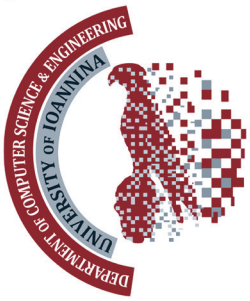
An Empirical Study on the Usage of Conventions and Rules for SQL Programming in FoSS

Aggelos Papamichail

Master Thesis

— ♦ —

Ioannina, June 2018



ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ & ΠΛΗΡΟΦΟΡΙΚΗΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
UNIVERSITY OF IOANNINA

An Empirical Study on the Usage of Conventions and Rules for SQL programming in FoSS

A Thesis

submitted to the designated

by the General Assembly of Special Composition

of the Department of Computer Science and Engineering

Examination Committee

by

Papamichail Aggelos

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

WITH SPECIALIZATION

IN SOFTWARE

University of Ioannina

June 2018

Examining Committee:

- **Zarras Apostolos**, Associate Professor, Dept. of Computer Science & Engineering, University of Ioannina (Supervisor)
- **Vassiliadis Panagiotis**, Associate Professor, Dept. of Computer Science & Engineering, University of Ioannina
- **Mamoulis Nikos**, Associate Professor, Dept. of Computer Science & Engineering, University of Ioannina

DEDICATION

To my parents Aristeidis and Augoula, for all the years they assisted in my academic pursuits.

To my sister Chrisanthi, her professionalism, work ethic and capacity will always be inspiring, a true pioneer!

ACKNOWLEDGMENTS

I would like to thank my thesis advisor Professor Apostolos Zarras of Department of Computer Science and Engineering at the University of Ioannina. His guidance, insights and remarks not only made this thesis possible but also formed the directions of my career as software engineer.

Furthermore I would like to thank Professor Panos Vassiliadis of Department of Computer Science and Engineering at the University of Ioannina, for his assistance in defining the concepts of this thesis.

TABLE OF CONTENTS

INDEX OF TABLES

INDEX OF FIGURES

ΕΚΤΕΤΑΜΕΝΗ ΠΕΡΙΛΗΨΗ

Παπαμιχαήλ Άγγελος, Μ.Δ.Ε. στην Πληροφορική, Τμήμα Μηχανικών Η/Υ και Πληροφορικής, Πανεπιστήμιο Ιωαννίνων, Ιούνιος 2018.
Εμπειρική Μελέτη Σχετικά με τη Χρήση Συμβάσεων και Κανόνων SQL σε Ελεύθερο και Ανοιχτό Λογισμικό.
Επιβλέπων: Ζάρρας Απόστολος, Αναπληρωτής Καθηγητής

Η εξέλιξη του λογισμικού είναι μία από τις σημαντικότερες πλευρές του software engineering με τη συντήρη, να απαιτεί τους μισούς από τους συνολικά διαθέσιμους πόρους. Η κοινότητα του software engineering έχει κάνει σημαντική πρόοδο ως προς την ποιότητα κώδικα μέσω της δημιουργίας και αξιοποίησης τεχνικών που βελτιστοποιούν την ανάπτυξη και συντήρησή του αλλά και της ανάπτυξης τεχνικών διαχείρισης πόρων. Πιο συγκεκριμένα, για την πλειονότητα των γλωσσών αντικειμενοστρεφούς προγραμματισμού υπάρχει ένα σύνολο κανόνων γραφής[1][2] που εξασφαλίζουν την ύπαρξη ομοιογένειας και αναγνωσιμότητας στο κώδικα. Παράλληλα ορίστηκαν μοτίβα γραφής κώδικα που ενισχύουν άμεσα την επεκτασιμότητά[3] και εξασφαλίζουν την χρήση αντικειμενοστρεφούς αρχών σχεδίασης[4].

Είναι γνωστό, ειδικά σε μεγάλης κλίμακας προγράμματα, ότι η ύπαρξη λογισμικού προϋποθέτει την ύπαρξη μιας ή περισσότερων βάσεων δεδομένων και το αντίστροφο, ενώ ακόμη πολλές φορές το ίδιο το λογισμικό δημιουργείται και εξελίσεται γύρω από μια βάση δεδομένων. Η πραγματικότητα λοιπόν ορίζει ως ίσα μέρη το λογισμικό και τις βάσεις. Εύλογα θα περιμένε κανείς την ύπαρξη πλούσιας και σε βάθος βιβλιογραφία για τις βάσεις αντίστοιχη αυτής του λογισμικού, όσον αφορά την δημιουργία και ανάπτυξή τους. Δυστυχώς όμως η βιβλιογραφία είναι σαφώς πιο περιορισμένη και αυτό

[1] https://www.oracle.com/technetwork/articles/javase/codeconvtoc-136057.html

[2] https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/inside-a-program/coding-conventions

[3] https://sourcemaking.com/design_patterns/factory_method

[4] https://sourcemaking.com/design_patterns/builder

μας ώθησε στο να ερευνήσουμε και να συνεισφέρουμε στο τομέα της ποιότητας των βάσεων δεδομένων.

Σε αυτή την εργασία επικεντρωνόμαστε στον προγραμματισμό SQL. Ο κύριος στόχος μας είναι να διευρενήσουμε το βαθμό στον οποίο οι προγραμματιστές αξιοποιούν συμβάσεις και στυλ γραφής κώδικα κατά τον ορισμό SQL σχημάτων. Για αυτό το λόγο, εισάγουμε ένα ιδανικό SQL στυλ που αποτελείται από ένα σύνολο κανόνων που προέρχονται από ό,τι καλύτερο υπάρχει στη σχετική βιβλιογραφία. Το ιδανικό στυλ καλύπτει ποικίλες πλευρές όσον αφόρα τη ποιότητα ενός SQL σχήματος. Για να ελέγξουμε την ποιότητα του σχήματος προτείνουμε ένα εργαλείο που επιτρέπει στους προγραμματιστές να ελέγξουν την ικανοποίηση των κανόνων του ιδανικού στυλ. Το εργαλείο αυτοματοποιεί πλήρως τον έλεγχο ικανοποίησης των κανόνων στην ιστορία ενός σχήματος, όπου ως ιστορία ορίζονται οι διαφορετικές εκδόσεις ενός σχήματος από την αρχή της ύπαρξής του ως την τελευταία γνωστή έκδοση σε εμάς. Παράλληλα με το βαθμό ικανοποίησης των κανόνων για την κάθε έκδοση, εξάγει στατιστικά στοιχεία που περιγράφουν το εύρος τιμών της ικανοποίησης (τυπική απόκλιση, μέσος όρος κ.α.) καθώς και τη συσχέτιση μεταξύ του μεγέθους του σχήματος και του βαθμού ικανοποίησης. Αξιοποιούμε τις δυνατότητες που μας παρέχει το εργαλείο διεξάγοντας μια μεγάλης κλίμακας μελέτη αποτελούμενη από 21 γνώστα, ανοιχτού κώδικα, προγράμματα (FoSS). Στη μελέτη ελέγχουμε αν τα SQL σχήματα των προγραμμάτων που συμμετέχουν σε αυτήν ακολουθούν τους κανόνες. Ακόμα, εξετάζουμε την εξέλιξη των στυλ που ακολουθούντε από το κάθε σχήμα. Παράλληλα, αναγνωρίζουμε μοτίβα εξέλιξης που περιγράφουν την ικανοποίηση των κανόνων κατά τη διάρκεια ζωής των σχημάτων. Συνεχίζοντας, διεξάγουμε μια λεπτομερή ανάλυση του κάθε κανόνα και αναγνωρίζουμε τα αντίστοιχα μοτίβα ικανοποίησης ή παραβίασής τους. Βασιζόμενοι στα παραπάνω μοτίβα, ορίζουμε ένα σταθμισμένο (weight) στυλ που αντικατοπτρίζει το εύρος χρήσης των κανόνων από τους προγραμματιστές στην πράξη. Τέλος, αξιολογούμε την απόσταση των ελεγθέντων σχήματων από το ιδεατό και σταθμισμένο στυλ.

Συγκεκριμένα, αποδείξαμε ότι ικανοποιούνται αρκετοί κανόνες στυλ γραφής SQL ενώ ο βαθμός ικανοποίησης επηρεάζεται από το σχήμα, την SQL οντότητα που ελέγχουμε (πίνακας ή κολώνα) αλλά και από τον ίδιο το κανόνα. Μελετώντας την εξέλιξη των σχημάτων, βρήκαμε ότι για τους πίνακες το πλήθος των κανόνων που αλλάζει

κυμαίνεται από 0 έως 7 και για τις κολώνες από 0 έως 8. Επίσης, οι αλλαγές μεγάλης κλίμακας συμβαίνουν σε μικρά χρονικά διάστηματα ως προς τη συνολική διάρκεια ζωής του σχήματος και ότι τα διαστήματα αυτά είναι συνεχή. Ο έλεγχος της διαφοράς του βαθμού ικανοποιήσης των κανόνων μεταξύ πρώτης και τελευταίας έκδοσης έδειξε την φύση των αλλαγών. Οι περισσότεροι κανόνες παραμένουν σταθεροί (13 για πίνακες, 14 για κολώνες), ορισμένοι εξελίσσονται αρνητικά (2 κανόνες πινάκων, 1 κολώνων) και τέλος ελάχιστοι εξελίσσονται θετικά (2 για τις κολώνες). Διερευνόντας τον βαθμό ικανοποίησης στην τελευταία γνωστή έκδοση βρήκαμε ότι οι κανόνες που ικανοποιούνται σε μεγάλο βαθμό είναι 11 για τους πίνακες 14 για τις κολώνες. Οι κανόνες που ικανοποιούνται σε μικρότερο βαθμό είναι 2 για τους πίνακες και 1 για τις κολώνες. Οι κανόνες που παραβιάζονται σε μεγάλο βαθμό είναι 2 για τους πίνακες και αυτοί που παραβιάζονται σε μικρότερο βαθμό είναι 2 και αφορούν τις κολώνες. Τέλος μέσω της κατάταξης των κανόνων με βάση τη ικανοποίησή τους από τους προγραμματιστές προσδιορίσαμε της αξία του κάθε κανόνα, όπως την αντιλαμβάνεται ο ίδιος ο προγραμματιστής.

ABSTRACT

Papamichail Aggelos, M.Sc. in Computer Science, Department of Computer Science and Engineering, University of Ioannina, Greece, June 2018.
An Empirical Study on the Usage of Conventions and Rules for SQL programming in FoSS.
Supervisor: Zarras Apostolos, Associate Professor

Software evolution is one of the most important aspects of software engineering with maintenance requiring about half of a project's resources. The software engineering community has made massive improvements in coding quality with the adoption of good practices, coding conventions and styles that optimize software development and maintenance. In this Thesis we focus on SQL programming. Our main objective is to investigate the extent to which developers employ coding conventions and styles in the definition of an SQL schema. To this end, we introduce a SQL style that consists of a set of style rules found in the state of the art. This style covers various aspects of the SQL schema quality. To assess schema quality we propose a style checking tool that allows developers to check the adherence to the rules of the defined style. We conduct a large scale case study consisting of 21 well-known FoSS projects. In our case study we check whether the SQL schemas of the examined projects adhere to the rules. Moreover, we investigate the evolution of the style that is followed by each schema. Taking a step further, we identify evolution patterns that describe the adherence to the rules over the lifetime of the examined schemas. Then, we perform a detailed analysis of the individual rules and we identify respective rule adherence and violation patterns. Based on the aforementioned patterns, we identify a weighted SQL style that reflects the extent to which the examined rules are employed in practice. Finally, we evaluate the distance of the examined schemas from the style we defined and the weighted style.

CHAPTER 1

INTRODUCTION: INITIATING SQL STYLE

Software evolution is one of the most important aspects of software engineering. It is well known that maintenance, which is part of software evolution will may require over half of a project's resources [1] [2], thus the existence of practices in managing resources and the creation of maintainable and extensible code is compulsory. So far, the software engineering community has done greatly in both resource management and code quality. In the last decades, several ways of managing projects have been proposed, from the more recent, the agile[5] model and DevOps[6] to older, like the waterfall. Moreover, there has been a massive improvement in coding quality practices with the adoption of refactoring and restructuring techniques and the creation of tools enabling developers to recognize and apply them as they code. Also, more refined and specific concepts were created to further improve the quality, like the code smells. Code smells are a well-known metaphor to describe symptoms of code decay or other issues with code quality which can lead to a variety of maintenance problems. They usually are diminished through globally accepted conventions and practices.

of course, one cannot think of large scale projects without the existence of one or more databases, co-evolving with the software of the project or even being its precursor, since many systems are built around databases. Since in a large scale project databases are of equal, or even greater importance, with software, there ought to be at least comparable bibliography on the subject of database's schema evolution (i.e. the ability of a database system to respond to changes in the real world by allowing the schema to evolve) or restructuring and refactoring techniques. Moreover, there ought to be styles, conventions, and good practices for SQL programming. Unfortunately, this is not the

[5] https://en.wikipedia.org/wiki/Agile_software_development#The_Agile_Manifesto

[6] https://en.wikipedia.org/wiki/DevOps

case; in the database community the state of the art in these research issues is far less rich both in depth and variety than in the software engineering community. Actually, when it comes to SQL programming there is no answer in the bibliography, to the following simple yet important question:

Do people care about SQL rules?

The aim of this thesis is to address the previous question, via a study that involves 21 database schemas found in respective open source projects. To this end, we define a SQL style [3.2] that consists of a set of rules and conventions, which have been proposed in the literature towards improving the readability and maintainability of a schema. These rules are style conventions and cover various styling quality aspects (writing style, methodological, lexicographical, SQL specific) of the SQL schema. Along with the proposed rules, we developed a fully automated SQL style checking tool that allows the developers to check the adherence of SQL schemas to the rules.

Based on the proposed rules, we begin with a coarse-grained analysis to assess the adherence of the examined schemas to the rules. We focus on the last know version (LKV) of the schemas. Our results reveal that developers take into account several rules. However, the extent to which they employ the rules varies, depending on the schema.

Next, we investigate the evolution of the schemas *from* their first known version (FKV) to the last known version available to us, to see if the adherence to the rules changes through time. We further investigate how this is done via respective evolution patterns and anti-patterns. Moreover, we evaluate the correlation between the evolution of the schemas' adherence to the rules and the size of the schemas. Our results reveal the following basic patterns: most rules (13 for tables rules, 13 for columns), follow the *fixed adherence evolution pattern*, i.e., the adherence of a schema to the rule does not evolve overtime; only few rules follow the *positive evolution pattern* (2 for columns rules), i.e., the adherence of a schema to the rule slightly improves overtime; few rules

follow the *negative evolution pattern* (2 for rules, 1 for colums), i.e., the adherence of a schema to the rule gets slightly worst overtime.

Taking a step further, we perform a fine-grained analysis that concerns each individual rule. Specifically, for each rule we identify adherence patterns, and/or violation anti-patterns: most rules follow the *strong adherence pattern*, i.e., more than 75% of the schema elements adhere to the rule; some rules follow the *weak adherence pattern*, i.e. the percentage of schema elements that follow the rule varies in [50%, 75%); few rules follow the strong violation pattern, i.e. less than 25% of the schema elements adhere to the rule; few some rules follow the weak violation pattern, i.e., the percentage of schema elements that follow the rule varies in [25%, 50%).

We introduce weights for the rules that reflect the extent to which they are followed in the examined schemas and propose a weighted SQL style, consisting of our initial list of rules, ranked according to their weights. Along with the rule based style we also consider the weight style based on the developers SQL rule preferences and adherence. In particular, we evaluate the distance between each of the examined schemas and the aforementioned styles, and we provide concrete examples that relate this distance with specific readability issues.

Roadmap. The structure of this thesis is as follows. In Chapter 2, we discuss related work on good practices, rules, conventions and techniques for the development of clean software. Moreover, we discuss related work on database schema evolution patterns, smells and refactorings. In Chapter 3 we present our methodology, the tool that we developed and the setup of our empirical study. In Chapter 4 we present our results. Finally, in Chapter 5 we conclude with a summary of our contribution and the future perspectives of this work.

CHAPTER 2

RELATED WORK

Primarily, the present study concerns two aspects of software engineering:

- The first aspect is focused on the database domain and is about schema evolution, see e.g. [3], SQL conventions and smells.
- The second is the existence of coding smells in general and practices to avoid them, mainly from the software domain.

The research of Sjøberg [3] on schema evolution and its consequences on related applications has been a breakthrough. Modification of database schemata, for example, is one kind of change which may significantly influence database applications. So, this paper presents a method and an appropriate tool [3] for measuring modifications to database schemata and their consequences. The resulting measurements serve as input to the design of change management tools. The temporal aspect of the grammatical database model of Laine and co-authors [4] is extended to the schema and thus the model possesses a schema capable of modification over time. The work of Roddick [5] presents an extension to SQL to handle some of the functionality provided by schema evolution in relational databases. Other early approaches consist the publication of Nguyen and Rieu [6] and the one of Barenjee and co-authors [7] that focus on object-oriented databases.

In the late ‘00's, partly due to accessibility to free open software, schema evolution in open source environments has been quite investigated, i.e. Curino et al. [8]. Those studies, however, focus on the statistical properties of the evolution and do not provide details on the mechanism that governs the evolution of data base schemata.

The results of Meir Lehman and co-authors [9] [10] include a set of rules on the mechanics of Software evolution [10], also known as the Laws on Software Evolution. More precisely, Lehman's Laws as stated in a more abstract form:

- *Law of Continuing Change:* An E-Type software system must be continually adapted or else it becomes progressively less satisfactory in use.
- *Law of Increasing Complexity*: As an E-type system is changed, its complexity increases and becomes more difficult to evolve unless work is done to maintain or reduce the complexity.
- *Law of Self-regulation*: Global E-type system evolution is feedback regulated.
- *Law of Conservation of Organizational Stability*: The work rate of an organization evolving an E-type software system tends to be constant over the operational lifetime of that system or phases of that lifetime.
- *Law of Conservation of Familiarity*: In general, the incremental growth (growth ratio trend) of E-type systems is constrained by the need to maintain familiarity.
- *Law of Continuing Growth*: The functional capability of E-type systems must be continually enhanced to maintain user satisfaction over system lifetime.
- *Law of Declining Quality*: Unless rigorously adapted and evolved to take into account changes in the operational environment, the quality of an E-type system will appear to be declining.
- *Law of Feedback System*: E-type evolution processes are multi-level, multi-loop, multi-agent feedback systems.

The breakthrough of Lehman's work has motivated Skoulis and his colleagues [11] to adapt the Laws of software evolution into schemata evolution. Subsequently, their research work identified patterns and regularities of schema evolution towards a better understanding of the underlying mechanism that governs it. By studying the evolution of the logical schema of eight databases, collecting and cleansing the available versions of the database schemata for the eight case studies and extracting the changes that have been performed in these versions, finally, their research came up with usable datasets subsequently analyzed. The main tool for this analysis came from the area of software engineering.

The research axis of Vassiliadis and Zarras [11] [12] [13]is the survival of a table in the context of schema evolution in open-source software. The authors found out that the probability of a table with a wide schema (i.e., a large number of attributes) being removed is systematically lower than average, as well as, that activity and duration are related to survival too. They proposed the *electrolysis pattern*, due to its diagrammatic representation, stating that dead and survivor tables live quite different lives: tables typically die shortly after birth, with short durations and mostly no updates, whereas survivors mostly live quiet lives with few updates, except for a small group of tables with high update ratios that are characterized by high durations and survival. In addition, they studied the phenomenon of gravitation to rigidity, stating that despite some valiant efforts, relational schemata suffer from the tendency of developers to minimize evolution as much as possible in order to minimize the resulting impact to the surrounding code.

What is also special about this research, is that it zoomed into the lives of tables in contrast with previous works that mostly focused on the macroscopic study of the entire database schema. Vassiliadis his colleagues identified four major patterns [11] on the relationship of such properties:

- The *Gamma pattern* studies the interrelationship of the schema size of a table at its birth with its overall duration and indicates that tables with large schemata tend to have long durations and avoid removal.
- The *Comet pattern* studies the interrelationship of the schema size of a table at its birth with its total amount of updates. This pattern indicates that the tables with most updates are frequently the ones with medium schema size.
- The *Inverse Gamma pattern* studies the interrelationship of the amount of updates in the life of a table with its duration and indicates that tables with medium or small durations produce amounts of updates lower than expected, whereas tables with long duration expose all sorts of update behavior.
- The *Empty Triangle pattern* studies the interrelationship of a table's version of birth with its overall duration. It indicates a significant absence of tables of medium or long durations that were removed.

Curino and his co-workers [8] created a tool, the so-called PRISM, containing recent theoretical advances in schema and query rewriting mapping [14] [15] [16]. This tool

addresses two main challenges of schema evolution, as considered by the authors, predictability and independence of the evolution process. PRISM solves the predictability challenge through describing the revisions of a schema with Schema Modification Operators (SMOs) and then through a static analysis upon the SMOs manages to give the database administrator knowledge about information preserving properties of the new schema and redundancy generation. The second challenge, independence, is tackled through the automation of the rewriting queries process.

Sharma and his co-workers [17] used the concept of code smells in the field of databases. They defined thirteen smells which were evaluated by developers through an online survey and conducted a large scale study in 357 industrial and 2568 open source projects. For each smell defined, developers had to characterize them based on their opinion as database smells, as a recommended practice or as neither smell nor recommended practice. They could also state that the context of use defined the practice as recommended or as a smell. They found that db smell detection is affected by the developers' subjectivity and acknowledged the importance of using a sophisticated tool in the form of a plug-in to automate the smell detection process on the go. The most frequent smell was index abuse (i.e. misused primary or foreign keys and even absence of a key) and interestingly the use of ORM frameworks did not affect the number of smells found in the project. The existence of certain smells in the schemata or queries increased the occurrence of other smells and that open source projects are prone to different smells than industrial ones.

