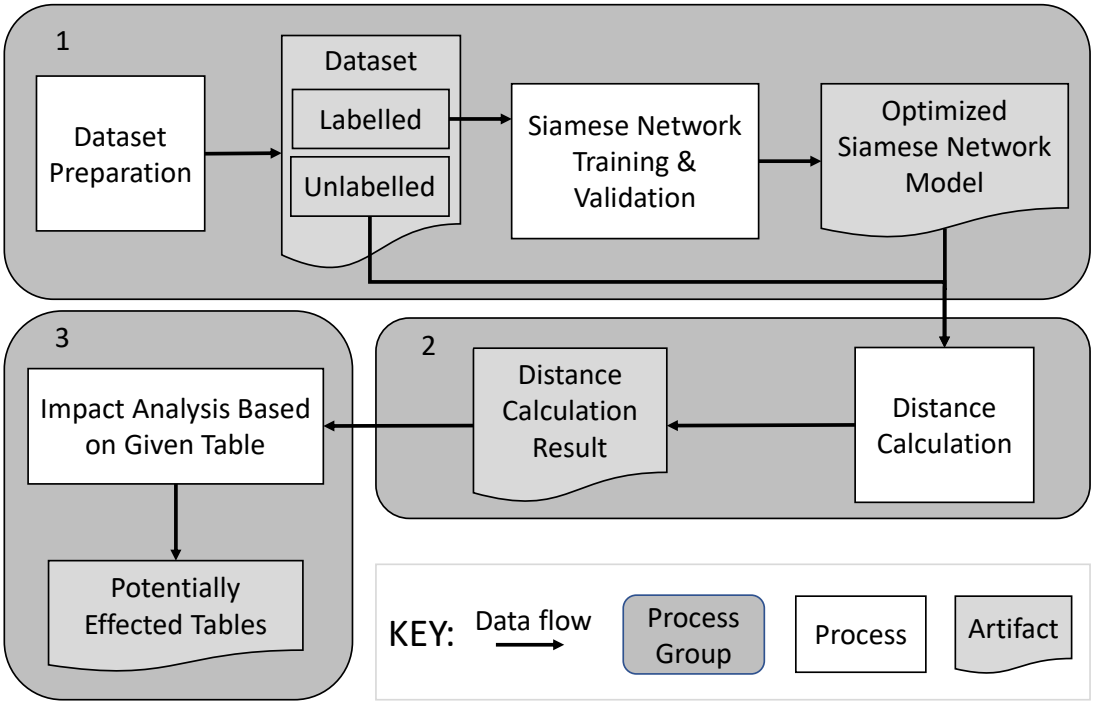


Figure 9: The overall process.

extracted all unique column names and marked them for each table as 1 if they exist in the table or 0 if they do not.

Algorithm 4 Populating the table that keeps table-column mappings (See Table 31)

```
T ← all tables in the dataset
for each t in T do
    insert a row for t    ▷ all the values in the new row are set as 0 by default
    C_t ← all columns in t
    for each c in C_t do
        update the value corresponding to cell [t, c] as 1
    end for
end for
```

This information is appended with the identification of the group for each table (i.e., Class) as determined by the domain experts. This is used for SN training and validation. A representative snippet from the created dataset is shown in Table 31.

Dataset preparation is followed by the model construction. SN is employed in this step. Positive and negative samples are prepared randomly in the form of table pairs based on the labels (i.e., Class information) in the dataset. These

Table 31: A representative sample snippet from the prepared dataset.

Table	Class	Column 1	Column 2	...	Column n
TABLE 1	0	1	1	...	0
TABLE 2	0	1	1	...	1
TABLE 3	1	0	0	...	0
TABLE 4	2	0	0	...	1
...	...	...	...	...	...
TABLE n	11	0	0	...	0

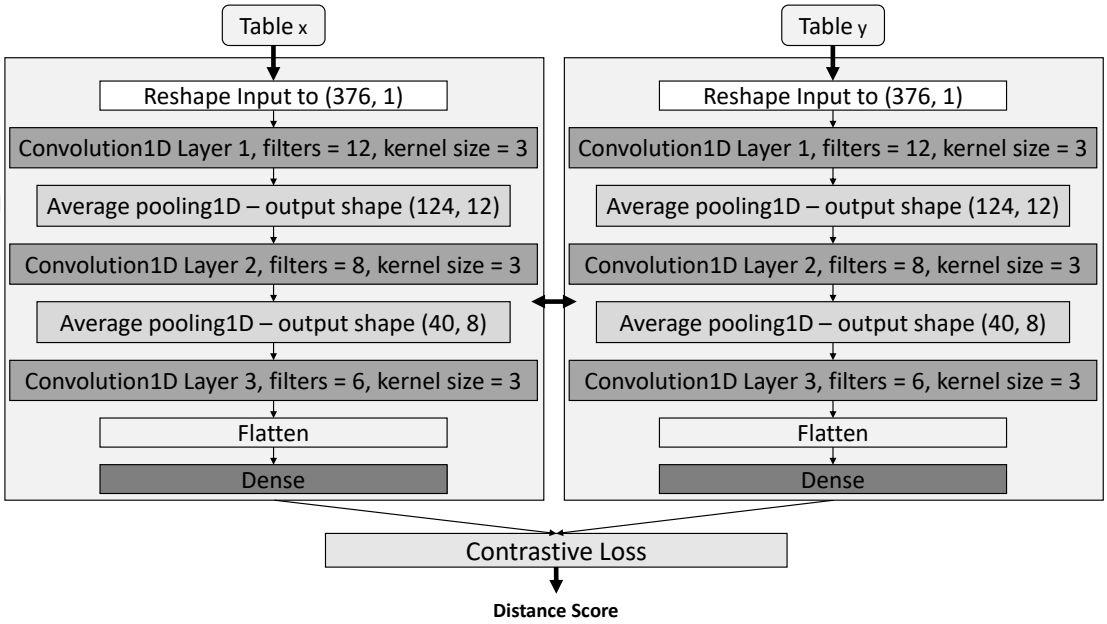


Figure 10: The proposed SN architecture.

samples are provided as input to the SN model for training and validation. The proposed SN architecture is shown in Figure 10. SN hyper-parameters such as *kernel*, *filters*, *number of layers* tuned empirically with several trials. Settings of these parameters are listed in Figure 10. *The number of epochs* and *batch size* are selected as 20 and 16, respectively. Average pooling is employed to avoid over-fitting [116]. Stratified 5-fold cross-validation [117] is used for validating our model.

5.1.2 Distance Calculation

Distance (dissimilarity) and similarity scores are inversely proportional and are used interchangeably in the literature. A distance score for each pair of tables is

calculated by using the optimized and trained SN model. The score takes values between 0 and 1. A distance score that is closer to zero is interpreted as the pair of tables being similar to each other, indicating an implicit relation. On the other hand, a distance score that is closer to 1 indicates that tables are very different and relation is unlikely.

Table 32 presents a sample snippet from the table expansion impact analysis results. Each row of this table is considered as a record in our results. Each record consists of the names of two tables and their pair-wise distance score predicted by the SN model.

Table 32: A sample snippet from distance calculation results.

Main Table	**Depended Table**	**Distance Score**
TABLE 1	TABLE 2	0.148
TABLE 3	TABLE 51	0.461
TABLE 5	TABLE 9	0.072
TABLE 11	TABLE 18	0.160
TABLE 45	TABLE 86	0.497
TABLE 106	TABLE 109	0.076

5.1.3 Impact Analysis Based on a Given Table

The last process group involves a single step that retrieves the list of tables that can be potentially affected as a result of a given table that is to be extended. Hereby, results are presented in the same format as shown in Table 32 by querying the records created by the second process group.

A demonstration of this step of the process is available online. We prepared a Web page[2], as the user interface of the tool developed for the third process. The tool works on the results obtained from the second process group, which are populated based on our case study as described in the following section. The web page shows table pairs for which the distance score is less than 0.5. The first column in the list of results shows the table to be extended as the main table. The

[2] `https://ersinersoy.github.io/dmeiausn/impact-analysis.html`

second column shows the table that will be affected if the main table is extended. The last column shows the distance score. Table names are anonymized due to confidentiality.

5.2 Industrial Case Study

In this section, we present an industrial case study for performing data model extension impact analysis using SN. We applied our approach on a *Customer Relationship Management (CRM)* application. This is a large-scale legacy application that uses an Oracle database. It is one of the main business support systems developed and maintained by Turkcell[3], which is the largest GSM operator in Turkey.

We aim at answering the following research questions:

- **RQ1.** How accurate is our data model extension impact analysis?
- **RQ2.** How does the accuracy of SN compare to the accuracy of the baseline approach?
- **RQ3.** How is the run-time performance of our approach?

The first two research questions are related to the accuracy of impact analysis. The last question is concerned with the run-time performance. In the following, we first explain the experimental setup and the properties of our dataset. Then, we present and discuss the results. We conclude the section with a discussion of threats to validity.

5.2.1 Experimental Setup

There are thousands of tables in the CRM application data model. There is no explicit or logical grouping of these tables to identify which of them will be affected if a table is extended. We asked domain experts to create a labeled dataset for a subset of these tables as discussed in the previous section. They came up with 12

[3] `http://www.turkcell.com.tr`

groups and randomly selected 10 tables for each group. This grouping is considered as the ground-truth for impact analysis such that if one of the tables is extended, the remaining 9 tables in the same group are assumed to be impacted. In total, 120 tables are selected and labeled manually. We extracted 376 unique columns from these tables. We performed 14,400 pairwise distance calculations (i.e., 120×120) by using the optimized and trained SN model. Finally, we removed records, which have a distance score equal to or higher than the 0.5 threshold value. 1459 records remained after this filtering. The dataset and the source code of our tool are shared on a public GitHub repository[4].

