

Graph metrics as predictors of schema evolution for  
relational databases.

A Thesis

submitted to the designated  
by the General Assembly of Special Composition  
of the Department of Computer Science and Engineering  
Examination Committee

by

Michail-Romanos Kolozoff

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE  
WITH SPECIALIZATION  
IN SOFTWARE

University of Ioannina

January 2017

## DEDICATION

---

To my mother, father, sister and all my brothers.  
To my family....

## ACKNOWLEDGMENTS

---

To anyone and everyone that has mentally, and otherwise helped me in succeeding this huge task. First and foremost my family, for the years of raising and supporting me. All my friends for always being there and letting me believe that I can achieve anything. And finally to my supervisor, Prof. Panos Vassiliadis for inspiring me from the very beginning with the art of software engineering, giving me courage in every difficulty that I stumbled upon, and genuinely making me love the field of Computer Science. Thank you all for making this thesis possible.

# TABLE OF CONTENTS

---

<b>Dedication</b>	<b>ii</b>	
<b>Acknowledgments</b>		<b>iii</b>
<b>Table of Contents</b>		<b>i</b>
<b>List of Tables</b>	<b>iv</b>	
<b>List of Figures</b>	<b>vi</b>	
<b>Abstract</b>	<b>viii</b>	
<b>Εκτεταμένη Περίληψη στα Ελληνικά</b>		<b>x</b>
<b>CHAPTER 1. Introduction</b>		<b>1</b>
1.1 Aim and Scope		1
1.2 Roadmap		3
<b>CHAPTER 2. Related work</b>		<b>5</b>
2.1 Case studies concerning schema evolution		5
2.2 Node importance in graphs		8
<b>CHAPTER 3. Background concepts</b>		<b>11</b>
3.1 Fundamental Definitions		11
3.2 Node and Edge Properties		14
3.2.1 Degree		14
3.2.2 Centrality and Prestige		14
3.2.3 Reciprocity and Transitivity		15

3.3	Graph Properties	16
3.3.1	Large weak component	16
3.4	Parmenidian Truth	18
<b>CHAPTER 4. Graph metrics evolution</b>		<b>21</b>
4.1	Experimental Setup	21
4.1.1	Atlas Trigger	21
4.1.2	BioSQL	22
4.1.3	EGEE-II: JRA1 Activity	23
4.1.4	Castor	23
4.1.5	SlashCode	24
4.1.6	Zabbix	24
4.2	Total number of node and edges	26
4.3	Diameter of Large Weak Component & number of Weak Components	29
4.4	How do the nodes and edges of Diachronic graph relate to the average graph snapshot	33
4.5	Summary of Findings	35
<b>CHAPTER 5. Evolution of table and foreign key metrics</b>		<b>37</b>
5.1	Simple degrees and their relationship to the table evolution	37
5.1.1	Statistical profile for tables with respect to graph properties	38
5.1.2	How simple degrees relate to the evolution of tables	46
5.2	Clustering Coefficient and its relationship to table evolution	50
5.2.1	Statistical profile for tables with respect to clustering coefficient	50
5.2.2	How clustering coefficient relates to the evolution of tables	52

5.3	Vertex Betweenness Centrality and its relationship to table evolution	54
5.3.1	Statistical profile for tables with respect to vertex betweenness	54
5.3.2	How does vertex betweenness relate to table evolution	56
5.3.3	Normalized Vertex Betweenness and its relationship to evolution	59
5.4	Edge Betweenness and its relationship to schema evolution	61
5.5	Summary of Findings	67
<b>CHAPTER 6. Software architecture and design – parmenidian truth</b>		<b>69</b>
6.1	Package Diagram	69
6.2	The core package	71
6.3	The export package	73
6.4	The model package	75
6.5	The gui package	77
6.6	The model.Loader package	79
6.7	The parmenidianEnumeration package	80
<b>CHAPTER 7. Conclusions and future work</b>		<b>83</b>
7.1	Summary	83
7.2	Future Work	84
<b>References</b>	<b>85</b>	
<b>Short CV</b>	<b>87</b>	

## LIST OF TABLES

---

Table 1	A description of the datasets we have used in this study	25
Table 2	Correlation of nodes and edges for the six datasets	27
Table 3	Pearson correlation for the studied metrics, with $ V $ standing for number of nodes, $ E $ for number of edges, $ C $ for number of weak components and $\delta$ for diameter	30
Table 4	Number of nodes contained in each dataset's lifetime	33
Table 5	Number of edges contained in each dataset's lifetime	33
Table 6	Number of nodes as percentage of the nodes of the Diachronic Graph	34
Table 7	Number of edges as percentage of the edges of the Diachronic Graph	34
Table 8	InDegree variants for specific nodes during evolution	37
Table 9	<i>InDegree</i> Breakdown for the 6 studied datasets. Each cell represents how many tables of the database have the respective average value (rounded) of the first column.	39
Table 10	<i>OutDegree</i> Breakdown for the 6 studied datasets. Each cell represents how many tables of the database have the respective average value (rounded) of the first column.	40
Table 11	Joint Distribution for the average in/out degree for the 6 studied datasets.	43
Table 12	Breakdown of node percentages per combination of degrees for the 6 studied datasets.	44
Table 13	Probability of survival with respect to total degree	47
Table 14	Clustering Coefficient Breakdown for the 6 studied datasets	50
Table 15	Probability of survival with respect to clustering coefficient	52

Table 16 Average Vertex Betweenness Breakdown for the 6 studied datasets	54
Table 17 Probability of survival with respect to avg. vertex betweenness	56
Table 18 Probability of survival with respect to normalized avg. vertex betweenness	59
Table 19 Breakdown of tables per category of EBC score and relationship to survival.	63
Table 20 Percentage of survivor tables per rank for all the studied data sets.	65



## LIST OF FIGURES

---

Figure 1 Diachronic graph of Egee along with its starting versions	13
Figure 2 A version of Atlas in Parmenidian Truth	17
Figure 3 Number of nodes and edges over time for the 6 studied data sets	26
Figure 4 Size of Diameter and Number of Weak Components over time for the 6 studied data sets	29
Figure 5 Percentage of Nodes and Edges within the LWC over time for the 6 studied datasets	31
Figure 6 Node Breakdown per Average InDegree for the 6 studied datasets	41
Figure 7 Node Breakdown per Average OutDegree for the 6 studied datasets	42
Figure 8 Distribution of nodes with respect to both their In and Out Degree scores for the 6 studied datasets	45
Figure 9 Contrasting InDegree scores with the last known appearance for the tables of the 6 studied datasets	48
Figure 10 Contrasting OutDegree scores with the last known appearance for the tables of the 6 studied datasets	49
Figure 11 Node Breakdown per Average Clust. Coeff. for the 6 studied datasets	51
Figure 12 Contrasting Clustering Coefficient scores with the last known appearance for the tables of the 6 studied datasets	53
Figure 13 Breakdown per Average Node Betweenness Centrality for the 6 studied datasets	55
Figure 14 Contrasting Average Vertex Betweenness scores with the last known appearance for the tables of the 6 studied datasets	57

Figure 15 Contrasting Norm. Avg. Vertex Betweenness scores with the last known appearance for the tables of the 6 studied datasets	60
Figure 16 Evolution of the 2-Core Components for the 6 studied datasets (figures are partially cropped to fit)	62
Figure 17 Edge Betweenness scores, ordered decreasingly, for all 6 data sets.	63
Figure 18 Package Diagram for Parmenidian Truth	70
Figure 19 Class Diagram for the core package	71
Figure 20 Class Diagram for the export package	74
Figure 21 Class Diagram for the model package	75
Figure 22 Class Diagram for the gui package	77
Figure 23 Class Diagram for the model.Loader package	79
Figure 24 Class Diagram for the ParmenidianEnumeration package	80

## ABSTRACT

---

Michail-Romanos Kolozoff. MSc in Computer Science, Department of Computer Science and Engineering, University of Ioannina, Greece. January 2017.

Graph metrics as predictors of schema evolution for relational databases.

Advisor: Panos Vassiliadis, Associate Professor.

Databases evolve over time and their evolution does not only concern their contents, but also their internal structure, or *schema*. Schema evolution impacts deeply, both the database itself, and the surrounding applications that need to adapt too. The study of the mechanisms and patterns via which database schemata evolve is important as it can allow the in-advance planning of design, maintenance and resource allocation with a view to the future.

In this thesis, we focus on the study of the evolution of foreign keys in the context of schema evolution. Foreign keys are mechanisms that constraint data entry in relational tables, imposing that the domain of the contents of a table's attribute is a subset of the contents of an attribute of another, lookup, table. Despite the importance of foreign keys, as an integrity constraint that guarantees consistency among the values of different tables, the study of their evolution is a topic that –to the best of our knowledge- has never been studied in the literature before.

We have studied the schema histories of a six free, open-source databases that contained foreign keys. To facilitate a quantitative study, we model each version of the schema as a graph, with tables as nodes and foreign keys as directed edges (stemming from the referencing table to the referenced one). Our findings concerning the growth of nodes verify previous results that schemata slowly grow over time in terms of tables. Moreover, we have come to several surprising, new findings in terms of the schema edges (foreign keys). Foreign keys appear to be fairly scarce in the projects that we have studied and they do not necessarily grow in synch with table growth. In fact, we have observed different “cultures” for the handling of foreign keys, ranging from full sync with the growth of nodes to the unexpected extreme of full removal of foreign keys from the schema of the database. Node degrees and

survival are related with an inverse gamma pattern: the few nodes with high degrees stand higher chances of survival than average. Similarly, nodes with inciting edges with high values for edge betweenness centrality frequently (but not always) stand higher chances to survive compared to the nodes with a single or zero inciting edges, which have significantly higher chances of removal.

## ΕΚΤΕΤΑΜΕΝΗ ΠΕΡΙΛΗΨΗ ΣΤΑ ΕΛΛΗΝΙΚΑ

---

Μιχάλης – Ρωμανός Κολοζώφ. ΜΔΕ στην Πληροφορική, Τμήμα Μηχανικών Η/Υ και Πληροφορικής, Πανεπιστήμιο Ιωαννίνων, Ιανουάριος 2017.

Γραφοθεωρητικές μετρικές για την πρόβλεψη της εξέλιξης σχημάτων βάσεων δεδομένων.

Επιβλέπων: Παναγιώτης Βασιλειάδης, Αναπληρωτής Καθηγητής.

Οι βάσεις δεδομένων εξελίσσονται με την πάροδο του χρόνου και η εξέλιξη τους δεν αφορά μόνο το περιεχόμενό τους, αλλά και την εσωτερική τους δομή, ή το Σχήμα. Η εξέλιξη του Σχήματος επιδρά βαθιά, στην ίδια τη βάση δεδομένων, και στις γύρω εφαρμογές που πρέπει να προσαρμοστούν πολύ, προκειμένου να αποφευχθεί (α) η αποτυχία λειτουργίας (λόγω αναφορών σε ανύπαρκτα στοιχεία του σχήματος), (β) η απώλεια πληροφοριών (λόγω της μη-συσχέτιση των νέων δεδομένων), ή (γ) οι σημασιολογικές αντιφάσεις (σε περίπτωση που η σημασιολογία των απόψεων αλλάξει). Η μελέτη των μηχανισμών και των προτύπων μέσω των οποίων εξελίσσονται τα σχήματα βάσης δεδομένων είναι σημαντική, καθώς μπορεί να επιτρέψει τον εκ των προτέρων σχεδιασμό για την ανάπτυξη εφαρμογών, συντήρησης της βάσης δεδομένων και την κατανομή των πόρων, με στόχο το μέλλον. Προς το παρόν οι γνώσεις μας για τέτοιους μηχανισμούς ή μοτίβα είναι ακόμα στα πρώτα της βήματα, κυρίως λόγω της επί μακρόν έλλειψης ιστοριών σχήματος. Μέχρι σήμερα, η σχετική βιβλιογραφία μετράει μόνο λίγες μελέτες επί του θέματος, αξιοποιώντας κυρίως τις ιστορίες σχήματος των ελεύθερων, ανοικτού κώδικα έργων που δημοσιεύουν ολόκληρο τον κώδικά τους (συμπεριλαμβανομένου του σχήματος της υποκείμενης βάσης δεδομένων τους) σε αποθήκες δημόσιου λογισμικού.

Σε αυτή την εργασία, θα επικεντρωθούμε στη μελέτη της εξέλιξης των ξένων κλειδιών στο πλαίσιο της εξέλιξης του σχήματος. Ξένα κλειδιά είναι οι μηχανισμοί που περιορίζουν την εισαγωγή δεδομένων σε σχεσιακούς πίνακες, επιβάλλοντας ότι ο τομέας των περιεχομένων του χαρακτηριστικού ενός πίνακα είναι ένα υποσύνολο των περιεχομένων ενός χαρακτηριστικού του άλλου, πίνακας αναζήτησης, (παρέχοντας ό, τι είναι επίσης γνωστό ως active domain

του αναφερόμενου χαρακτηριστικού). Έτσι, τα ξένα κλειδιά αποτελούν βασικό μηχανισμό για τη διασφάλιση της ακεραιότητας των δεδομένων και τη συνοχή μεταξύ των τιμών σε διαφορετικούς πίνακες. Παρά τη σημασία των ξένων κλειδιών, ως περιορισμό ακεραιότητας, η μελέτη της εξέλιξής τους είναι ένα θέμα που – από όσο γνωρίζουμε – δεν έχει ποτέ μελετηθεί στη βιβλιογραφία πριν. Η συσχετιζόμενη δουλειά έχει επικεντρωθεί στην μελέτη για την εξέλιξη των πινάκων και των χαρακτηριστικών, αφήνοντας τη μελέτη των ξένων κλειδιών ανέγγιχτη.

Έχουμε συλλέξει τις ιστορίες σχημάτων από έξι βάσεις δεδομένων ανοικτού κώδικα που περιείχαν ξένα κλειδιά, και τις επεξεργαστήκαμε για να ανακαλύψουμε τις αλλαγές που συνέβησαν στα συνεχόμενα releases τους. Έτσι, η ιστορία των εκδόσεων έχει συμπληρωθεί από ένα ιστορικό μεταβάσεων μεταξύ μεταγενέστερες εκδόσεις. Στη συνέχεια, μελετήσαμε τα χαρακτηριστικά της εξέλιξης των ξένων κλειδιών. Για να διευκολυνθεί μια ποσοτική μελέτη, μοντελοποιήσαμε κάθε σχήμα ως γράφημα, με τους πίνακες ως κόμβους και τα ξένα κλειδιά ως κατευθυνόμενες ακμές (που προκύπτει από τον πίνακα αντιστοίχισης με τον αναφερόμενο). Επιπλέον, με την πράξη της ένωση της ιστορίας αυτών των γραφημάτων, δημιουργήσαμε τον Διαχρονικό Γράφο της ιστορίας του σχήματος ο οποίος περιέχει όλους τους πίνακες και όλα τα ξένα κλειδιά που υπήρξαν ποτέ στην ιστορία του σχήματος. Βάση αυτού του μοντελοποιημένου γραφήματος, αξιολογήσαμε πολλές γράφο-θεωρητικές ιδιότητες, τόσο όσον αφορά το σύνολο του σχήματος αλλά και από την άποψη των μεμονωμένων κόμβων και ακμών. Η προσπάθειά μας έχει διευκολυνθεί από ένα εργαλείο που έχουμε αναπτύξει, με το όνομα Παρμενίδεια Αλήθεια, το οποίο (α) κατασκευάζει το Διαχρονικό Γράφο της ιστορίας, (β) τον χρησιμοποιεί ως μέσο συνοχής για την οπτικοποίηση της κάθε διαφορετικής έκδοσης του σχήματος, (γ) εξάγει την ιστορία του σχήματος ως μια παρουσίαση σε PowerPoint, βίντεο, ένα σύνολο εικόνων, και ένα σύνολο αρχείων GraphML, και (δ) παράγει αναφορές για διαφορετικές γραφοθεωρητικές μετρικές σε csv αρχεία, διευκολύνοντας έτσι την μετέπειτα ανάλυσή τους.