Celko [18], created a comprehensive textbook on database smells and how to avoid them. Those smells are found in a wide field of SQL use; in names, in the scales for attributes, formatting, in the data declaration language and other. In this study we were particularly interested in the practices referred to naming conventions and use of punctuations and spacing which were combined with Holywell's[7] point of view on SQL style conventions. To our delight we found that Tushra's db smells contain some of Celko's smells, like the index abuse, for example.

[7]https://www.sqlstyle.guide/

Robert Martin wrote Clean Code, [19], a masterpiece describing in hideous detail how to write elegant and efficient code. In the world of a developer where deadlines may exist in a daily basis, developers tend to write messy code in an effort of accomplishing the tasks on time. Unfortunately, *as the mess builds the productivity decreases* to the point where a project is unmaintainable and developers rebel against the managers asking for a system redesign. Martin describes thoroughly the importance of meaningful names, good functions, proper formatting etc. and overall manages to create a roadmap for someone *who cares* about writing elegant and efficient code. Clean Code and SQL Programming Style are in a sense like Plutarch's parallel lives. Clean Code assesses readability, maintainability and ultimately, extensibility from an object oriented coding point of view while the second from the standard's SQL. Both influenced greatly our thesis, especially the reasoning behind each SQL style rule.
Lastly, in the domain of static analysis for SQL there has been conducted research for measuring the complexity of embedded queries or even the creation of tools able to reconstruct them, see [20] [21] [22].

The present study lies in the area of empirical studies that assess adherence to coding conventions and best practices. Let us refer to important contributions to this research area.
Smit et al [23] investigated code convention adherence in evolving software and got to the conclusion that the violations may increase or decrease depending on the particular project. Butler et al. [24] evaluated the quality of identifier names in 8 established open source Java applications libraries, using a set of 12 identifier naming guidelines and found statistically significant associations between flawed identifiers and code quality issues by a static analysis tool. Focused on naming conventions, Butler et al. [25] found that developers follow naming conventions to a certain degree, but adherence to specific conventions varies widely, depending on the project.
The work of Buse et. al [26] showed positive correlations between readability, comments and blank lines but negative correlations between readability, identifiers and line length. The empirical study of Lawrie et al. [27] was about the impact of names in source code comprehension. Further, the empirical studies of Binkley et al. investigated

the impact of identifier style to source code comprehension [22] and the impact of vocabulary normalization on software engineering activities [21].

The paper of Capiluppi et al. [28] reports on the empirical analysis of two major forges where OSS projects are hosted. Results from this analysis form a complex picture; visually, all the selected metrics show a clear divide between the two forges, the first forge that provides a set of guidelines and coding standards in the form of a coding style for the developers and the second studied forge, SourceForge, which imposes no formal coding standards on developers. From the statistical standpoint, however, clear distinctions cannot be drawn amongst these quality related measures in the two forge samples.

CHAPTER 3

SQL STYLE: DEFINING RULES, CONVENTIONS AND METHODOLOGY

To cover the need for an adequate SQL style checking tool, in this thesis we created Dbsea. In this chapter we discuss the modus operandi of the proposed tool and the fundamental concepts of this thesis.

3.1Overview

The proposed tool takes as input either a single version of a particular schema, or the entire evolution history of the schema, which consists of a set of schema versions, from the birth of the schema to its last known version. It is also possible to conduct style analysis in the evolution history of multiple schemata, in this case Dbsea will parse the schemata in depth first fashion. The SQL style checking analysis is based on a set of rules introduced in two well-known sources of good coding practices, namely J. Celko's SQL programming style [18], and R. Martin's book on *clean code* [19]. Moreover, we consider rules derived from our own experience as developers.

Concerning their scope we can divide the rules that we consider in two categories, table rules and column rules. Overall, we have 15 rules for tables and 17 rules for column, with some rules being common in both cases.

Regarding the intent/nature of the rules' we can divide them in four categories as follows:

- *Style of writing:* This category concerns the formatting of the SQL schema specification. Hence, this category includes rules that have a visual impact on the SQL schema specification, like the use of Pascal Case for table/column names. Also includes practices like the use of spaces, the capitalization of the first letter and other that affect the readability of the SQL schema specification (e.g., Universal Type of Case, Ends with Letter or Number).
- *Methodological:* This category contains rules concerning the systematic use of particular naming patterns. An example is naming "id" every primary key in a schema.
- *Language Specifics:* This category includes rules derived from SQL limitations or other specific constraints, like the length of a name being less than thirty characters. (e.g., Proper Length, Reserved Word).
- *Lexicological:* This category contains rules that concern part of speech issues for the terms used for the specification of the different schema elements (e.g., Contains Verb, Contains Only Singular).

The style checking analysis is done in following two levels:

Schema Level Analysis (SLA): At this level we examine a given schema as a whole, measuring the number of elements (i.e., tables or columns, depending on the scope of the rule), that follow each one of the rules that we consider.

Table Level Analysis (TLA): At this level, we focus on each schema element (i.e., tables or columns, depending on the scope of the rule), checking its adherence to each one of the rules that we consider.

While checking a schema, for each level of analysis the tool produces as output respective CSV files that contain information about tables and columns based on the adherence of the rules. For each of those CSV files the tool produces another file containing statistics about the nature of the dataset.

The overall flow of the analysis is show in the activity diagram of Figure 1.

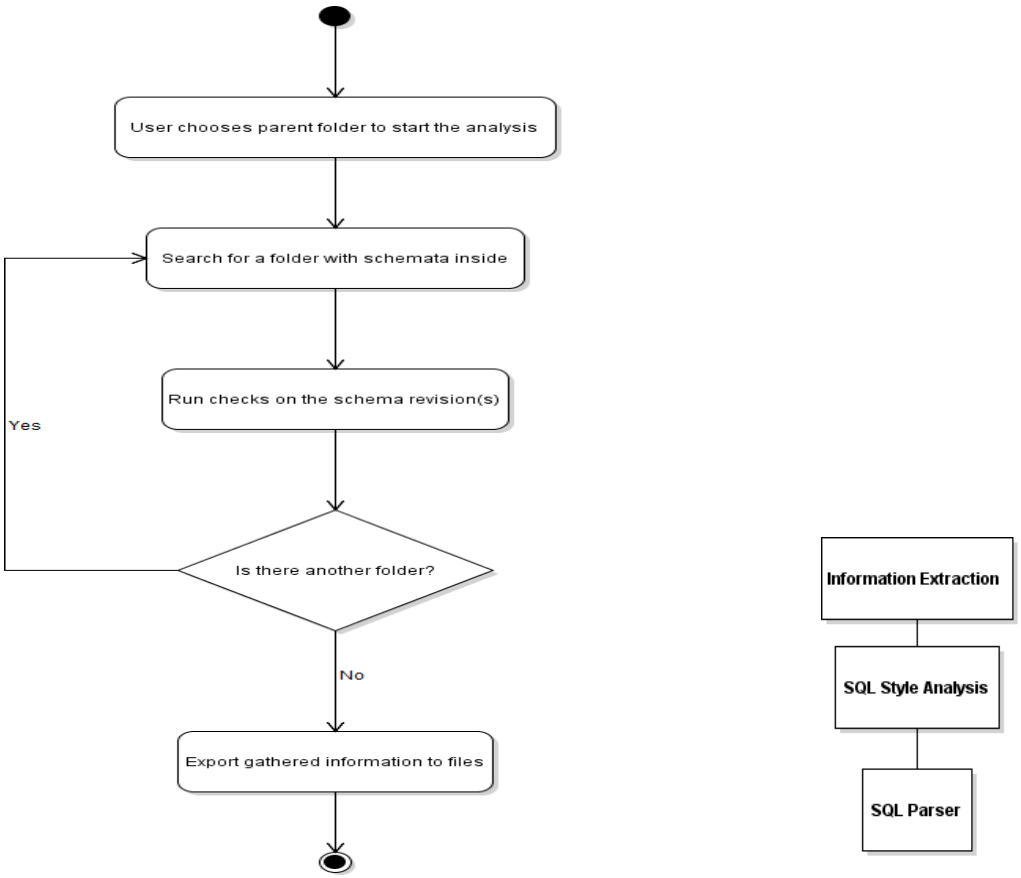


Figure 1: Activity diagram and architecture of Dbsea.

3.2 SQL Style Rules

Introduction and Lineage

Differently from typical programming languages like C, C++, C#, or Java, for SQL there is no widely accepted programming style. Nevertheless, there exist certain general guidelines and rules found in different sources [18] [29] [19] that can be considered in our investigation.

Based on these sources and our personal experience as developers, we define a catalog of rules for SQL scripting. In detail, Celko relies on the ISO-11179[8] metadata standards and his experience to define rules regarding the readability of SQL code. This is, to our knowledge, the first attempt towards a unified SQL Style. On the other hand, Robert C. Martin provides general coding conventions regarding the style of writing, for example the code length, the readiness of code like the use of pronounciable names, avoidance of hungarian notation, the use of nouns for class names and more, and so on. This work inspired us to create some additional rules that will assist us to a better assesment of the meaningfulness and readability of the names used in an SQL specification

Overall, we consider fifteen rules for tables and seventeen rules for columns. In the following sub-sections we discuss in detail the rational of the examined rules. Moreover, in Table 1, we provide the classification of the rules with respect to their intent, origin, and scope.

SQL Style Rules Description and Rationale

Use universal type of case, (UTC) Origin : Authors

In general, coding conventions should not vary throughout a project. As developers, we prefer the code to be predictable and homogeneous. In our context, a schema should follow the same type of case for every element name.

[8] https://www.iso.org/standard/60341.html

Rule	*Type*	*Lineage*	*Tables*	*Columns*	*Acronym*
Use universal type of case	*Style of writing*	*Authors*	*√*	*√*	*UTC*
Always start with letter	*Style of writing*	*Celko*	*√*	*√*	*SWL*
Always end with letter or number	*Style of writing*	*Celko*	*√*	*√*	*EWL*
Avoid camelCase	*Style of writing*	*Celko*	*√*	*√*	*ACC*
Avoid consecutive underscores	*Style of writing*	*Celko*	*√*	*√*	*ACU*
Avoid using spaces	*Style of writing*	*Celko*	*√*	*√*	*AUS*
Avoid special characters	*Style of writing*	*Celko*	*√*	*√*	*ASC*
Avoid using delimiters	*Style of writing*	*Celko*	*√*	*√*	*AUD*
Start with capital	*Style of writing*	*Celko*	*√*		*SWC*
Avoid concatenation of table names	*Methodological*	*Celko*	*√*		*ACN*
Use standardized postfix	*Methodological*	*Celko*		*√*	*USP*
Avoid "id" as identifier	*Methodological*	*Celko*		*√*	*AII*
Avoid defining name by place	*Methodological*	*Celko*		*√*	*NBP*
Use different name for columns	*Methodological*	*Celko*		*√*	*DNC*
Use proper length	*SQL*	*Celko*	*√*	*√*	*UPL*
Avoid reserved words	*SQL*	*Celko*	*√*	*√*	*ARW*
Use more words	*Lexicological*	*Authors*	*√*	*√*	*UMW*
Columns in singular	*Lexicological*	*Celko*		*√*	*CIS*
Tables in plural	*Lexicological*	*Celko*	*√*		*TIP*
Avoid using verbs	*Lexicological*	*R. Martin*	*√*	*√*	*AUV*

Table 1: SQL style rules classification, regarding their intent also origin and scope of use.

This rule tends to be violated by new developers who don't adhere to the coding conventions of the organization of corporation they work for. The violation could lead to frustration when a developer who writes scripts suddenly finds out the half of the columns are in lower case with underscores and the other half is camelCase.

To check for adherence to this rule in the tool we consider six different cases that are commonly used in practice, namely, *upper case*, *lower case*, *upper case with underscore*, *lower case with underscore*, *PascalCase*, and *camelCase.* For a given schema (respectively table), we identify the type of case that is followed by each table (respectively column). Then, we find the maximum number of tables (columns) that

follow the same type of case. Ideally, this number should be equal to the total number of tables (columns) that constitute the given schema (respectively tables).

Always start with a letter *(SWL)*, Origin : Celko

The name of an SQL element should begin with a letter, this is the norm in real life why this should not be the case the code?

Always end with letter or number (EWL), Origin : Celko

The name of an SQL element should end with a letter of a number. Using other characters can create confusion, for example if we use underscore we could think that something is missing, especially if the universal type of case contains underscores. If we use something other than an underscore, one could easily think of it as typing error inserted from the developer by mistake.

Avoid camelCase (ACC), Origin : Celko

Celko recommends avoiding camelCase, both in table and column names. camelCase tends to disrupt the flow or reading by forcing the eye to focus on case changes, instead of focusing on whole words. Moreover as ACC conflicts with another rule (SWC) that demands table names should start with capital letters.

Avoid consecutive underscores (ACU), Origin : Celko

Consecutive underscores have little to no use in naming and even less appealingness to the eye. Celko gives a real world problem with them; if a developer was to review a printed version of a schema he would face the difficulty of reading names with consecutive underscores because underscores are hard to be counted in a printed version.

Avoid special characters (ASC), Origin : Celko

Several DBMSs do not allow special characters like $, #, or @. This means that the use of special characters in SQL element names can potentially lead to compatibility errors.

	Standard SQL	IBM	Oracle	Microsoft
First Character	Letter	Letter, $#@	Letter	Letter, #@
Later Characters	Letter, Digit, _	Letter, Digit, $#@_	Letter, Digit, $#)	Letter, Digit, #@_
Case Sensitive?	No	No	Nonquoted identifier	Optional
Term		Ordinary identifier	Nonquoted identifier	Regular identifier

Figure 2: Supported special characters and information about the characters use in various DBMS.

Avoid using spaces (AUS), Origin : Celko

Developers are not used to read names with spaces, so why use them? Also spaces will make scripts more prone to SQL syntax errors, like in the example below, taken from Celko's book where the first statement is correct and the second generated by ADO [30] is wrong.

```
INSERT INTO Table ([field with space]) VALUES (value);

INSERT INTO Table (field with space) VALUES (value);
```

So we could have compatibility errors like with special characters.

Avoid using delimiters (AUD), Origin : Celko

The use of delimiters in SQL elements names should be avoided. The main reason for using delimiters in SQL is case sensitivity and compatibility. Case sensitivity rules vary

from product to product. As stated in the Oracle Database SQL Language Reference[9] *non quoted identifiers are not case sensitive. Oracle interprets them as uppercase. Quoted identifiers are case sensitive*. This means that by enclosing strings inside delimiters we can create different names for example:

"employees"

"Employees"

"EMPLOYEES"

Every one of the above words is a different name in Oracle's SQL. In the contrary, the names below are the same:

employees

EMPLOYEES

"EMPLOYEES"

According to MSSQL[10], delimiters should be used for two reasons:

i. When reserved words are used for object names or parts of object names.
ii. When you are using characters that are not listed as qualified identifiers.

IBM[11] also limits the use of delimiters in the use of reserved words or when character does not qualify as an ordinary identifier.

Concluding, the usage and meaning of identifiers varies with the underlying system such variance impairs migration capabilities, thus identifiers should be avoided.

[9] https://docs.oracle.com/cd/A97630_01/server.920/a96540/sql_elements9a.htm

[10] https://docs.microsoft.com/en-us/sql/?view=sql-server-2017.

[11] https://www.ibm.com/support/knowledgecenter/en/ssw_ibm_i_72/db2/rbafzintro.htm

Start with capital (SWC), Origin : Celko

The names of composite schema elements like tables, views and procedures should be capitalized. Capitalization is used to start a sentence in most of the languages So its already a globally used practice to separate different contexts. Reading scripts is in a way the same as reading sentences, we search for different contexts, we might as well use a practice to help us separate them. Celko outlines an exception, when a name naturally begins with lowercase.

Avoid concatenation of table names (ACN), Origin : Celko

Concatenation is typically used when a developer wants to show a relationship between tables. Since databases tend to have many relationships this practice could create a lot of tables having a part of their name common. From our own experience, names would also be getting longer thus one should either create name with length bigger that thirty characters or induce acronyms. Acronyms would make the database harder to read and would increase the learning curve of the database for a new developer on the project.

As a last note, a relationship based name between two tables is always suggested, but in the case where we need to invent one, for example by using acronyms, we better use name concatenation.

Use standardized postfix (USP), Origin : Celko

Many corporations or organisms use standardized postfixes. For instance, a postfix could be "_id" referring to a key of a table or "_cat" where cat stands for category. In the aforementioned example the meaning of "_cat" is not obvious. However, this is not really a problem, as long as the used postfix has conceptual relationship inside the database. Generally. the purpose behind the using uniform postfixes is mapping entities of the outside world with objects inside the database.

To check the adherence to this rule in Dbsea we employ a list of standard, widely used postfixes. Then, we check whether these postfixes are used as parts of the names that are used in a given schema.

Avoid "id" as identifier (AII), Origin : Celko

This rule is a well-known, popular sin among developers. Usually it is found in databases where the primary key is an auto-increment integer. Using "id" as the primary key makes this rule conceptual as defined in the Overview, everyone can easily understand that a column with this name concerns a primary key. Unfortunately using id in such manner conveys two problems.

The first problem is the vagueness. "id" is *a means of identification,* and that's exactly the limitation, there is no context at all, using only "a means" lacks context.

The second problem is that if *"id"* is used in a conceptual revealing manner, this means setting all, or the at least the majority of primary keys in the database as *"id"*, we would end up having a great number of queries with statements like the one below :

Transactions.ID = TransactionId. When primary keys are named as ID and foreign keys are created through the concatenation of the table' name, where the primary key exists, and the word *"id"* as Celko humorously pointed *"it quickly becomes a game of looking for the period"*.

Avoid defining name by place (NBP), Origin : Celko

Imagine changing your car's plate while travelling from town to town. This would change your car's identity and thus is forbidden by law. An SQL element doesn't really differ from the car and plate paradigm. Its name should be based on its identity, not the place where it is found, otherwise one may end up having different names for the same element even if this element is a structural component of a database. The validity of this rule is easy to be seen in the following simple example where in the first query the rule is violated. There is a vehicle plate in both TRAFFIC_LAW_INFRINGEMENTS and CARS tables, but in the first one the plate is named as *TRAFFIC_LAW_INFRINGEMENTS_PLATE* and in the second as *CARS_PLATE* they should both be named as *PLATE*:

```
SELECT TRAFFIC_LAW_INFRINGEMENTS.OWNER
FROM TRAFFIC_LAW_INFRINGEMENTS
INNER JOIN CARS ON
TRAFFIC_LAW_INFRINGEMENTS.TRAFFIC_LAW_INFRINGEMENTS_PLATE =
CARS.CARS_PLATE


SELECT TRAFFIC_LAW_INFRINGEMENTS.OWNER
FROM TRAFFIC_LAW_INFRINGEMENTS
INNER JOIN CARS ON TRAFFIC_LAW_INFRINGEMENTS.PLATE = CARS.PLATE
```

SQL Snippet 1: In the first select statement we do not use the *Name defined by place* rule, the second select much easier to read.

It is quite common for this violation to occur when COBOL naming logic is used. The COBOL naming logic dictates that the name of the fields should have the name of the file in which they exist as a prefix in their name, so when a migration takes place with minimum effort, old entities are mapped in the new system in the exact way they were in old system.

Another reason is that developers *have habits* [19] they tend to use throughout their carrier.

To assess the adherence to this rule in the tool we check if the name of a particular table is used as prefix in the names of the table's column names.

Use different name for columns (DNC), Origin : Celko

Although in it is possible to create column names that are same as the table names, this is not considered a good practice.

Firstly, this practice violates other rules discussed in the following like

Columns should be in singular (CIS), Origin : *Celko*. Secondly by doing so we ought to think if a table represents a collection or if a column represents a single characteristic.

Use proper length (UPL), Origin : Celko

The SQL-92 standards define a maximum identifier length of 18 characters. More modern DBMSs allow for more than 30. The problem lies in complexity; Celko states that *"if you cannot say it in 18 characters, then you have a problem".* The statement might be an exaggeration but it still holds value. It's to our best interest to not exhaust or abuse the features new technologies have to offer, and keep *the identifiers simple* [19].

We consider a violation of this rule occurring when the length of a name is greater than 30 characters.

Avoid reserved words (ARW), Origin : Celko

Reserved words are a list of terms that have some special meaning in SQL. If used in the context of the DBMS, they tend to convey their meaning in a straightforward manner. On the contrary, if they are found in a table definition as column names chances are that these names would be vague and vagueness should be avoided

For example if we were to use *LANGUAGE (reserved word) as a column name* inside two different tables, where in the first one we would refer to the user language and in the second to the program's language wouldn't we create confusion?

Usually confusion leads to errors. The use of reserved words as names should be avoided.

Use more words (UMW), Origin : Authors

often, in our everyday life we use acronyms for well-known entities like a big corporation or inside our close social circle for easiness of communication. Likely, in the case of an SQL schema, used in the context of a real-world project this is not always

the case; developers may come and go and the well-known entities usually have, or at least should, distinct names. We believe that by using distinct, simple names, we can avoid inducing acronyms to our SQL schema. Acronyms need translation, words don't.

To assess the adherence to this rule, we divide a name into parts, based on the type of case it assumes. Then, we check whether these parts correspond to actual words (specifically, nouns, adverbs and adjectives) based on WordNet[12], a large lexical database. Nouns, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept. Synsets are interlinked by means of conceptual-semantic and lexical relations. The resulting network of meaningfully related words and concepts can be navigated like a graph. WordNet enables us to determine if a string is an actual English word; this is not an out of the box feature, to induce such functionality we used two widely known libraries [https://nlp.stanford.edu/software/[edu.smu.tspell.wordnet].

Columns should be in singular (CIS), Origin : Celko

Celko in his description of ISO-11179-4 for scalar elements states that their name should be in singular. Tables, by nature are collections, while columns specify properties that are common, to all members of the collection. Thus it is better to define a property with a singular noun and not using any word in plural at all.