5.2.2 Evaluation Metrics

In the following, we introduce a set of definitions followed by the equations regarding the set of metrics considered for evaluation.

- **TN (True Negative):** The number of cases where a table is not listed among the tables that are impacted and there is indeed no impact for that table.
- **FP (False Positive):** The number of cases where a table is listed among the tables that are impacted although there is no impact for that table.
- **TP (True Positive):** The number of cases where a table is listed among the tables that are impacted and there is indeed an impact for that table.
- **FN (False Negative):** The number of cases where a table is not listed among the tables that are impacted although there is an impact for that table.

$$Accuracy = \frac{TN + TP}{TN + FP + TP + FN} \tag{13}$$

$$Precision = \frac{TP}{TP + FP} \tag{14}$$

[4] `https://github.com/ersinersoy/dmeiausn`

$$Recall = \frac{TP}{TP + FN} \tag{15}$$

$$F1\ Score = \frac{2 * Precision * Recall}{Precision + Recall} \tag{16}$$

Due to the unbalanced nature of the dataset, the accuracy score (See Equation 13) does not lead to meaningful results for our experimental setup. 13,200 table pairs out of a total of 14,400 pairs fall into the TN category in our dataset. Hence, it is very easy to reach very high accuracy scores regardless of the employed technique, strategy, and parameter settings. Instead, we relied on the F1 score metric (See Equation 16), which is commonly used for balancing precision (See Equation 14) and recall (See Equation 15) scores [118].

We used the threshold value 0.5 as it is generally used [119] for SN. Distance scores lower than 0.5 are considered as similar, i.e., the corresponding tables are related, and changing one can impact the other. Distance scores that are equal to or greater than 0.5 are considered as dissimilar, i.e., the corresponding tables are not related and as such, there is no change impact expected.

We compared the results with those obtained with a baseline technique [40] that employs cosine similarity metric [120] to calculate the similarity between each pair of tables. Hereby, a threshold value has to be determined such that change impact is expected if the similarity is above the determined threshold. We calculated F1 Scores achieved with the whole dataset for various threshold values as listed in Table 33.

We set the threshold as 0.6 for which the best F1 score can be obtained with the baseline approach when the whole dataset is considered. We employed stratified 5-fold cross-validation [117] for evaluation. Hereby, the dataset is randomly divided into 5 subsets such that each subset has the same proportion of related table pairs and unrelated table pairs. We repeated tests 5 times and calculated the F1 score for each test. One of the 5 subsets is used in each test as the test set. The remaining 4 subsets are used for training the SN model. Note that the baseline approach does not require any training and it can be directly applied to the test set. We also repeated 5-fold cross-validation 100 times, where the dataset is divided into 5

Table 33: F1 Score achieved with cosine similarity for various threshold values.

Threshold value	**F1 Score**
0	15.38%
0.05	46.91%
0.1	55.38%
0.15	64.00%
0.2	69.08%
0.25	75.52%
0.3	78.59%
0.35	79.63%
0.4	82.53%
0.45	87.27%
0.5	91.20%
0.55	94.00%
0.6	**95.15%**
0.65	91.77%
0.7	85.96%
0.75	72.20%
0.8	65.02%
0.85	52.03%
0.9	41.27%
0.95	32.40%
1	18.46%

subsets randomly in each of these repetitions. We present and discuss the results in the following.

5.2.3 Results and Discussions

Figure 11 depicts a box plot that shows the distribution of F1 scores obtained with both SN and the baseline approach over 100 repetitions of 5-fold cross-validation performed with our dataset. Results regarding a single run can be viewed online[5], where F1 scores obtained with SN and the baseline approach are 95.86% and 94.46%, respectively. In overall tests (See Figure 11), the minimum, mean and maximum F1 scores obtained with SN are **92.35%**, **96.1%** and **98.4%**, respectively. On the other hand, these values are **91.93%**, **95.79%** and **98.26%**, respectively for the baseline approach. Therefore, we can conclude regarding *RQ1* and *RQ2* that SN leads to high accuracy in change impact analysis (above 92%) and it performs better than the baseline approach. Since the dataset is relatively small and it does not involve complex dependencies, SN performs only slightly better than the baseline approach. We expect that the difference between the two would be more significant for more complex and larger datasets.

We measured the time spent for the analysis process to answer *RQ3*. SN training and validation take less than one minute. Pairwise distance calculation for all the table pairs in the dataset takes approximately ten minutes. Hence, both the training and prediction times are inconsiderable. Finding data model extension impacts on other tables is a trivial task for domain experts but it is not the case for novice developers. Therefore, novice developers have to request help from domain experts frequently. However, this is not an easy request for domain experts since there are usually a few domain experts in organizations, who are almost always occupied. Novice developers might have to wait for days to reach a domain expert. Therefore, our approach and tool can significantly reduce the time and effort required for change impact analysis.

[5]dmeiausn.ipynb in the GitHub repository

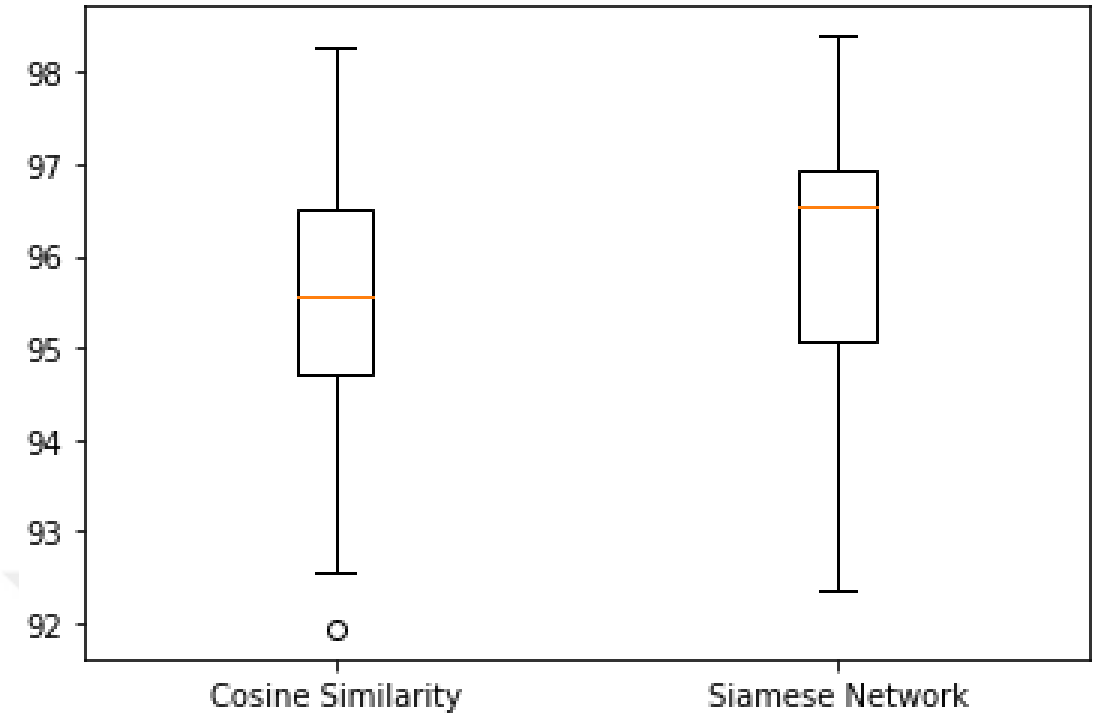


Figure 11: Comparison of F1 scores over 100 repetitions of 5-fold cross-validation.

5.2.4 Threats to Validity and Limitations

There exist internal and construct validity threats because of the fact that our evaluations inevitably rely on expert opinions. Their opinions are consulted for preparing the dataset, i.e., labeling the table pairs. There is also an external validity threat since we performed a single case study from a particular domain. The study provides relevant results since the subject system is a real legacy system from the industry. However, the number of experimental objects must be increased to improve the generalizability of the results. We should also note that our approach aims at detecting the potential impact of data model extensions. It might not be necessary to make changes in tables after every data model modification even though the corresponding tables are related/similar. This decision depends on the system, the reason for change, and the business processes.

5.3 Related Work and Our Contributions

An analysis of the literature reveals that the majority of studies on dependency management and change impact analysis have focused on the analysis of source code [32]. There are only a few studies [21, 13, 121] that extend the scope of the analysis on parts of the system other than source code and the data model of data-intensive systems in particular. On the other hand, the majority of the studies that consider changes on database models [28], focus on the impact of these changes on the source code [122, 34, 123, 35, 124, 24, 20], test cases [125] or both of these artifacts [23, 37, 36]. In this study, we do not aim at relating database elements with source code or test code. Our goal is to reveal implicit relations among database tables and analyze the impact of changes in database tables on other tables.

There are tools that automatically adapt and synchronize queries and views after schema changes [22, 39]. These tools apply automated transformations that keep these database elements consistent with the tables they rely on. However, such transformations can be defined only when there exist explicit and structural relations among them. Change impact analysis can also be easily automated in this case. Full automation is not trivial in our case since we do not assume the existence of any explicit and structural relationship between the modified table and the impacted tables. We have two more differences with respect to related studies in this domain. First, we focus on the impact of changes in tables on other tables rather than queries and views. Second, we focus on extensions (i.e., adding a new column to a table) rather than deletions and modifications. The impact of table extensions is neglected in the literature unlike other types of changes [29].