Τα ευρήματά μας συνοψίζονται ως εξής:

Τα Σχήματα αυξάνονται με την πάροδο του χρόνου από την σκοπιά των κόμβων (πίνακες). Η ανάπτυξη είναι ομαλή και αργή, με αρκετές περιόδους ηρεμίας. Αυτό είναι ένα πολύ γνωστό αποτέλεσμα από την υπάρχουσα βιβλιογραφία που έχει επίσης επαληθευτεί από τη μελέτη μας. Οι ακμές (ξένα κλειδιά) δεν αναπτύσσονται απαραίτητα σε συγχρονισμό με την ανάπτυξη των πινάκων. Στην πραγματικότητα, έχουμε παρατηρήσει διαφορετικές "κουλτούρες" για το χειρισμό των ξένων κλειδιών. Σε δύο περιπτώσεις που αφορούν επιστημονικές βάσεις δεδομένων (Atlas, Biosql), τα ξένα κλειδιά αποτελούν αναπόσπαστο μέρος του σχήματος, εκτείνονται σε ένα τεράστιο ποσοστό των πινάκων και συν-

εξελισσονται μαζί τους. Σε δύο άλλες περιπτώσεις, επίσης επιστημονικής φύσης (Egee και Castor), μόνο ένα υποσύνολο των πινάκων συμμετέχουν σε συσχετίσεις ξένων κλειδιών και η εξέλιξή τους είναι διττή: Η Egee (με πολύ μικρού μεγέθους Σχήμα) έχει μια ισχυρή συσχέτιση μεταξύ πινάκων και εξέλιξης ξένων κλειδιών, ενώ ο Castor (με ένα μικρό ποσοστό των πινάκων να εμπλέκονται σε ξένα κλειδιά) έχει ανάμικτη συμπεριφορά σε όλη την ιστορία της εξέλιξής του. Η μεγαλύτερη έκπληξη ήρθε από τα σχήματα Συστήματος Διαχείρισης Περιεχομένου (CMS), SlashCode και Zabbix, όπου τα ξένα κλειδιά εμπλέκονται μόνο σε μια μικρή μειοψηφία των πινάκων. Προς μεγάλη μας έκπληξη, τα ξένα κλειδιά σε αυτά τα έργα, μετά από μια περίοδο ανάπτυξης, αφαιρούνται από το σύστημα (α) με μια απότομη απομάκρυνση στην πρώτη περίπτωση, και (β) με μια αργή αλλά σταθερή ταχύτητα απομάκρυνσης στην δεύτερη. Η συνολική μας εντύπωση είναι ότι, με την εξαίρεση κάποιων περιβαλλόντων με αυστηρή τήρηση των υπαγορεύσεων της σχεσιακής θεωρίας, τα ξένα κλειδιά είναι σπάνια και ενίοτε ανεπιθύμητα.

Όσον αφορά τη συμπεριφορά μεμονωμένων κόμβων, φαίνεται ότι ο βαθμός και η επιβίωση σχετίζονται θετικά μεταξύ τους, και, στην πραγματικότητα, υπάρχει ένα μοτίβο αντίστροφου - Γ στο συσχετισμό βαθμού και επιβίωσης. Το μοτίβο προτείνει ότι οι χαμηλοί σε βαθμούς κόμβοι φέρουν μη αμελητέα πιθανότητα απομάκρυνσης για τους αντίστοιχους πίνακες- ταυτόχρονα, οι υψηλόβαθμοι, αν και αρκετά σπάνια, φέρουν πιθανότητα διαγραφής 5% -19% χαμηλότερα από το μέσο όρο. Έχουμε μελετήσει επίσης κόμβους για τον συντελεστή ομαδοποίησή τους και την κεντρικότητά τους με μάλλον αμφίβολα αποτελέσματα όσον αφορά την επιβίωση του κόμβου. Τέλος, φαίνεται ότι οι κόμβοι που εφάπτονται με υψηλά score σε betweenness - centrality, συχνά (αλλά όχι πάντα) έχουν υψηλότερες πιθανότητες επιβίωσης από κόμβους με μία ή καμία προσκείμενη ακμή.





# CHAPTER 1.

## INTRODUCTION

---

### 1.1 Aim and Scope

### 1.2 Roadmap

---

### 1.1 Aim and Scope

Databases evolve over time and their evolution does not only concern their contents, but also their internal structure, or *schema*. Schema evolution impacts deeply, both the database itself, and the surrounding applications that need to adapt too, in order to avoid (a) crashing (due to references to inexistent schema elements), (b) information loss (due to the non-referencing of newly added data), or (c) semantic inconsistencies (in case the semantics of views change). The study of the mechanisms and patterns via which database schemata evolve is important as it can allow the in-advance planning of application development, database maintenance and resource allocation with a view to the future. Still, our knowledge of such mechanisms or patterns is still in its early steps, mainly due to the longtime lack of schema histories. To this day, the related literature counts only a handful of studies of the topic, mainly exploiting the schema histories of free, open source projects that publish their entire code (including the schema of their underlying database) to public software repositories.

In this thesis, we focus on the study of the evolution of foreign keys in the context of schema evolution. Foreign keys are mechanisms that constraint data entry in relational tables, imposing that the domain of the contents of a table's attribute is a subset of the contents of an attribute of another, *lookup*, table (providing what is also known as *active domain* of the referencing attribute). Thus, foreign keys are a key

mechanism to guarantee data integrity and consistency between the values of different tables. Despite the importance of foreign keys, as an integrity constraint, the study of their evolution is a topic that –to the best of our knowledge- has never been studied in the literature before. Related work has worked on the evolution of tables and attributes, leaving the study of foreign keys untouched.

We have collected the schema histories of a six free open-source databases that contained foreign keys, and processed them to discover the changes that occurred between subsequent releases. Thus, the history of versions has been complemented by a history of transitions between sequential versions. Subsequently, we studied the characteristics of foreign key evolution. We want to see if the topology of relations within a relational schema has anything to do with their behavior concerning the evolution of the schema. To this end, we structure each version of a schema as a graph, with relations as nodes and foreign keys as edges. We also introduce the idea of the Diachronic Graph, a graph that encompasses a node for every table that has ever appeared in the history of the database and an edge for every foreign key that has ever appeared in the history of the database. One can think of the Diachronic Graph as the superimposition of all the graphs of the different versions – equivalently, the graph of each version is the projection of the Diachronic Graph for the tables and foreign keys that are present in that particular version. Based on this graph modeling, we assess several graph-metric properties, both in terms of the entire schema and in terms of individual nodes and edges. Our effort has been facilitated by a tool we have developed, called *Parmenidian Truth*, that (a) constructs the Diachronic Graph of the history, (b) uses it as the means of consistently visualizing the different version of the schema, (c) exports the story of the schema as a PowerPoint presentation, a video, a set of images, and a set of GraphML files, and (d) reports different graph theoretic measures in csv files, thus facilitating their subsequent analysis.

The main research question we address is: is there any inherent property of the graph constructs that forecasts their liability to change? We want to understand if graph-theoretic properties like the degree or the centrality of a relational table are related somehow to its evolutionary history. We start with traditional correlation analysis and depending on the results we direct research towards algorithms that reveal the internal mechanisms of this evolution.

Schemata grow over time in terms of nodes (tables). Growth is smooth and slow, with several periods of calmness. This is a well-known result from the existing literature that is also verified by our study too. Edges (foreign keys) do not necessarily grow in synch with table growth. In fact, we have observed different “cultures” for the handling of foreign keys. In two cases concerning scientific databases (*Atlas*, *Biosql*), foreign keys are an integral part of the schema, span a vast

percentage of tables and co-evolve with them. In two other cases, also of scientific nature (Egee and Castor), only a subset of tables were involved in foreign key relationships and their evolution is biased: Egee (with a very small schema size) has a strong correlation of table and foreign key evolution, whereas Castor (with a small percentage of tables being involved in foreign keys) has mixed behavior throughout its history. The biggest surprise came from the data sets of Content Management System (CMS) nature, SlashCode and Zabbix, where foreign keys involved only a small minority of tables. To our big surprise, foreign keys in these projects, after a period of growth, are removed from the system (a) with a steep removal in the first case and (b) a slow but constant removal rate in the latter. Our overall impression is that, with the exception of few environments with a strict adherence to the dictations of relational theory, foreign keys are scarce and occasionally unwanted.

Concerning the behavior of individual nodes, it appears that degree and survival are positively related, and, in fact, there is an inverse-gamma pattern in the correlation of degree and survival. The pattern suggests that low degrees carry a non-negligible probability of removal for the corresponding tables; at the same time, high degrees, although scarce enough, carry a probability of removal 5%-19% lower than average. We have also studied nodes for their clustering coefficient and vertex betweenness centrality with rather inconclusive results in terms of node survival. Finally, it appears that nodes touched by edges with high edge betweenness centrality, frequently (but not always) have higher chances of survival than nodes with a single or zero incident edges.

## 1.2 Roadmap

In Chapter 2, we review the state of the art that is closely related to our problem. In Chapter 3, we discuss background concepts around foreign keys, graph metrics as well as the data extraction process and a summary of the tool we have developed for our work. In Chapter 4, we discuss the evolution of foreign keys viewed from the point of view of the entire schema. In Chapter 5, we focus on graph metrics for individual tables and foreign keys and discuss how these graph metrics related to the survival of tables. In Chapter 6, we discuss the internal architecture of the tool we have developed, Parmenidian Truth. Finally, in Chapter 7, we conclude our discussion and offer suggestions for follow up work.



## CHAPTER 2.

### RELATED WORK

---

#### 2.1 Case studies concerning schema evolution

#### 2.2 Node importance in graphs

---

#### 2.1 Case studies concerning schema evolution

The first case study we will discuss [Kara01], concerns the impacts that database schema evolution has to the application level at a macroscopic level. This research was conducted for the purpose of providing a technology for maintaining consistency between database schemata and their corresponding applications, in an object-oriented context, and, at the same time, providing the evaluation of this technology from the perspective of the schema and application developers. The authors applied the *transitive closure algorithm*, which takes as input a directed acyclic graph with components as nodes and relationships between them as edges and finds all components reachable from a given component, in their tool where schema and application components are represented as nodes and the relationship between them as edges. The kinds of relationship between components are inheritance, encapsulation, aggregation, and usage. Based on this idea, the authors created a tool called SEMT (Schema Evolution Management Tool), where the components are extracted from the source files and the relationships between them are identified. The evaluation was focused on performance and usability. The general user satisfaction was relatively high. In the end, the purpose of this paper was to highlight that the ensuring of consistency between a database schema and applications during schema evolution is a non-trivial task. So, by applying knowledge from the software change

impact analysis and software visualization in the area of database schema evolution the SEMT tool was created.

The next case study that we discuss in this section was performed by Sjøberg [Sjob93], where a method for measuring modifications to database schemata and their consequences, was presented. For this purpose, a measuring tool, *Thesaurus*, was built to monitor the evolution of a large, industrial database application, a health management system (HMS) for a period of 18 months. The Thesaurus tool assists in keeping track of the use of names in the HMS application. An important requirement of this tool was that its contents should not be manually maintained. This was achieved by periodically scheduled, source program and database schema scans, in order to detect record changes. The resulting publication of this research effort [Sjob93] reports on how the schema changed. During the period of the study, the number of relations increased from 23 to 55 (139% increase) and the number of fields increased from 178 to 666 (274%). The most interesting result, however, is the fact that every relation had been changed. At the beginning of the development, almost all changes were additions. After the system provided a prototype and later went into production use, there was not a diminution in the number of changes, but the additions and deletions were more nearly in balance. Having measured the consequences of the schema changes on the application programs, the author suggests that the results confirm that change management tools are needed.

Our next case study [CMTZ08] presents an in-depth analysis of the evolution history of the Wikipedia database and its schema. For the purpose of this study a schema evolution tool is produced for the analysis of a real-life web information system. The authors studied the evolution of the MediaWiki software, which is a browser-based web-application under the hood of many applications, and most importantly, Wikipedia. First, the authors studied the size of MediaWiki DB schema in history in terms of the number of tables and columns. The number of tables has increased from 17 to 34 (100% increase) and the number of columns from 100 to 242 (142%). The schema growth is due to three main driving forces, specifically, (a) performance improvement, (b) addition of new features, and, (c) the growing need for preservation of the history of the contents of the database. The authors also classified changes in schema size in two different ways, macroscopically, by focusing on schema change, index/key adjustments, rollback to previous versions and documentation changes, and microscopically, by making use of the Schema Modification Operators, which are SQL queries like *Create Table*, *Drop Table*, *Rename table* etc. So, in order to study the effect of schema evolution on the frontend

applications, the authors analyzed the impact of the schema changes on a variety of different sets of queries and resulted to the conclusion that MediaWiki had undergone through a very intensive schema evolution as a result of a cooperative, multi-party development, something that is common in leading-edge Web Information projects.

In another line of work, the main goal of [LiNe09] is the study of *Collateral evolution of applications and databases*. This term is used in order to denote potential inconsistencies that arise when a database and the application programs using that database do not evolve simultaneously. In this work, object of study was the relationship between the evolution of an application and the evolution of the database system used to store the necessary data. Apparently, if these lines of evolution are not performed in sync, this may lead to collateral effects such as data loss, program failure, or decreased performance. To collect empirical evidence on the collateral evolution of application programs and databases, the authors followed the following steps:

- Firstly, an evolution study that identified changes to database schemas was performed
- Secondly, the evolution of database file formats was studied
- Next, an investigation in how application programs and database management systems cope with these changes was done
- Finally, some solutions for facilitating and ensuring the safety of applications and database evolution were given.

The conclusion of this study is that the current co-evolution approaches are inadequate and further research towards this goal is needed.

In the work of [SkVZ14], the authors performed a large-case study on the evolution of open-source databases and made observations regarding Lehman's laws of software evolution. For the purpose of this case study eight datasets were collected and cleansed. The resulting histories were then processed by a tool, Hecate, which was built for this purpose and proceeds in the following sequence of steps:

- First, Hecate, detects changes at both the attribute and the relation level
- Second, the tool produces the differences between two subsequent committed versions for all the subsequent versions in the schema history. This transcript

of changes is a sequence of deltas for each transition from a version to the next.

- Finally, based on all this sequence of detected changes, Hecate, produces statistical measures that characterize not only the overall evolution of the entire schema, but also the evolution of individual tables.

This study made several contributions regarding the evolution of real-world open-source databases, the most interesting of which are:

- The schema evolution happens in bursts and in grouped periods of evolutionary activity and not as a continuous process
- All datasets had the tendency to grow over time
- Age results to a reduction in the density of changes
- Change is quite more frequent in the beginning
- The schema size of a certain version of the database can be accurately estimated via a regressive formula that exploits the amount of changes in previous versions

## 2.2 Node importance in graphs

Another study [WhSm03] addresses the definition and computation of the importance of nodes in a graph, relative to one or more root nodes. The authors study a selection of different algorithms from social networks and graph theory, like Markov models and Web analysis. Then, these algorithms are evaluated based on simulated data on toy graphs as well as on real-work networks. Regarding node importance, a number of different approaches have been developed, like the centrality of a node in a social network, as well as the embedding of a social network data in latent Euclidean spaces. In the area of Web graphs, computer scientists have proposed a number of algorithms like HITS and PageRank. The purpose of [WhSm03] is to determine the relative importance of nodes in a graph with respect to a set of root nodes. Using graph-theoretic notions of distance defined explicitly on the graph as a general framework for estimating relative importance the authors indicate the following algorithms:

- *Shortest paths*, an important metric for measuring pair-wise relations in a graph



- *K-shortest paths*, for taking into consideration paths that are usually longer than the shortest path but important nonetheless
- *K-short Node-Disjoint Paths*, which refers to sets of k-short paths that have neither edges nor nodes in common

Thereinafter, the authors make a conceptually different approach, as to view the graph as representing a stochastic process, more specifically, a first-order Markov chain. So, they study focuses on the ensuing algorithms:

- *Markov Centrality*, where the inverse of the mean first-passage time in the Markov chain is examined
- *PageRank with Priors*, where the PageRank algorithm is extended to generate personalized ranks, and therefore is retrofitted into the current framework
- *HITS with Priors*, similar to the prior modification, the HITS algorithm is extended to fit the same framework
- *K-step Markov*, similar to the PageRank with priors and Hits with priors, but this time the procedure ends after K steps

Finally, those algorithms were tested in both simulated and real-world data with promising results, for the identification of relative-importance nodes.