To check for this rule, we employ respective functionalities of the WordNet libraries that allow us to determine if a particular word is in singular.

[12] https://wordnet.princeton.edu/

Tables should be in plural (TIP), Origin : Celko

Just like the columns should be in singular, ISO-11179-4 states that tables should be in plural and plural only. Celko states that for this rule there is an exception, tables consisting of only one record.

Evaluating this rule involves corresponding functionalities of the WordNet libraries that allow us to determine if a particular word is in plural.

Avoid using verbs (AUV), Origin : R. Martin

Both Celko and Martin suggest the use of verbs in method names because verbs are intention revealing. A scalar element or a table should not be able to act; we have procedures, functions and queries in general for this exact reason so why would end up with names containing words describing an action?

The evaluation of this rule is based on the same part-of speech identification functionalities of the WordNet libraries that we employed in the case of UMW.

3.3 SQL Style Checking, Approach and Datasets

In the Levels of Analysis we explain our approach on researching the SQL style subject, the two levels of analyses, as mentioned above. In the next subsection, Examined Schemata, we present the schemata used to conduct our research. There are three main type of projects in which the schemata belong, the first is scientific, the second is medical and the third one is content management systems.

3.4 Levels of Analysis

Schema Level Analysis

In Schema Level Analysis (SLA) we conduct a gross-grained SQL style analysis in the examined schemas, without drilling into the individual tables or columns of the different schema versions and their evolution. So, in this level we see the tables/columns of a schema as a whole and we count the ones that follow each rule. Moreover, we focus on high-level statistical measures like the mean, median, standard deviation of the number of tables or columns that follow each rule divided by the respected total sum of tables/columns in a schema. We further consider possible correlations between the number of objects that follow the rules with the size of the schema.

To characterize the extent of use of the rules in a given schema, we introduce the *SRAD* metric which stands for *Schema Rule Adherence Degree*.

SRAD for each rule concerning the table names in a schema is:

$$SRAD = \frac{\#\text{table names of the schema that follow a rule}}{\#\text{table names in the schema}}$$

For the table names of a schema when used in a figure, or chart, *SRAD* might also be referred as *Tables_SRAD*.

SRAD for each rule concerning the column names is:

$$SRAD = \frac{\#\text{column names of the schema that follow a rule}}{\#\text{column names in the schema}}$$

SRAD for the table names of a schema when used in a figure, or chart might also be referred as *Columns_SRAD*.

To facilitate SLA, Dbsea creates a csv file where in the vertical dimension we have the revisions and in the horizontal dimension we give the percentage of use for each rule.

Next, we present a simple example. The script that is given in SQL Snippet 2, the script is from a particular version of one of the Examined Schemata .The script is only a fraction of the schema.

We focus on three rules for table names, (*UTC, SWC, TIP)*, the format of results for these rules, as generated in the csv file, is depicted in Table 3.

In this example we see 2 tables. All of them follow the Universal type of case rule, so the value of the respective column is 100%. There is no table starting with a capital letter and none of the tables contains nouns in plural. Hence, the values of the respective columns are 0%.

```
CREATE TABLE l1_trigger_type (
        l1tty_id NUMBER(10)NOT NULL,
        …
        CONSTRAINT l1tty_nmver UNIQUE (l1tty_name, l1tty_version)
        );

CREATE TABLE l1_muctpi_info (
        l1mi_id NUMBER(10) NOT NULL,
        …
        l1mi_used CHAR NOT NULL,
        CONSTRAINT muctpi_pk PRIMARY KEY (l1mi_id),
        CONSTRAINT muctpi_nmver UNIQUE (l1mi_name, l1mi_version)
        );
```

SQL Snippet 2: A snapshot taken from the Atlas case study.

Total tables	UTC	SWC	TIP
2	100%	0%	0%

Table 2: The results of *SLA* for SQL Snippet 2.

Table Level Analysis

In Table Level of Analysis (TLA) we perform a fine-grained SQL style analysis in the examined schemas. In each version, we check each table on its own. Specifically, we examine each schema element (i.e., tables or columns, depending on the scope of the rule), checking its adherence to each one of the rules that we consider. In the case of tables we assess whether a table rule is followed or not in a zero or one fashion. In the case of columns we measure the number of table columns that follow a rule and divide this number with the total number of columns of this table. To facilitate TLA, *Dbsea* creates a CSV file where in the vertical dimension for each version we have its tables (or the sum of the columns of a table) and in the horizontal the value that measures the satisfaction level of rule.

To characterize the extent of use of the rules we introduce the *TRAD* metric which stands for *Table Rule Adherence Degree*.

TRAD describes the adherence of a rule for a table name or for the column names of a table. More precisely:

TRAD for a table name:

$$TRAD = \begin{cases} 1, rule\ is\ adhered \\ 0, otherwise \end{cases}$$

For the column names of a table:

$$TRAD = \frac{\#\text{column names of the table that follow the rule}}{\#\text{column names in table}}$$

Getting back to the example of Atlas, in TLA lets check if the schema tables follow the following rules: *Always begin with letter*, *Avoid concatenation of table names and Use proper length.* Table 3 gives the results, as generated in the csv file. In the first column we see the tables. Both columns start with letter, none of them has the name of the table they exist concatenated and both have proper length. In the next columns we see that

both tables begin with a letter. Their names are not concatenations of other table names. Finally, the length of the table names is appropriate.

Columns	SWL	ACN	UPL
l1_trigger_type	1	0	1
l1_muctpi_info	1	0	1

Table 3: The results of *TLA* for SQL Snippet 2.

3.5 Examined Schemata

In this section we give a brief description for every dataset used in our study. The collection of the datasets that we consider in the study was done by Athanasios Pappas[13].

In the collection there are three types of projects; scientific, medical and Content Management Systems. In detail, we have five scientific projects from CERN, two medical projects and eleven CMS projects. For more information about the datasets please refer to thebibliography.

In Table 4 we provide basic information about the schemata of our case study.

Scientific projects:

1. ATLAS[14]: is a particle physics experiment at the Large Hadron Collider at CERN.

[13] https://github.com/DAINTINESS-Group/EvolutionDatasets

[14] https://atlas.cern/

2. CASTOR[15]: is a hierarchical storage management system which was developed at CERN for archiving physics data.
3. SRM2: a client system for CASTOR.
4. DQ2[16]: a data management system for atlas.

Schema	#Revisions	#Tables at birth	#Tables at LKV	#Columns at birth	#Columns at LKV
Atlas	84	73	56	709	857
SRM2	58	11	11	54	84
CASTOR2	192	62	74	632	838
DQ2MySQL	54	10	26	116	184
EGEEMySQL	16	6	9	34	63
Coppermine	117	8	22	85	169
e107	17	33	34	261	274
Joomla 1,5	45	35	36	307	321
NucleusCMS	4	20	20	110	112
phpBB	133	61	65	613	565
phpwiki	21	10	10	33	49
SlashCode	398	42	87	259	610
TikiWiki	153	207	215	1528	1628
Typo3	98	10	23	122	421
DekiWiki	16	28	40	204	315
wikimedia	322	17	50	100	318
Zabbix Oracle	27	47	48	312	313
OpenCart	165	48	114	74	230
XOOPS	7	31	32	297	129
Medbiosql	47	21	28	227	731
Ensembl	528	19	75	82	486

Table 4: General information about the schemata.

5. EGEE[17]: is a series of efforts to provide access to high-throughput computing resources across Europe using grid computing techniques.

Medical projects:

[15] http://castor.web.cern.ch

[16] https://www-zeuthen.desy.de/technisches_seminar/texte/dq2.pdf

[17] http://information-technology.web.cern.ch/about/projects/eu/egee-iii

1. Ensembl[18]: is a project that produces genome databases for vertebrates and other eukaryotic species.
2. BioSQL[19]: is a joint effort between the OBF projects (BioPerl, BioJava etc), to support a shared database schema for storing sequence data.

Content Management Systems:

1. Typo3[20]: is an enterprise CMS for managing any kind of digital content.
2. PhpBB[21]: is an Internet forum package written in PHP.
3. PhpWiki [22] : is a WikiWeb clone in PHP that supports multiple storage backends, dynamic hyperlinking, and more.
4. Slashcode[23]: is the site for All Things Slash.
5. ZABBIX[24]: is an enterprise-class open source distributed monitoring solution that provides information about numerous parameters of a network and the health and integrity of a server.
6. e107[25]: is a website system written in PHP and MySQL for the creation of dynamic sites providing a flexible admin area.
7. Coppermine[26]: is a photo gallery with a MySQL database, some of its features are user management, private galleries and automatic thumbnail creation.
8. DekiWiki[27]: is a popular commercially supported wiki platform for creating content and mashups using a wiki interface.
9. Nucleus[28]: is a simple CMS based on PHP and JavaScript.

---

[18] https://www.ensembl.org/index.html]

[19] http://biosql.org

[20] https://typo3.com

[21] https://www.phpbb.com

[22] https://sourceforge.net/projects/phpwiki

[23] http://www.slashcode.com/www.slashcode.com/about.shtml

[24] https://www.zabbix.com

[25] https://github.com/e107inc/e107

[26] http://coppermine-gallery.net

[27] https://www.osalt.com/mindtouch

[28] https://sourceforge.net/projects/nucleuscms

10. OpenCart[29]: is an ecommerce platform for online merchants.
11. TikiWiki[30]: is a powerful Content Management System (CMS).
12. XOOPS[31]: is a web application platform in PHP and MySQL for developing small or large community websites.
13. MediaWiki[32]: is a wiki software package written in PHP that serves as the platform for Wikipedia and the other Wikimedia projects.
14. Joomla! 1.5[33]: is a content management system (CMS) for publishing web content.

[29] https://github.com/opencart/opencart

[30] https://tiki.org/tiki-index.php

[31] https://xoops.org

[32] https://github.com/wikimedia/mediawiki

[33] https://github.com/joomla/joomla-cms

CHAPTER 4

EXPERIMENTAL STUDY: STATUS QUO OF SQL STYLE AND A DOSE OF IDEALISM

4.1Do People Care About SQL Style Rules?

Coding conventions are important in the case of conventional software, as evidenced in the related research and practice. Developers typically follow specific coding styles and standard coding conventions, at least those who care. The issue is what happens in the case of SQL, where the lack of related research and empirical studies is profound. To answer the initial question of this thesis, we checked if the rules that we defined in section SQL Style Rules, are followed by the schemata that we consider in our study.

To come up with a quick answer to the question we calculate the average values and the standard deviation of the rules SRAD for tables and columns, based on all the datasets of our case study. The respective results are given in.

So, in Table 5 we observe for most of the table rules high average SRAD values that range from 95% to 100%, with medium-low standard deviations, ranging from 0% to 22%. Only four rules (i.e., SWC, TIP, UMW, ACN) have lower average SRAD values that range from 15% to 70%. In Table 7, we observe a similar situation for the column rules. Most of the column rules come with high average SRAD values that range from 93% to 100%, with medium-low standard deviations, ranging from 0% to 13%. Only four rules (i.e., USP, AUV, UMW, CIS) have lower average SRAD values that range from 28% to 65%.

	Average SRAD	STD of SRAD
UTC	99%	3%
UPL	98%	6%
SWL	95%	22%
EWL	100%	0%
UMW	56%	24%
TIP	17%	19%
SWC	15%	36%
ACC	100%	0%
ARW	100%	0%
ACU	95%	22%
AUS	100%	0%
ASC	100%	0%
AUD	100%	0%
AUV	100%	0%
ACN	70%	21%

Table 5: Average and standard deviation of the last known versions of the schemata for each one of the table rules SRAD.

	Average SRAD	STD of SRAD
UTC	94%	14%
UPL	100%	0%
SWL	100%	0%
EWL	100%	0%
UMW	59%	21%
USP	28%	14%
CIS	65%	23%
ACC	93%	18%
ARW	100%	0%
ACU	100%	0%
AUS	100%	0%
ASC	100%	0%
AUD	100%	0%
AUV	51%	18%
AII	98%	4%
ACN	100%	0%
NBP	93%	16%

Table 6: Average and standard deviation of the last known versions of the schemata for each one of the column rules SRAD.

Overall, we have a first sign that people do care about SQL style rules to a certain extent. To get a more detailed view of what happens in our case studies we further calculate for each schema *the percentage of rules followed by all of the schema elements* (tables/columns) and *the percentage of rules that are not followed by any element*.

Moreover, we look at *the percentage of rules followed by at least some of the schema elements* and *the percentage of rules that are not followed by some elements*. In detail, we focus on the LKV of each schema. For the table rules we provide a bar chart that gives:

- the percentage of rules followed by all tables (i.e. SRAD = 100%),
- the percentage of rules not followed by any tables (i.e. the percentage of rules for which SRAD = 0%),
- the percentage of rules followed by some tables (i.e. the percentage of rules for which SRAD > 0%),
- the percentage of rules not followed by some tables (i.e. the percentage of rules for which SRAD < 100%).

Similarly, for the column rules we provide a bar chart that gives:

- the percentage of rules followed by all columns (i.e. the percentage of rules for which SRAD = 100%),
- the percentage of rules not followed by any column (i.e. the percentage of rules for which SRAD = 0%),
- the percentage of rules followed by some columns (i.e. the percentage of rules for which SRAD > 0%),
- the percentage of rules not followed by some columns (i.e. the percentage of rules for which SRAD < 100%).

To dive into further details, for each schema we also provide bar charts that show what happens with each table. In particular, for the table rules we provide a bar chart that gives *for each table* the percentage of rules that are followed by the table (i.e. the percentage of rules for which TRAD = 1) and the percentage of rules that are not followed by the table (i.e. the percentage of rules for which TRAD = 0). Similarly for the column rules we provide a bar chart that gives *for each table* the percentage of rules followed by all columns (i.e. the percentage of rules for which TRAD = 100%), the percentage of rules not followed by any column (i.e. the percentage of rules for which TRAD = 0%), the percentage of rules followed by some columns (i.e. the percentage of rules for which TRAD > 0%), the percentage of rules not followed by some columns (i.e. the percentage of rules for which TRAD < 100%).

The respective results for SRAD are given in Figure 3 and Figure 4. The figures group the results of the different cases studies alphabetically. Similarly, the results for TRAD are illustrated in pages 86 and 88 in

Appendix A

Concerning the detailed results, in all schemas we observe the following *schema-level adherence pattern*:

Tables:

- The percentage of rules that hold at least for some tables is high, ranging from 80% to 100%.
- The percentage of rules that are violated by some tables is medium, varying from 13% to 40%.
- The percentage of rules that hold at least for some tables is always higher than the percentage of rules that are not followed by some tables.
- The percentage of rules that hold for all tables is medium high, varying from 60% to 87%.
- The percentage of rules that do not hold for any table is low, ranging from 0% to 20%.
- The percentage of rules that hold for all tables is always higher than the percentage of rules that do not hold for any table.

Columns:

- All rules hold at least for some columns is 100% in all the schemata.
- The percentage of rules that hold for all columns is medium high, varying from 59% to 76%.
- The percentage of rules that do not hold for some columns is low medium, ranging from 24% to 41%.

Jointly:

- The percentage of rules that do not hold for any table is always higher than the respective percentage for columns in a schema.
- The percentage of rules adhered by all the tables is usually lower than the respective percentage for columns.

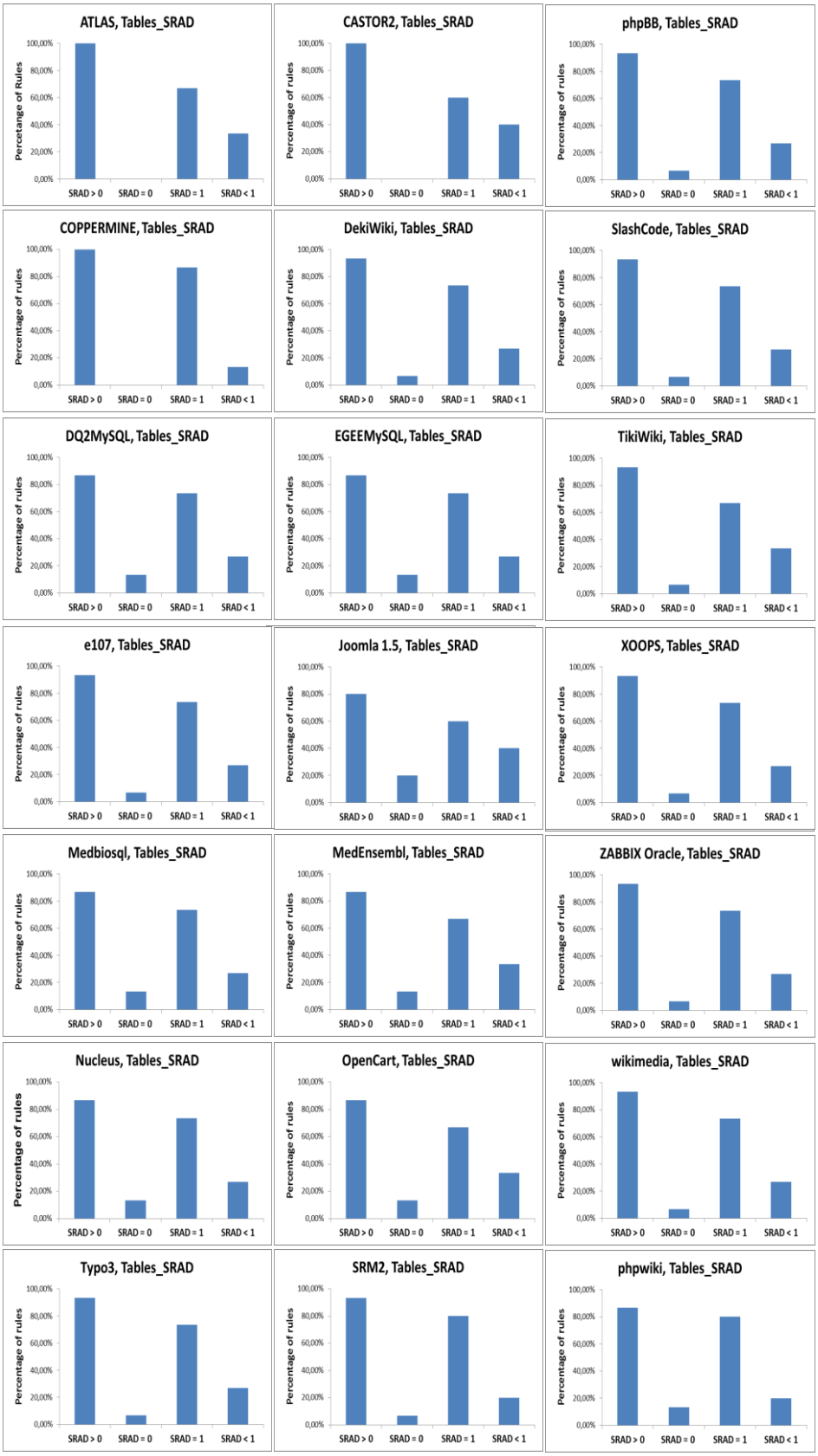


Figure 3: Percentages representing the number of table rules in each schema that have SRAD > 0%, SRAD = 0%, SRAD = 100% and SRAD < 100% respectively.

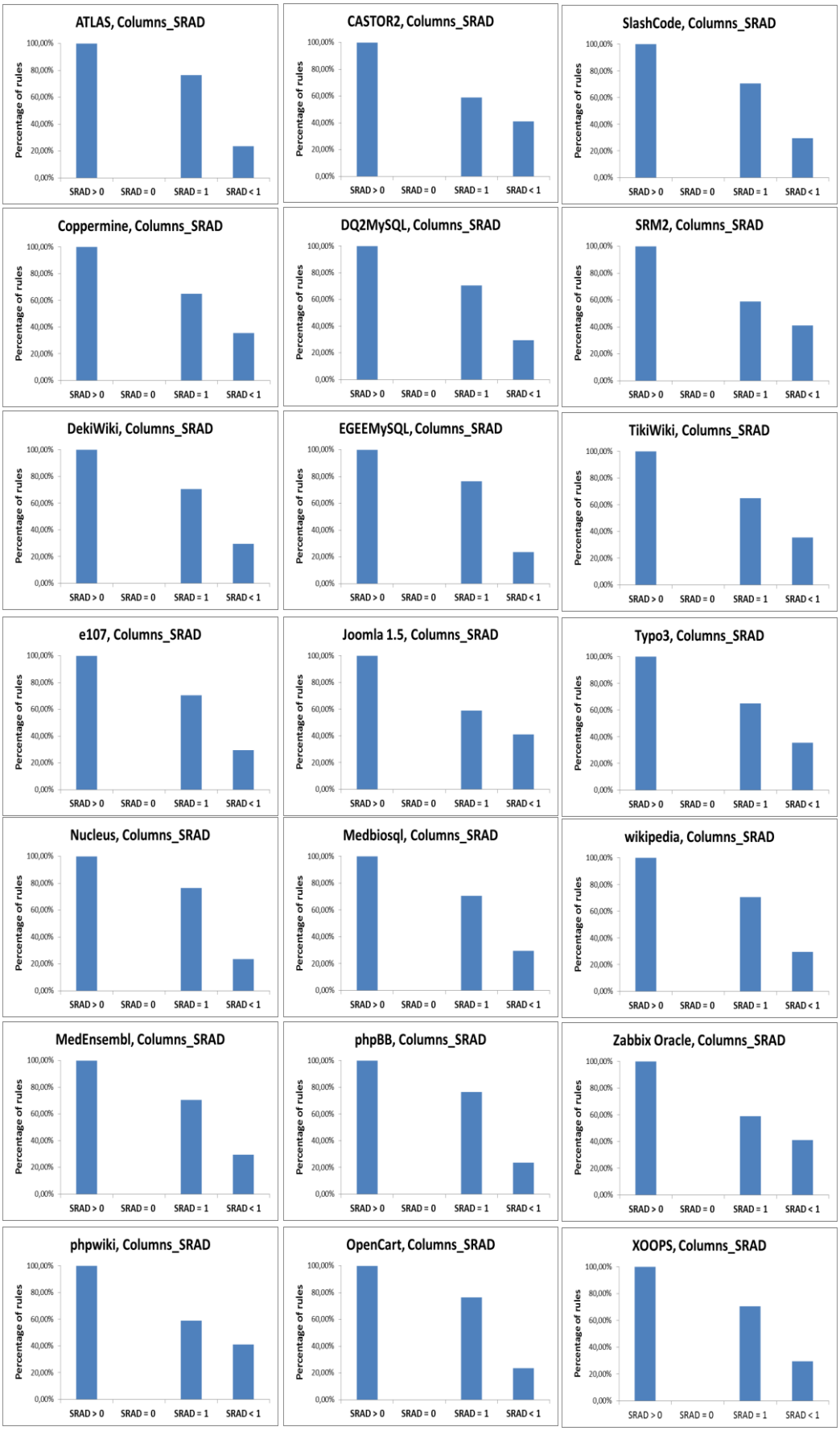


Figure 4: Percentages representing the number of column rules in each schema that have SRAD > 0%, SRAD = 0%, SRAD = 100% and SRAD < 100% respectively.