Change impact analysis can be supported with visualization. The lack of such a support is one of the acknowledgment problems regarding database maintenance [27]. There are only a few studies [126] for visualizing changes on database models to allow interactive and visual analysis. This is also an interesting line of

research with relevant challenges to be addressed. However, we consider visualization out of the scope of this study. Our goal is to directly pinpoint database tables that might be affected by an extension of a particular table. Visualization can further improve this process with a better support for system comprehension.

We recently introduced a tool [40] for analyzing the impact of table extensions on other tables. We consider that tool as the baseline for evaluating the approach proposed in this work, which is based on SN. SN have been so far used for image recognition [127, 128], face recognition [129, 130] and signature verification [131, 132]. To the best of our knowledge, this is the first time that SN are used for detecting data model extension impact analysis.

CHAPTER VI

CONCLUSIONS

Nowadays, the majority of enterprise applications employ data-tier software and they are highly coupled with databases. Constant maintenance of database systems is needed since inevitable changes become necessary in time. Unfortunately, these changes may hinder the alignment of architectural modules and may lead to inconsistencies that are exposed to users as failures. In this thesis, we proposed techniques and tools to deal with three challenging maintenance activities that are related to data-tier software: *i)* correct placement of a new object (e.g., procedure, function, package) in data-tier software, *ii)* impact of data model changes on programs that rely on the data model, and *iii)* impact of data model extensions on the data model itself. The usual practice is to use domain expert knowledge during these maintenance activities. However, domain experts are rarely available, if any remains at all. Our main goal is to provide tool support for the three challenging maintenance activities, reduce the need for domain expert knowledge, reduce the overall effort and reduce the number of errors made in the process.

To address the first challenge, we proposed a novel approach to automatically pinpoint the correct placement of a new object among architectural modules. In particular, we adopted a classification approach to determine the placement of PL/SQL objects among candidate schemas of an enterprise application. An artificial neural network model is used to implicitly represent the module decomposition of the application. We performed industrial case studies with two large-scale legacy systems from the telecommunications domain. We showed that our model can pinpoint the relevant schema for a new object with an accuracy of 86.7% and 89% for these two subject systems. We also conceived a simple approach, where

an object is assumed to belong to the schema that includes most of the other interdependent objects. This baseline approach reached 55.5% and 57.4% accuracy for the same subject systems, respectively. The proposed approach can be applied to other systems, possibly from various domains to improve the generalizability of our results. One can also consider the usage of other classification algorithms to improve the accuracy of prediction.

To address the second challenge, we introduced an approach and a tool for analyzing table and column dependencies of PL/SQL programs that are developed as data access tiers of business software. The tool parses PL/SQL programs to analyze the dependencies of procedures on tables and views in the database. Unlike prior studies, our tool can analyze queries that are created dynamically and that use multiple tables as well as PL/SQL-specific features. We evaluated its accuracy with an industrial PL/SQL program from the telecommunications domain. The accuracy of the tool is reviewed and approved by domain experts. We also used the tool for effort estimation as an additional case study. We estimated effort for two common refactoring types by using the output of the tool. We also collected effort estimations from domain experts for the same refactoring types. We observed high consistency between the automated estimations and manual estimations in the case studies. Based on these observations, we can say that our tool can perform extensive table and column level dependency analysis on PL/SQL programs, which can be exploited for many tasks other than change impact analysis such as software architecture recovery, effort estimation, and fault localization. Our tool can still be improved in future work for handling exceptional cases in PL/SQL data manipulation mechanisms such as multiple assignments, complex assignments, string concatenation, and missing table information. Details regarding these cases are listed in Table 28.

To address the third challenge, we introduced a novel approach and a tool for automated extension impact analysis. Our analysis focuses on the impact of extensions of database tables on the other tables. We employed a Siamese network

model for this analysis. We performed an industrial case study based on a real legacy system. We compared the accuracy of our approach with respect to that of a recently proposed baseline technique. Results show that our predictions are more accurate, achieving the mean F1 score of 96.1%. Our tool support can guide novice developers in scoping their impact analysis by highlighting only relevant tables in case of data model extensions.

APPENDIX A

TOP 10 ACCURACY RESULTS FOR EXTENDING PL/SQL PROGRAMS

Top ten accuracy values achieved with the datasets for *CRM* application *CRM-T*, *CRM-TV*, *CRM-TVPF* and *CRM-ALL* are depicted in Table 34. There are two parts in each cell of Table 34 for each dataset. The first part shows the best hyperparameters for the top 10 results. The second part shows the results for each of these settings. Hereby, the first row of the second part lists the accuracy scores. The second row of the second part list macro scores of precision, recall, and F1. Finally, the third row of the second part list weighted scores of precision, recall, and F1. Also, top ten accuracy values achieved with the datasets for *CMS* application *CMS-T*, *CMS-TVPF* and *CMS-ALL* are depicted in Table 35.

Table 34: Top 10 accuracy values obtained with *CRM* case study datasets, and the corresponding ANN hyperparameters

DATASET	Top 10(Hidden Layers & # of Neurons, Alpha value, Activation Function, Accuracy, Macro Scores of Precision & Recall & F1, Weighted Scores of Precision & Recall & F1)									
	Top 1	Top 2	Top 3	Top 4	Top 5	Top 6	Top 7	Top 8	Top 9	Top 10
CRM-T	(500, 1750) 0.0001 Sigmoid **83.4** 61.6 - 50.6 - 52.8 80.6 - 83.4 - 81.0	(750, 250) 0.0005 Sigmoid 83.0 61.2 - 51.6 - 53.5 80.6 - 83.0 - 81.0	(750, 50) 0.002 Sigmoid 82.8 64.6 - 50.0 - 53.1 80.8 - 82.8 - 80.7	(2000, 150) 0.0005 Sigmoid 82.7 70.7 - 54.1 - **57.6** 82.7 - 82.7 - **81.2**	(500, 1750) 0.0015 Sigmoid 82.7 60.5 - 50.8 - 52.6 80.4 - 82.7 - 80.5	(750, 500) 0.001 Sigmoid 82.7 56.5 - 50.6 - 52.2 79.3 - 82.7 - 80.4	(2750, 150) 0.002 Sigmoid 82.6 62.4 - 52.7 - 55.1 80.7 - 82.6 - 80.8	(750, 2250) 0.0005 Sigmoid 82.6 58.6 - 49.4 - 51.3 79.5 - 82.6 - 80.1	(750, 250) 0.0015 Sigmoid 82.5 61.2 - 51.0 - 53.5 80.1 - 82.5 - 80.5	(500, 2500) 0.0015 Sigmoid 82.4 60.5 - 50.5 - 52.6 80.1 - 82.4 - 80.3
CRM-TV	(750, 2750) 0.0001 Sigmoid **83.8** 64.0 - 53.0 - 55.1 81.5 - 83.8 - 81.5	(750, 750) 0.0005 Sigmoid 83.4 60.6 - 51.7 - **53.3** 81.0 - 83.4 - 81.2	(500, 1000) 0.0001 Sigmoid 83.2 61.3 - 53.8 - **55.3** 81.5 - 83.2 - **81.6**	(750, 3000) 0.001 Sigmoid 83.2 59.9 - 52.6 - 54.0 80.4 - 83.2 - 81.1	(750, 150) 0.0001 Sigmoid 83.1 62.0 - 53.4 - 55.2 81.4 - 83.1 - 81.4	(750, 2250) 0.0001 Sigmoid 83.1 63.2 - 52.3 - 54.4 81.1 - 83.1 - 81.1	(750, 150) 0.001 Sigmoid 83.1 61.7 - 51.9 - 53.7 81.2 - 83.1 - 81.2	(500, 2750) 0.001 Sigmoid 83.0 62.5 - 53.1 - 55.0 81.0 - 83.0 - 81.2	(500, 2000) 0.0005 Sigmoid 83.0 62.1 - 53.4 - 54.9 81.5 - 83.0 - 81.3	(750, 150) 0.002 Sigmoid 83.0 61.9 - 51.6 - 53.4 81.1 - 83.0 - 81.0
CRM-TVPF	(50, 2000) 0.001 Sigmoid **84.0** 64.9 - 56.7 - **58.7** 82.6 - 84.0 - **82.7**	(200, 2250) 0.0005 Sigmoid 83.8 64.8 - 55.3 - 57.6 82.1 - 83.8 - 82.3	(100, 1000) 0.0005 Sigmoid 83.7 62.2 - 55.2 - 56.9 81.9 - 83.7 - 82.2	(150, 3000) 0.001 Sigmoid 83.7 63.5 - 54.8 - 56.8 82.2 - 83.7 - 82.2	(200, 1750) 0.0005 Sigmoid 83.7 64.6 - 55.8 - 57.6 82.5 - 83.7 - 82.4	(150, 1500) 0.001 Sigmoid 83.6 61.8 - 54.2 - 55.7 81.7 - 83.6 - 82.0	(150, 2500) 0.0015 Sigmoid 83.6 63.4 - 53.8 - 55.9 82.1 - 83.6 - 82.0	(100, 2000) 0.002 Sigmoid 83.6 61.4 - 54.1 - 55.8 81.7 - 83.6 - 82.0	(200, 2000) 0.0005 Sigmoid 83.6 64.8 - 55.6 - 58.0 82.2 - 83.6 - 82.3	(200, 2250) 0.0001 Sigmoid 83.6 64.7 - 55.3 - 57.4 82.0 - 83.6 - 82.1
CRM-ALL	(500, 750) 0.0001 Sigmoid **86.7** 82.4 - 67.2 - **72.0** 86.6 - 86.7 - **85.8**	(750, 1250) 0.001 Sigmoid 86.2 80.0 - 66.2 - 70.4 86.0 - 86.2 - 85.2	(750, 200) 0.001 Sigmoid 86.2 80.4 - 64.2 - 68.9 85.4 - 86.2 - 84.7	(500, 250) 0.0001 Sigmoid 86.1 79.2 - 67.2 - 70.6 85.7 - 86.1 - 85.1	(750, 3000) 0.0001 Sigmoid 86.1 80.5 - 66.4 - 70.5 86.1 - 86.1 - 85.0	(750, 200) 0.0001 Sigmoid 86.1 80.2 - 66.0 - 70.3 85.5 - 86.1 - 84.9	(500, 2000) 0.0001 Sigmoid 86.1 82.1 - 65.3 - 69.9 86.2 - 86.1 - 84.9	(500, 1000) 0.0001 Sigmoid 86.0 79.2 - 65.3 - 69.6 85.5 - 86.0 - 84.9	(500, 1250) 0.0001 Sigmoid 86.0 77.2 - 64.9 - 68.7 85.2 - 86.0 - 84.8	(500, 3000) 0.0001 Sigmoid 86.0 76.0 - 64.5 - 67.8 84.8 - 86.0 - 84.6