In this master thesis, we try to find out which graph-theoretic properties, if any, are helpful for predicting a relational database's schema evolution. At first, we study each schema's graph macroscopically, trying to find if there is any correlation regarding the number of nodes, edges, weak components and diameter. Afterwards, we consider metrics that microscopically describe the graph and contain information about its topography, such as the in and out degree of a node, its clustering coefficient, and the vertex and edge betweenness.



## CHAPTER 3.

### BACKGROUND CONCEPTS

---

#### 3.1 Fundamental Definitions

#### 3.2 Node and Edge Properties

#### 3.3 Graph Properties

#### 3.4 Parmenidian Truth

---

### 3.1 Fundamental Definitions

The schema or intention of a relation is defined as a triplet (name, set of attributes, primary key). A foreign key constraint is a pair between a set of attributes in a certain relation  $R$  (called the source of the foreign key) and a set of attributes in a relation  $R$  (called the target of the foreign key). The foreign key constraint requires a 1:1 mapping between the attributes of the source and the target. As usual, at the extensional level, the semantics of the foreign key denote a subset relation between the instances of the source and the instances of the target. For the purpose of this thesis, we treat a relational database schema as a set of relations along with their foreign key constraints.

We model the database schema as a directed graph  $G(V, E)$ , with relations as nodes and foreign keys as directed edges, originating from their source and targeted to their target. If two relations have more than one foreign key with the same direction,

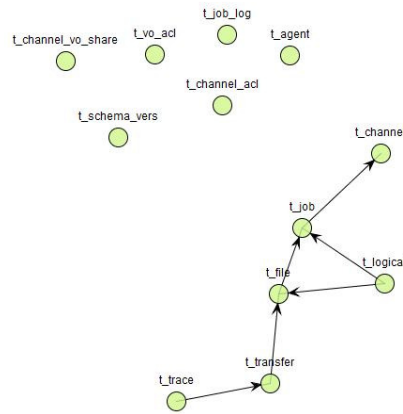
the single edge that connects them is annotated with all the foreign key pairs involved.

The evolution history of each database schema can be thought of as (a) a sequence of versions, but also as (b) a sequence of revisions. Unless otherwise specified, we will treat the term history under the semantics of the former of the two representations.

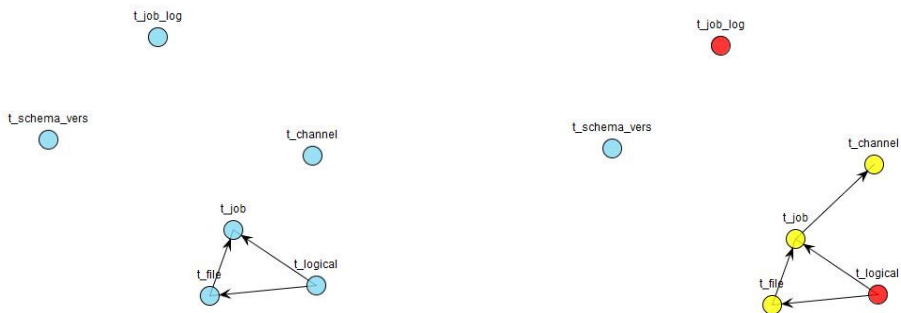
**Schemata as graphs.** Each version of the schema  $v^i$  is a graph  $G^i(V^i, E^i)$ . A transition between two subsequent versions of the history involves a set of changes involving relation additions, relation deletions, relation updates (attribute additions or deletions, change of attribute data types, changes of primary keys), as well as foreign key additions and deletions.

**Diachronic Graph.** Assuming an evolution history  $H=\{v^1, \dots, v^n\}$  for the database schema, the Diachronic Graph of the database schema is a graph  $G^\Omega (V^\Omega, E^\Omega)$  where (a)  $V^\Omega$  is the union of all  $V^i$ , and (b)  $E^\Omega$  is the union of all  $E^i$ .

As an example, the evolution history of the Egee dataset is presented in Figure 1. The first graph represents Egees' Diachronic graph and the following graphs represent a few versions with deletions and additions that shaped the Diachronic graph. The idea is that nodes are only added to the diachronic graph; thus it is the outcome of progressively adding all the table additions to the original version, without removing any. Observe that in the second version, two tables are deleted; yet they are present in the Diachronic Graph. In terms of coloring, a node that is deleted is colored red, nodes that are added are colored green and nodes with internal updates (e.g., attribute additions, deletions, etc) are yellow.

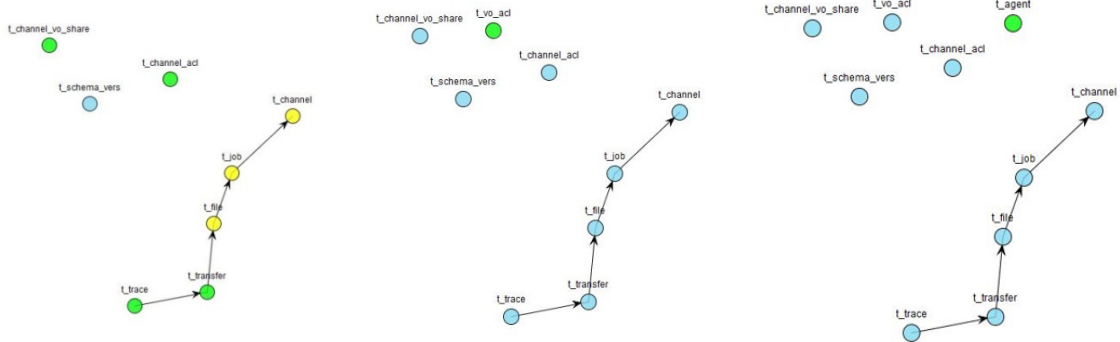


Diachronic Graph



Start: v. 1.0.1

Deletions: v. 1.0.2



Additions: v. 1.0.8

Additions: v. 1.0.15

Additions: v. 1.0.17

(final v.)

Figure 1 Diachronic graph of Egee along with its starting versions

## 3.2 Node and Edge Properties

As we will work with a graph representation of our schemata, we will attempt a graph-metric assessment of how the schema evolves. In other words, we will assess how important measurable properties of the graph evolve over time. In our deliberations, we employ several graph metrics that can be classified as (a) applicable to individual nodes and (b) to the entire graph<sup>1</sup>.

### 3.2.1 Degree

*Degree of a node.* The degree of a node,  $degree(v)$ , is the number of edges incident to the node.

*In-Degree of a node.* The in-degree of a node,  $in-degree(v)$ , is the number of incoming edges to the node.

*Out-Degree of a node:* The out-degree of a node,  $out-degree(v)$ , is the number of outgoing edges to the node.

### 3.2.2 Centrality and Prestige

Given two arbitrary nodes of a graph, it is possible that more than one paths exist that connect them. The distance between two nodes in a graph is the number of edges in a shortest path connecting them. If there is no path connecting the two nodes, i.e., if they belong to different connected components, then, conventionally their distance is defined as infinite.

*Eccentricity.* The eccentricity  $ecc(v)$  of a node  $v$  in a graph, is the maximum distance of  $v$  with respect to any other node in the graph. Conventionally, low values of eccentricity indicate nodes with a central position in the graph, thus a low eccentricity is a good indicator of a node's centrality.

---

<sup>1</sup> All the definitions are based on the online dictionary of graph metrics available at <http://reference.wolfram.com/language/Combinatorica/guide/GraphProperties.html>

The *Betweenness-Centrality* is an indicator of a node's centrality in a network and it is equal to the percentage of shortest paths from all nodes to all others that pass through that node. A node with high betweenness centrality has a large influence on the transfer of items through the network, under the assumption that item transfer follows the shortest paths [Brand01]. The same metric was used for both nodes and edges.

### 3.2.3 Reciprocity and Transitivity

*Clustering coefficient.* The clustering coefficient  $cc(v)$  of a node  $v$  is defined as follows:

- If  $degree(v)$  belongs  $\{0,1\}$ ,  $cc(v) := 0$
- Let  $N(v)$  denote the set of neighbors of  $v$ , i.e. every node  $u$ , such that an edge  $(u,v)$  or  $(v,u)$  belongs to the graph. Observe that in this definition, we treat the graph as undirected. Assume that the cardinality of  $N(v)$  is  $n$ , i.e.,  $v$  has  $n$  neighbors. Then, the interconnectivity between the members of  $N(v)$ , is quantified as the number of edges between members of  $N(v)$ , denoted as  $E_N(v)$ . Then,  $cc(v)$  is defined as the division of  $E_N(v)$  by the number of edges that could possibly exist between the members of the neighborhood of  $v$ , which is  $n \cdot (n-1) \cdot 0.5$ .

$$CC(v) = \frac{E_N(v)}{n(n-1) \frac{1}{2}}$$

Less formally, the clustering coefficient  $cc(v)$  of a node  $v$  is the fraction of  $v$ 's neighbors that are also neighbors of each other [Newm03].

### 3.3 Graph Properties

Assuming a graph  $G(V, E)$ , we can also measure the following properties of the graph.

*Number of Edges.* The number of edges,  $|E|$ , that are contained in the graph.

*Number of Nodes.* The number of nodes,  $|V|$ , that are contained in the graph.

*Number of Weak Components.* A weak component is defined as a maximal subgraph with at least 2 nodes, in which all pairs of nodes in the subgraph are reachable from one another in the underlying undirected subgraph.

Then, the nodes of the graph are partitioned into disjoint, non-adjacent partitions,  $V = \{V_1 \cup \dots \cup V_n\}$ , whose number we measure as the number of weak components of the graph. Note that we only consider non-trivial weak components that include at least two nodes.

*Diameter.* The diameter of a graph is the maximum eccentricity of any node in the graph, or equivalently, the maximum shortest distance between any pair of nodes.

In case there exists a pair of nodes for whom there is no path, the diameter is infinite. In order to compute the diameter of a database's graph, we have treated it as undirected.

#### 3.3.1 Large weak component

Since all the datasets that we studied for the purpose of this thesis are disconnected graphs<sup>2</sup>, thus the diameter would be infinite, we need to find a way to assess the "size" of the graph via its diameter.

We addressed this issue by defining as *Weak component* a maximal subgraph which would be connected if we ignored the direction of the edges.

---

<sup>2</sup> It is interesting to consider that it takes just a single table without foreign keys to make the graph of the schema of any relational database fall in this category.



We define the *large weak component (LWC)* of the graph as the *weak component* that includes the largest number of nodes among all components of the graph.

Initially, we introduced the notion of a *large weak component* in order to compute its diameter, as an approximation of the diameter of the entire graph. As our research progressed, we discovered the potential value of the *large weak component*. Firstly, we observed that it contains, in almost every dataset the highest percentage of edges with respect to the whole graph and secondly, it contains a set of nodes and edges which survive until the end of each dataset's lifetime. So we computed the graph metrics for it as well, trying to correlate them with the ones of the entire graph.

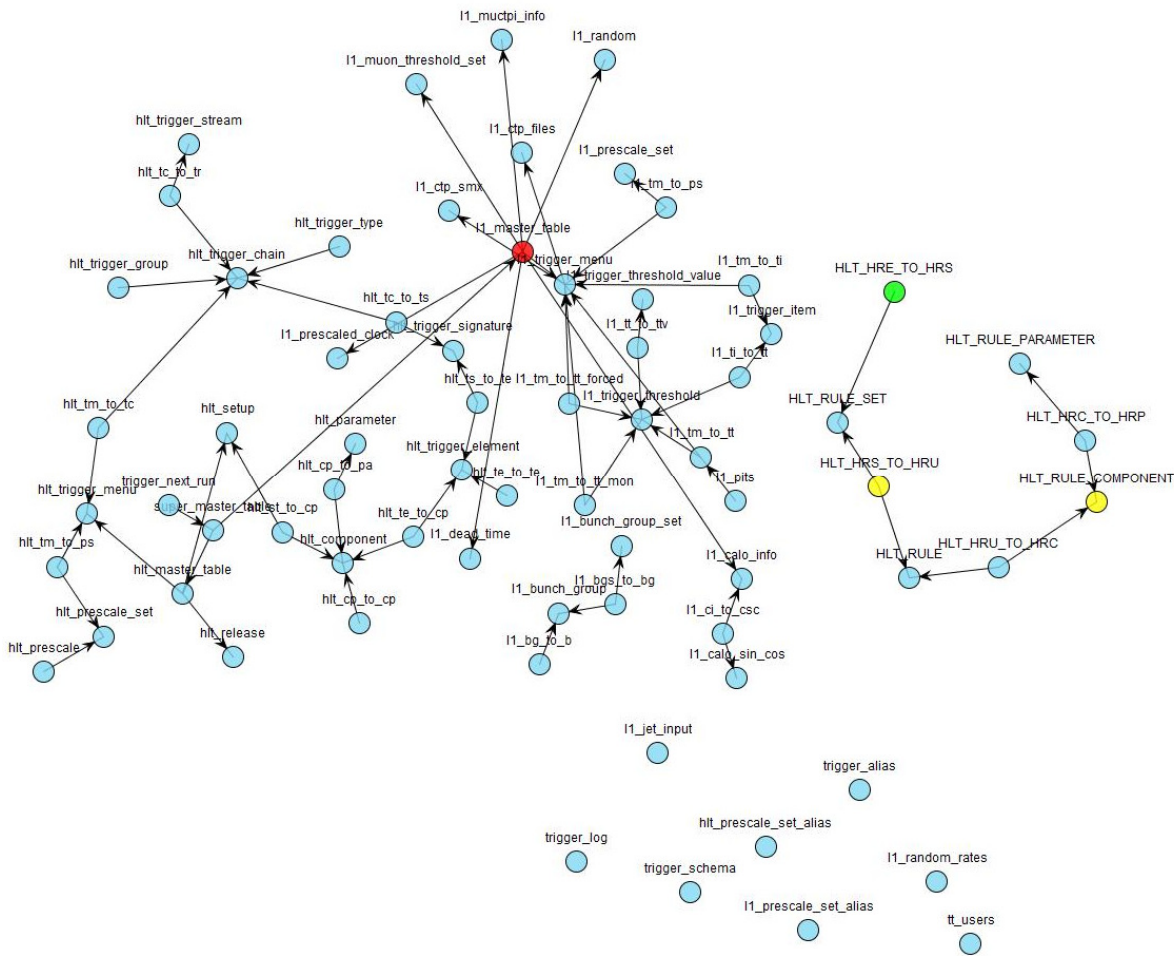


Figure 2 A version of Atlas in Parmenidian Truth

### 3.4 Parmenidian Truth

The tool **Parmenidian Truth** [<https://github.com/DAINTINESS-Group/ParmenidianTruth>] is a tool with the goal of producing the evolution of the schema of a relational database as a movie. Given the history of a database, expressed as a sequence of data definition files, and consequently, a sequence of differences between subsequent versions, Parmenidian truth visualizes each version of the database schema as a graph, with tables as nodes and foreign keys as edges and produces a PowerPoint presentation, with one slide per version (appropriately annotated with color to highlight the tables affected by change). Along with the appropriate visualization provisions, the result is practically a movie on how the schema of the database has evolved.