4.2 Does The Adherence To Sql Style Rules Evolve Over Time?

In RQ1 we used the LKV of each schema to assess whether the examined schemata adhere to SQL style rules. In our second research question we start from the birth of a schema and move towards the LKV; we analyze the history of the schema to find out *which rules evolve.* To this end, we use typical statistic measures (i.e., average, median, mode, standard deviation, min, max and skewness) to describe the distribution of SRAD for each rule in the history of the schema.

For each schema we further provide a line chart. Specifically, each line in the chart corresponds to a rule whose SRAD changes over the history of the schema versions; the x-axis of the chart depicts the history of the schema versions in terms of version IDs, while the y-axis depicts the value of SRAD for the particular rule. The line charts are depicted in Figure 5 *and* Figure 6. Lastly we use the Kendall correlations provided between the rules' SRAD and the size of a schema, provided by DBsea. We find and identify the significance of correlation between the rules' SRAD fluctuation across the history of a schema and it's changes in size (i.e. number of tables or columns in a given revision). For each type of SQL element we divide the values of correlations in two categories the positive and negative, each category is further divided in three based on the range of the correlation values. The three buckets refer to:

1. low correlation
    a. positive: (0%, 30%]
    b. negative: [-30%, 0%]
2. medium correlation
    a. positive: (30%, 60%]
    b. negative: [-60%, -30%)
3. high correlation
    a. positive: (60%, 100%]
    b. negative: [-100%, -60%]

The results are shown in Figure 8.

As a summary of our results, in Table 7 and

Table 8 we provide for each rule the *maximum evolution range of SRAD* (i.e., the difference between the maximum and the minimum SRAD as an absolute value) across the examined schemas.

Overall, the typical *schema-level adherence evolution pattern* that we observe in all schemas and across the different levels of detailed results is summarized below:

- Table and column rules do evolve during the life of a schema (see Figure 5 and Figure 6) .
- However, only some rules change (see Table 9) with the number of rules that change in the examined schemas varying from zero to seven for tables and zero to eight for columns.

  Typically the rules that change the most are lexicological and methodological, followed by the writing style rules. The range of the changes of *SRAD* may vary a lot depending on the schemas, the rules, and the schema elements involved. In the case of table rules for instance the maximum range of SRAD varies from an astonishing 91% for ACN to a minor 3% for *SWL* (Table 3 left). Moreover, in the case of columns the maximum range of SRAD varies from 41% for *USP* a minor 1% (Table 3 right).
- If a rule changes significantly (>10%), the biggest portion of rule's SRAD fluctuation will occur in a minor fraction of the schema's history.

The most interesting line charts for tables are from the schemata Coppermine, SRM2, Typo3, DQ2 and Ensembl. Coppermine managed in less than ten consecutive revisions to increase TIP's SRAD by 40%. SRM2 in less than five revisions increased ACN's SRAD by 70%. OpenCart decreased in only one revision its UMW's SRAD by 35% and kept it this way with minor fluctuations until the LKV. Typo3 in got ACN's SRAD decreased by 20% in less than ten revisions and in the next ten it decreased by 20%. The same phenomenon with Typo3 was found in DQ2 for ACN's SRAD, but in even smaller number of revisions, less than 3 revisions in decreased by 40% and in the next three increased by 30%. The most lively schema, based on the number of rules having their SRAD fluctuating and the multitude of SRAD is Ensembl which also is has the longest history (>500 schemata). Ensembl has an interesting fluctuation of UTC's

SRAD falling from a full adherence to about 80% and after fifty revisions returning back to 100% SRAD.

The most interesting line charts for columns are from BioSQL, DQ2, EGEE, Phpwiki, SRM2 and Wikimedia. In BioSQL the SRADs of UMW, USP and NBP change by 10% in less than five consecutive revisions. In DQ2 in less than five revisions there are significant changes for five rules, AUV decreases by 15% and then increases by about 15%, UMW and CIS increase by 20% and then decrease by approximately 15%. EGEE in less than two revisions has its AUV's SRAD increased by 10%, CIS's, USP's and UMW's SRAD decreased by approximately 8%. In the case of Phpwiki in less than three revisions CIS's and UMW's increase by 25% and AUV's decreases by 20%. SRM2 in less than five versions has the SRADs of AUV increased by 20%, CIS's and UTC's by 10%, while its ACC's decreased by 10%. Lastly wikimedia in less than two revisions has the SRAD of AUV decreased by 15% and NBP's increased by 10%.

We focus specifically on the UTC of each schema, because the SRAD of UTC is based on the most popular type of case in a schema it hides changes in the actual type of case used by the schema. Our argument is more clear through an extreme example; if we a schema was using uppercase with underscores in all of its names UTC's SRAD would be 100%, if the type of case was to change in the next revision in all of the names to PascalCase the SRAD would still be 100%. We explored the possibility of significant changes in the primary type of case with four charts (see Figure 6), two describing the change in the UTC for each schema (one for tables, one for columns for *SLA*) between the last and the first version and two charts describing the change in the used type of case for each schema respectively.

Interestingly, UTC changes through the variance type of cases usage. For example in SRM2, columns use three different type of cases, the usage for two those decreased by 6% and 2% while for the third increased by 7% (see Figure 4) the UTC's SRAD increased by 8%. Tables on the other hand have a more stable *UTC* and the different type of cases used in a schema are less.

Tables rules correlation with size

For tables the positive correlation is fairly low both in terms of number of schemata having positive correlation and in correlation's value. For UTC we there is a schema with low and one with medium correlation from the twenty one of the case study. TIP and SWC both have two schemata with low correlations. ACC has two low correlated schemata and one with medium and lastly ACN has two medium correlations. Overall it seems that table rules adherence is not correlated positively with the schema's size.

On the contrary, we observe some degree of negative correlation for a few rules. More specifically TIP and ACN are medium or highly correlated with the schema size. For TIP's SRAD there are four schemas highly and three medium correlated. For ACN the are seven with medium and five with high. Concluding, from the eight rules found having correlation with the table size of the schemata, six of them had insignificant positive or negative correlation and two of them had medium to strong negative correlation.

Columns rules correlation with size

Columns have greater correlation than tables and in more rules. The rules that had significant positive correlation where UTC, UMW, USP, CIS, AII and NBP. More specifically, for UTC the positive correlation has two weak, four medium and one high when the number of schemata where changes exist are eight, UMW has four high, one medium and five minor in the eighteen datasets where changes of SRAD occur. USP in a total of eighteen datasets has eight minor, three medium and two strong while AII in a total of eight datasets has four of them highly correlated and one medium correlated. Lastly NBP from a total of ten datasets with SRAD changes, has four of them being highly correlated, one medium and two low. The verdict is that column rules have, in some cases, strong correlations especially for style of writing, lexicological and methodological rules, leaving out the SQL specifics.

In the negative correlations there are two rules being strongly correlated with size, CIS and AUV. For CIS there are four low, three medium and four highly correlated

schemata and in AUV there are three with low, five with medium and seven with high correlation.

SRADs' distribution description during evolution

In Table 11 and Table 12 we show two descriptions for each of the SQL elements. For tables we used Ensembl and OpenCart, for columns Wikimedia and Typo3. It should be noted that these table *are* representative for all the schemata. We observe for all the rules the lack of skewness when we have standard deviation or some skewness when there is not important standard deviations (e.g. UPL in Wikimedia). Standard deviation is below 10% and usually the average, median and mode have very similar values. Interestingly we observe in the case where minimum and maximum values of SRAD for a given rule have a significant difference, mode is usually closer to the maximum value.

	UTC	UPL	SWL	UMW	TIP	SWC	ACC	ACN
ATLAS	14%	0%	0%	14%	5%	14%	0%	12%
CASTOR2	10%	0%	0%	21%	8%	10%	10%	38%
SRM2	0%	0%	0%	27%	0%	0%	0%	91%
DQ2MySQL	0%	0%	0%	7%	0%	0%	0%	61%
EGEEMySQL	0%	0%	0%	11%	0%	0%	0%	25%
Coppermine	0%	0%	0%	10%	43%	0%	0%	0%
Zabbix	0%	0%	0%	1%	2%	0%	0%	3%
DekiWiki	0%	0%	0%	11%	6%	0%	0%	4%
e107	0%	0%	0%	1%	0%	0%	0%	0%
Nucleus	0%	0%	0%	0%	0%	0%	0%	0%
OpenCart	0%	6%	0%	36%	0%	0%	0%	21%
phpBB	0%	0%	0%	5%	2%	0%	0%	8%
phpwiki	0%	0%	0%	10%	0%	0%	0%	0%
SlashCode	0%	0%	0%	18%	16%	0%	0%	13%
TikiWiki	0%	0%	0%	1%	1%	0%	0%	3%
wikimedia	3%	3%	3%	17%	9%	0%	0%	22%
Typo3	0%	6%	0%	27%	28%	0%	0%	30%
XOOPS	0%	0%	0%	2%	0%	0%	0%	2%
Joomla 1,5	0%	0%	0%	0%	1%	0%	0%	3%
biosql	0%	0%	0%	17%	5%	0%	0%	35%
Ensembl	19%	3%	0%	26%	5%	3%	15%	35%
Maximum range	*19%*	*6%*	*3%*	*36%*	*43%*	*14%*	*15%*	*91%*

Table 7: Table's SRAD ranges through the evolution of each schema, rules with zero range in all of the schemata are missing; SRAD range varies greatly based on the rule or the schema.

	UTC	UPL	EWL	UMW	USP	CIS	ACC	AUV	AII	DNC	NBP
ATLAS	0%	0%	0%	4%	6%	6%	0%	6%	0%	0%	0%
CASTOR2	23%	0%	0%	13%	10%	24%	39%	23%	9%	0%	0%
SRM2	12%	0%	0%	10%	4%	10%	12%	20%	8%	0%	4%
DQ2	0%	0%	0%	19%	16%	19%	0%	11%	8%	0%	0%
EGEE	0%	0%	0%	11%	14%	12%	0%	13%	0%	0%	0%
Coppermine	3%	0%	0%	9%	5%	8%	2%	9%	0%	0%	0%
Zabbix	0%	0%	1%	2%	1%	2%	0%	1%	0%	0%	1%
DekiWiki	0%	0%	0%	7%	7%	8%	0%	8%	0%	0%	2%
e107	0%	0%	0%	2%	1%	1%	0%	1%	0%	0%	1%
Nucleus	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
OpenCart	0%	0%	0%	3%	6%	3%	0%	4%	0%	0%	14%
phpBB	0%	0%	0%	4%	6%	1%	0%	1%	0%	0%	0%
phpwiki	0%	0%	0%	13%	7%	19%	0%	18%	4%	1%	11%
SlashCode	2%	0%	0%	7%	9%	8%	2%	9%	2%	0%	0%
TikiWiki	1%	0%	0%	1%	0%	1%	1%	0%	1%	0%	0%
wikimedia	0%	1%	0%	13%	3%	9%	0%	21%	0%	1%	30%
Typo3	6%	0%	0%	8%	7%	13%	4%	5%	1%	0%	0%
XOOPS	0%	0%	0%	0%	1%	0%	0%	1%	0%	0%	1%
Joomla 1.5	0%	0%	0%	1%	1%	2%	0%	2%	0%	0%	0%
Biosql	2%	0%	0%	15%	7%	4%	1%	8%	0%	0%	17%
Ensembl	5%	0%	0%	29%	41%	28%	5%	18%	15%	0%	17%
Maximum range	*23%*	*1%*	*1%*	*29%*	*41%*	*28%*	*39%*	*23%*	*15%*	*1%*	*30%*

Table 8: Column's SRAD ranges through the evolution of each schema, rules with zero range in all of the schemata are missing; SRAD range varies greatly based on the rule or the schema.

	#table rules that change	#column rules that change
ATLAS	5	4
CASTOR2	6	7
SRM2	2	8
DQ2	3	5
EGEE	2	4
Coppermine	2	6
Zabbix	3	6
DekiWiki	3	5
e107	1	5
Nucleus	0	0
OpenCart	3	5
phpBB	3	4
phpwiki	1	7
SlashCode	3	7
TikiWiki	3	5
wikimedia	6	7
Typo3	4	7
XOOPS	2	3
Joomla 1,5	2	4
Biosql	3	7
Ensembl	7	8

Table 9: Number of rules that change in each schema during the evolution.

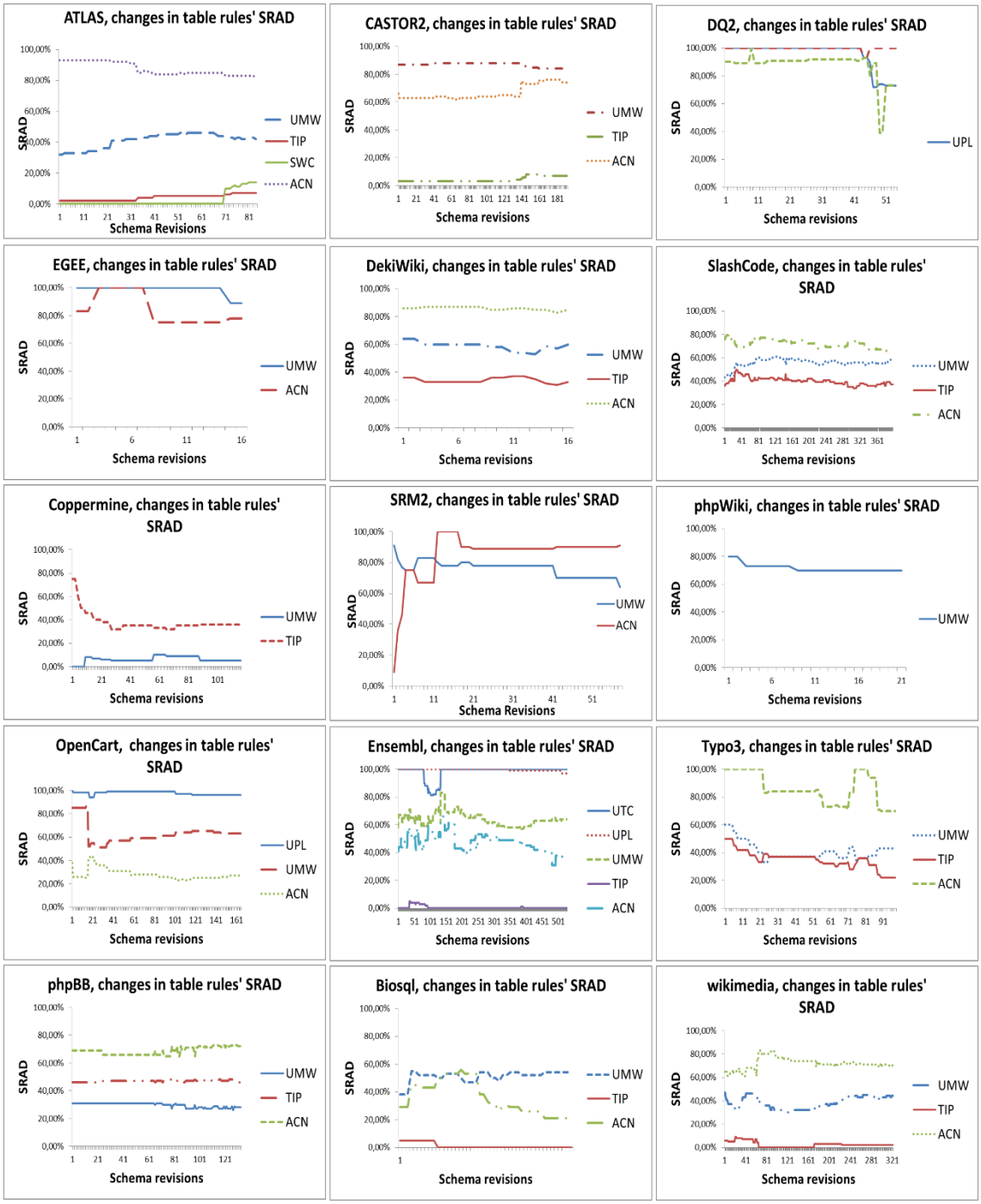


Figure 5: Changes in SRAD for table rules during the evolution of the schemata.

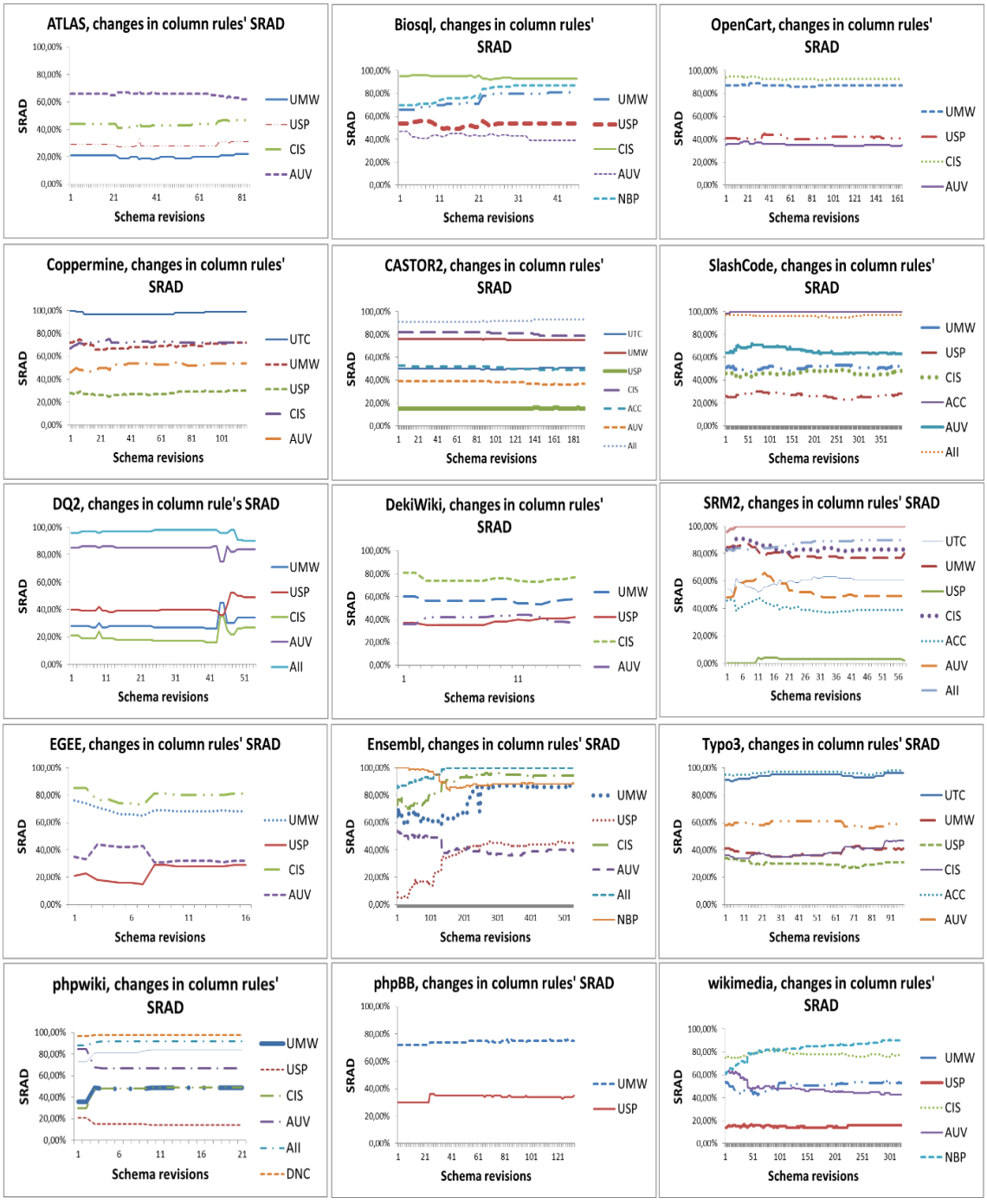


Figure 6: Changes in SRAD for column rules during the evolution of the schemata.