Table 35: Top 10 accuracy values obtained with *CMS* case study datasets, and the corresponding ANN hyperparameters

DATASET	Top 10(Hidden Layers & # of Neurons, Alpha value, Activation Function, Accuracy, Macro Scores of Precision & Recall & F1, Weighted Scores of Precision & Recall & F1)									
	Top 1	Top 2	Top 3	Top 4	Top 5	Top 6	Top 7	Top 8	Top 9	Top 10
CMS-T	(50, 750) 0.0001 Sigmoid	(250, 2250) 0.0001 Sigmoid	(3000, 2000) 0.0001 Sigmoid	(50, 200) 0.0001 Sigmoid	(50, 150) 0.0001 Sigmoid	(50, 2500) 0.0015 Sigmoid	(3000, 250) 0.0001 Sigmoid	(50, 100) 0.0001 Sigmoid	(50, 1250) 0.0001 Sigmoid	(50, 2250) 0.0001 Sigmoid
	87.0 76.3 - 65.3 - 68.3 86.0 - 87.0 - 85.9	87.0 75.3 - 64.4 - 67.2 85.8 - 87.0 - 85.7	87.0 73.1 - 62.5 - 65.2 84.7 - 87.0 - 85.2	86.9 76.8 - 65.9 - **68.9** 86.3 - 86.9 - **86.0**	86.9 76.6 - 65.5 - 68.5 86.2 - 86.9 - 85.9	86.9 74.9 - 66.1 - 68.2 86.0 - 86.9 - 85.9	86.8 73.3 - 63.6 - 65.8 85.1 - 86.8 - 85.3	86.7 75.1 - 65.3 - 68.0 85.9 - 86.7 - 85.8	86.7 74.2 - 65.3 - 67.8 85.8 - 86.7 - 85.7	86.7 73.4 - 65.5 - 67.7 85.8 - 86.7 - 85.8
CMS-TVPF	(1500, 150) 0.0005 Tanh	(1750, 150) 0.0001 Tanh	(1500, 150) 0.0001 Tanh	(1750, 250) 0.002 Tanh	(750, 150) 0.0005 Tanh	(750, 150) 0.001 Tanh	(1500, 50) 0.0005 Tanh	(1750, 150) 0.002 Tanh	(750, 500) 0.0001 Tanh	(750, 750) 0.002 Tanh
	87.4 77.2 - 67.8 - **70.4** 87.0 - 87.4 - **86.7**	87.3 77.6 - 67.2 - 70.2 86.9 - 87.3 - 86.6	87.2 75.6 - 66.5 - 68.9 86.5 - 87.2 - 86.3	87.1 77.2 - 67.9 - 70.0 87.1 - 87.1 - 86.5	87.1 76.3 - 66.8 - 69.9 86.6 - 87.1 - 86.4	87.1 76.8 - 66.9 - 69.8 86.8 - 87.1 - 86.4	87.1 76.5 - 67.3 - 69.7 86.8 - 87.1 - 86.4	87.1 77.3 - 67.2 - 69.6 87.2 - 87.1 - 86.5	87.1 75.8 - 66.2 - 69.1 86.5 - 87.1 - 86.3	86.9 76.1 - 66.6 - 69.5 86.6 - 86.9 - 86.3
CMS-ALL	(1000, 500) 0.0001 Sigmoid	(1750, 1000) 0.0001 Sigmoid	(500, 1500) 0.001 Tanh	(1500, 150) 0.0001 Sigmoid	(1500, 100) 0.0001 Sigmoid	(750, 750) 0.001 Tanh	(50, 250) 0.001 Tanh	(1750, 1250) 0.0001 Sigmoid	(1000, 100) 0.0001 Sigmoid	(500, 1750) 0.0005 Sigmoid
	89.0 86.5 - 77.8 - **80.0** 89.7 - 89.0 - **88.7**	88.7 83.8 - 77.2 - 78.5 89.1 - 88.7 - 88.4	88.7 80.5 - 78.4 - 78.1 88.9 - 88.7 - 88.5	88.5 84.1 - 76.7 - 77.8 88.9 - 88.5 - 88.3	88.5 83.5 - 76.2 - 77.8 88.8 - 88.5 - 88.2	88.3 80.2 - 79.1 - 78.6 88.9 - 88.3 - 88.2	88.3 80.7 - 77.0 - 78.3 88.3 - 88.3 - 88.1	88.3 84.3 - 77.2 - 78.1 89.2 - 88.3 - 88.1	88.2 83.5 - 77.5 - 79.4 88.5 - 88.2 - 88.0	88.2 83.3 - 76.1 - 78.0 88.5 - 88.2 - 87.8

APPENDIX B

PL/SQL PROGRAMS DATA MANIPULATION LANGUAGE(DML) TYPES

There are five different DML(Data manipulation language) types on PL/SQL programs(See 12). These are single table DML statements, Multi table DML statements, Subqueries (for example select in select), Cursor statements and dynamic SQL. Cursor statements is specific for PL/SQL programs and there are two different usage cases. These are Explicit cursor declaration(See 13) and Cursor with variable(See 14). Dynamic SQL statement has three different usage cases. These are Direct SQL(See 15), SQL as a Variable(See 16) and SQL as a result(See 17). Table 36 and Table 37 presents examples for these DML types.

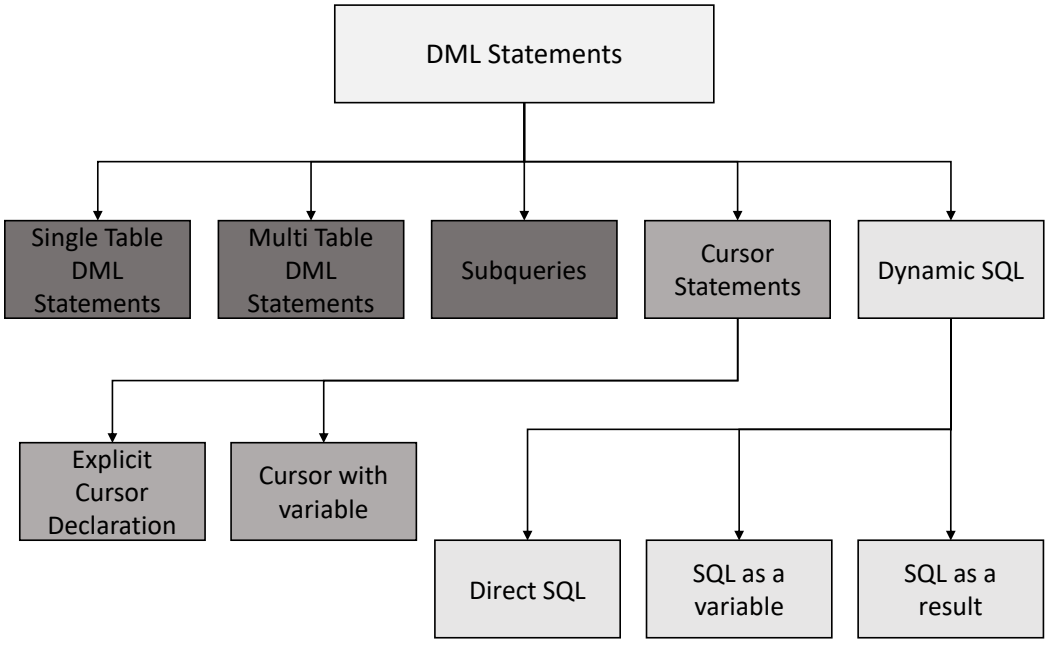
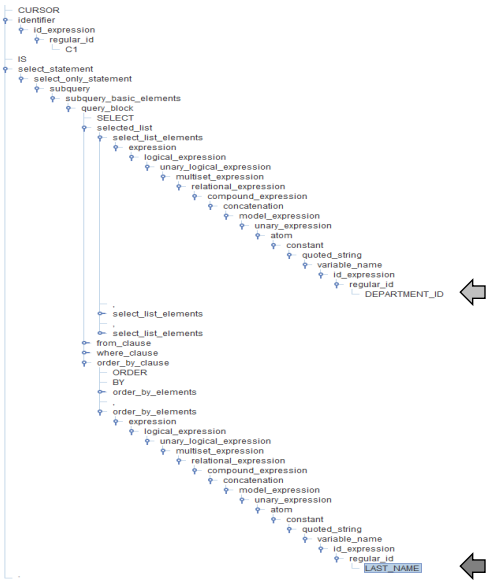