A fundamental requirement for the smooth advancement of the movie is that tables do not change place in the screen once a new version of the schema is displayed. So, we need to produce a “global” positioning of tables, with each table retaining its coordinates along the entire history of the database. To this end, we introduce the idea of the Diachronic Graph, a graph that encompasses a node for every table that has ever appeared in the history of the database and an edge for every foreign key that has ever appeared in the history of the database. One can think of the Diachronic Graph as the superimposition of all the graphs of the different versions – equivalently, the graph of each version is the projection of the Diachronic Graph for the tables and foreign keys that are present in that particular version.

The tool proceeds as follows:

- The input to Parmenidian Truth is the history of a database's schema, expressed as a sequence of versions as well as the changes that appear during each transition among subsequent versions.
- The tool locates every table and every foreign key that takes part in a database's lifetime in all its versions.
- Then, the tool constructs the Diachronic Graph of the database, which is a graph that contains the union of the database's tables and its foreign keys throughout its entire history
- The tool automatically places the nodes of the diachronic graph in a two dimensional surface; this layout is retained for all versions, and consequently, each table gets fixed coordinates for all the versions of the history.
- For each version of the database schema, we project the graph that corresponds to it as a subgraph of the Diachronic Graph, retaining the

coordinates of the nodes as previously computed and export the result as an image file

- The tool automatically constructs a PowerPoint Presentation, where each version comes with a slide that contains the respective image file
- The PowerPoint Presentation can be converted to wmv and mp4 files if the user wishes
- The history of the database schema as well as the Diachronic Graph are subjects to the extraction of graph-based measures (per version, per table, or overall) that characterize the evolution of both the schema in its entirety and its constituent tables



## CHAPTER 4.

### GRAPH METRICS EVOLUTION

---

#### 4.1 Experimental Setup

#### 4.2 Total number of nodes and edges

#### 4.3 Diameter of Large Weak Component and number of Weak Components

#### 4.4 How do the nodes and edges of Diachronic graph relate to evolution

#### 4.5 Summary of Findings

---

#### 4.1 Experimental Setup

In this section, we provide a brief description and some key statistics for the datasets that we have studied. In the case where preprocessing was needed for a dataset to be imported in our tool, we provide a detailed analysis of all the actions that were taken, for cleansing it.

##### 4.1.1 Atlas Trigger

ATLAS (<https://twiki.cern.ch/twiki/bin/view/Atlas/TriggerDAQ>, <http://atlasexperiment.org/trigger.html>, and [http://atdaq-sw.cern.ch/cgi-bin/viewcvs-atlas.cgi/offline/Trigger/TrigConfiguration/TrigDb/share/sql/combined\\_schema.sql](http://atdaq-sw.cern.ch/cgi-bin/viewcvs-atlas.cgi/offline/Trigger/TrigConfiguration/TrigDb/share/sql/combined_schema.sql)) is a particle physics experiment

at the Large Hadron Collider at CERN, the European Organization for Nuclear Research based on Geneva, Switzerland. ATLAS is notably known for its attempt to find the Higgs boson, although its scientific aims are much broader. Trigger is one of the software modules used in the ATLAS project and it is responsible of filtering the very large amounts of data collected by the Collider and storing them in its database. This database uses the Oracle RDBMS. This dataset has a lifespan of 2 years and it consists of 85 revisions. Atlas started its life with 56 tables and 61 foreign key relationships and ended it with 73 tables (approximately 30% schema growth with respect to its starting size) and 63 foreign key relationships (approximately 3% foreign key growth).

The *diachronic graph* of Atlas consists of 88 tables and 88 edges – foreign key relationships. Unfortunately, although the schema history was publicly available at the time that we performed the data collection, currently, the data are unavailable from their original source.

#### 4.1.2 BioSQL

BioSQL([http://biosql.org/wiki/Main\\_Page](http://biosql.org/wiki/Main_Page) and <https://github.com/biosql/biosql/blob/master/sql/biosql-mysql.sql>) is a generic relational schema with the aim of providing a unified access to data from various sources such as GenBank or Swissport that store genomic data like sequences, features, etc. BioSQL facilitates data storage and interoperability for the different projects of the Open Bioinformatics Foundation (OBF) projects (those include BioPerl, BioPython, BioJava, and BioRuby) that are open source toolkits for the manipulation of these data. The currently supported Relation Database Management Systems (RDBMSs) are MySQL, PostgreSQL, Oracle and SQLite. This dataset has a lifespan of 6.6 years and it consists of 47 revisions. BioSQL started its life with 21 tables and 17 foreign key relationships and ended it with 28 tables (approximately 33% schema growth) and 43 foreign key relationships (approximately 53% foreign key growth).

The *diachronic graph* of BioSQL consists of 45 tables and 79 edges – foreign key relationships. In some schemata of BioSQL whenever a double field was declared it was annotated as *Double Precision*, while our parser expected to read a *double* value. So, the preprocessing that took place, was simply the detection of the keyword *precision* followed after the keyword *double* and whenever detected we erased it.

### 4.1.3 EGEE-II: JRA1 Activity

Egee (<http://egee-jra1.web.cern.ch/egee-jra1>) is the EU-funded project Enabling Grids for E-SciencE, more generally known as EGEE, whose goals was to provide researchers with access to computational Grids. EGEE's goal was to provide researchers in academia and industry with round-the-clock access to major computing resources, independent of geographic location. The infrastructure will support distributed research communities, which share common Grid computing needs and are prepared to integrate their own computing infrastructures and agree on common access policies. This database uses the Oracle and MySQL RDBMS. This dataset has a lifespan of 4 years and consists of 17 revisions. EGEE started its life with 6 tables and 3 foreign key relationships and ended it with 10 tables (approximately 67% schema growth) and 4 foreign key relationships (approximately 33% foreign key growth).

The *diachronic graph* of EGEE consists of 12 tables and 6 edges – foreign key relationships. Like the aforementioned project Atlas, hosted by CERN, direct access to Egee data is no longer available.

### 4.1.4 Castor

CASTOR (<http://castor.web.cern.ch/> previously at <http://castor-obsolete-201310.web.cern.ch/>) stands for the CERN Advanced STORAge manager, is a hierarchical storage management (HSM) system developed at CERN used to store physics production files and user files. Files can be stored, listed, retrieved and remotely accessed using CASTOR command-line tools or user applications that were developed using the CASTOR API. This database uses the Oracle RDBMS. This dataset has a lifespan of 3 years and consists of 194 revisions. Castor started its life with 62 tables and only 6 foreign key relationships and ended it with 74 tables (approximately 20% schema growth) and 10 foreign key relationships (approximately 67% foreign key growth).

The *diachronic graph* of Castor consists of 91 tables and 13 edges – foreign key relationships. In most of Castor's schemata additionally, to declaring and modifying tables as well as, foreign keys, other actions also took place like, building and organizing indexes, declaring and executing stored procedures, creating triggers. However, we only needed to parse the declaration of tables and foreign keys. So, for preprocessing Castor, we built a script containing regular expressions that captivated only the information we needed, and stored it to a new set of schemata forming our new Castor dataset.

### 4.1.5 SlashCode

SlashCode (<http://www.slashcode.com/> and <http://slashcode.git.sourceforge.net/> ) is host of Slash, an architecture for building web sites, most famously known for supporting the Slashdot website. It is written in Perl and built on top of Apache. This database uses the MySQL RDBMS. This dataset has a lifespan of 12.5 years and consists of 399 revisions. SlashCode started its life with 42 tables and 0 foreign key relationships and ended it with 87 tables (approximately 108% schema growth) and 0 foreign key relationships (0% foreign key growth), however during SlashCode's schema evolution foreign key relationships made their appearance, but faded in the end.

The *diachronic graph* of SlashCode consists of 126 tables and 47 edges- foreign key relationships.

### 4.1.6 Zabbix

Zabbix (<http://www.zabbix.com/>) is an open source distributed monitoring solution that can be used for the monitoring of networks, servers and virtual machines. This database uses the MySQL RDBMS. This dataset has a lifespan of 10.8 years and consists of 160 revisions. Zabbix started its life with 15 tables and 10 foreign key relationships and ended it with 48 tables (220% schema growth) and 2 foreign key relationships (80% foreign key loss).

The *diachronic graph* of Zabbix consists of 58 tables and 38 edges – foreign key relationships. For this dataset we used the PostgreSQL version available online. Most of Zabbix's schemata contained some external procedure calls that we needed to remove for parsing it correctly. Additionally, we also corrected minor syntactic failures like missing parenthesis when declaring foreign keys. These actions, are the preprocessing that took place for this dataset.



Dataset	Versions	Lifetime	Tables Start	Tables End	Tables @ Diach.	Table Growth	FKs Start	FKs End	FKs @ Diach.	FK Growth
Atlas	85	2 Y, 7 M	56	73	88	30%	61	63	88	3%
BioSQL	47	6 Y, 7 M	21	28	45	33%	17	43	79	153%
Egee	17	4Y	6	10	12	67%	3	4	6	33%
Castor	194	3Y	62	74	91	20%	6	10	13	67%
SlashCode	399	12 Y, 6 M	42	87	126	108%	0	0	47	-
Zabbix	160	10 Y, 10 M	15	48	58	220%	10	2	38	-80%

Table 1 A description of the datasets we have used in this study

## 4.2 Total number of node and edges

In this section we visualize the evolution of the aforementioned datasets, based on the total number of nodes and edges throughout their entire lifetime.

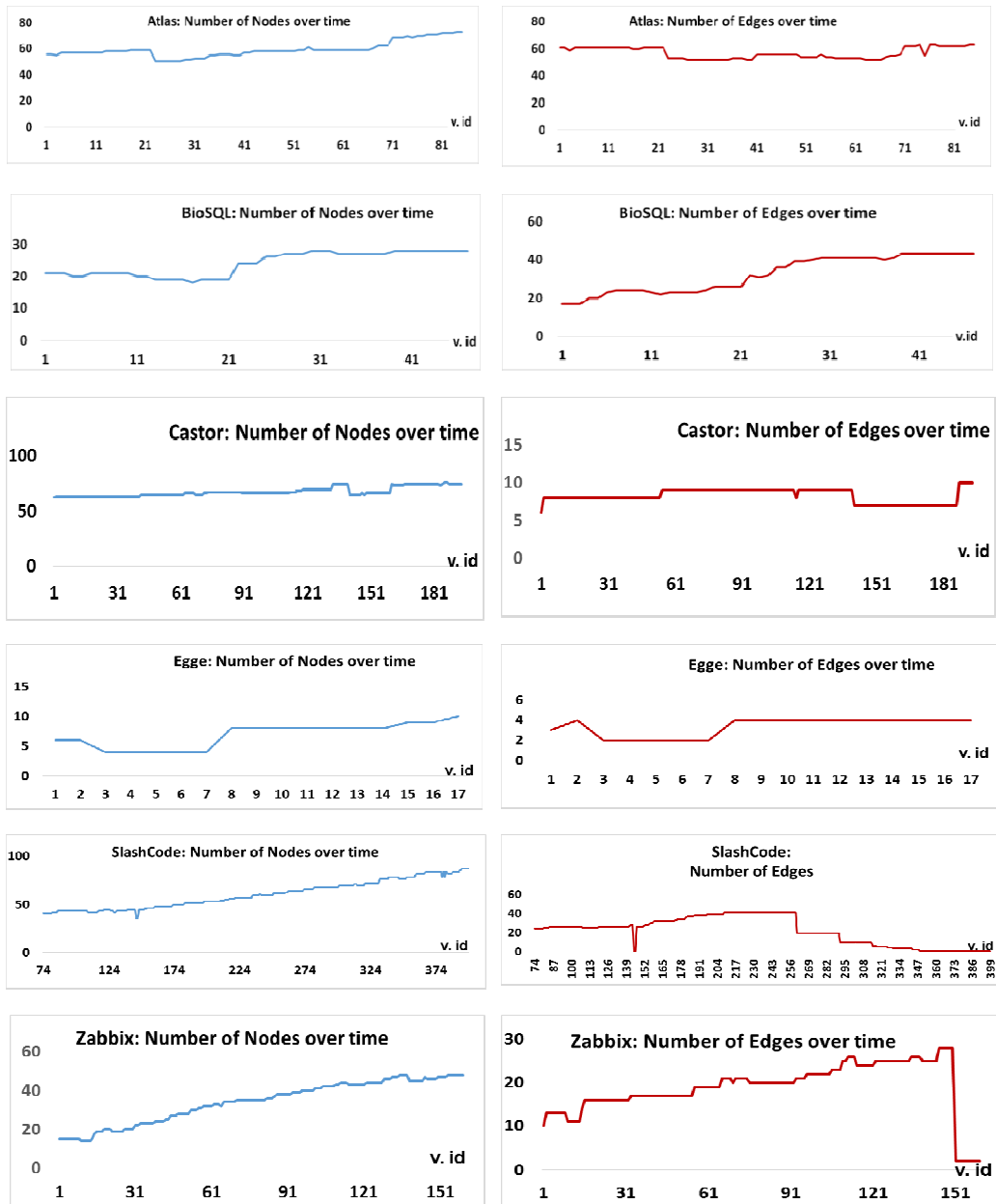


Figure 3 Number of nodes and edges over time for the 6 studied data sets

In the following discussions, we comment on the trends of the number of edges and nodes. Moreover, we computed the Pearson correlation between the evolution of the size of nodes and the evolution of the size of edges for each respective dataset. In the cases where there is a high correlation between them we can continue our research by studying only one of the two. The results are gathered in Table 2.

	Egee	BioSQL	Atlas	Castor	SlashCode	Zabbix
Pearson correlation for Nodes and Edges	94.79%	96%	71.60%	11%	-67%	42%

Table 2 Correlation of nodes and edges for the six datasets

We observe that in the first two datasets, Egee and BioSQL, there is a high correlation between the size of nodes and edges. Observing the trend-lines corresponding to total number of tables and edges respectively, we can clearly see that they follow the same pattern. Meaning that the addition of a table would probably mean the addition of an edge two. So, as far as these two datasets are concerned, there is no need of studying separately, tables and edges, choosing one will suffice.

In the Atlas case, we can only say that there is a medium correlation between the nodes and edges. Their respective trend-lines seem to follow a pattern, this correlation though is not as strong as before.

In the last three datasets the correlation is very small, and in the SlashCode case it is even negative, meaning that the evolution of schema size and the size of foreign key relationships are inversely proportional.

As we can see from the graphical representations of Figure 3, Castor is a relatively quiet dataset. Both of the trend-lines corresponding to schema growth and foreign key relationship growth are almost flat. But during the end of Castor’s evolution we observe that there is a fall in the number of edges while the number of tables remains unaffected. At the end Castor finishes its evolution with 74 tables and only 10 edges. It is noteworthy, that the versions of Castor in Oracle and MySQL never had foreign keys. Observing the whole of Castor’s lifetime, we can conclude that is a database

whose administrators, from the start till the end, do not favor the existence of foreign keys. Thus the correlation between the total number of tables and edges is only 11%.

The Zabbix dataset also has a low correlation between total number of tables and foreign keys. Observing its respective trend-lines we can see that edges were added and removed throughout its lifetime while tables kept rising. Finally, near the end, at version 151, the majority of the edges was removed, while at the same time the total number of tables was hardly affected, ending its evolution with the total of 48 tables and only two edges. Again in this dataset the existence of a high number of foreign keys compared to the total number of tables was not favored. This is fairly obvious, since all the foreign keys are massively deleted.

Observe that in the case of the SlashCode dataset, we started studying it after its 74<sup>th</sup> version. We followed this approach because before that revision no edges were present rendering most of our metrics meaningless. In this thesis we study graph – theoretic properties and a graph without edges can hardly qualify as a graph, thus our approach of studying SlashCode after its edges appear at version 74. Much like the Zabbix case, the trend-line corresponding to the total number of edges contains its ups and downs, comparing to the trend-line corresponding to the total number of tables, which mostly keeps rising. In the same way as in the case of Zabbix, all the edges disappear near the end. Thus SlashCode ends its evolutions with 87 tables and 0 edges. SlashCode is a dataset with 399 revisions, from which, only after the 74<sup>th</sup>, edges make their appearance, and in the end all of them are removed. Once again we can say that foreign key relationships were not favored, justifying the negative Pearson score.