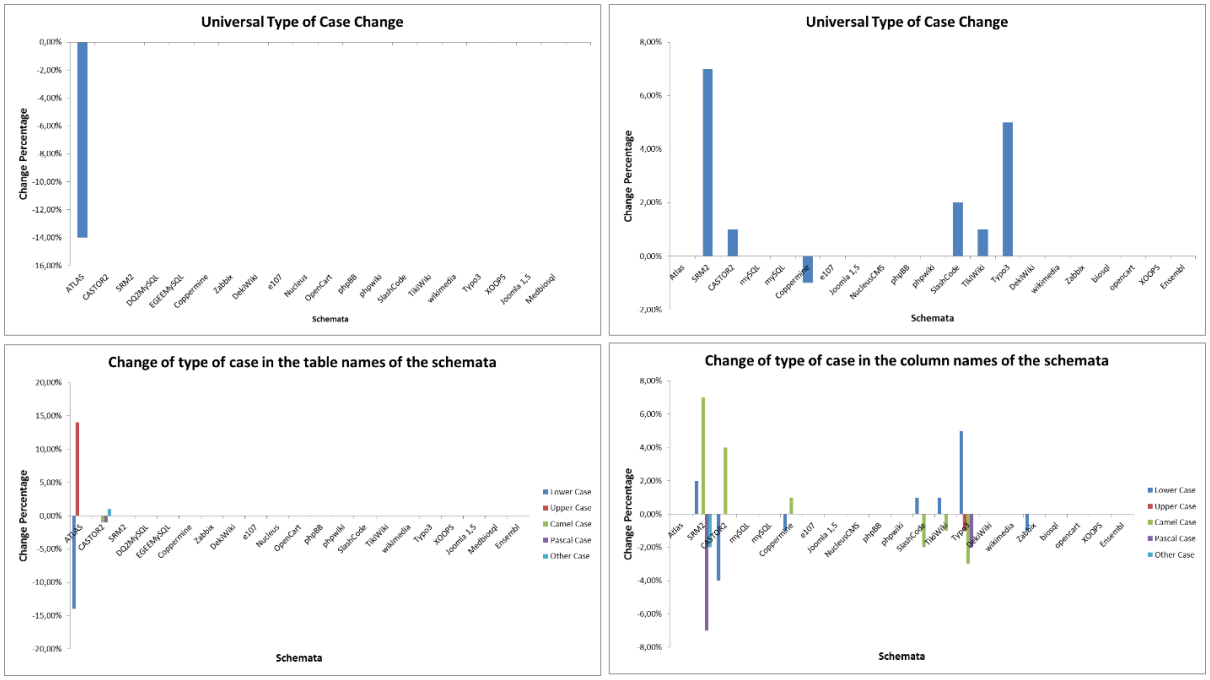


Figure 7: UTC related information the first two horizontal figures refer to the tables, the second to columns.

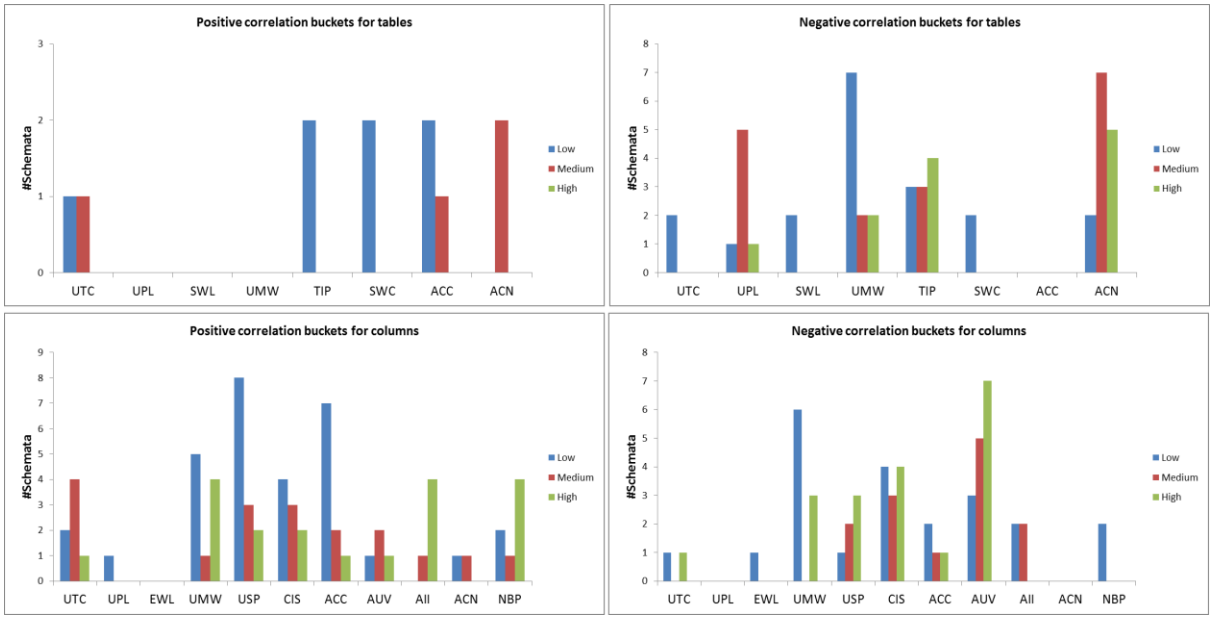


Figure 8: Number of schemata having weak, medium and high correlation, for each rule.

Ensembl	Skewness	Median	STD	Mode	Average	Max	Min
UTC	-2,86	100%	5%	100%	98%	100%	81%
UPL	-2	100%	1%	100%	100%	100%	97%
SWL	0	100%	0%	100%	100%	100%	100%
EWL	0	100%	0%	100%	100%	100%	100%
UMW	1,34	63%	5%	63%	64%	83%	57%
TIP	2,72	0%	1%	0%	0%	5%	0%
SWC	2,78	0%	1%	0%	0%	3%	0%
ACC	-2,91	100%	3%	100%	99%	100%	85%
ARW	0	100%	0%	100%	100%	100%	100%
ACU	0	100%	0%	100%	100%	100%	100%
AUS	0	100%	0%	100%	100%	100%	100%
ASC	0	100%	0%	100%	100%	100%	100%
AUD	0	100%	0%	100%	100%	100%	100%
AUV	0	100%	0%	100%	100%	100%	100%
ACN	0,29	49%	7%	49%	48%	66%	31%

OpenCart	Skewness	Median	STD	Mode	Average	Max	Min
UTC	0	100%	0%	100%	100%	100%	100%
UPL	-0,53	98%	1%	99%	98%	100%	94%
SWL	0	100%	0%	100%	100%	100%	100%
EWL	0	100%	0%	100%	100%	100%	100%
UMW	1,62	61%	8%	59%	63%	87%	51%
TIP	0	0%	0%	0%	0%	0%	0%
SWC	0	0%	0%	0%	0%	0%	0%
ACC	0	100%	0%	100%	100%	100%	100%
ARW	0	100%	0%	100%	100%	100%	100%
ACU	0	100%	0%	100%	100%	100%	100%
AUS	0	100%	0%	100%	100%	100%	100%
ASC	0	100%	0%	100%	100%	100%	100%
AUD	0	100%	0%	100%	100%	100%	100%
AUV	0	100%	0%	100%	100%	100%	100%
ACN	1,7	26%	5%	25%	28%	44%	23%

Table 10: Ensembl’s and OpenCart’s statistical description of the rules’ SRAD distribution during evolution for tables.

wikimedia	Skewness	Median	STD	Mode	Average	Max	Min
UTC	0	100%	0%	100%	100%	100%	100%
UPL	-4,5	100%	0%	100%	100%	100%	99%
SWL	0	100%	0%	100%	100%	100%	100%
EWL	0	100%	0%	100%	100%	100%	100%
UMW	-0,95	51%	3%	53%	51%	55%	42%
USP	-0,12	15%	1%	16%	15%	17%	14%
CIS	0,33	78%	2%	78%	78%	83%	74%
ACC	0	100%	0%	100%	100%	100%	100%
ARW	0	100%	0%	100%	100%	100%	100%
ACU	0	100%	0%	100%	100%	100%	100%
AUS	0	100%	0%	100%	100%	100%	100%
ASC	0	100%	0%	100%	100%	100%	100%
AUD	0	100%	0%	100%	100%	100%	100%
AUV	1,77	47%	5%	48%	48%	64%	43%
AII	0	100%	0%	100%	100%	100%	100%
ACN	-7,15	100%	0%	100%	100%	100%	99%
NBP	-1,57	85%	7%	86%	83%	91%	61%

Typo3	Skewness	Median	STD	Mode	Average	Max	Min
UTC	-0,89	95%	1%	95%	94%	96%	90%
UPL	0	100%	0%	100%	100%	100%	100%
SWL	0	100%	0%	100%	100%	100%	100%
EWL	0	100%	0%	100%	100%	100%	100%
UMW	0,14	38%	2%	41%	38%	43%	35%
USP	0,26	30%	2%	30%	30%	34%	27%
CIS	0,96	37%	4%	36%	38%	47%	34%
ACC	-0,41	97%	1%	97%	96%	98%	94%
ARW	0	100%	0%	100%	100%	100%	100%
ACU	0	100%	0%	100%	100%	100%	100%
AUS	0	100%	0%	100%	100%	100%	100%
ASC	0	100%	0%	100%	100%	100%	100%
AUD	0	100%	0%	100%	100%	100%	100%
AUV	-0,38	60%	2%	61%	59%	61%	56%
AII	-1,49	100%	0%	100%	100%	100%	99%
ACN	0	100%	0%	100%	100%	100%	100%
NBP	0	100%	0%	100%	100%	100%	100%

Table 11: Wikimedia’s and Typo3’s statistical description of the rules’ SRAD distribution during evolution for columns.

4.3 What are the Evolution Patterns of Sql Style Rules?

According to the schema-level SQL style evolution pattern that we identified in our second research question, the adherence of a schema to SQL style rules does evolve overtime. Typically, this happens only for a few SQL style rules, the number of which ranges from zero to eight. In this section, we investigate the SQL style evolution in more detail. In particular, the issue is whether the adherence of a schema to SQL style rules, increase, decrease or remains stable overtime.

To address the aforementioned issue for a particular rule we rely on the following methodology:

- We introduce the Schema Rule Adherence Evolution Degree (SRAED) that measures the respective difference between the SRAD in the LKV of the schema and the SRAD in first known version (FKV) of the schema as follows: $SRAED = SRAD_{LKV} - SRAD_{FKV}$.
- Then, we consider the following characterizations:
    - The schema adherence to the rule is fixed if $0\% \leq SRAED \leq 1\%$.
    - The schema adherence to the rule is positive if $SRAED > 1\%$.
    - The schema adherence to the rule is negative if $SRAED < 0\%$.

Table 12 gives the probability of fixed, positive and negative for table rules, across the examined schemas. Similarly,
Table 13 gives the probability of fixed, positive and negative for column rules, across the examined schemas.

Table Rules	Positive	Fixed	Negative	AVG Change	STDev
UTC	0,00%	95,24%	4,76%	-14%	0%
UPL	0,00%	85,71%	14,29%	-11%	14%
SWL	0,00%	100,00%	0,00%	0%	0%
EWL	0,00%	100,00%	0,00%	0%	0%
UMW	33,33%	23,81%	42,86%	-10%	9%
TIP	9,52%	61,90%	28,57%	-16%	17%
SWC	4,76%	95,24%	0,00%	8%	9%
ACC	0,00%	100,00%	0,00%	0%	0%
ARW	0,00%	100,00%	0,00%	0%	0%
ACU	0,00%	100,00%	0,00%	0%	0%
AUS	0,00%	100,00%	0,00%	0%	0%
ASC	0,00%	100,00%	0,00%	0%	0%
AUD	0,00%	100,00%	0,00%	0%	0%
AUV	0,00%	100,00%	0,00%	0%	0%
ACN	23,81%	23,81%	52,38%	-10%	8%

Table 12: Probability of fixed, positive and negative for table rules, across the examined schemas for table rules. AVG (average) Change and STDev (standard deviation) describe the distribution of SRAD in schemata with the higher propability in between of being positive or negative.

Column Rules	Positive	Fixed	Negative	AVG Change	STDev
UTC	14,29%	85,71%	0,00%	3%	3%
UPL	0,00%	100,00%	0,00%	0%	0%
SWL	0,00%	100,00%	0,00%	0%	0%
EWL	0,00%	100,00%	0,00%	-3%	2%
UMW	23,81%	52,38%	23,81%	12%	8%
USP	47,62%	38,10%	14,29%	7%	10%
CIS	38,10%	28,57%	33,33%	2%	7%
ACC	9,52%	80,95%	9,52%	2%	2%
ARW	0,00%	100,00%	0,00%	0%	0%
ACU	0,00%	100,00%	0,00%	0%	0%
AUS	0,00%	100,00%	0,00%	0%	0%
ASC	0,00%	100,00%	0,00%	0%	0%
AUD	0,00%	100,00%	0,00%	0%	0%
AUV	14,29%	42,86%	42,86%	-5%	7%
AII	19,05%	76,19%	4,76%	7%	5%
DNC	0,00%	100,00%	0,00%	0%	0%
NBP	23,81%	71,43%	9,52%	11%	10%

Table 13: Probability of fixed, positive and negative for table rules, across the examined schemas for column rules. AVG (average) Change and STDev (standard deviation) describe the distribution of SRAD in schemata with the higher propability in between of being positive or negative.

Based on the results, for both table and column rules we consider the following *rule-level adherence evolution patterns*:

- Fixed adherence evolution pattern: A rule follows this pattern if most likely SRAED will be fixed (vs being positive or negative), throughout a schema evolution history.
- Positive adherence evolution pattern: A rule follows this pattern if most likely SRAED will be positive, throughout a schema evolution history.
- Negative adherence evolution (anti-)pattern: A rule follows this anti-pattern if most likely SRAED will be negative, throughout a schema evolution history.

Following, we discuss in more detail the case of each rule.

4.4 Table Rules Evolution

Instances of the fixed adherence evolution pattern: ARW, ACU, AUS, ASC, AUD, AUV, ACC, SWL, and EWL are all strong instances of the fixed adherence evolution pattern, their SRAED is zero and there are no fluctuations of SRAD during the schema evolution history.

SWC follows the fixed adherence evolution pattern. In particular, SRAED for SWC is fixed in all schemas except ATLAS, where its value is positive.

In UTC we observe only a minor tendency to negative adherence evolution, as in 20 schemas SRAED is fixed and in one schema it has a negative value with a decrease of 14%. This indicates that people are aware of the way they name their tables in a schema and they agree on the type of case they use. Interestingly enough as shown in RQ2 in Table 7, there are four schemata where *UTC* changes and only ATLAS, in the last version has changes from the first version. This indicates that developers try to correct the type of case with SQL specific refactorings [31].

In UPL, SRAED is negative in three of the examined schemas with an average SRAD reduction of 11% and a STD of 14%. Adherence to UPL seems to get worse as the evolution of the schema continues. A possible answer for this anti-pattern could be the co-occurrence with the ACN violation, since when table names are created through

concatenating the names of multiple tables, the name tends to lengthy. Another reason could be inexperienced developers who not follow the schema specific SQL style or just the use of plain bad names.

Regarding TIP, SRAED is positive in only two datasets, fixed in thirteen and negative in six. In the latter six schemas the average SRAD is 16%. Overall, TIP changes in thirteen datasets, but only eight have different SRAD between FKV and LKV. TIP has the highest reduction of SRAD, compared to the other table rules and is one of the less respected rules with an average SRAD of 17% of average SRAD across the datasets.

Instances of the positive adherence evolution pattern: Unfortunately, there is no table rule that follows the positive adherence evolution pattern.

Instances of the negative adherence evolution anti-pattern: UMW and ACN are instances of the negative adherence evolution anti-pattern.

In UMW we observe seven schemas with positive SRAED, five fixed and nine schemas with negative SRAED. The average SRAD reduction is about 10% with an STD of 9%. UMW is not only an instance of the negative adherence evolution anti-pattern, but also a common sin among developers since the average SRAD in the LKV is only 56%.

The most definite instance of the negative evolution anti-pattern is ACN, with negative SRAED in eleven schemas, positive SRAED in five schemas and fixed SRAED in five schemas. Concerning the negative SRAED the average reduction in SRAD is 10%. ACN changes in seventeen datasets, from those there is one returning to its initial SRAD, DekiWiki having max range in SRAD during its evolution 4%.

4.5 Column Rules Evolution

Instances of the fixed adherence evolution pattern: UPL, SWL, EWL, ARW, ACU, AUS, ASC, AUD and DNC are all strong instances of the fixed adherence evolution pattern. It should be noted that there are some insignificant SRAD changes in their evolution and limited in a few number of datasets (≤ 2).

Concerning UTC, SRAED is expected to be fixed, meaning the developers just keep on following the original type of case convention; with less probability it may be positive with small improvement (3% is the average), as we observe in three schemas where SRAD increases. UTC has SRAD fluctuations during the evolution of the schema in eight schemas, the average SRAD is high at 94% and the standard deviation is 14%. It is interesting to note that in five of those schemas the value of SRAD in LKV is equal to the value of SRAD in FKV.

For ACC, SRAED is fixed in most schemas. There are two exceptions with positive SRAED and two exceptions with negative SRAED. The value of SRAD fluctuates in eight schemas and camelCase is used as main type of case in SRM2 and was widely used in CASTOR2; as a reminder both of those schemata belong to the same project. AII SRAED is positive in six schemas, negative in one schema and fixed in sixteen schemas. Adherence improves by an average of 7%, indicating that developers tend to favor meaningful primary keys over the "id". It should be noted that the schemata in our case study followed to great extent the AII rule, the average SRAD in the LKV across the datasets is 98%. This might not be the case for schemata in general since the violation of this rule, not only from our experience[34], is a common practice for some developers.

For columns, UMW follows the fixed adherence evolution pattern, as the value of SRAED for this rule is fixed in eleven schemas.

NBP had changes during the evolution in ten datasets. Managed to improve from FKV to LKV by an average of 7% and became negative in one, the other five returned to their initial SRAD. Generally NBP was a widely adhered rule with an average of 93% in LKV and standard deviation of 16%.

Instances of the positive adherence evolution pattern: USP and CIS follow the positive adherence evolution pattern. Specifically, for USP the average SRAD improvement is 7% with a standard deviation of 10%. CIS is a *borderline* instance of the pattern, as

[34] https://softwareengineering.stackexchange.com/questions/114728/why-is-naming-a-tables-primary-key-column-id-considered-bad-practice

SRAED is positive in eight schemas, fixed in six schemas and negative in seven schemas and on top of that average SRAD improvement is 2%.

Instances of the negative adherence evolution anti-pattern: AUV is the only instance of the negative adherence evolution anti-pattern. AUV's majority consists of nine datasets with an average SRAD reduction of 5% and standard deviation of 7%. The number of datasets having SRAD fluctuation is nineteen from those seven manage to return to the initial SRAD.

4.6 Which are the Adherence/Violation Patterns of SQL Style Rules?

So far, our study revealed that the developers do care about rules and conventions in SQL programming. Moreover, our studied showed that typically the adherence to the rules does not evolve dramatically throughout the schema evolution history.

The evolution patterns that we discovered show that schemas for the most part, are bound to the style they were created with. Consequently, without loss of generality *in the remainder of our study we focus on the LKV* of the examined schemas. Our goal hereafter is to investigate the extent to which schemas adhere to SQL rules and conventions in the end of the known evolution. Is the reason behind the rigidity phenomenon adequate quality? Are the schemas clean enough to make changes in style, infrequent?

To address the aforementioned issues we characterize the adherence of a schema to a particular rule with respect to the following characterizations:

- Strong adherence: the adherence of the schema to the rule is strong if $SRAD \geq 75\%$.
- Weak adherence: the adherence of the schema to the rule is weak if $50\% \leq SRAD < 75\%$.

Similarly, we characterize the violation of a rule by a particular schema as follows:

- Strong violation: the rule is strongly violated by the schema if $SRAD < 25\%$.

- Strong violation: the rule is weakly violated by the schema if $25\% \leq SRAD < 50\%$.

Based on the examined schemas, for each rule we calculate the probability of the aforementioned characterizations. The results for table rules and column rules are given in Table 14 and Table 15 accordingly.

Table rules	Strong Adherence P(SRAD ≥ 75%)	Weak Adherence P(50% ≤ SRAD < 75%)	Strong Violation P(SRAD < 25%)	Weak Violation P(25% ≤ SRAD < 50%)
UTC	100,00%	0,00%	0,00%	0,00%
UPL	95,24%	4,76%	0,00%	0,00%
SWL	95,24%	0,00%	4,76%	0,00%
EWL	100,00%	0,00%	0,00%	0,00%
UMW	23,81%	42,86%	9,52%	23,81%
TIP	0,00%	4,76%	66,67%	28,57%
SWC	14,29%	0,00%	85,71%	0,00%
ACC	100,00%	0,00%	0,00%	0,00%
ARW	100,00%	0,00%	0,00%	0,00%
ACU	95,24%	0,00%	4,76%	0,00%
AUS	100,00%	0,00%	0,00%	0,00%
ASC	100,00%	0,00%	0,00%	0,00%
AUD	100,00%	0,00%	0,00%	0,00%
AUV	100,00%	0,00%	0,00%	0,00%
ACN	38,10%	47,62%	4,76%	9,52%

Table 14: Probability of strong/weak adherence/violation for table rules

According to the rule adherence/violation probabilities, we further introduce the following *rule-level adherence/violation patterns*:

- Strong adherence pattern: a rule follows this pattern if most likely the adherence of a schema to the rule will be strong.
- Weak adherence pattern: a rule follows this pattern if most likely the adherence of a schema to the rule will be weak.
- Strong violation (anti-)pattern: a rule follows this anti-pattern if most likely a schema strongly violates the rule.

- Weak violation (anti-)pattern: a rule follows this anti-pattern if most likely a schema weakly violates the rule.