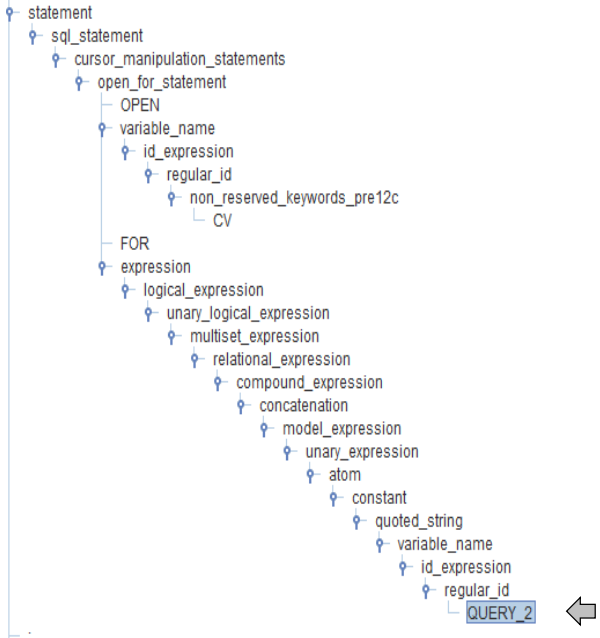
Figure 12: PLSQL Programs DML Types




```
DECLARE

CURSOR C1 IS
SELECT department_id, last_name, salary
FROM employees t
WHERE salary > ( SELECT AVG(salary)
FROM employees
WHERE t.department_id = department_id
)
ORDER BY department_id, last_name;

R1 C1%ROWTYPE;;


BEGIN

LOOP
    FETCH  C1  INTO R1;

    EXIT WHEN C1%NOTFOUND;

    END LOOP;


END;
```

* PLSQL Abstract Syntax Tree(AST) generated from ANTLR-PLSQL Parser

Figure 13: Explicit cursor declaration example




```
DECLARE


query_2 VARCHAR2(200) :=
'SELECT * FROM employees
WHERE REGEXP_LIKE (job_id,
''[ACADFIMKSA]_M[ANGR]'')
ORDER BY job_id';

BEGIN

OPEN cv FOR query_2;
……

END;
```

Figure 14: Cursor with variable example

```
EXECUTE IMMEDIATE 'DELETE FROM dept WHERE deptno = :num' USING dept_id;
```

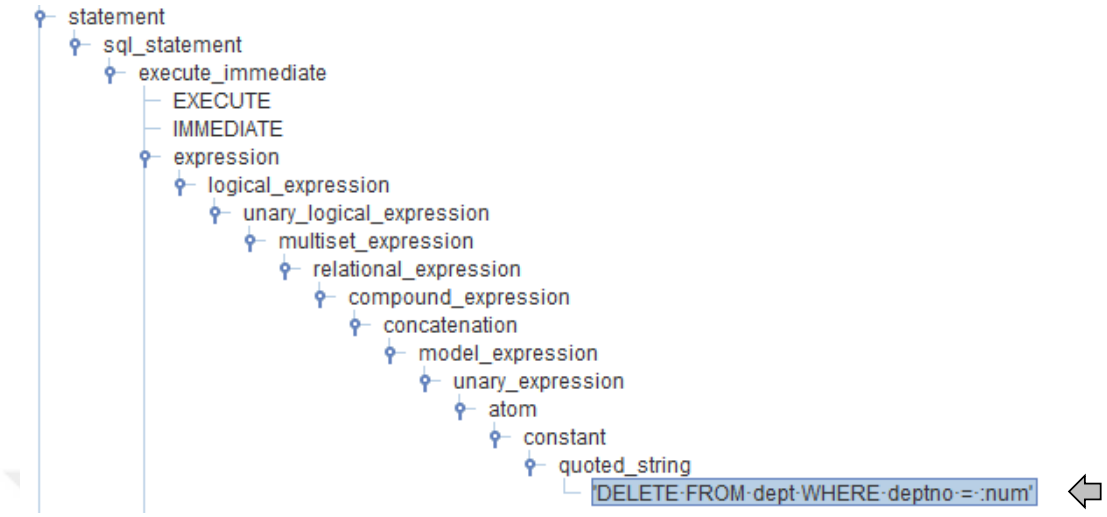


Figure 15: Direct SQL example

```
sql_stmt := 'SELECT dept_name FROM emp WHERE empno = :id';
EXECUTE IMMEDIATE sql_stmt INTO emp_dept USING emp_id;
```

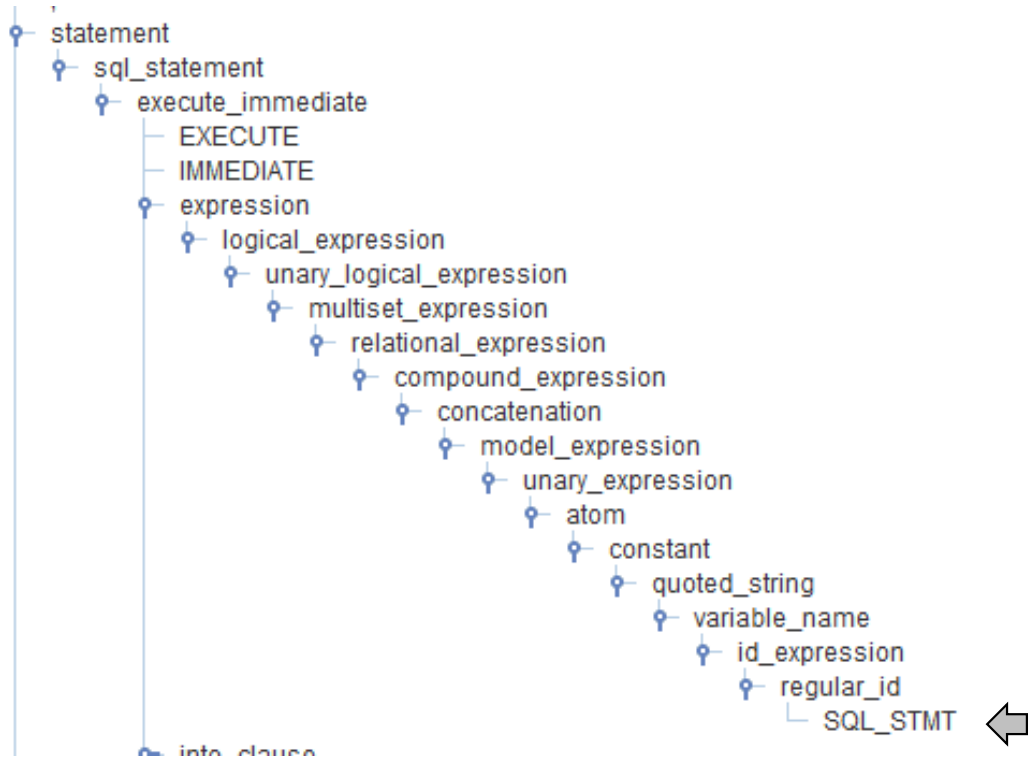


Figure 16: SQL as a variable example

```
SELECT tsql INTO v_sql FROM sqlcommandtable WHERE nid=1;
EXECUTE IMMEDIATE v_sql;
```

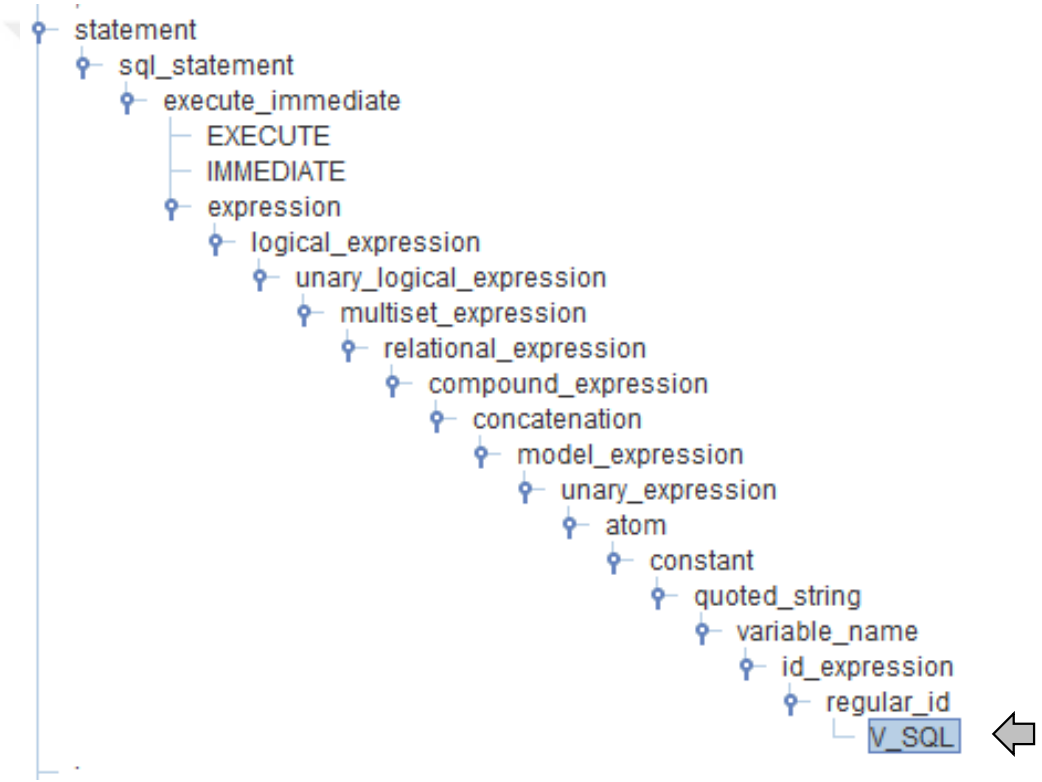


Figure 17: SQL as a result example

Table 36: Types of SQL statements with an example code snippet for each (DML: Data Manipulation Language, ST: Single Table, MT: Multi Table).