### 4.3 Diameter of Large Weak Component & number of Weak Components

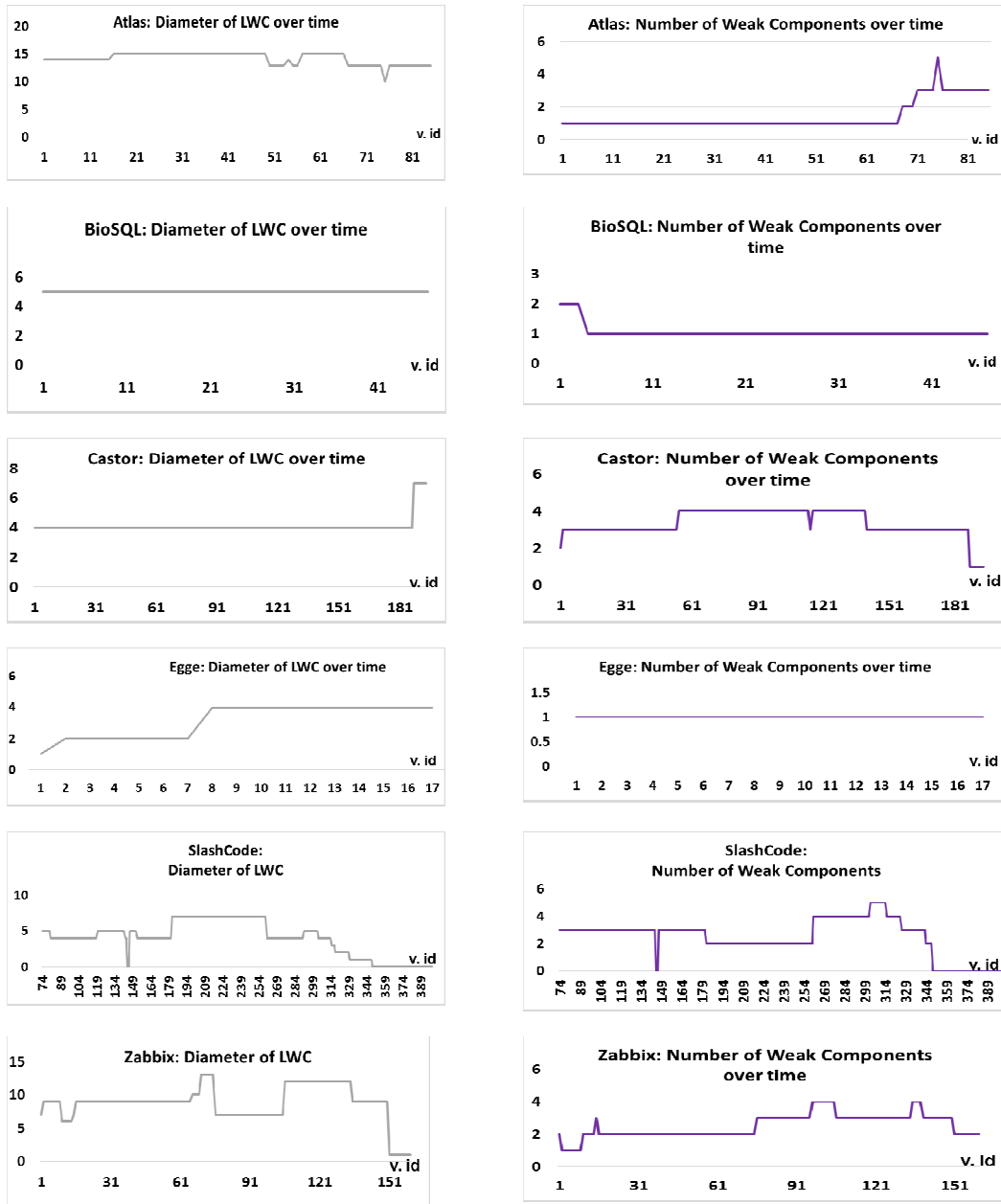


Figure 4 Size of Diameter and Number of Weak Components over time for the 6 studied data sets

In this section we study the evolution of the Diameter of the Large Weak Component, as well as the number of weak components that exist in the database's schema.

The LWC is an important part of our study since we use it to measure some important metrics such as the approximation of the diameter of the graph. As we can see in Figure 4, the diameter of the LWC is fairly stable over time. Atlas BioSQL and Castor2 have pretty much a stable diameter throughout their entire lifetime. Egee reaches this stability soon enough. Lastly, the two CMSs have rather abrupt changes in both their diameter and in their number of weak components. SlashCode loses suddenly all of its edges for just a version making that negative spike, we could argue that this was a wrong commit. But nonetheless SlashCode comes with a period where it progressively loses all of its edges at the end of its lifetime. Zabbix follows pretty much the same pattern of change as SlashCode but this time the changes and the loss of all the edges is abrupt; in any case, Zabbix loses all its edges at the end as well.

Overall, it is interesting to observe that, in the two data sets where the diameter evolved with disruptions, peaks and valleys (as opposed to the smooth progress observed in other data sets), in the end, the data set lost all of its edges. Although we cannot generalize the phenomenon as a rule, it appears that, if foreign keys are not so welcome by the developers who maintain the system, there are early signs of it in the heartbeat of the schema.

	Atlas	Biosql	Castor	Egee	Slash Code	Zabbix
$ V  -  E $	71.6%	96%	11%	94.79%	-67%	42%
$ V  -  C $	80.65%	-18%	-14%	-	-39%	67%
$ E  -  C $	41.60%	-37%	32%	-	21%	60%
$ V  - LWC \delta$	-62.97%	-	43%	82.75%	-60%	-1%
$ E  - LWC \delta$	-35.64%	-	46%	75.61%	92%	73%
$ C  - LWC \delta$	-76.42%	-	-65%	-	39%	9%

Table 3 Pearson correlation for the studied metrics, with  $|V|$  standing for number of nodes,  $|E|$  for number of edges,  $|C|$  for number of weak components and  $\delta$  for diameter

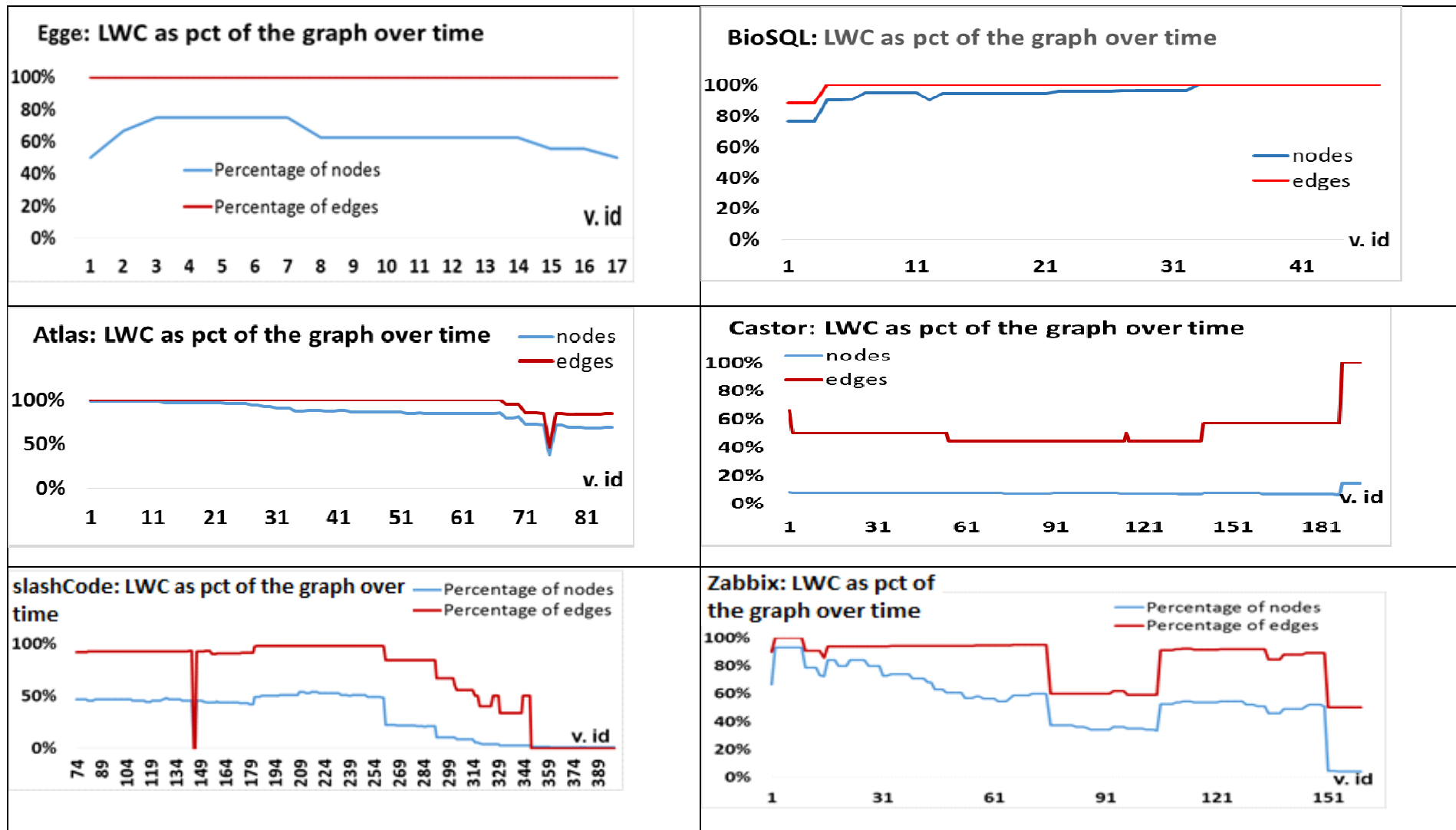


Figure 5 Percentage of Nodes and Edges within the LWC over time for the 6 studied datasets

In Figure 5, the percentage of nodes and edges that is contained in the large weak component is shown. It is interesting to observe that, in almost every dataset that we studied for the purpose of this thesis, the LWC contains the majority of the edges of the entire graph. Moreover, the percentage of the edges contained in the LWC is always higher than the percentage of nodes. This means, that there is only one grand neighborhood where nodes are connected, and not small, isolated neighborhoods of nodes. Our conclusion from these observations is that the foreign keys are either spanning the entire schema (like the cases of Atlas and BioSQL), or at least, a large part of it (Zabbix, Egee and SlashCode), or completely neglected (like the case of Castor.)



#### 4.4 How do the nodes and edges of Diachronic graph relate to the average graph snapshot

As we discussed above, the *diachronic graph* is the union of nodes and edges of each revision of the database’s schema. The main purpose of its creation, is to pinpoint and finalize the coordinates of each table, so that it has fixed coordinates as we render through the schema evolution. In this section however, we try to see how the number of nodes and edges that are contained in the *diachronic graph*, relates to each version’s number of nodes and edges, throughout the database’s evolution.

# Nodes	Atlas	BioSQL	Castor	Egee	SlashCode	Zabbix
<b>D.G.</b>	88	45	91	12	126	58
<b>Average</b>	59.44	23.85	67.19	6.82	56.01	34.07
<b>Max</b>	73	28	76	10	87	48
<b>Min</b>	51	18	62	4	34	14

Table 4 Number of nodes contained in each dataset’s lifetime

In Tables 4 and 5, for each dataset, we report the average, maximum and minimum number of nodes or edges. To compute this value, we take the actual number of nodes and edges, for each version in the history of the data set and we apply the respective aggregate function.

# Edges	Atlas	BioSQL	Castor	Egee	SlashCode	Zabbix
<b>D.G.</b>	88	79	13	6	47	38
<b>Average</b>	56.93	32.23	8.26	3.35	17.51	18.89
<b>Max</b>	63	43	10	4	41	28
<b>Min</b>	52	17	6	2	0	2

Table 5 Number of edges contained in each dataset’s lifetime

Non-surprisingly, the measures of the *diachronic graph* are the highest of all four categories. The *diachronic graph* is the union of all the tables and edges that participate in a database’s schema evolution, as a result it is expected to gather the highest values of total nodes and edges.

By dividing, the contents of all the 3 last lines of Tables 4 and 5 over the respective first line (i.e. the size of the Diachronic Graph), we compute the respective numbers as percentages over the Diachronic Graph. We list the measurements of number of nodes in Table 6 and respective measurement for edges in Table 7.

As we can see in Tables 6 and 7, on average, a version has approximately 60% of the nodes of the diachronic graph and 50% of its edges, although the variation per data set differs a lot. The standard deviation between the individual percentages is around 10% for both edges and nodes. The minimum value has a quite large range for the different data sets and is clearly related with the update rate of a dataset: datasets with low update activity start high and just evolve towards larger values. Datasets with larger rates for table birth and death start from lower values and reach lower maximum values too.

# Nodes as pct over DG	Atlas	BioSQL	Castor	Egee	Slash Code	Zabbix	<i>Avg</i>	<i>stdev</i>
<b>Average</b>	68%	53%	74%	57%	44%	59%	59%	10%
<b>Max</b>	83%	62%	84%	83%	69%	83%	77%	9%
<b>Min</b>	58%	40%	68%	33%	27%	24%	42%	18%

Table 6 Number of nodes as percentage of the nodes of the Diachronic Graph

# Edges as pct over DG	Atlas	BioSQL	Castor	Egee	Slash Code	Zabbix	<i>Avg</i>	<i>stdev</i>
<b>Average</b>	65%	41%	64%	56%	37%	50%	52%	11%
<b>Max</b>	72%	54%	77%	67%	87%	74%	72%	11%
<b>Min</b>	59%	22%	46%	33%	0%	5%	28%	23%

Table 7 Number of edges as percentage of the edges of the Diachronic Graph

## 4.5 Summary of Findings

In this chapter, we have calculated metrics regarding the size and structure of the graph. Our findings can be summarized as follows:

- *Total number of nodes and edges.* In this section we calculated the total number of nodes and edges for each respective dataset. Next, we tried to correlate these two measures in order to see if there is any correlation, or not, between them. It is interesting to see that two datasets share a strong correlation between those measures while a third one shares a weaker correlation. Nonetheless, it is important to note that the former 3 datasets contain a steady number of edges and keep them alive until the end of our observation. Opposing to the latter datasets that their total edge count over time is unstable, and tend to lose all their edges at the end.
- *Diameter* is typically constant with values ranging between 1 and 4.
- *Number of weak components* is typically low between 1 of 3. *The largest weak Component* contains most of the times, more than 60% of the nodes and more than 80% of the edges of the graph.
- *Number of nodes* increases slowly, with periods of calmness
- *Number of edges* increases also, but not fully in sync with *number of nodes*. It depends on the dataset.



## CHAPTER 5.

### EVOLUTION OF TABLE AND FOREIGN KEY METRICS

- 
- 5.1 Simple degrees and their relationship to the table evolution
  - 5.2 Clustering Coefficient and its relationship to table evolution
  - 5.3 Vertex Betweenness Centrality and its relationship to table evolution
  - 5.4 Edge Betweenness and its relationship to schema evolution
  - 5.5 Summary of Findings
- 

#### 5.1 Simple degrees and their relationship to the table evolution

InDegree Breakdown	inDegree @Diach	inDegree @Birth	inDegree @Death	inDegree AVG
Biodatabase	1	1	1	1.00
Term_relationship	1	0	1	0.47
Cache_corba_support	0	0	0	0.00

Table 8 InDegree variants for specific nodes during evolution

We have used our tool *Parmenidian Truth*, to produce the in- and out-degree of each relation, represented as a node, in every version of the database's life. As a result, we came up with the specific measurements of the in/out degrees for each version of the history of the schema, plus one extra value derived from the Diachronic Graph. Apart from these measurements, we have calculated for each table, the degrees at the time of its birth, its last known version and the average value over their lifetime. In Table 8, we indicatively list a few tables with the variants of their *In-Degree* metrics.