Column rules	*Strong Adherence P(SRAD ≥ 75%)*	Weak Adherence P(50% ≤ SRAD < 75%)	Strong Violation P(SRAD < 25%)	Weak Violation P(25% ≤ SRAD < 50%)
UTC	85,71%	14,29%	0,00%	0,00%
UPL	100,00%	0,00%	0,00%	0,00%
SWL	100,00%	0,00%	0,00%	0,00%
EWL	100,00%	0,00%	0,00%	0,00%
UMW	28,57%	38,10%	9,52%	23,81%
USP	0,00%	4,76%	38,10%	57,14%
CIS	47,62%	19,05%	4,76%	28,57%
ACC	85,71%	4,76%	0,00%	9,52%
ARW	100,00%	0,00%	0,00%	0,00%
ACU	100,00%	0,00%	0,00%	0,00%
AUS	100,00%	0,00%	0,00%	0,00%
ASC	100,00%	0,00%	0,00%	0,00%
AUD	100,00%	0,00%	0,00%	0,00%
AUV	9,52%	33,33%	0,00%	57,14%
AII	100,00%	0,00%	0,00%	0,00%
ACN	100,00%	0,00%	0,00%	0,00%
NBP	95,24%	0,00%	0,00%	4,76%

Table 15: Probability of strong/weak adherence/violation for column rules.

The characterization of table and column rules with respect to the (anti-)patterns that they follow is also given in Table 15 and Table 16, respectively.

In Table 14, we observe that eleven table rules follow the strong adherence pattern, with the respective adherence probabilities varying from 95.24% to 100%. Two tables rules, namely UMW and ACN, adhere to the weak adherence pattern, with adherence probabilities 42.86% and 47.62%, respectively. Finally, two table rules, namely TIP and SWC, conform with the strong violation anti-pattern, with violation probabilities 66.67% and 85.71%, respectively. Table 15 we observe fourteen column rules that follow the strong adherence pattern, with the respective adherence probabilities ranging

from 47.62% to 100%. Taking a closer look, CIS is the Achilles' heel in this set of rule since there is an important minority of six schemas that weakly violate the rule it to a certain degree (SRAD < 50%), and one schema that strongly violates the rule (SRAD < 25%)

UMW is the only rule that follows the weak adherence pattern with probability 38.10%. If we view UMW together with CIS then we have a worrying result both rules are very important in creating meaningful names. Finally, two column rules, namely USP and SUV, conform with the weak violation anti-pattern, with an equal violation probabilities (57.14%).

4.7 Which SQL Style(s) is(are) Actually Followed in Practice?

Getting back to the starting point of our study, we introduced a list of SQL style rules which can be considered as a *SQL style* that should be followed. However, in practice we see that the SQL style rules are not followed to the same extent by the developers. Taking a step further, our next goal is to introduce the style that is actually followed by the developers, which we call the *weighted SQL style*, and evaluate the examined schemas, with respect to the weighted style and the ideal style.

To begin, we introduce a weighted formula that we call the *Rule Adherence Degree (RAD)*, which allows us to rank the rules of the rule based style based on the extent to which they are followed in practice. In particular, the value of RAD for a rule is calculated with respect to the probabilities of strong/weak adherence/violation to/of the rule as follows:

$$RAD = \begin{pmatrix} w_{strong}^{adherence} * P(\text{SRAD} \geq 75\%) + \\ w_{weak}^{adherence} * P(50\% \leq \text{SRAD} < 75\%) - \\ w_{strong}^{violation} * P(\text{SRAD} < 25\%) - \\ w_{weak}^{violation} * P(25\% \leq \text{SRAD} < 50\%) \end{pmatrix}$$

As a generalization of our ranking approach, in the above formula we weight the strong/weak adherence/violation probabilities with respective weights, which can vary

from 0 to 1. However, by default in our study we consider the following default values: $w_{strong}^{adherence} = 1$, $w_{weak}^{adherence} = 0.5$, $w_{strong}^{violation} = 1$, $w_{weak}^{violation} = 0.5$.

Table rule	RAD	Column rule	RAD
UTC	1	UPL	1
UPL	1	SWL	1
EWL	1	EWL	1
ACC	1	ARW	1
ARW	1	ACU	1
AUS	1	AUS	1
ASC	1	ASC	1
AUD	1	AUD	1
AUV	1	AII	1
SWL	0,9	ACN	1
ACU	0,9	NBP	1
ACN	0,7	UTC	0,9
UMW	0,3	ACC	0,9
SWC	-0,7	CIS	0,4
TIP	-0,9	UMW	0,2
		AUV	0,1
		USP	-0,4

Table 16: Weighted SQL style - table and column rules ranked with respect to RAD.

Table 16 provides the weighted SQL style, i.e., the list of the SQL style rules, ranked with respect to RAD. The ranking of table rules is given on the left part of the table, while the ranking of the column rules is provided in the left part of the table.

The ranking for the rules is in alignment with the adherence patterns and the violation anti-patterns that we defined. Rules that follow the strong adherence/violation patterns have weights equal to 1, while rules that conform to the weak adherence/violation patterns have smaller weights. Rules that follow the strong/weak adherence patterns have positive weights, while rules that conform to the strong/weak violation patterns have negative weights.

To move on, we assess the examined schemas with respect to their distance from the weighted and the rule based style. The distance reveals if schemas are have common styles, in the case the inbetween distance is small. The distance also shows how far or close the schemas are to the rule based style and how these distances compare to the respective distances of the weighted style.

To this end, for the each style we consider:

- A vector that consists of the values of RAD for table rules.
- A vector that comprises the values of RAD for column rules.

For the weighted style the values of the vectors are given in Table 16, while for the ideal style all values are equal to 1. Similarly, we calculate respective vectors for each one of the examined schema and measure the Euclidean distance between the schema vectors and the style vectors. The resulted distances, ranked in an increasing order, for the weighted and the rule based style are given in Table 17 and Table 18, respectively.

A basic observation in the results is that the distance between the examined schemas and the weighted style is relatively small. Concerning the tables rules the distance ranges from 0.01 to 1.83, while for column rules it is even smaller varying from 0.01 to 0.80. On the contrary, the distance between the examined schemas and the rule based style is quite larger. Regarding the tables rules the distance ranges from 3.52 to 5.85, while for column rules it is even smaller varying from 3.89 to 4.80.

Table 19 gives indicative examples of table names taken from SlashCode (i.e., the schema whose style is closer to the weighted style), phpwiki (i.e., the schema whose style is closer to the rule based style), and Joomla (i.e., the schema whose style is farther from both the weighted and the rule based style). We can easily observe that the table names in phpwiki are almost perfect. On the other hand, the table names in SlashCode are poor due to the use of several acronyms (violation of UMW), and the concatenation of table names (violation of ACN). The table names in Joomla are also poor with several issues, as they start with special characters (violation of SWL), the different terms are not separated in some way (violation of UMW), and so on.

Moreover, Table 20 gives indicative examples of column names taken from Ensembl (i.e., the schema whose style is closer to the weighted style), OpenCart (i.e., the schema whose style is closer to the rule based style), and Castor2 (i.e., the schema whose style is farther from both the weighted and the rule based style). Again we observe that the column names in OpenCart are quite simple and easy to read. On the other hand, the column names in Ensemble are not so clear, as they include several acronyms (violation of UMW). Finally, the column names in Castor are also poor with several problems like not being in singular, using acronyms, CamelCase, and so on.

Schema	Distance between the table rules vectors		Schema	Distance between the column rules vectors
SlashCode	0.01		Ensembl	0.01
ATLAS	0.04		wikimedia	0.02
Medbiosql	0.05		Coppermine	0.03
TikiWiki	0.05		EGEE	0.06
OpenCart	0.06		Joomla 1.5	0.07
XOOPS	0.10		phpBB	0.07
Typo3	0.11		Medbiosql	0.07
phpBB	0.11		DekiWiki	0.09
Ensembl	0.13		SlashCode	0.11
DekiWiki	0.14		OpenCart	0.11
DQ2MySQL	0.17		Zabbix Oracle	0.14
Zabbixc	0.24		Atlas	0.15
wikimedia	0.27		Typo3	0.17
SRM2	0.28		XOOPS	0.21
e107	0.28		NucleusCMS	0.25
NucleusCMS	0.37		phpwiki	0.27
CASTOR2	0.41		DQ2MySQL	0.36
EGEEMySQL	0.43		TikiWiki	0.41
phpwiki	0.50		e107	0.61
Coppermine	0.65		SRM2	0.75
Joomla 1.5	1.83		CASTOR2	0.80

Table 17: Euclidean distance between the styles of the examined schemas and the weighted style.

Schema	Distance between the table rules vectors		Schema	Distance between the column rules vectors
phpwiki	3.52		OpenCart	3.89
EGEEMySQL	3.59		phpBB	3.93
NucleusCMS	3.66		EGEEMySQL	3.94
e107	3.75		Coppermine	3.97
wikimedia	3.76		Ensembl	4.01
DQ2MySQL	3.86		wikimedia	4.02
DekiWiki	3.89		Joomla 1.5	4.07
Ensembl	3.90		Medbiosql	4.07
Typo3	3.92		DekiWiki	4.09
XOOPS	3.92		SlashCode	4.11
OpenCart	3.97		Zabbix Oracle	4.14
TikiWiki	3.97		Atlas	4.15
Medbiosql	3.98		Typo3	4.17
ATLAS	3.99		XOOPS	4.21
SlashCode	4.01		NucleusCMS	4.25
phpBB	4.13		phpwiki	4.27
Zabbix Oracle	4.26		DQ2MySQL	4.36
SRM2	4.30		TikiWiki	4.41
CASTOR2	4.43		e107	4.61
Coppermine	4.68		SRM2	4.75
Joomla 1.5	5.85		CASTOR2	4.80

Table 18: Euclidean distance between the styles of the examined schemas and the rule based style.

SlashCode	Phpwiki	Joomla 1.5
accesslog_admin	link	#__banner
accesslog_artcom	nonempty	#__bannerclient
al2_log	link	#__bannertrack
al2_log_comments	session	#__categories
al2_types	recent	#__components

Table 19: Examples of table names taken from SlashCode (i.e., the schema whose style is closer to the weighted style), phpwiki (i.e., the schema whose style is closer to the rule based style), and Joomla (i.e., the schema whose style is farther from both the weighted and the rule based
style).

Ensembl	OpenCart	CASTOR2
asm_seq_region_id	address_id	Flags
seq_region_id	customer_id	userName
seq_region_start	firstname	Euid
seq_region_end	email	Egid
exc_seq_region_id	telephone	Mask
exc_seq_region_start	password	Pid
exc_seq_region_end	salt	Machine
attrib_type_id	website	creationTime

Table 20: Examples of column names taken from Ensembl (i.e., the schema whose style is closer to the weighted style), OpenCart (i.e., the schema whose style is closer to the rule based style), and Castor2 (i.e., the schema whose style is farther from both the weighted and the rule based style)

4.8 Threats to Validity

Construct Validity

Construct validity concerns the appropriateness of observations made on the basis of measurements, taken during the case study. Concerning our SQL style checking tool, we used unit tests for all the SQL style rules to rule out deficiencies in the implementation. Additionally, we manually tested the correctness of our tool via an artificial evolution assessment scenarios. In particular, we took samples of few schemata and created a history. Then we checked manually the validity of the results provided by DBsea. The most complicated rules, i.e., the ones belonging in the lexicological category, are based on WordNet, a state of art thesaurus and they were implemented with the use of libraries developed by well-respected institutions. Statistical measurements where made with Apache Commons Math the biggest open-source library of mathematical functions and utilities for Java. To process the DML files we use a well-known parser ANTLR, widely used in both academia and industry[35] that ensures us for the correctness of DBsea's input.

Internal Validity

Internal validity is the extent to which a causal conclusion based on a study is warranted, which is determined by the degree to which a study minimizes systematic error, a tendency of supporting particular outcomes. The results of our thesis are based on observations made in the majorities of the schemata. The schemata per se did not have significant changes, from one revision to a next revision. Major changes could indicate the existence of an abnormal event in the history of a schema, like a total restructuring of the database or any other event, that could lead us to wrong assumptions. Throughout this thesis the only questionable results concern the weak/medium correlations of style changes with the size of a schema and the possibility of misinterpretation for some

[35] http://www.antlr.org/testimonials.html

rules' SRAD values or fluctuations by the reader, countermeasures were taken in both cases in the form of clearly stated comments upon the results.

External validity

Our study has been conducted in a well-defined context, FoSS databases. We used a respected number of databases with variance in the respective fields of use. Those databases had also variety concerning the size of their schema and the extent of their history. The number of revisions for the databases ranges from 4 to 528 while the number of tables from 9 to 215. Thus, we believe that those schemata are representative for the case of open source projects. In the case of industry related databases we would generalize the conclusions of this thesis with precaution since usually (and hopefully) industry projects have stricter demands in quality. We would also advise against the generalization of our results to the SQL style of queries, as they were not in the scope of our study.

CHAPTER 5

CONCLUSIONS: SCHEMAS AND ELEGANCE

5.1 An Interesting Future

In this chapter, we discuss fundamental observations, conjectures and patterns that have been detected in our study. As a reminder, the context of our study (i.e FoSS projects) sets some limitations to the generalization of our results to closed projects. Having said that we firmly believe that our conclusions hold strong in the described context and to make our thesis more precise and clear we distinguish between (a) the most important results and, (b) further results. We firstly provide the reader, a recap.

We introduced an rule based style consisting of rules and conventions based on a literature review and created a tool that enables developers to evaluate their schemata against the afformentioned style. We performed a large scale empirical study involving 21 well-known schemata from open source projects that vary in their respective fields of use. We checked the adherence of those schemata to the rule based style and assessed the evolution of the schemas with respect to their adherence to the rules. Through this assessment we found certain evolution patterns, discussed in the remainder. We performed a detailed analysis of each rule and identified respective adherence and violation patterns. We continued by ranking the rules based on the extent to which they are adopted by the schemata of the study. We used the rankings to derive a weighted style that reflects the developers' perception on the applicability of the rules in practice. Finally we compared the distance of the examined schemas' distance from the rule based and the weighted style.

Most important results

We identified several interesting patterns summarized below:

Schema-level adherence pattern

Tables

The percentage of rules that hold at least for some tables is high, ranging from 80% to 100% while rules that are not followed by some tables is medium, varying from 13% to 40%.The percentage of rules that hold at least for some tables is always higher than the percentage of rules that are not followed by some tables. The percentage of rules that hold for all tables is medium, high, varying from 60% to 87%, while the percentage of rules that do not hold for any table is low, ranging from 0% to 20%. Lastly, the percentage of rules that hold for all tables is always higher than the percentage of rules that do not hold for any table.

Columns

All rules hold at least for some columns in all of the schemata. The percentage of rules that hold for all columns is medium high, varying from 59% to 76%. Finally, the percentage of rules that do not hold for some columns is low medium, ranging from 24% to 41%.

Schema-level adherence evolution pattern

Table and column rules do evolve during the life of a schema, however, only a few rules change. The number of rules that change in the examined schemas varies from zero to seven for tables and zero to eight for columns.
Typically, the rules that change the most are lexicological, methodological, or writing style rules. The magnitude of *SRAD* fluctuations depends on the schema, the rules, and the schema elements involved.

If a rule changes significantly (>10%), the biggest portion of rule's SRAD fluctuation will occur in a minor fraction of the schema's history.

Rule-level adherence evolution patterns

We found that most rules follow the fixed adherence evolution pattern, i.e., the schemas' adherence to the rules does not increase or decrease overtime; in other words, for the most part a schema's style will be as good as it was in the birth of the schema. Few other rules follow the positive (resp. negative) evolution pattern, i.e., .e., the schemas' adherence to the rules increases (resp. decreases) overtime.

Rule-level adherence/violation patterns

Most rules follow the strong adherence pattern, i.e., more than 75% of the schema elements adhere to the rule. Moreover, few rules follow the weak adherence pattern, i.e. the percentage of schema elements that follow the rule varies in [50%, 75%). Also, few rules follow the strong violation pattern, i.e. less than 25% of the schema elements adhere to the rule. Finally, few some rules follow the weak violation pattern, i.e., the percentage of schema elements that follow the rule varies in [25%, 50%).

Further outcomes

We looked at the Kendall correlation of a rule's SRAD with the schema's size (i.e. number of tables or columns) across the schema's evolution. For table rules, we did not find any positive correlation. On the contrary, we found a strong negative correlation for two table rules, namely TIP and ACN. For certain column rules we observed strong positive correlations, especially for lexicological, methodological, and writing style rules. CIS and AUV were found to have strong negative correlation.

We specifically focused on the UTC's SRAD fluctuations during the evolution and found that tables do not change their type of case as much as columns do. Interestingly schemata having more than one type of case, in their birth or early life ,by the last known version, typically the use of multiple type of cases was reduced.

5.1 An Interesting Future

Our work takes a first step towards assessing the importance of good practices in SQL programming. Nevertheless, there is still room for further research to this direction. In particular, Sharma et al. [17] gathered a large number of different schemata. We would like to extract from their massive datasets the SQL code that refers to the definition of schemata and repeat some key experiments of this work. One could also investigate the possible existence of correlations between database smells [17] and SQL style rule violations, to assess whether bad SQL schemas come with a bad style. If bad schemas do come with bad style, then it would be interesting to enhance our tool to enable schema quality assessment by taking into account both of these aspects. If both aspects were taken into consideration and correlation between db smells and SQL style existed; if a large dataset of schemata was publically available with the information about the date of birth, date of death, number of revisions in-between and the revisions per se, one could create a tool able to predict a schema's life expectancy. This tool would have a classifier trained with the aforementioned dataset and would be able to decide, if given as input a db smell assessment and SQL style assessment, the fate for the schema or even the whole project, failure or continuous evolution.

Another open issue is to investigate if births and deaths of schema elements impact style and vice versa. This objective could be accomplished by using Hecate and DBsea jointly, and search for correlations between schema adherence to rules and table/column insertions/deletions.

BIBLIOGRAPHY

[1] RD Banker, SM Datar, CF Kemerer, D Zweig, "Software Complexity and Software Maintenance Costs," *ACM,* pp. 81-94, 1993.

[2] M. J.-G. Juan Carlos Granja-Alvarez, "Method for Estimating MaintenanceCost in a Software Project: A CaseStudy," *Journal of software maintenance,* p. 161–175, 1998.

[3] D. Sjøberg, "Quantifying schema evolution," *Information and Software Technology,* pp. 35-44, 1993.

[4] LA Belady, MM Lehman , "A model of large program development," *IBM Syst.J.15,* p. 225–252, 1976.

[5] J. F. Roddick, "SQL/SE - A Query Language Extension for Databases Supporting Schema Evolution," *SIGMOD Record,* 1992.

[6] GT Nguyen, D Rieu, "Schema evolution in object-oriented datebase systems," *Data and Knowledge Engineering,* pp. 43-67, 1989.

[7] J Banerjee, W Kim, HJ Kim, HF Korth, "Semantics and Implementation of Schema Evolution in Object-oriented Databases," *SIGMOD,* pp. 311-322, 1987.

[8] CA Curino, HJ Moon, MW Ham, C Zaniolo , "The PRISM Workwench: Database Schema Evolution without Tears," *IEEE,* pp. 1523-1526, 2009.

[9] MM Lehman, JF Ramil , "Rules and Tools for Software Evolution Planning and Management, Software Evolution and Feedback: Theory and Practice," *Annals of Software Engineering,* pp. 15-44 , 2006.

[10] MM Lehman, JF Ramil, PD Wernick, DE Perry , "Metrics and laws of software evolution - the nineties view," *Proceedings of the 4th IEEE International Software Metrics Symposium,* pp. 20-34, 1997.

[11] P Vassiliadis, AV Zarras, I Skoulis , "Gravitating to Rigidity: Patterns of Schema Evolution -and its Absence- in the Lives of Tables," *Information Systems,* pp. 24-46, 2017.

[12] P Vassiliadis, AV Zarras, I Skoulis , "“Growing up with stability: How open-source relational databases evolve”, Information Systems," *Information Systems,* pp. 363-385, 2015.

[13] P Vassiliadis, AV Zarras, "Schema Evolution Survival Guide for Tables: Avoid Rigid Childhood and You 're En Route to a Quiet Life," *Journal of Data Semantics (JODS),* pp. 221-241, 2017.

[14] A Deutsch, V Tannen , "Mars: A system for publishing XML from mixed and redundant storage," *VLDB,* pp. 201-212, 2003.

[15] R. Fagin, "Inverting schema mappings," *ACM Transactions on Database Systems* , p. Article 25, 2007.

[16] PA Bernstein, TJ Green, S Melnik, A Nash , "Implementing mapping composition" *VLDB,* pp. 333-353, 2008.

[17] T Sharma, M Fragkoulis, S Rizou, M Bruntink, "Smelly Relations: Measuring and Understanding Database," *ICSE,* p. Article 4, 2018.

[18] J. Celko, SQL Programming Style.

[19] R. C. Martin, Clean Code: A Handbook of Agile Software Craftsmanship, 2008.

[20] C Gould, Z Su, P Devanbu , "JDBC Checker: A Static Analysis Tool for SQL/JDBC Applications," *ICSE,* p. 697–698, 2004.

[21] D. J. L. D. Binkley, "The Impact of Vocabulary Normalization," *Journal of Software: Evolution and Process,* p. 255–273, 2015.

[22] D Binkley, M Davis, D Lawrie, JI Maletic, "The Impact of Identifier Style on Effort and Comprehension," *Empirical,* p. 219–276, 2013.

[23] M Smit, B Gergel, HJ Hoover , "Code Convention Adherence in Evolving Software," *ICSME,* p. 504–507, 2011.

[24] S Butler, M Wermelinger, Y Yu, H Sharp , "Relating Identifier Naming Flaws and Code Quality: An Empirical Study," *Working Conference on Reverse Engineering,* 2009.

[25] S Butler, M Wermelinger, Y Yu , "Investigating Naming Convention," *ICSME,* p. 41–50, 2015.

[26] RPL Buse, WR Weimer , "Learning a Metric for Code Readability," *IEEE,* p. 546–558, 2010.

[27] D Lawrie, C Morrell, H Feild, D Binkley , "What’s in a Name? A Study of Identifiers," *ICPC,* p. 3–12, 2006.