SQL Statement Types	**Example**
ST DML Statement	SELECT NAME INTO v_name FROM CUSTOMER;
MT DML Statement	SELECT E.NAME, S.SALARY INTO v_name,v_salary FROM EMPLOYEE E, EMP_SALARY S WHERE E.ID=S.ID;
Subquery	UPDATE CUSTOMER C SET TOTAL_DEPT= (SELECT SUM (PRICE) FROM ORDER WHERE CUST_ID =1) WHERE C.ID=1;

Explicit Cursor	DECLARE
	CURSOR C1 IS
	SELECT E.NAME,S.SALARY
	FROM EMPLOYEE E, CUST_SALARY S
	WHERE E.ID=S.ID;
	BEGIN
	OPEN C1;
	END;

Cursor with String	DECLARE
	cv SYS_REFCURSOR;
	BEGIN
	OPEN cv FOR 'SELECT E.NAME,S.SALARY
	FROM EMPLOYEE E, CUST_SALARY S
	WHERE E.ID=S.ID';
	END;

Cursor with Variable	v_Cursor := 'SELECT E.NAME,S.SALARY FROM EMPLOYEE E, CUST_SALARY S WHERE E.ID=S.ID'; OPEN cv FOR v_Cursor;

Table 37: Types of Dynamic SQL statements with an example code snippet for each (ST: Single Table, MT: Multi Table, D-SQL: Dynamic SQL).

SQL Statement Types	**Example**
ST D-SQL with String	EXECUTE IMMEDIATE 'INSERT INTO CUST_SALARY VALUES (:1,:2)' USING v_One, v_Two;
ST D-SQL with Variable	v_SQL_STMT := 'INSERT INTO CUST_SALARY VALUES (:1,:2)'; EXECUTE IMMEDIATE v_SQL_STMT USING v_One, v_Two;
ST D-SQL with SQL Result	SELECT DYNAMIC_SQL INTO v_SQL_STMT FROM PARAMETER WHERE NID = v_One; EXECUTE IMMEDIATE v_SQL_STMT;
MT D-SQL with String	EXECUTE IMMEDIATE 'INSERT INTO EMP_SALARY SELECT ID , SALARY FROM NEW_EMP_SALARY';
MT D-SQL with Variable	v_SQL_STMT:= 'INSERT INTO EMP_SALARY SELECT ID, SALARY FROM NEW_EMP_SALARY'; EXECUTE IMMEDIATE v_SQL_STMT ;
MT D-SQL with SQL Result	SELECT DYNAMIC_SQL INTO v_SQL_STMT FROM PARAMETER WHERE NID = v_One; EXECUTE IMMEDIATE v_SQL_STMT;

REFERENCES

[1] M. M. Lehman and L. A. Belady, *Program evolution: processes of software change*. Academic Press Professional, Inc., 1985.

[2] C. Chen, R. Alfayez, K. Srisopha, B. Boehm, and L. Shi, "Why is it important to measure maintainability, and what are the best ways to do it?," in *Proceedings of the 39th International Conference on Software Engineering Companion*, p. 377–378, 2017.

[3] D. L. Parnas, "On the criteria to be used in decomposing systems into modules," *Communications of the ACM*, vol. 15, no. 12, pp. 1053–1058, 1972.

[4] M. Altınışık, E. Ersoy, and H. Sözer, "Evaluating software architecture erosion for pl/sql programs," in *Proceedings of the 11th European Conference on Software Architecture: Companion Proceedings*, pp. 159–165, ACM, 2017.

[5] S. Yau, J. Collofello, and T. MacGregor, "Ripple effect analysis of software maintenance," in *Proceedings of the International Conference on Computer Software and Applications*, pp. 60–65, 1978.

[6] M. T. Tessema, G. Tesfom, M. A. Faircloth, M. Tesfagiorgis, and P. Teckle, "The "great resignation": Causes, consequences, and creative hr management strategies," *Journal of Human Resource and Sustainability Studies*, vol. 10, no. 1, pp. 161–178, 2022.

[7] G. Gulesir, *Evolvable Behavior Specifications Using Context-Sensitive Wildcards*. PhD thesis, University of Twente, 2008.

[8] T. Lutellier, D. Chollak, J. Garcia, L. Tan, D. Rayside, N. Medvidović, and R. Kroeger, "Measuring the impact of code dependencies on software architecture recovery techniques," *IEEE Transactions on Software Engineering*, vol. 44, no. 2, pp. 159–181, 2018.

[9] S. Ducasse and D. Pollet, "Software architecture reconstruction: A process-oriented taxonomy," *IEEE Transactions on Software Engineering*, vol. 35, no. 4, pp. 573 – 591, 2009.

[10] H. Abdeen, S. Ducasse, H. Sahraoui, and I. Alloui, "Automatic package coupling and cycle minimization," in *Proceedings of the 16th Working Conference on Reverse Engineering*, pp. 103–112, 2009.

[11] K. Praditwong, M. Harman, and X. Yao, "Software module clustering as a multi-objective search problem," *IEEE Transactions on Software Engineering*, vol. 37, no. 2, pp. 264–282, 2011.

[12] G. Bavota, A. D. Lucia, A. Marcus, and R. Oliveto, "Using structural and semantic measures to improve software modularization," *Empirical Software Engineering*, vol. 18, no. 5, pp. 901 – 932, 2013.

[13] M. Altınışık and H. Sözer, "Automated procedure clustering for reverse engineering pl/sql programs," in *Proceedings of the 31st ACM/SIGAPP Symposium on Applied Computing*, pp. 1440–1445, 2016.

[14] C. Patel, A. Hamou-Lhadj, and J. Rilling, "Software clustering using dynamic analysis and static dependencies," in *Proceedings of the 13th European Conference on Software Maintenance and Reengineering*, pp. 27–36, 2009.

[15] B. Mitchell and S. Mancoridis, "On the automatic modularization of software systems using the bunch tool," *IEEE Transactions on Software Engineering*, vol. 32, no. 3, pp. 193 – 208, 2006.

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

[17] E. Ersoy and H. Sözer, "Data model extension impact analysis," in *Proceedings of the 15th Turkish National Software Engineering Symposium*, pp. 1–6, 2021.

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

[19] A. Cleve, "Program analysis and transformation for data-intensive system evolution," in *Proceedings of the IEEE International Conference on Software Maintenance*, pp. 1–6, 2010.

[20] D. Qiu, B. Li, and Z. Su, "An empirical analysis of the co-evolution of schema and code in database applications," in *Proceedings of the 9th Joint Meeting on Foundations of Software Engineering*, p. 125–135, 2013.

[21] M. Goeminne, A. Decan, and T. Mens, "Co-evolving code-related and database-related changes in a data-intensive software system," in *Proceedings of the IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering*, pp. 353–357, 2014.

[22] L. Caruccio, G. Polese, and G. Tortora, "Synchronization of queries and views upon schema evolutions: A survey," *ACM Transactions on Database Systems*, vol. 41, no. 2, pp. 1–41, 2016.

[23] A. Jaimoon and T. Suwannasart, "Impact analysis of database schema changes on hibernate source code and test cases," in *Proceedings of the 3rd International Conference on Software and E-Business*, p. 120–123, 2019.

[24] L. Meurice, C. Nagy, and A. Cleve, "Detecting and preventing program inconsistencies under database schema evolution," in *IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pp. 262–273, IEEE, 2016.

[25] E. O'Neil, "Object/relational mapping 2008: Hibernate and the Entity Data Model (EDM)," in *Proceedings of the ACM SIGMOD international conference on Management of data*, pp. 1351–1356, 2008.

[26] G. Gregory and C. Bauer, *Java Persistence with Hibernate*. Simon and Schuster, 2015.

[27] J. Delplanque, A. Etien, N. Anquetil, and O. Auverlot, "Relational database schema evolution: An industrial case study," in *Proceedings of the 34th IEEE International Conference on Software Maintenance and Evolution*, pp. 635–644, 2018.

[28] A. Maule, W. Emmerich, and D. Rosenblum, "Impact analysis of database schema changes," in *Proceedings of the 30th international conference on Software Engineering*, p. 451–460, 2008.

[29] D. Kuhn, "Data dictionary fundamentals," in *Pro Oracle Database 12c Administration*, pp. 259–275, Springer, 2013.

[30] C. Potts, "Software-engineering research revisited," *IEEE software*, vol. 10, no. 5, pp. 19–28, 1993.

[31] S. Feuerstein and B. Pribyl, *Oracle pl/sql Programming*. " O'Reilly Media, Inc.", 2005.

[32] B. Li, X. Sun, H. Leung, and S. Zhang, "A survey of code-based change impact analysis techniques," *Software Testing, Verification and Reliability*, vol. 23, no. 8, pp. 613–646, 2013.