The first problem we had to address was the decision on which measurement we could base our analysis on, as, it was possible for us to choose from a variant of values for only a specific metric. Specifically, the involved variants of the in-degree are defined as follows:

- The *inDegree@Diach* is the *In-Degree* value of the respective node as measured in the Diachronic graph.
- The *inDegree@Birth* is the *In-Degree* value of the respective node as measured in the first revision of the database's schema.
- The *inDegree@Death* is the *In-Degree* value of the respective node as measured in the final version of the database history for survivors and the time of death for deleted tables.
- Lastly, the *inDegree AVG* is the average *In-Degree* value of the respective node throughout the database's lifetime. It is important to note that whenever a node was absent in a revision, that revision was not taken into account when computing the average.

Given the above possibilities, we resorted to the average value of each degree as the most representative value since it takes into consideration the entire lifetime of the node. In other words, every version in which the node under examination is alive, contributes equivalently instead of assigning a distinct and arbitrary value of a single version as in the cases of Birth, Death, or even in Diachronic.

### 5.1.1 Statistical profile for tables with respect to graph properties

Firstly, we list the number of tables per value of the *in and outDegree AVG* metric Tables 9 and 10, and provide their bar-chart. Then, we study their joint distribution for all the datasets and depict it in Table 11.

<b>inDegree AVG</b>	<b>Egee</b>	<b>BioSQL</b>	<b>Atlas</b>	<b>Castor</b>	<b>SlashCode</b>	<b>Zabbix</b>
0	8	30	48	81	114	42
1	3	6	20	8	7	7
2	1	2	11	1	2	6
3	-	2	5	1	-	2
4	-	2	-	-	1	-
5	-	-	2	-	-	1
6	-	-	1	-	-	-
7	-	-	-	-	1	-
8	-	1	-	-	-	-
9	-	1	-	-	-	-
10	-	-	-	-	-	-
11	-	-	-	-	-	-
12	-	-	-	-	-	-
13	-	-	-	-	-	-
14	-	-	-	-	-	-
15	-	1	-	-	-	-
16	-	-	-	-	-	-
17	-	-	-	-	-	-
18	-	-	-	-	-	-

Table 9 *InDegree* Breakdown for the 6 studied datasets. Each cell represents how many tables of the database have the respective average value (rounded) of the first column.

outDegree AVG	Egee	BioSQL	Atlas	Castor	SlashCode	Zabbix
0	7	7	43	83	95	32
1	4	14	14	4	25	19
2	1	22	28	-	6	7
3	-	2	1	-	-	-
4	-	-	1	4	-	-
5	-	-	-	-	-	-
6	-	-	-	-	-	-
7	-	-	-	-	-	-
8	-	-	1	-	-	-

Table 10 *OutDegree* Breakdown for the 6 studied datasets. Each cell represents how many tables of the database have the respective average value (rounded) of the first column.



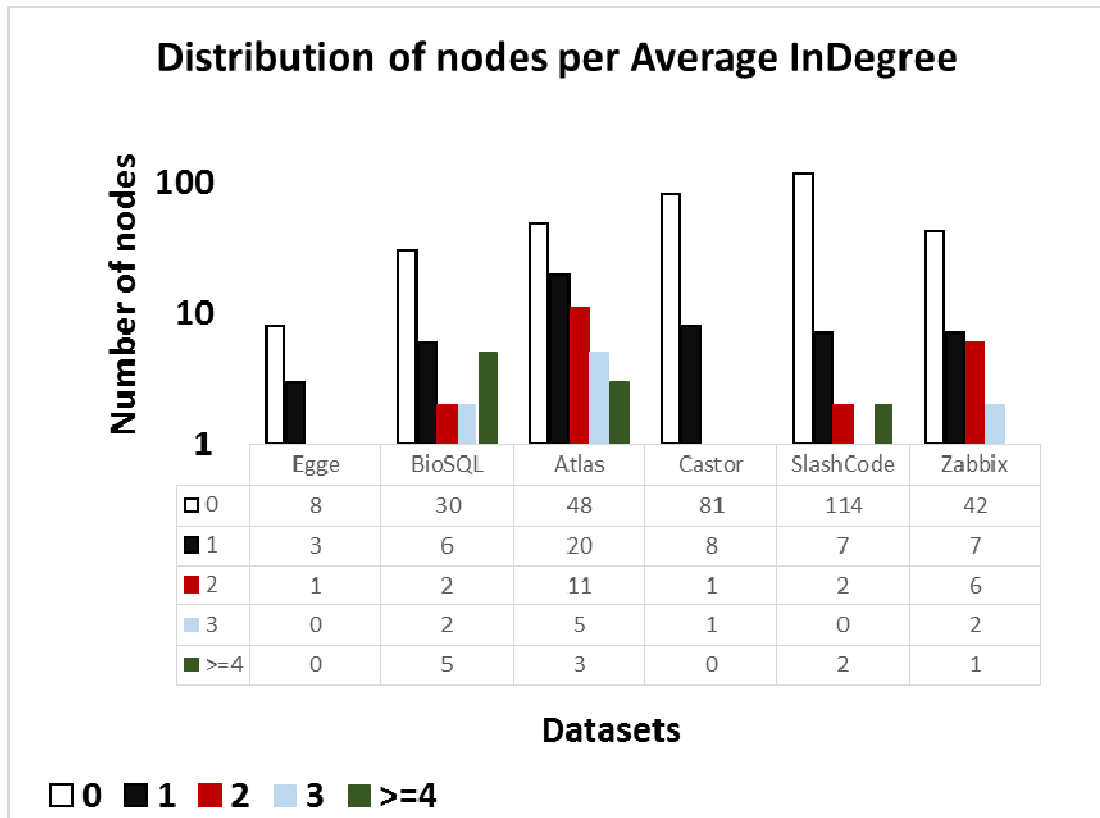


Figure 6 Node Breakdown per Average InDegree for the 6 studied datasets

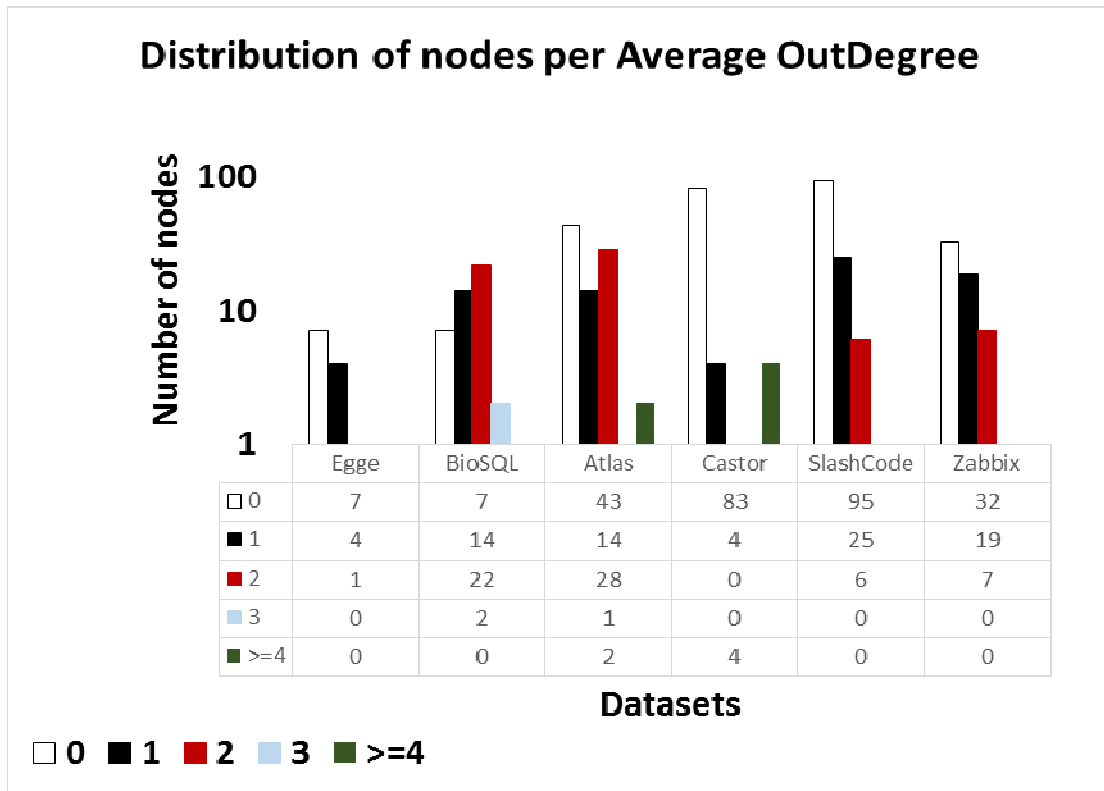


Figure 7 Node Breakdown per Average OutDegree for the 6 studied datasets

InDegree	outDegree	Egee	BioSQL	Atlas	Castor	SlashCode	Zabbix
0	0	6	2	11	75	90	23
0	1	1	9	11	2	22	12
0	2	1	19	25	4	2	7
0	3	-	0	1	-	-	-
1	0	1	3	15	6	3	4
1	1	2	1	1	2	2	3
1	2	-	1	2	-	2	-
1	3	-	1	-	-	-	-
1	4	-	-	1	-	-	-
1	8	-	-	1	-	-	-
2	0	-	-	10	1	-	4
2	1	1	1	1	-	-	2
2	2	-	1	-	-	2	-
3	0	-	1	5	1	1	1
3	1	-	1	-	-	-	1
4	0	-	1	1	-	-	-
4	1	-	-	-	-	1	-
4	3	-	1	-	-	-	-
5	0	-	-	1	-	-	-
5	1	-	-	1	-	-	1
6	2	-	-	1	-	-	-
7	0	-	-	-	-	1	-
8	2	-	1	-	-	-	-
9	1	-	1	-	-	-	-
15	1	-	1	-	-	-	-

Table 11 Joint Distribution for the average in/out degree for the 6 studied datasets.

The rows are ordered by in- and out-degree value. Each cell represents how many tables of the database have the respective combination of values of the first two

columns. Note that these numbers refer to the tables of the entire lifetime of each dataset.

Apart from the descriptive statistical analysis of the two degrees in question, we also assess the breakdown of tables with respect to both of them. In Table 12, we provide an aggregated overview for the breakdown of tables and the absence of edges for all the 6 datasets of our study. It is noteworthy, that from all the datasets that we have studied, only BioSQL and Atlas have a low number of nodes that are not incident to an edge. In two cases, Castor and SlashCode, the vast majority of tables are actually without edges and in another two, Egee and Zabbix, tables without edges are the largest group.

In-degree	Out-degree	Slash					
		Egee	BioSQL	Atlas	Castor	Code	Zabbix
0	0	50%	4%	12.5%	82%	71%	40%
≠0	0	8%	11%	36%	9%	4%	16%
0	≠0	17%	62%	42%	7%	19%	33%
≠0	≠0	25%	22%	9%	2%	6%	12%

Table 12 Breakdown of node percentages per combination of degrees for the 6 studied datasets.

More observations from the above numbers include:

- A small percentage of nodes serve as a bridge (i.e. tables with both incoming and outgoing edges) between nodes.
- The number of nodes classified as sinks (nodes that have incoming edges of foreign keys as lookups but no outgoing one) is smaller than the number of nodes classified as fountains (nodes that reference other tables as foreign keys but do not receive any edge themselves).

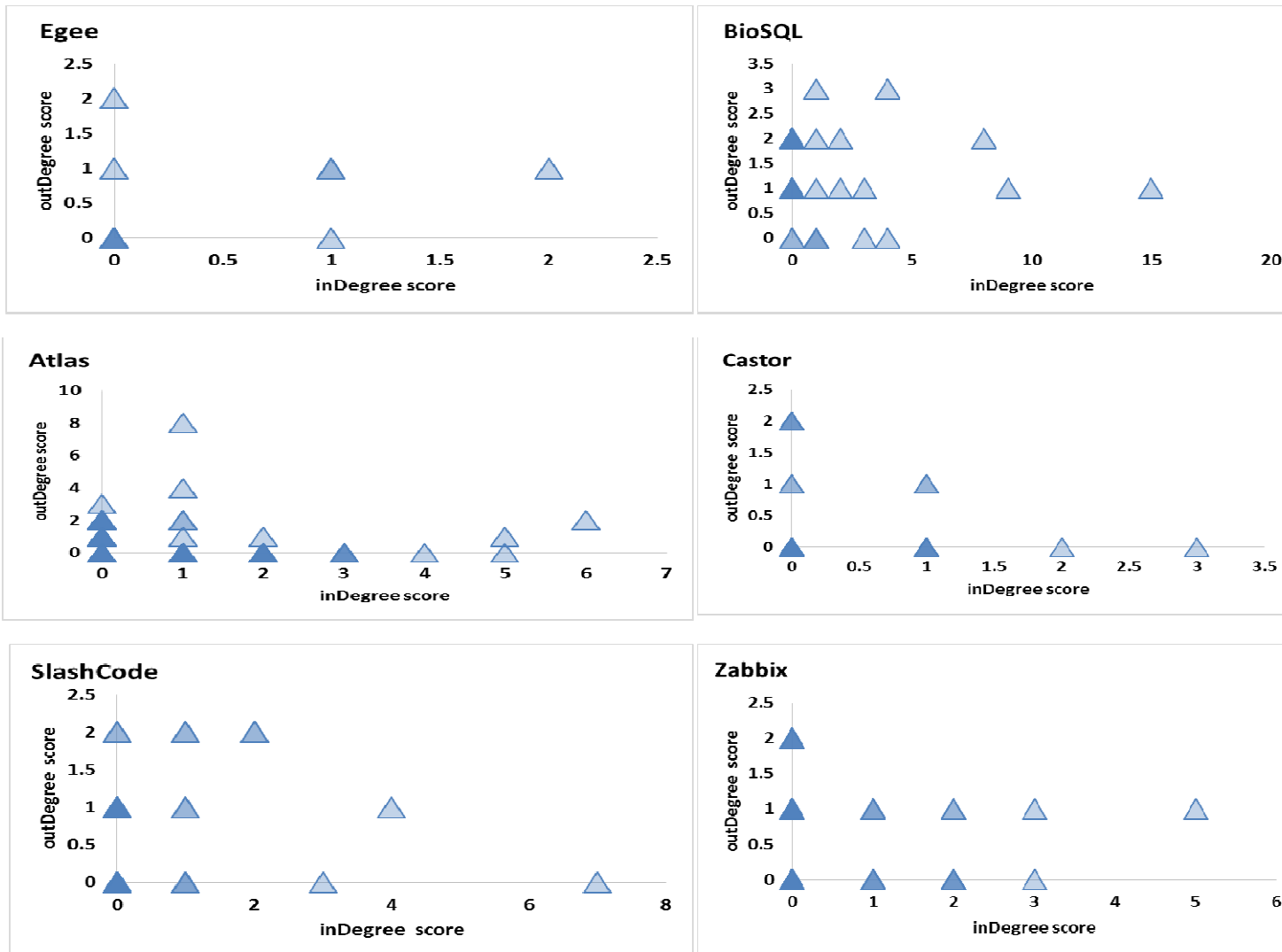


Figure 8 Distribution of nodes with respect to both their In and Out Degree scores for the 6 studied datasets

In Fig. 8, we use scatterplots to help us comprehend how in- and out- degree scores are distributed in each of the 6 studied datasets. The horizontal axis of each scatterplot represents the in-degree of a table and the vertical axis of the scatterplot represents the out-degree of a table. Each point in the scatterplot corresponds to a table. We use transparency in the coloring of the triangles representing tables; therefore, intense color in a particular triangle signifies high concentration of tables in the same  $(x,y)$  point of the chart, placed one on top of the other. As Figure 8 demonstrates, the majority of nodes are concentrated to low in- and out- degree scores representing the lookup and the fact tables respectively.