[28] C. B. K. B. A. Capiluppi, "Quality Factors and Coding Standards - a Comparison Between Open Source Forges," *Electronic Notes in Theoretical Computer Science,* pp. 89-103, 2009.

[29] S. Holywell. [Online]. Available: http://www.sqlstyle.guide/.

[30] A. Microsoft, "https://docs.microsoft.com/en-us/sql/connect/ado-net/microsoft-ado-net-for-sql-server?view=sql-server-2017," [Online]. Available: https://docs.microsoft.com/en-us/sql/connect/ado-net/microsoft-ado-net-for-sql-server?view=sql-server-2017.

[31] P. J. S. Scott J Ambler, "Refactoring Databases: Evolutionary Database Design".

[32] N. P. M. Simon, "SQL Code Complexity Analysis," *ICAI2010*, pp. 353-359, 2010.

[33] X Fu, X Lu, B Peltsverger, S Chen, K Qian "A static analysis framework for detecting SQL injection vulnerabilities," *COMPSAC*, pp. 87-96, 2007.

[34] H van den Brink, R van der Leek, J Visser, "Quality Assessment for Embedded SQL", 7th IEEE International Working Conference on Source Code Analysis and Manipulation," *SCAM*, pp. 163-170, 2007.

[35] A Nash, PA Bernstein, S Melnik, "Composition of mappings given by embedded dependencies," *PODS*, pp. 172-183, 2005.

[36] PA Bernstein, TJ Green, S Melnik, A Nash, "Implementing mapping composition," *VLDB*, p. 333–353, 2008.

[37] D. Sjøberg, "The Thesaurus - A Tool for Meta Data Management," *Technical Report FIDE/91/6*, p. Project Number 3070, 1991.

[38] JF Roddick, SIGMOD, "Grammatical database model," *Information Systems*, pp. 257-267, 1979.

[39] A Yamashita, L Moonen, "Do developers care about code smells? An exploratory survey," *20th Working Conference on Reverse Engineering*, 2013.

[40] C Boogerd, L Moonen, "Assessing the Value of Coding Standards: An Empirical Study," *ICSME*, p. 277–286, 2008.

[41] D Binkley, D Lawrie, "The Impact of Vocabulary Normalization," *Journal of Software: Evolution and Process*, p. 255–273, 2015.

[42] I Skoulis, P Vassiliadis, A Zarras, "How is Life for a Table in an Evolving Relational Schema? Birth, Death and Everything in Between," *Conceptual Modeling Lecture Notes in Computer Science*, pp. 453-466, 2015.

[43] I Skoulis, P Vassiliadis, A Zarras, "Open-Source Databases: Within, Outside, or Beyond Lehman's Laws of Software Evolution?," *CAiSE*, 2014.

Appendix A

FURTHER STATISTICS

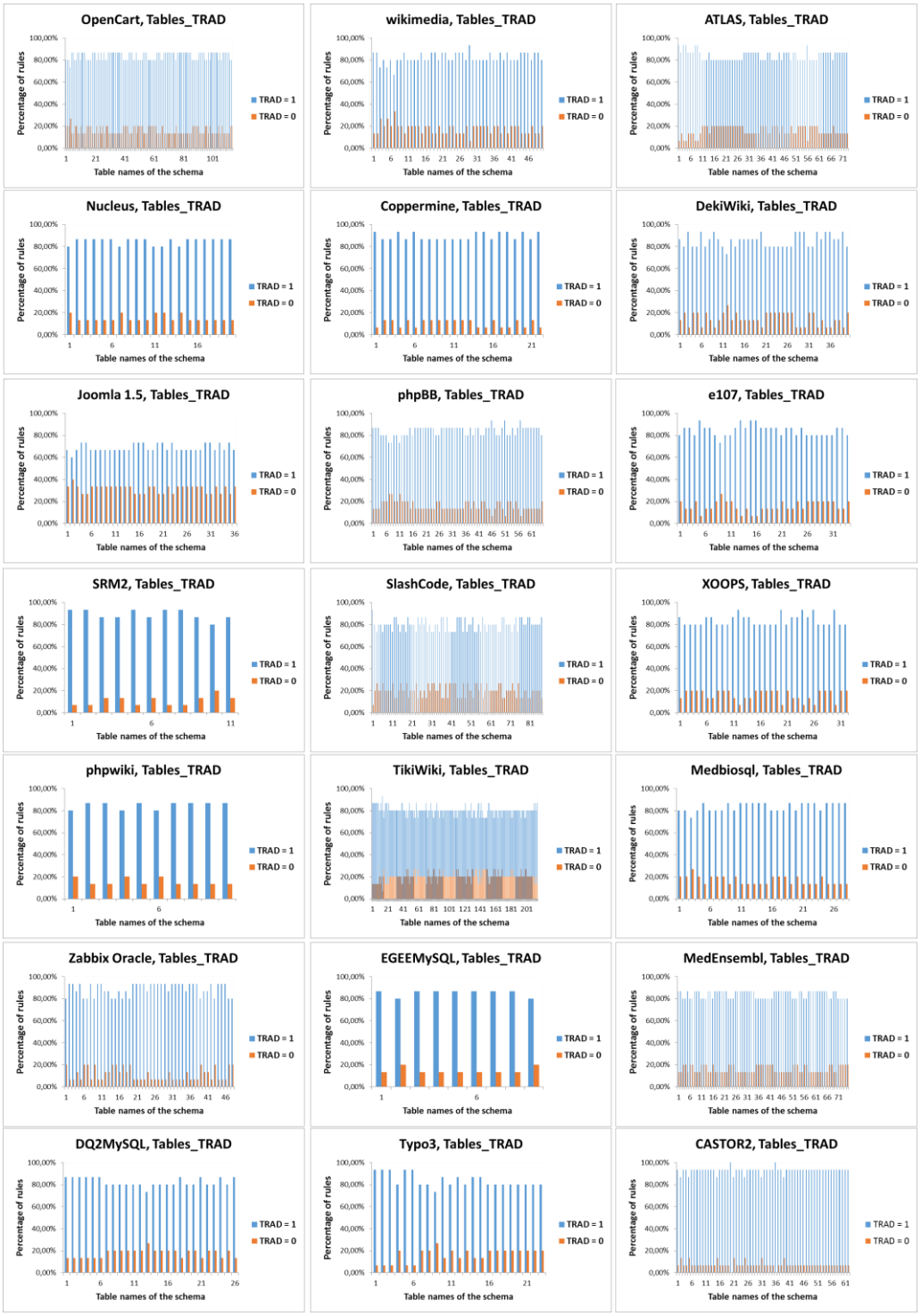


Figure 9: Schemata tables TRAD.

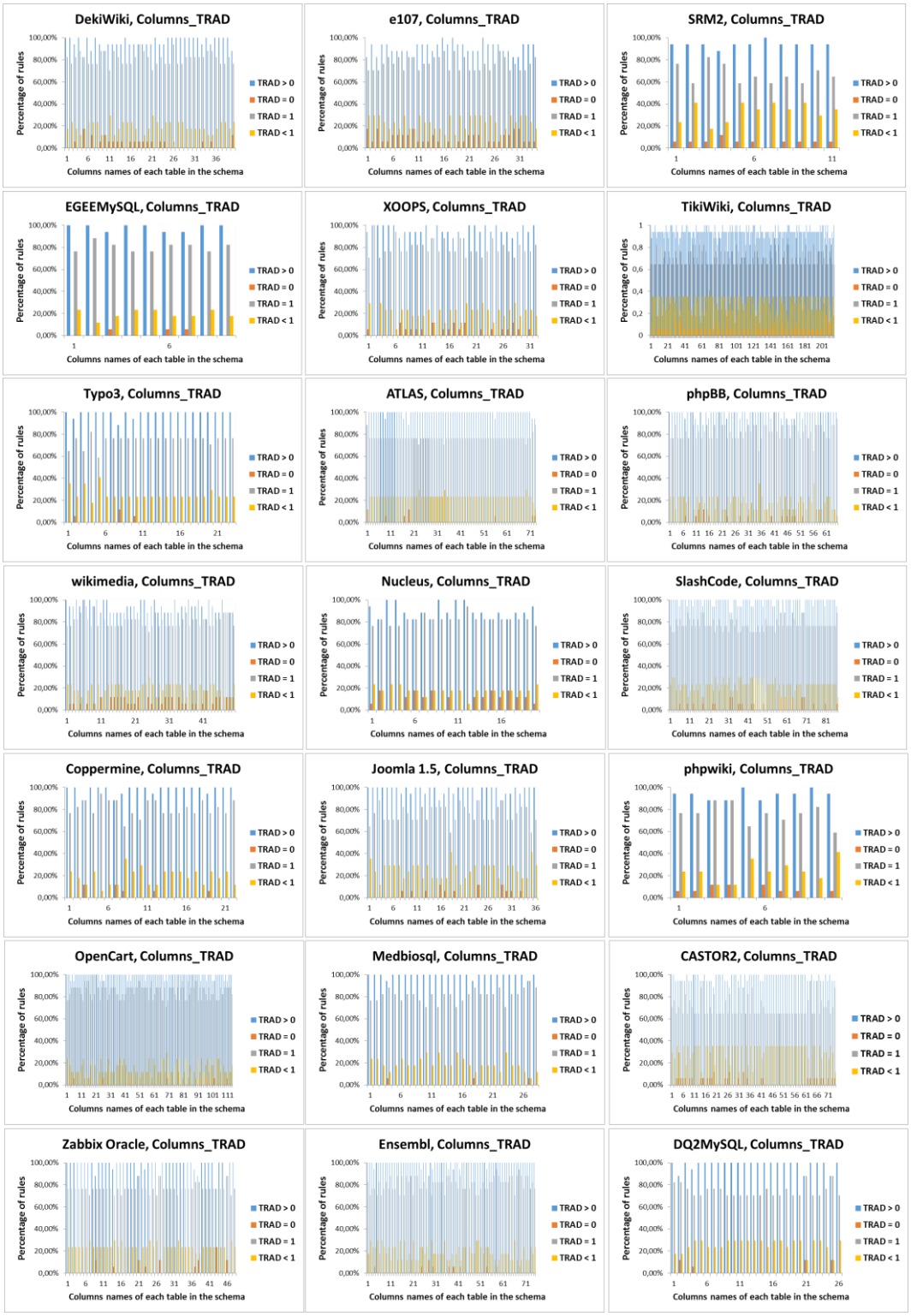


Figure 10: Schemata column TRAD.

APPENDIX B

TOOL RELATED INFORMATION

Dbsea in a Nutshell

DBSEA is based upon three main functionalities.

First of all the SQL parsing. Borrowed by HECATE the parser takes as input Data Definition Language (DDL) files. Data Definition Language has similar syntax to a computer programming language and is used for defining data structures, such as database schemata. Most common Data Definition statements in SQL are the CREATE TABLE, ALTER and DROP. As the statements are identified they are classified in the corresponding objects, the same objects HECATE use. For more information how HECATE saves the information into the memory please refer to the Diploma thesis of Ioannis Skoulis.

Second main functionality is the checking for rule adherence for columns and tables. The rule adherence is measured in two dimensions. The first dimension is the schema dimension, which means that for every schema we measure for each rule its use in percentage based on the objects we are assessing, tables or columns. The second dimension does a more fine grained analysis, checks the adherence of every rule for each table of a schema. The are two outcomes from this mode, the rule compliance for every table in a schema a version and the columns compliance with rules for each table inside the schema. The last main functionality is the exportation of the mined information.

For each dimension are created four files, two files containing statistical properties about the rule adherence across the evolution for the sql elements and two containing

the rule adherence across the evolution of the sql elements per se. The files contain as prefix of their name the schema's name they refer to.

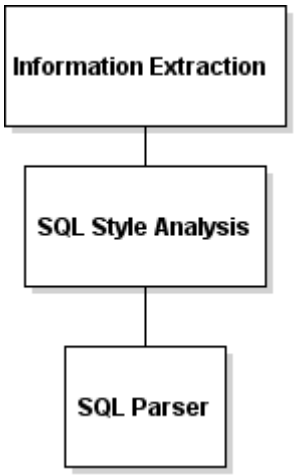


Figure 11 : DBsea in a nutshell, the main three functionalities.

Architecture

DBsea is consists of ten packages as seen in Figure 12. Below is given a brief explanation for each package.

- The package dbsea. Here lies the main class of the program responsible for starting up the GUI.
- The package dbsea.gui.swing contains all the classes that are responsible for User Interface.
- The package dbsea.stylecore is responsible of combining the various modules dbsea has, to execute the analysis flow. This package has the functionality to traverse the folders to find schemata, check the style for the versions of a schema and export the retrieved information.
- The package dbsea.parser. The SQL parser of the Tool.
- The package dbsea.tablestylecheck. As the name implies this package includes all the style checks for tables.
- The package dbsea.columnsstylecheck similarly to the tablestylecheck implements the necessary checks for the columns of a table.

- The package dbsea.sql has the objects that will represent SQL entities in the memory.
- The package dbsea.generalchecks contains style checks common between columns and tables. Both dbsea.columnsstylecheck and dbsea.tablestylecheck depend on dbsea.generalchecks.
- The package dbsea.wordnetchecks is where the natural language processing functionality of dbsea lies.
- The package dbsea.statistics, keeps track of the retrieved information. It also has the functionality of exportation of statistics to csv files.

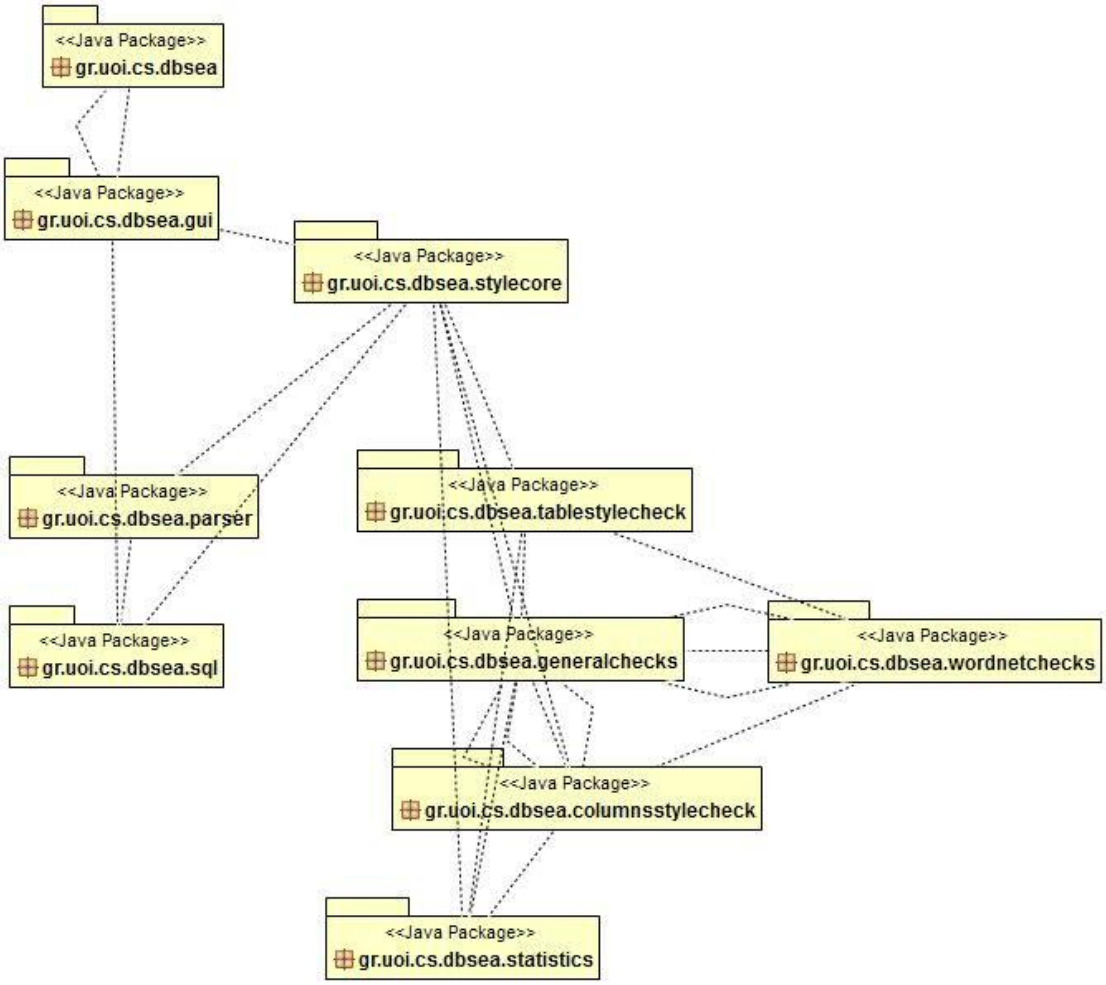


Figure 12 Package Diagram for Schemata Style Extraction Tool

We break down the afformentioned packages to give more details about their class and existing dependencies.

Package dbsea

Contains the main static class of dbsea and fires the user interface.

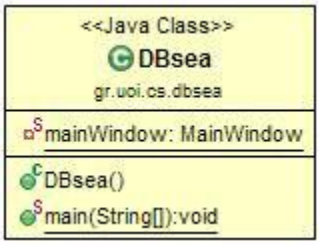


Figure 13 : dbsea contains the main static class and starts the User Interface.

Package dbsea.stylecore

This package contains one class which implements two functionalities of great value.

The SchemaStyleAnalysis class is responsible for the implementation of the main flow, the algorithm which dictates the way the crawling between folders is done, the execution of checks and the extraction of the gathered information to csv files.

As stated above, the two dimensions of check analysis are the one of the Schema and the one of the Table. The multiple dimensionality approach is implemented in the methodscheckSchemaHistoryStyleByRuleAndExport and checkSchemaHistoryStyle-ByTableAndExport. The schema flow is described in Algorithm 1 and the table's flow is implemented in a similar manner.

In the method traversePaths takes place the file crawler that searches for folders with the name "schemata". traversePaths searches the folders starting from the parent folder, the one selected in the GUI in a Depth First fashion.

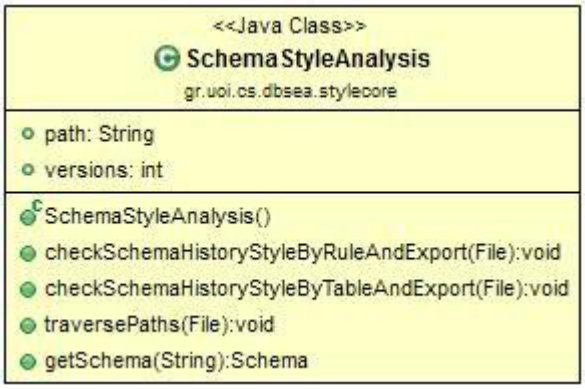


Figure 14 : This class implements the flow for style analysis.

Algorithm 1 Schema Dimension Algorithm

Input Versions of a Schema (as *Schemata*)
Output CSV files for tables, columns

```
1:  procedure CHECKSCHEMAHISTORYSTYLEBYRULEANDEXPORT
2:      Set up for style analysis
3:      for each schema version i in Schemata do
4:          for each table j in schema do
5:              for each colun k in table do
6:                  Run column checks
7:              Run table checks
8:          Run Table check for name Concatenation
9:          Write current statistics to file
10:     Clear statistics
```

Algorithm 1 : Style Extraction Algorithm from a schema's point of view.

Package dbsea.tablestylecheck

This package contains TableCheck the class implementing the rules about the tables of a Schema.

TableCheck contains 6 rules in total.

1. Name contains plural

2. Name starts with capital
3. Contains Verb
4. Name is concatenated in another table's name
5. Name contains Only singular
6. Name contains Prefix (not used in the analysis)

TableCheck is also responsible of updating the respective metrics for the above rules. This class is also responsible for the manipulation of the files where the statistics will be held. This is accomplished through aggregation with *TableStastistics* in *dbsea.statistics* package. TableCheck implements the method *runchecks* which is where the table checks are executed, it is a single point of maintenance if more checks are realized and are meant to be used for the style extraction analysis.

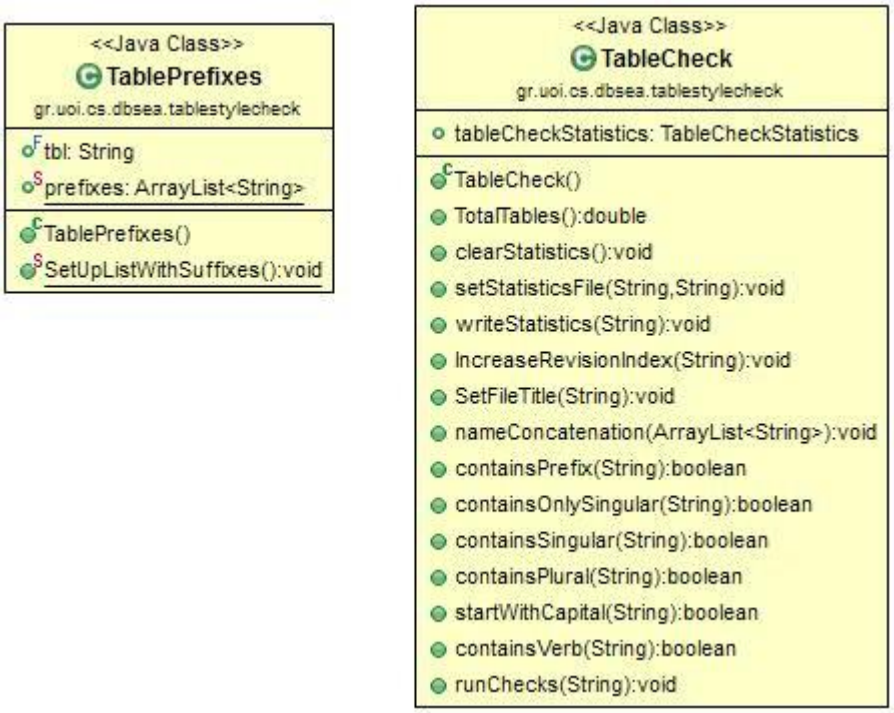


Figure 15 : The classes in tablestylecheck contain the rules about a table entity.