[33] L. Meurice, C. Nagy, and A. Cleve, "Static analysis of dynamic database usage in java systems," in *International Conference on Advanced Information Systems Engineering*, pp. 491–506, Springer, 2016.

[34] Y. Wang, J. Dong, R. Shah, and I. Dillig, "Synthesizing database programs for schema refactoring," in *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 286–300, 2019.

[35] A. Maule, W. Emmerich, and D. S. Rosenblum, "Impact analysis of relational schema changes on native language queries," *RN*, vol. 6, p. 13, 2001.

[36] C. Sriarpanon and T. Suwannasart, "A source code and test cases impact analysis tool for database schema changes," in *Proceedings of the International MultiConference of Engineers and Computer Scientists*, vol. 1, 2015.

[37] S. K. Gardikiotis and N. Malevris, "A two-folded impact analysis of schema changes on database applications," *International Journal of Automation and Computing*, vol. 6, no. 2, pp. 109–123, 2009.

[38] A. Cleve, "Program analysis and transformation for data-intensive system evolution," in *IEEE International Conference on Software Maintenance*, pp. 1–6, IEEE, 2010.

[39] C. Curino, H. J. Moon, A. Deutsch, and C. Zaniolo, "Automating the database schema evolution process," *The VLDB Journal*, vol. 22, no. 1, pp. 73–98, 2013.

[40] E. Ersoy and H. Sözer, "Data model extension impact analysis," in *Proceedings of the 15th Turkish National Software Engineering Symposium*, pp. 1–6, 2021.

[41] E. Ersoy and H. Sozer, "Using artificial neural networks to provide guidance in extending pl/sql programs," *Software Quality Journal*, 2022.

[42] E. Ersoy and H. Sözer, "Effort estimation for architectural refactoring of data tier software," in *Proceedings of the 19th International Conference on Software Architecture*, pp. 1–10, 2022.

[43] "Oracle Database Online Documentation 11g Release developing and using stored procedures." `http://docs.oracle.com/cd/B28359_01/appdev.111/b28843/tdddg_procedures.htm`. Accessed in October 2019.

[44] R. Strniša, P. Sewell, and M. Parkinson, "The Java module system: Core design and semantic definition," in *Proceedings of the 22nd Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages and Applications*, p. 499–514, 2007.

[45] L. V. Fausett, *Fundamentals of Neural Networks: Architectures, Algorithms, and Applications*. Prentice-Hall, 1994.

[46] G. Zhang, B. E. Patuwo, and M. Y. Hu, "Forecasting with artificial neural networks:: The state of the art," *International journal of forecasting*, vol. 14, no. 1, pp. 35–62, 1998.

[47] S. H. Walker and D. B. Duncan, "Estimation of the probability of an event as a function of several independent variables," *Biometrika*, vol. 54, no. 1-2, pp. 167–179, 1967.

[48] D. R. Cox, "The regression analysis of binary sequences," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 215–242, 1958.

[49] P. Harrington, "Machine learning in action," *Shelter Island, NY: Manning Publications Co*, 2012.

[50] J. R. Quinlan, "Induction of decision trees," *Machine learning*, vol. 1, no. 1, pp. 81–106, 1986.

[51] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the fifth annual workshop on Computational learning theory*, pp. 144–152, ACM, 1992.

[52] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[53] T. K. Ho, "Random decision forests," in *Proceedings of 3rd international conference on document analysis and recognition*, vol. 1, pp. 278–282, IEEE, 1995.

[54] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah, "Signature verification using a" siamese" time delay neural network," *Advances in neural information processing systems*, vol. 6, 1993.

[55] S. Chopra, R. Hadsell, and Y. LeCun, "Learning a similarity metric discriminatively, with application to face verification," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, pp. 539–546, IEEE, 2005.

[56] X. Zhang, F. X. Yu, S. Kumar, and S.-F. Chang, "Learning spread-out local feature descriptors," in *Proceedings of the IEEE international conference on computer vision*, pp. 4595–4603, 2017.

[57] H. Hindy, C. Tachtatzis, R. Atkinson, E. Bayne, and X. Bellekens, "Developing a siamese network for intrusion detection systems," in *Proceedings of the 1st Workshop on Machine Learning and Systems*, pp. 120–126, 2021.

[58] R. Kohavi *et al.*, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Ijcai*, vol. 14, pp. 1137–1145, Montreal, Canada, 1995.

[59] A. Gelman, A. Jakulin, M. G. Pittau, Y.-S. Su, *et al.*, "A weakly informative default prior distribution for logistic and other regression models," *The Annals of Applied Statistics*, vol. 2, no. 4, pp. 1360–1383, 2008.

[60] D. L. Chester, "Why two hidden layers are better than one," in *Proceedings of the international joint conference on neural networks*, vol. 1, pp. 265–268, 1990.

[61] G. Panchal, A. Ganatra, Y. Kosta, and D. Panchal, "Behaviour analysis of multilayer perceptronswith multiple hidden neurons and hidden layers," *International Journal of Computer Theory and Engineering*, vol. 3, no. 2, p. 332, 2011.

[62] G. Zaccone, M. R. Karim, and A. Menshawy, *Deep Learning with TensorFlow*. Packt Publishing Ltd, 2017.

[63] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, no. Feb, pp. 281–305, 2012.

[64] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, "Scikit-learn: Machine learning in python," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[65] A. Wawer, R. Nielek, and A. Wierzbicki, "Predicting webpage credibility using linguistic features," in *Proceedings of the 23rd international conference on world wide web*, pp. 1135–1140, 2014.

[66] K. Henderson, *The guru's guide to Transact-SQL*. Addison-Wesley Professional, 2000.

[67] D. Bales, *Java programming with Oracle JDBC*. " O'Reilly Media, Inc.", 2002.

[68] M. Nelson, "A survey of reverse engineering and program comprehension," *CoRR*, vol. abs/cs/0503068, 2005.

[69] C. Sun, J. Zhou, J. Cao, M. Jin, C. Liu, and Y. Shen, "ReArchJBs: a tool for automated software architecture recovery of javabeans-based applications," in *Proceedings of the 16th Australian Software Engineering Conference*, pp. 270–280, 2005.

[70] L. Qingshan et al., "Architecture recovery and abstraction from the perspective of processes," in *WCRE*, pp. 57–66, 2005.

[71] G. Guo, J. Atlee, and R. Kazman, "A software architecture reconstruction method," in *Proceedings of the First Working Conference on Software Architecture*, (Deventer, The Netherlands, The Netherlands), pp. 15–34, 1999.

[72] T. Callo, P. America, and P. Avgeriou, "A top-down approach to construct execution views of a large software-intensive system," *Journal of Software: Evolution and Process*, vol. 25, no. 3, pp. 233–260, 2013.

[73] L. Xiao, Y. Cai, and R. Kazman, "Design rule spaces: A new form of architecture insight," in *Proceedings of the 36th International Conference on Software Engineering*, pp. 967–977, 2014.

[74] A. Ouni, R. G. Kula, M. Kessentini, T. Ishio, D. M. German, and K. Inoue, "Search-based software library recommendation using multi-objective optimization," *Information and Software Technology*, vol. 83, pp. 55–75, 2017.

[75] M. W. Mkaouer, M. Kessentini, M. Ó. Cinnéide, S. Hayashi, and K. Deb, "A robust multi-objective approach to balance severity and importance of refactoring opportunities," *Empirical Software Engineering*, vol. 22, no. 2, pp. 894–927, 2017.

[76] M. Kessentini, U. Mansoor, M. Wimmer, A. Ouni, and K. Deb, "Search-based detection of model level changes," *Empirical Software Engineering*, vol. 22, no. 2, pp. 670–715, 2017.

[77] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[78] A. Ghannem, M. Kessentini, M. S. Hamdi, and G. El Boussaidi, "Model refactoring by example: A multi-objective search based software engineering approach," *Journal of Software: Evolution and Process*, vol. 30, no. 4, p. e1916, 2018.

[79] U. Mansoor, M. Kessentini, B. R. Maxim, and K. Deb, "Multi-objective code-smells detection using good and bad design examples," *Software Quality Journal*, vol. 25, no. 2, pp. 529–552, 2017.

[80] K. Deb, *Multi-objective optimization using evolutionary algorithms*, vol. 16. John Wiley & Sons, 2001.

[81] M. Laser, N. Medvidović, D. Le, and J. Garcia, "ARCADE: an extensible workbench for architecture recovery, change, and decay evaluation," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 1546–1550, 2020.

[82] J. Garcia, D. Popescu, G. Edwards, and N. Medvidovic, "Toward a catalogue of architectural bad smells," in *Proceedings of the International Conference on the Quality of Software Architectures*, pp. 146–162, 2009.

[83] O. Chaparro, J. Aponte, F. Ortega, and A. Marcus, "Towards the automatic extraction of structural business rules from legacy databases," in *Proceedings of the 19th Working Conference on Reverse Engineering*, pp. 479–488, 2012.

[84] E. Ersoy, K. Kaya, M. Altınışık, and H. Sözer, "Using hypergraph clustering for software architecture reconstruction of data-tier software," in *European Conference on Software Architecture*, pp. 326–333, Springer, 2016.

[85] M. Habringer, M. Moser, and J. Pichler, "Reverse engineering PL/SQL legacy code: An experience report," in *Proceedings of the IEEE International Conference on Software Maintenance and Evolution*, pp. 553–556, 2014.