### 5.1.2 How simple degrees relate to the evolution of tables

In this section, we provide a deeper investigation of the relationship of the in- and out- degree metrics with the survival of tables.

The following research question concerns whether the different degrees of a node can predict its survival. To assess how survival is related to degree in the graph, we correlated the two measures; the result is visualized it in Figures 9 and 10. In the scatter plots of the both figures, the horizontal axis depicts the last known version of a table (thus, survivors are placed at the rightmost part of the figure, and all the other tables are dead tables). The vertical axis depicts the in and out degree of the nodes. A vertical red line discriminates the survivors from the dead tables.

The scatterplots provide two observations:

- With few exceptions, the higher the degree of a node is the more possible is, for the node to survive,
- In all 6 datasets, there is an *inverse-gamma pattern* in the correlation of degree and survival. The pattern suggests that low degrees carry a non-negligible probability of removal for the corresponding tables; at the same time, high degrees carry a significantly lower probability of removal.

Again, due to the use of transparency in the coloring of triangles, intense coloring means high concentration of triangles, overlaid one on top of the other, at the same  $(x,y)$  point in the chart.

	#nodes	#nodes with deg>2	#nodes survivor	# nodes survivor && deg>2	pct nodes with deg>2	pct nodes survivor	P(survivor   deg>2)	P(deg>2   survivor)
Egee	12	1	10	1	8%	83%	100%	10%
BioSQL	45	11	28	8	24%	62%	73%	29%
Atlas	88	15	73	14	17%	82%	87%	18%
Castor	91	1	74	1	1%	81%	100%	1.3%
SlashCode	126	7	87	6	6%	69%	86%	6.7%
Zabbix	58	5	48	5	9%	82%	100%	11%

Table 13 Probability of survival with respect to total degree

In Table 13 we can see that nodes with a degree higher than 2 have significantly higher chances of survival than the average probability of survival for the entire dataset

Specifically, the column  $P(\text{survivor} | \text{deg} > 2)$  improves the survival rate contrasted to column  $\text{pct nodes survivor}$  with improvements in the area of 17%-19% with the exception of Atlas (only 5%) and BioSQL (11%). In 3 cases, the survival rate reaches 100%. Unfortunately, the result involves a small amount of tables in the datasets (11% - 24% in column  $\text{pct nodes with deg} > 2$ ) and thus the predictive ability of this pattern is constrained to a small percentage of cases.

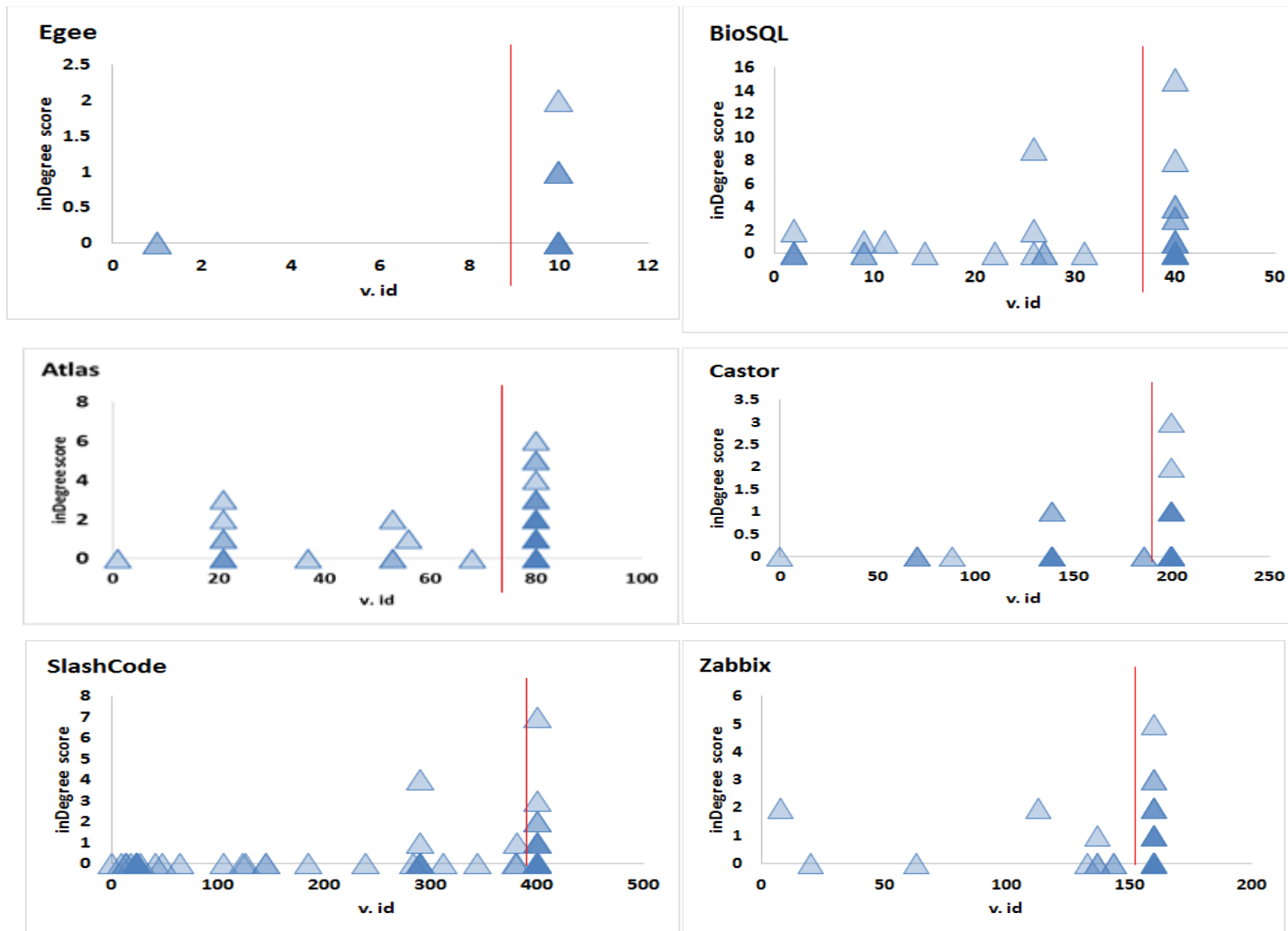


Figure 9 Contrasting InDegree scores with the last known appearance for the tables of the 6 studied datasets



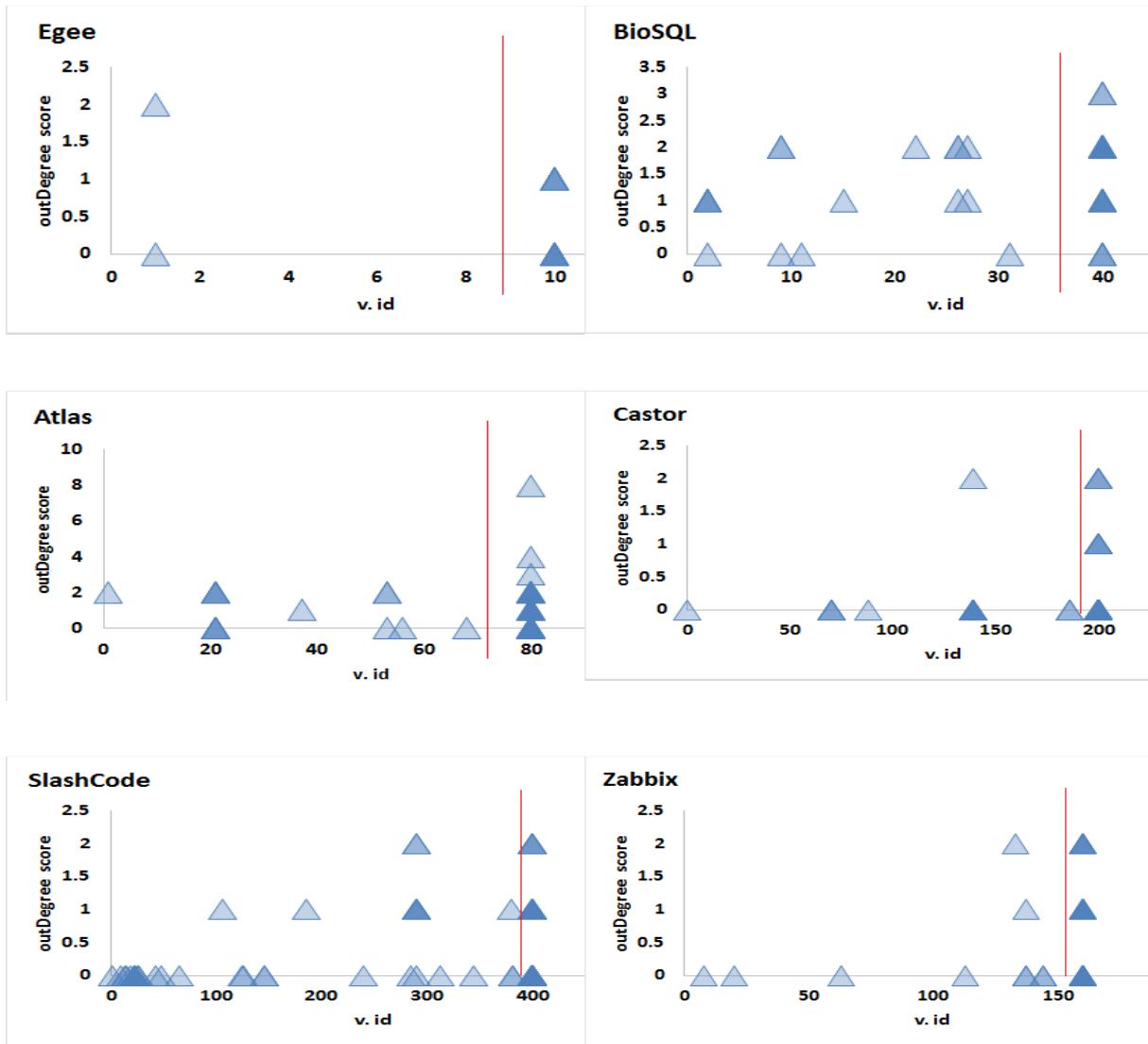


Figure 10 Contrasting OutDegree scores with the last known appearance for the tables of the 6 studied datasets

## 5.2 Clustering Coefficient and its relationship to table evolution

Apart from the effect of a local measure of graph topology, like the different kinds of degrees, we wanted to assess the effect of neighborhood-based measures to the behavior of nodes. To this end, we have assessed the profile of clustering coefficient, presented in this subsection, and betweenness centrality, presented in the following subsection. Clustering Coefficient denotes the “tight coupling” of neighbors in the neighborhood of a node (see Chapter 3 for the formal definition).

### 5.2.1 Statistical profile for tables with respect to clustering coefficient

We have assessed the clustering coefficient for each node, in each version of the schema where it was present. Then, we averaged this measurement to obtain the *average clustering coefficient* of a node. During the computation of the average value per node, we skipped the schema versions where the node was not present. Table 14 lists the aggregated results for each dataset. Below in Figure 11, we provide the bar-chart of the clustering coefficient scores for the 6 studied datasets

Clustering Coefficient	Egee	BioSQL	Atlas	Castor	SlashCode	Zabbix
0	9	29	79	91	115	55
(0,0.1)	1	1	3	-	4	2
[0.1,0.2)	1	3	2	-	2	-
[0.2,0.3)	-	2	-	-	3	-
[0.3,0.4)	-	1	-	-	-	-
[0.4,0.5)	-	-	-	-	1	-
[0.5,0.6)	-	1	1	-	-	-
[0.6,0.7)	-	1	-	-	-	-
[0.7,0.8)	-	-	-	-	1	-
[0.8,0.9)	-	2	-	-	-	-
[0.9,1)	-	-	-	-	-	-
1	1	5	1	-	-	1

Table 14 Clustering Coefficient Breakdown for the 6 studied datasets

As we can see in Table 14 and Figure 11 the clustering coefficient of the 6 studied datasets is very low. However, this seems to be an expected result considering that we study the field of databases, where two nodes pointing to a third does not necessarily mean that they are connected. Again, due to the use of transparency in the coloring of triangles, intense coloring means high concentration of triangles, overlaid one on top of the other, at the same (x,y) point in the chart.

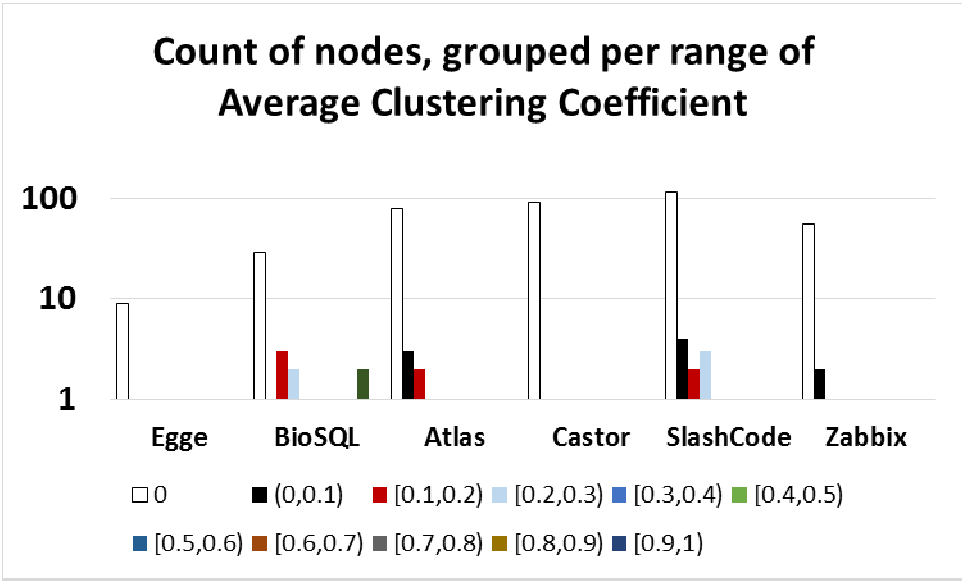


Figure 11 Node Breakdown per Average Clust. Coeff. for the 6 studied datasets

## 5.2.2 How clustering coefficient relates to the evolution of tables

Having computed the average clustering-coefficient score for each table during its lifetime, the natural research question was to evaluate if this metric can predict whether or not a node will survive. So we correlated survival and clustering coefficient, and visualize the results in Figure 12. Again, the horizontal axis signifies the last known version of a table. Tables that reach the last version are survivors and are separated from the rest of the data set via a red vertical line.

Apparently, this method seems that cannot very well predict the survivability of a node. However, it is important to highlight that in cases where the database's schema is a graph rich in edges, like the Atlas or BioSQL dataset this method performs much better as opposed to the other datasets.

Again observe, that the datasets with a satisfactory number of edges tend to follow the aforementioned *inverse-gamma pattern*.