Package dbsea.columnsstylecheck

In Figure 16 are represented the classes in the package which implements the column specific rules. Those rules are enlisted below.

1. Name contains uniform postfix

2. Name contains only Singular
3. Name contains only Singular and not Plural
4. Contains Verb
5. Name equals "id"
6. Column Name is the same as the table

ColumnCheck is responsible of updating the respective metrics tables of the above rules as well as preparing the file both functions being done through aggregation with *ColumnStastistics* in *dbsea.statistics* package. This class implements the method *runchecks*, similar to the one used in TableCheck and executes the columns checks.

ColumnCheck is using ColumnCheckStatistics to clear the metrics, set the title of the file, the rules for columns generic and column specific and to write to the files. UniformSuffixes is where the suffixes proposes in Celko's book are held.

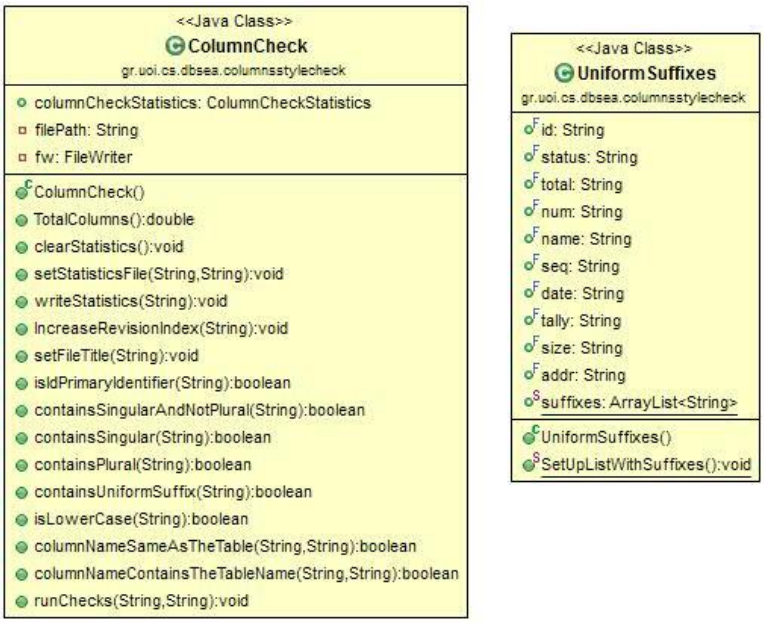


Figure 16 : The classes in columnsstylecheck contain the rules about a table entity.

Package dbsea.generalchecks

The classes of generalchecks are found in Figure 17.

Figure 17 : The classes in generalchecks contain the rules shared by tables and columns as well as some helper classes.

The main class of this package is GenericCheck since inside her reside the checks for conducted for both tables and columns. The generic checks are:

1. Lower Case
2. Upper Case
3. Pascal Case
4. Underscore Case with Lower Case
5. Underscore Case With Upper Case
6. Underscore Case with some other Case
7. Proper length
8. Begins with letter

9. Ends with letter or number
10. Words more than strings in name
11. Camel Case
12. Is reserved Keyword
13. Contains consecutive underscores
14. Contains space
15. Contains special character
16. Contains delimiters

As show in Figure 20, ColumnCheck and TableCheck have inheritance relationship with GenericChecks.

GenericChecks has a runChecks method too and is called inside the runChecks of the two aforementioned classes, having the same obligation, to run all the checks implemented inside GenericCheck.

The enum TypeofCases defines the type of cases that can be found in names. Those are :

1. Lowercase
2. Uppercase
3. Pascal case
4. Camel case
5. Lowercase with underscores
6. Uppercase with underscore
7. Other case with underscore

All the types are self-explanatory except the OtherCaseWithUnderscore. This one is defined as the type of case which is NOT one of the others. An example of this type of case is the table “CPG_albums” from Coppermine, this table’s name consists of uppercase characters, an underscore and lowercase characters.

The class CaseCheck, as the name reveals is responsible of deciding for a given string, its type of case.

The class ReservedWords is responsible for loading all the reserved words MsSQL and MySQL that have been defined in a resource's file and of answering about the adherene of Is reserved Keyword rule. The reserved words are those of the date of creation of the class 15/12/17.

Package dbsea.statistics

In Figure 18 are found the classes responsible for information extraction.

Each check has a corresponding class. GenericCheckStatistics, TableCheckStatistics and ColumnCheckStatistics are used via aggregation inside the tablestylecheck and the columnstylecheck package.

Package dbsea.wordnetchecks

Inside this package is found the pinnacle of the SQL style rules the class WordnetCheck.

WordnetCheck has the ability to determine if a name contains

- Noun
- Verb
- Adjective (not used in the style analysis)

Or if a name is

- Plural
- Singular

Also measures the strings and the actual words inside a name. An actual word is a string which is recognized by WorldNet as word. This package uses the libraries edu.smu.tspell and edu.stanford.nlp. Those two are combined in an ad hoc solution to give as the ability to measure meaningfulness inside a name.

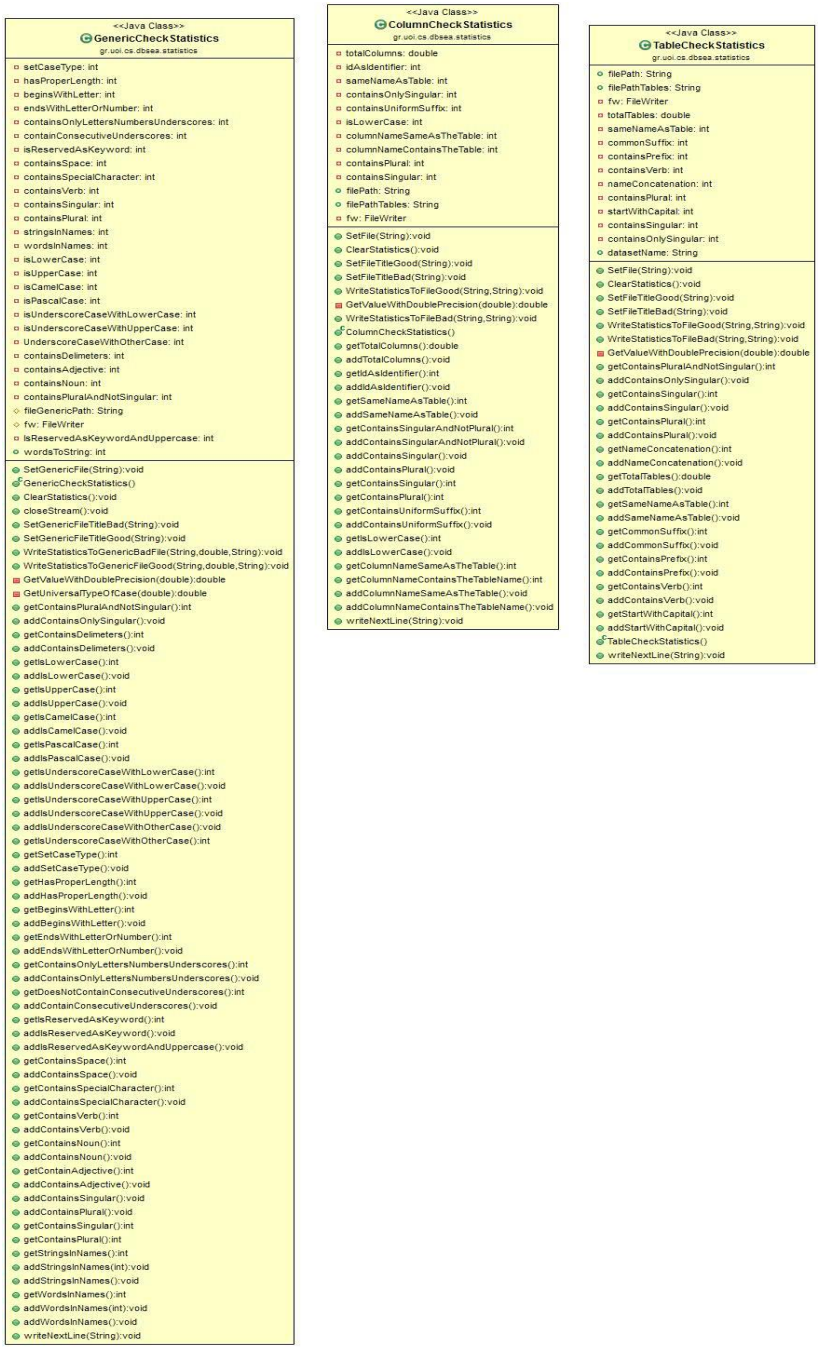


Figure 18 : The classes from the package statistics, responsible of keep track of the violated/adhered rules, setting up the title of column, table file and of writing the metrics.

This solution is possible through the use of posTags (Part of Speech Tags) from edu.stanford.nlp. For more information about WorldNet please refer to the official site[36].

Package Swing

The gui package contains the classes which give the user the ability to interact with dbsea.

The *MainWindow* is the graphical environment the user confronts when *DBsea* is started. It has two main options *File* and *Help.*

The *File* option enables the user to select a parent folder or close the program. By choosing a parent folder the initiation of the SQL Style Analysis begins through the call of the method traversePaths. This functionality resides inside *OpenFolderDialog.*

The *Help* option is is based upon the classes *InstructionsDialog* and *AboutDialog*. The user gets access to a simple and straightforward tutorial on how to use DBsea as well as to information about the author.

[36] https://wordnet.princeton.edu/

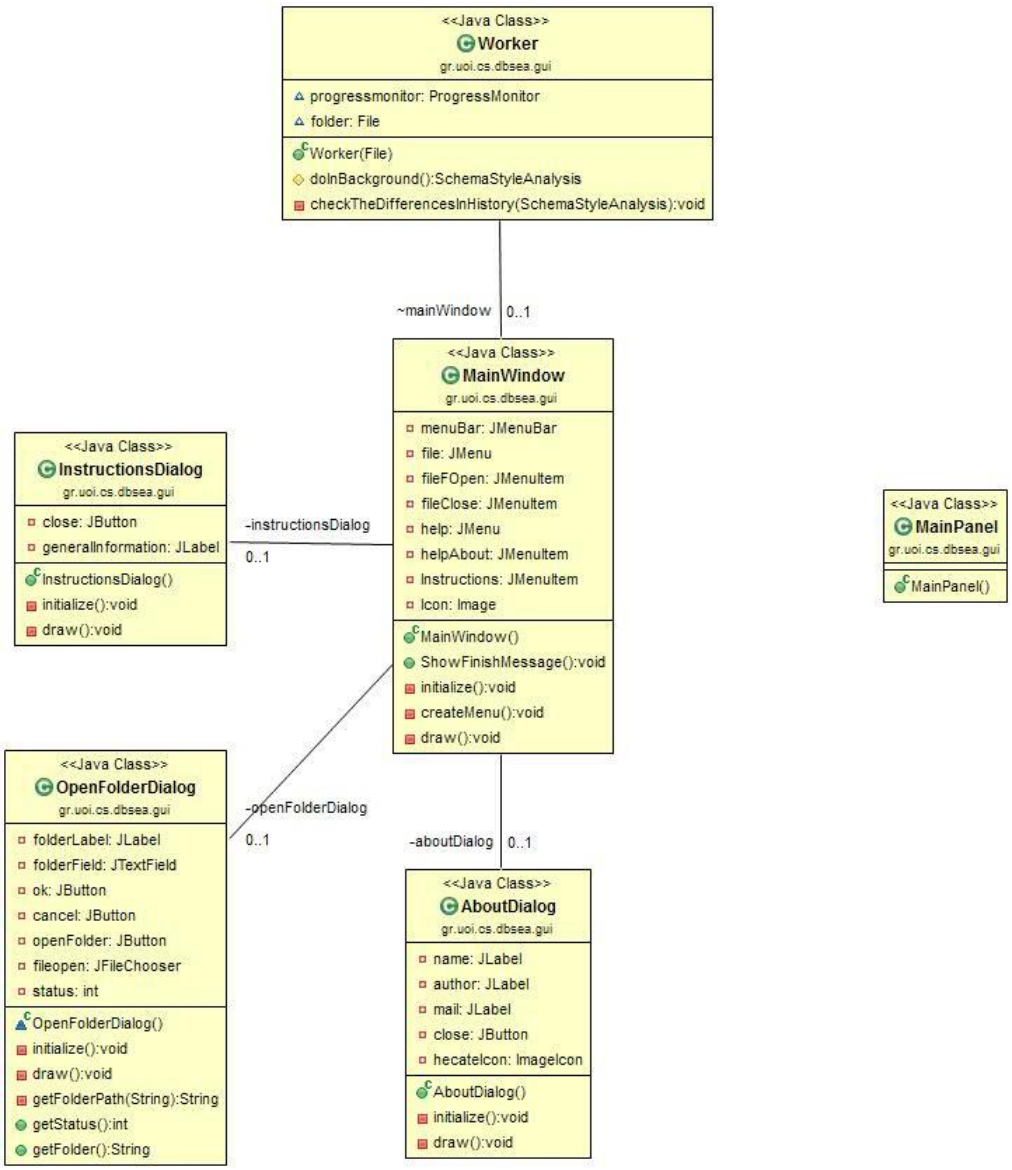


Figure 19 : Implementation of UI, gui package

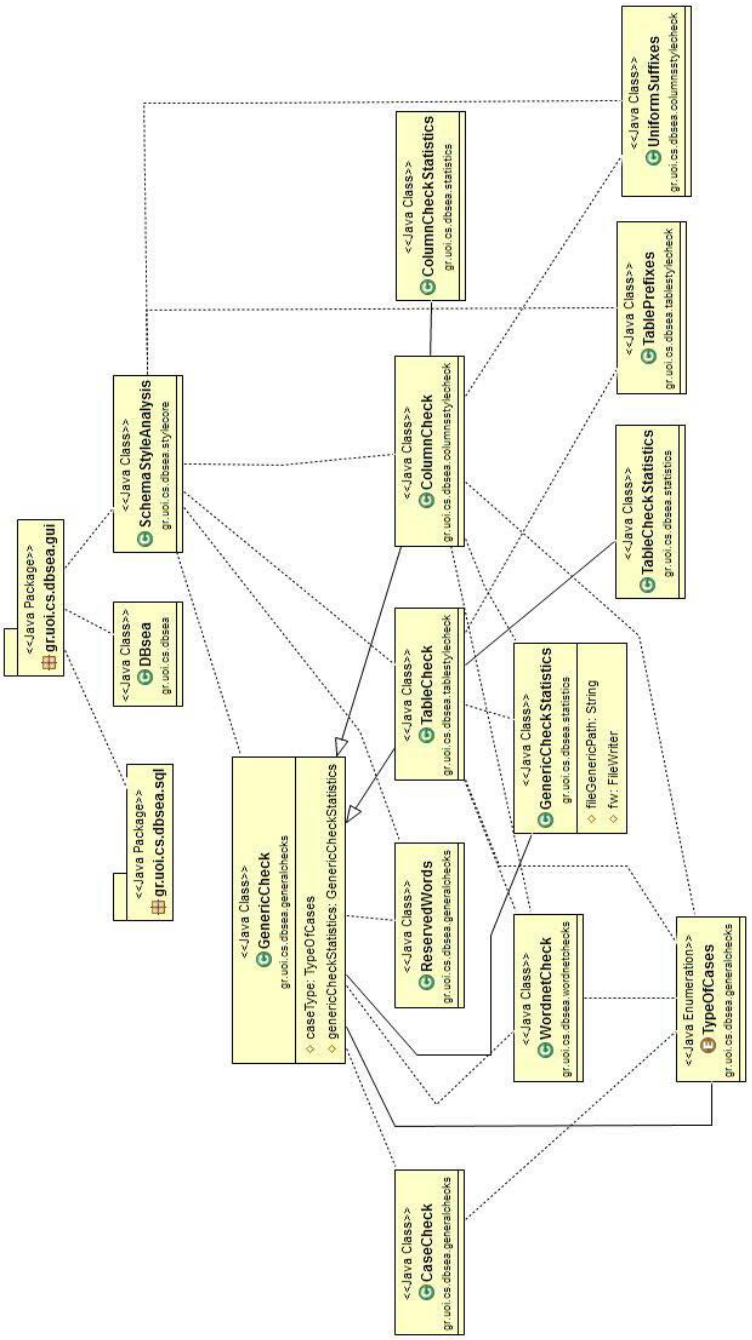


Figure 20 : The whole class diagram of SSet

Reserved Words

Oracle reserved words

```
ACCESS
ADD
ALL
ALTER
AND
ANY
AS
ASC
AUDIT
BETWEEN
BY
CHAR
CHECK
CLUSTER
COLUMN
COLUMN_VALUE (See Note 1 at the end of this list)
COMMENT
COMPRESS
CONNECT
CREATE
CURRENT
DATE
DECIMAL
DEFAULT
DELETE
DESC
DISTINCT
DROP
ELSE
EXCLUSIVE
EXISTS
FILE
FLOAT
FOR
FROM
GRANT
GROUP
HAVING
IDENTIFIED
IMMEDIATE
IN
INCREMENT
INDEX
INITIAL
INSERT
INTEGER
INTERSECT
INTO
```

IS
LEVEL
LIKE
LOCK
LONG
MAXEXTENTS
MINUS
MLSLABEL
MODE
MODIFY
NESTED_TABLE_ID
NOAUDIT
NOCOMPRESS
NOT
NOWAIT
NULL
NUMBER
OF
OFFLINE
ON
ONLINE
OPTION
OR
ORDER
PCTFREE
PRIOR
PUBLIC
RAW
RENAME
RESOURCE
REVOKE
ROW
ROWID
ROWNUM
ROWS
SELECT
SESSION
SET
SHARE
SIZE
SMALLINT
START
SUCCESSFUL
SYNONYM
SYSDATE
TABLE
THEN
TO
TRIGGER
UID
UNION
UNIQUE
UPDATE
USER

```
VALIDATE
VALUES
VARCHAR
VARCHAR2
VIEW
WHENEVER
WHERE
```

MsSQL reserved words

```
ADD
EXTERNAL
PROCEDURE
ALL
FETCH
PUBLIC
ALTER
FILE
RAISERROR
AND
FILLFACTOR
READ
ANY
FOR
READTEXT
AS
FOREIGN
RECONFIGURE
ASC
FREETEXT
REFERENCES
AUTHORIZATION
FREETEXTTABLE
REPLICATION
BACKUP
FROM
RESTORE
BEGIN
FULL
RESTRICT
BETWEEN
FUNCTION
RETURN
BREAK
GOTO
REVERT
BROWSE
GRANT
REVOKE
BULK
GROUP
RIGHT
```

```
BY
HAVING
ROLLBACK
CASCADE
HOLDLOCK
ROWCOUNT
CASE
IDENTITY
ROWGUIDCOL
CHECK
IDENTITY_INSERT
RULE
CHECKPOINT
IDENTITYCOL
SAVE
CLOSE
IF
SCHEMA
CLUSTERED
IN
SECURITYAUDIT
COALESCE
INDEX
SELECT
COLLATE
INNER
SEMANTICKEYPHRASETABLE
COLUMN
INSERT
SEMANTICSIMILARITYDETAILSTABLE
COMMIT
INTERSECT
SEMANTICSIMILARITYTABLE
COMPUTE
INTO
SESSION_USER
CONSTRAINT
IS
SET
CONTAINS
JOIN
SETUSER
CONTAINSTABLE
KEY
SHUTDOWN
CONTINUE
KILL
SOME
CONVERT
LEFT
STATISTICS
CREATE
LIKE
SYSTEM_USER
```

```
CROSS
LINENO
TABLE
CURRENT
LOAD
TABLESAMPLE
CURRENT_DATE
MERGE
TEXTSIZE
CURRENT_TIME
NATIONAL
THEN
CURRENT_TIMESTAMP
NOCHECK
TO
CURRENT_USER
NONCLUSTERED
TOP
CURSOR
NOT
TRAN
DATABASE
NULL
TRANSACTION
DBCC
NULLIF
TRIGGER
DEALLOCATE
OF
TRUNCATE
DECLARE
OFF
TRY_CONVERT
DEFAULT
OFFSETS
TSEQUAL
DELETE
ON
UNION
DENY
OPEN
UNIQUE
DESC
OPENDATASOURCE
UNPIVOT
DISK
OPENQUERY
UPDATE
DISTINCT
OPENROWSET
UPDATETEXT
DISTRIBUTED
OPENXML
USE
```

```
DOUBLE
OPTION
USER
DROP
OR
VALUES
DUMP
ORDER
VARYING
ELSE
OUTER
VIEW
END
OVER
WAITFOR
ERRLVL
PERCENT
WHEN
ESCAPE
PIVOT
WHERE
EXCEPT
PLAN
WHILE
EXEC
PRECISION
WITH
EXECUTE
PRIMARY
WITHIN GROUP
EXISTS
PRINT
WRITETEXT
EXIT
PROC
```

Uniform Suffixes

id: a unique identifier such as a column that is a primary key.

status: flag value or some other status of any type such as public finalation_status.

total: the total or sum of a collection of values.

num: denotes the field contains any kind of number.

name: signifies a name such as first_name.

seq: contains a contiguous sequence of values.

date: denotes a column that contains the date of something.

tally: a count.

size: the size of something such as a file size or clothing.

addr: an address for the record could be physical or intangible such as ip_addr.

SHORT VITA

Papamichail Aggelos was born in Ioannina in 1991. He received his BSc degree from the Computer Science Department of University of Ioannina in July 2015. In January 2016 he became a MSc student in the same institution under the supervision of Zarras Apostolos. In July of 2017 he started working as a software developer in the banking department of Natech S.A.