[86] S. K. Gardikiotis, N. Malevris, and T. Konstantinou, "A structural approach towards the maintenance of database applications," in *Proceedings of the International Database Engineering and Applications Symposium*, pp. 277–282, 2004.

[87] W. E. Wong and Y. Qi, "Bp neural network-based effective fault localization," *International Journal of Software Engineering and Knowledge Engineering*, vol. 19, no. 4, pp. 573–597, 2009.

[88] W. E. Wong, V. Debroy, R. Golden, X. Xu, and B. M. Thuraisingham, "Effective software fault localization using an RBF neural network," *IEEE Transactions on Reliability*, vol. 61, no. 1, pp. 149–169, 2012.

[89] A. Lee, C. H. Cheng, and J. Balakrishnan, "Software development cost estimation: integrating neural network with cluster analysis," *Information & Management*, vol. 34, no. 1, pp. 1–9, 1998.

[90] A. Heiat, "Comparison of artificial neural network and regression models for estimating software development effort," *Information and software Technology*, vol. 44, no. 15, pp. 911–922, 2002.

[91] A. Idri, T. M. Khoshgoftaar, and A. Abran, "Can neural networks be easily interpreted in software cost estimation?," in *IEEE World Congress on Computational Intelligence. IEEE International Conference on Fuzzy Systems. FUZZ-IEEE'02. Proceedings (Cat. No. 02CH37291)*, vol. 2, pp. 1162–1167, IEEE, 2002.

[92] R. Schwanke and S. Hanson, "Using neural networks to modularize software," *Machine Learning*, vol. 15, no. 2, pp. 137–168, 1994.

[93] A. Furda, C. Fidge, O. Zimmermann, W. Kelly, and A. Barros, "Migrating enterprise legacy source code to microservices: On multitenancy, statefulness, and data consistency," *IEEE Software*, vol. 35, no. 3, pp. 63–72, 2018.

[94] J. Fritzsch, J. Bogner, A. Zimmermann, and S. Wagner, "From monolith to microservices: A classification of refactoring approaches," in *Proceedings of the First International Workshop on Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment*, pp. 128–141, 2019.

[95] L. Nunes, N. Santos, and A. Silva, "From a monolith to a microservices architecture: An approach based on transactional contexts," in *Proceedings of the 13th European Conference on Software Architecture*, pp. 37–52, 2019.

[96] N. Gonçalves, D. Faustino, A. Silva, and M. Portela, "Monolith modularization towards microservices: Refactoring and performance trade-offs," in *Proceedings of the IEEE 18th International Conference on Software Architecture Companion*, pp. 1–8, 2021.

[97] H. Sozer, B. Tekinerdogan, and M. Aksit, "FLORA: a framework for decomposing software architecture to introduce local recovery," *Software Practice and Experience*, vol. 39, no. 10, pp. 869–889, 2009.

[98] F. Ozturk, E. Sarili, H. Sozer, and B. Aktemur, "Effort estimation for architectural refactoring to introduce module isolation," in *Proceedings of the 8th European Conference on Software Architecture*, pp. 300–307, 2014.

[99] "Oracle database link." `https://docs.oracle.com/cd/E18283_01/server.112/e17120/ds_concepts002.htm`. Accessed in July 2021.

[100] T. Bain, L. Davidson, R. Dewson, and C. Hawkins, *SQL Server 2000 Stored Procedures Handbook*. APress, 2003.

[101] "Oracle vs MSSQL." `https://docs.oracle.com/cd/E10405_01/appdev.120/e10379/ss_oracle_compared.htm`. Accessed in July 2021.

[102] T. Parr and R. Quong, "ANTLR: A predicated-LL(k) parser generator," *Software: Practice and Experience*, vol. 25, no. 7, pp. 789–810, 1995.

[103] "Antlr plsql grammar parser." `https://github.com/antlr/grammars-v4/tree/master/sql/plsql`. Accessed in July 2021.

[104] T. Parr, *The Definitive ANTLR 4 Reference*. Pragmatic Bookshelf, 2013.

[105] B. Sam-Bodden, *Beginning POJOs: Lightweight Java Web Development Using Plain Old Java Objects in Spring, Hibernate, and Tapestry*. Apress, 2006.

[106] I. Alsmadi and M. Nuser, "Evaluation of cost estimation metrics: Towards a unified terminology," *Journal of Computing and Information Technology*, vol. 25, no. 1, pp. 23–34, 2013.

[107] "Pl/sql code encryption." `https://docs.oracle.com/cd/B19306_01/appdev.102/b14258/d_crypto.htm\#BJFGFDFG`. Accessed in July 2021.

[108] C. Wohlin, P. Runeson, M. Host, M. Ohlsson, B. Regnell, and A. Wesslen, *Experimentation in Software Engineering*. Berlin, Heidelberg: Springer-Verlag, 2012.

[109] 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, 2015.

[110] H. Weber, A. Cleve, L. Meurice, R. Bermudez, and J. Javier, "Managing technical debt in database schemas of critical software," in *Proceedings of the Sixth International Workshop on Managing Technical Debt*, pp. 43–46, 2014.

[111] M. Albarak, R. Bahsoon, I. Ozkaya, and R. Nord, "Managing technical debt in database normalization," *IEEE Transactions on Software Engineering*, pp. 1–1, 2020. Early Access.

[112] A. Cleve, *Program analysis and transformation for data-intensive system evolution*. PhD thesis, University of Namur, 2009.

[113] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[114] "Dba dependency column data dictionary view." `http://rwijk.blogspot.com/2008/10/dbadependencycolumns.html`. Accessed in July 2021.

[115] N. Ketkar, "Introduction to keras," in *Deep learning with Python*, pp. 97–111, Springer, 2017.

[116] M. D. Zeiler and R. Fergus, "Stochastic pooling for regularization of deep convolutional neural networks," *arXiv preprint arXiv:1301.3557*, 2013.

[117] R. J. May, H. R. Maier, and G. C. Dandy, "Data splitting for artificial neural networks using som-based stratified sampling," *Neural Networks*, vol. 23, no. 2, pp. 283–294, 2010.

[118] A. Fujino, H. Isozaki, and J. Suzuki, "Multi-label text categorization with model combination based on f1-score maximization," in *Proceedings of the Third International Joint Conference on Natural Language Processing: Volume-II*, 2008.

[119] N. An and W. Qi Yan, "Multitarget tracking using siamese neural networks," *ACM Transactions on Multimidia Computing Communications and Applications*, vol. 17, no. 2s, pp. 1–16, 2021.

[120] J. Whissell and C. Clarke, "Effective measures for inter-document similarity," in *Proceedings of the 22nd ACM international conference on Information and Knowledge Management*, p. 1361–1370, 2013.

[121] E. Ersoy, M. Altinisik, and H. Sözer, "PL/SQL programlari icin veri tabani bagimlilik analizi(database dependency analysis for PL/SQL programs)," in *Proceedings of the 11th Turkish National Software Engineering Symposium*, vol. 1980 of *CEUR Workshop Proceedings*, pp. 114–125, 2017.

[122] A. Karahasanovic and D. I. Sjoberg, "Visualizing impacts of database schema changes-a controlled experiment," in *Proceedings IEEE Symposia on Human-Centric Computing Languages and Environments (Cat. No. 01TH8587)*, pp. 358–365, IEEE, 2001.

[123] G. Nguyen and D. Rieu, "Schema change propagation in object-oriented databases," in *Proceedings of the IFIP 11th World Computer Congress*, pp. 815–820, 1989.

[124] A. Karahasanovic, "Identifying impacts of database schema changes on applications," *Proceedings of the 8th Doctoral Consortium at the CAiSE*, vol. 1, pp. 93–104, 2001.

[125] J. Jainae and T. Suwannasart, "A framework for test case impact analysis of database schema changes using use cases," *International Journal of Engineering and Technology*, vol. 6, no. 3, p. 186, 2014.

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

[127] I. Melekhov, J. Kannala, and E. Rahtu, "Siamese network features for image matching," in *23rd international conference on pattern recognition (ICPR)*, pp. 378–383, IEEE, 2016.

[128] G. Koch, R. Zemel, R. Salakhutdinov, *et al.*, "Siamese neural networks for one-shot image recognition," in *ICML deep learning workshop*, vol. 2, pp. 1–30, Lille, 2015.

[129] L. Song, D. Gong, Z. Li, C. Liu, and W. Liu, "Occlusion robust face recognition based on mask learning with pairwise differential siamese network," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 773–782, 2019.

[130] W. Wang, J. Yang, J. Xiao, S. Li, and D. Zhou, "Face recognition based on deep learning," in *International Conference on Human Centered Computing*, pp. 812–820, Springer, 2014.

[131] S. Dey, A. Dutta, J. I. Toledo, S. K. Ghosh, J. Lladós, and U. Pal, "Signet: Convolutional siamese network for writer independent offline signature verification," *arXiv preprint arXiv:1707.02131*, 2017.

[132] K. Ahrabian and B. BabaAli, "Usage of autoencoders and siamese networks for online handwritten signature verification," *Neural computing and applications*, vol. 31, no. 12, pp. 9321–9334, 2019.

VITA

Ersin Ersoy received his associate degree in Computer Programming from Marmara University, 1999-2001. He graduated in Computer Engineering and Computer Education undergraduate programs from Kocaeli University, MBA from Bilgi University in 2012, and M.Sc. in Computer Engineering from Ozyegin University in 2016. He has been working as a senior manager at Turkcell/Paycell since 2000. He is responsible for Mobil Payment and Data Analytics technology solutions.