	#nodes	#nodes with cc>0	#nodes survivor	# nodes survivor && cc>0	pct nodes with cc>0	pct nodes survivor	P(survivor   cc>0)	P(cc>0   survivor)
Egee	12	3	8	2	25%	66%	67%	25%
BioSQL	45	16	28	12	35%	62%	75%	43%
Atlas	88	9	73	8	10%	82%	89%	11%
Castor	91	0	74	0	0%	81%	0%	0%
SlashCode	126	11	87	8	8%	69%	73%	9.2%
Zabbix	58	3	48	2	5%	82%	67%	4.2%

Table 15 Probability of survival with respect to clustering coefficient

In Table 15, we complement this original visualization with concrete number. This improvement in the survival rate for nodes with high clustering coefficient (cc>0) compared to the average survival rate is insignificant; in one case, Zabbix the survival rate is even lower than average (but we deal with a very small number of involved tables)

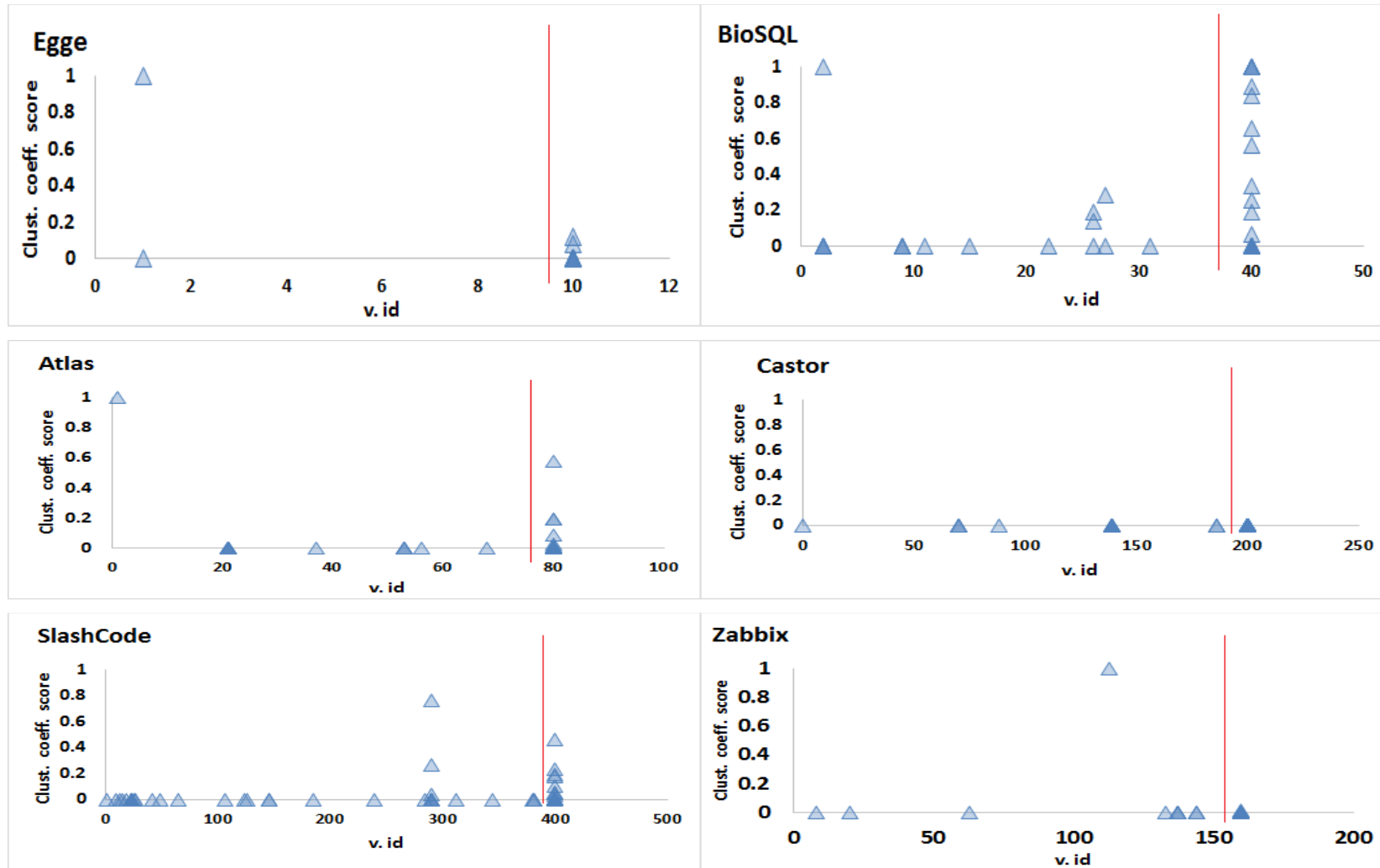


Figure 12 Contrasting Clustering Coefficient scores with the last known appearance for the tables of the 6 studied datasets

### 5.3 Vertex Betweenness Centrality and its relationship to table evolution

Apart from the clustering coefficient, Vertex Betweenness Centrality is another measure of topology characterizing a node in the graph of a schema. Whereas clustering coefficient refers to the local behavior of the neighborhood of a node, its betweenness centrality measures its position in the entire graph, by assessing the number of shortest paths that pass from it.

#### 5.3.1 Statistical profile for tables with respect to vertex betweenness

We have assessed Betweenness Centrality for every node in every version of the graph. Then, we calculated the Average Vertex Betweenness score. We provide the score breakdown as well as the corresponding barchart in Table 16 and Figure 13 respectively

Average Vertex Betweenness	Egee	BioSQL	Atlas	Castor	SlashCode	Zabbix
0	9	37	81	89	120	51
(0,5)	3	3	2	2	2	7
[5,10)	0	2	3	0	3	0
[10,15)	0	1	1	0	1	0
[15,20)	0	1	1	0	0	0
[20,25)	0	1	0	0	0	0

Table 16 Average Vertex Betweenness Breakdown for the 6 studied datasets

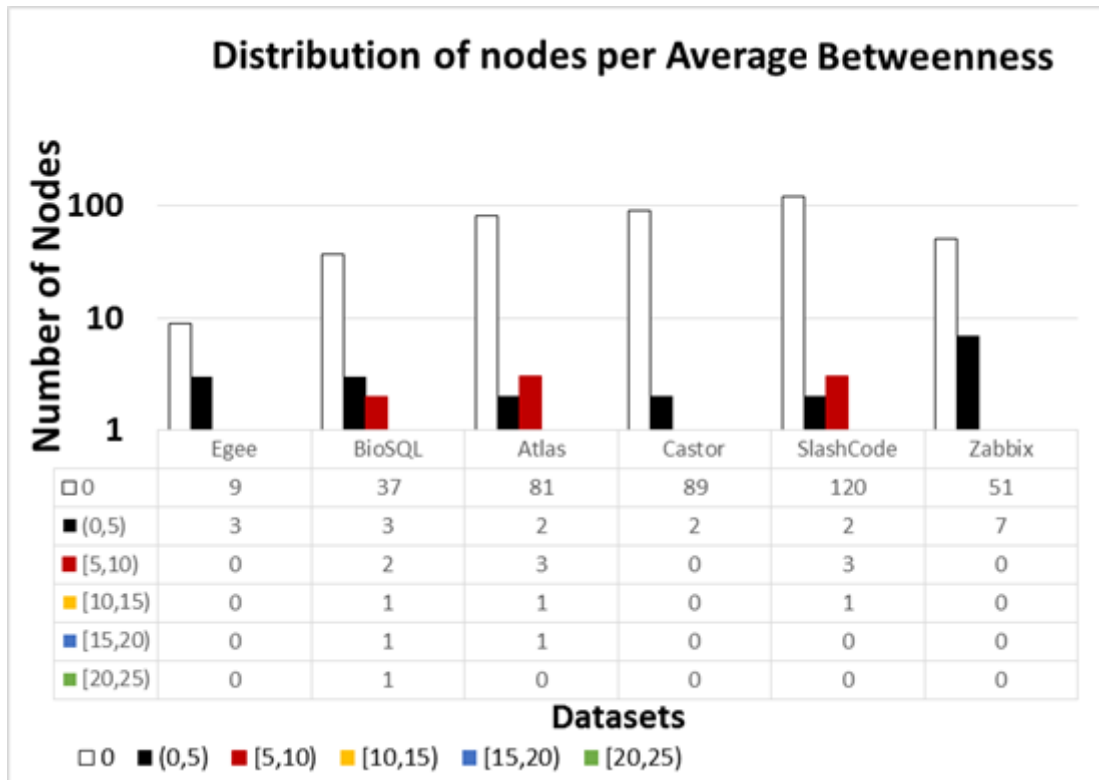


Figure 13 Breakdown per Average Node Betweenness Centrality for the 6 studied datasets

### 5.3.2 How does vertex betweenness relate to table evolution

In this section we are trying to find if we can use the Average Vertex Betweenness in order to predict if a vertex is a survivor.

In Figure 14 we provide the result for the 6 studied datasets.

	#nodes	#nodes with AVBC>=1	#nodes survivor	# nodes survivor && AVB>=1	pct nodes with >=1	pct nodes AVBCsurvivor	P(survivor   AVBC >=1)	P(AVBC >1   survivor)
Egee	12	3	10	3	25%	83%	100%	30%
BioSQL	45	6	28	4	13%	62%	66%	14.2%
Atlas	88	6	73	6	6%	82%	100%	8%
Castor	91	0	74	0	0%	81%	-	0%
SlashCode	126	4	87	3	3%	69%	75%	3%
Zabbix	58	2	48	2	12%	82%	100%	14.5%

Table 17 Probability of survival with respect to avg. vertex betweenness



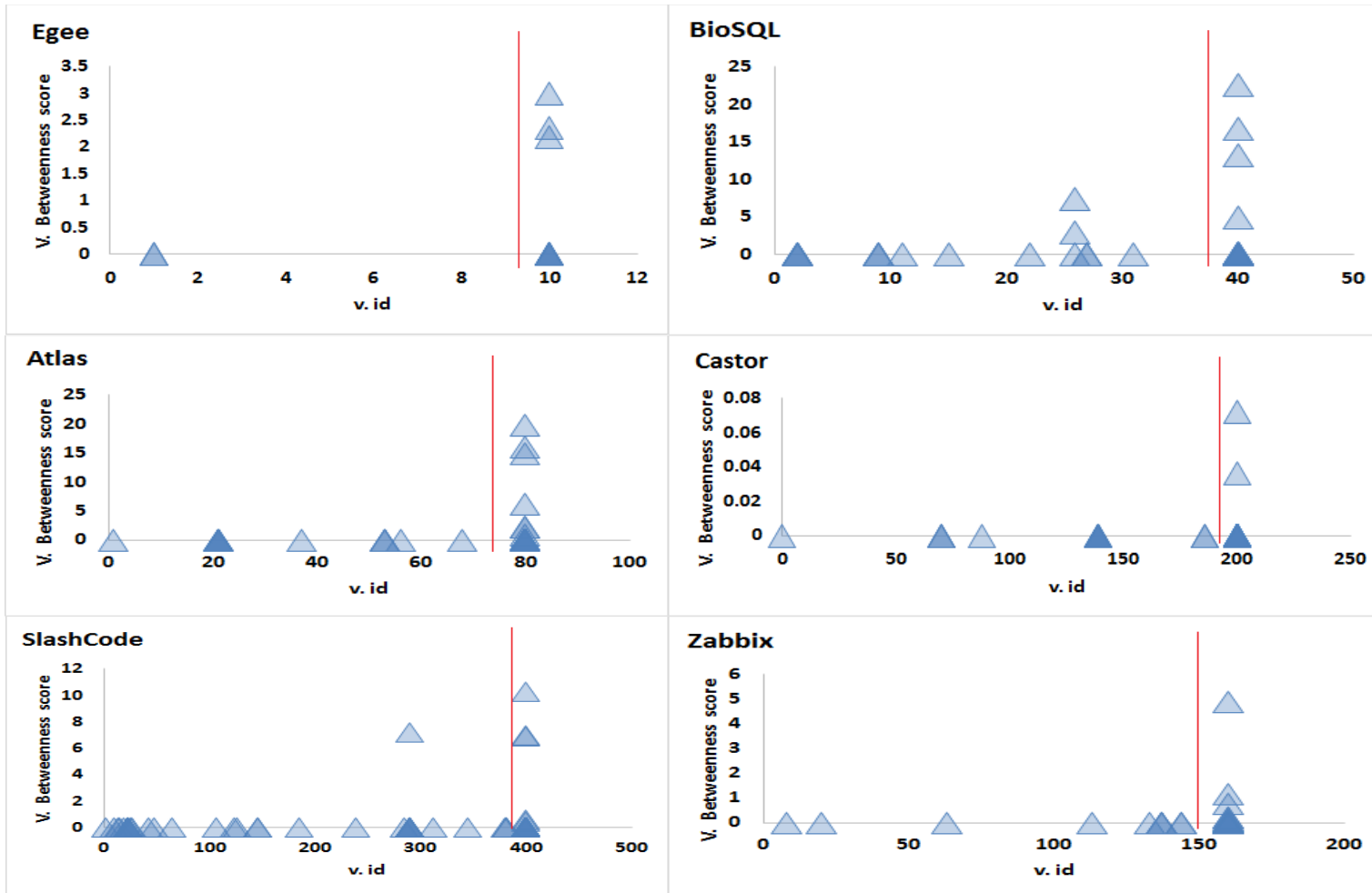


Figure 14 Contrasting Average Vertex Betweenness scores with the last known appearance for the tables of the 6 studied datasets

How does Average Vertex Betweenness Centrality relate to survival? In a systematic attempt, we assessed the correlation of betweenness centrality and survival and the results are depicted in the Table 18. Specifically, we evaluated whether a vertex with an Average Vertex Betweenness Centrality score (in short, AVBC) greater or equal to 1 is a survivor (by computing the fraction of survivor nodes with  $AVBC \geq 1$  over the population of nodes with  $AVBC \geq 1$ ), as well as what percentage of survivors do nodes with high Average Vertex Betweenness constitute (as the fraction of survivor nodes with  $AVBC \geq 1$  over the population of survivors).

Can we use Average Vertex Betweenness Centrality as a predictor of survival? Observe that this method has extremely high predicting precision in all 6 datasets except from the cases of BioSQL and SlashCode. Specifically, in 4 out of 6 data set, an  $AVBC \geq 1$  signified a 100% probability of survival. At the same time, survival can be related to other parameters too. This is why, the percentage of high centrality nodes among survivors is too small. *So, overall, although Average Vertex Betweenness Centrality cannot predict death or survival in the general case, it can predict survival for the few cases where its value is relatively high.*

Some specific findings follow.

- After careful examination in the BioSQL dataset we found a very interesting result. The two vertices that (a)  $AVBC \geq 1$  and (b) did not survive, turned out to be simple renames and not actual deletions of the corresponding tables, meaning that, in essence, our method also had a 100% precision as well, in the BioSQL dataset.
- In the case of the SlashCode dataset, there was a single non-survivor table with  $AVBC \geq 1$ . This table was indeed deleted, but its neighborhood was deleted too. This means that it was not some random deletion, but in fact this vertex was important as with its neighborhood, they had some functionality that was deemed unnecessary by the database architect. So, this does not justify why AVBC did not achieve a 100% precision but it does point out that the specific dataset underwent to a radical change in its evolution.

### 5.3.3 Normalized Vertex Betweenness and its relationship to evolution

To facilitate a homogeneous comparison of the individual results of the different data sets, we normalize the Vertex Betweenness Centrality by dividing the Vertex Betweenness Centrality with the total number of possible paths between nodes.

**Definition:** Given a graph of  $N$  nodes, the Normalized Average Vertex Betweenness Centrality of a node is its Average Vertex Betweenness Centrality divided by  $N \cdot (N - 1)$ .

	#nodes	#nodes with NAVBC>0	#nodes survivor	# nodes survivor && NAVBC>0	pct nodes with NAVBC>0	pct nodes survivor	P(survivor   NAVBC >0)	P(NAVBC >0   survivor)
Egee	12	3	10	3	25%	83%	100%	30%
BioSQL	45	8	28	4	17%	62%	50%	14.2%
Atlas	88	7	73	3	7%	82%	42%	4%
Castor	91	2	74	2	2%	81%	100%	2%
SlashCode	126	6	87	5	4%	69%	83%	5.7%
Zabbix	58	7	48	7	12%	82%	100%	14.5%

Table 18 Probability of survival with respect to normalized avg. vertex betweenness

It is interesting to note that NAVBC performs much differently than AVBC. To be specific for the Atlas and BioSQL datasets, datasets that are rich in edges, the probability to survive provided that you have non zero NAVB drops drastically while for the rest of the datasets it skyrockets. The reason for the discrepancy with the results of the previous subsection is due to the higher numbers of tables fulfilling the filtering criterion of  $NAVBC > 0$ .

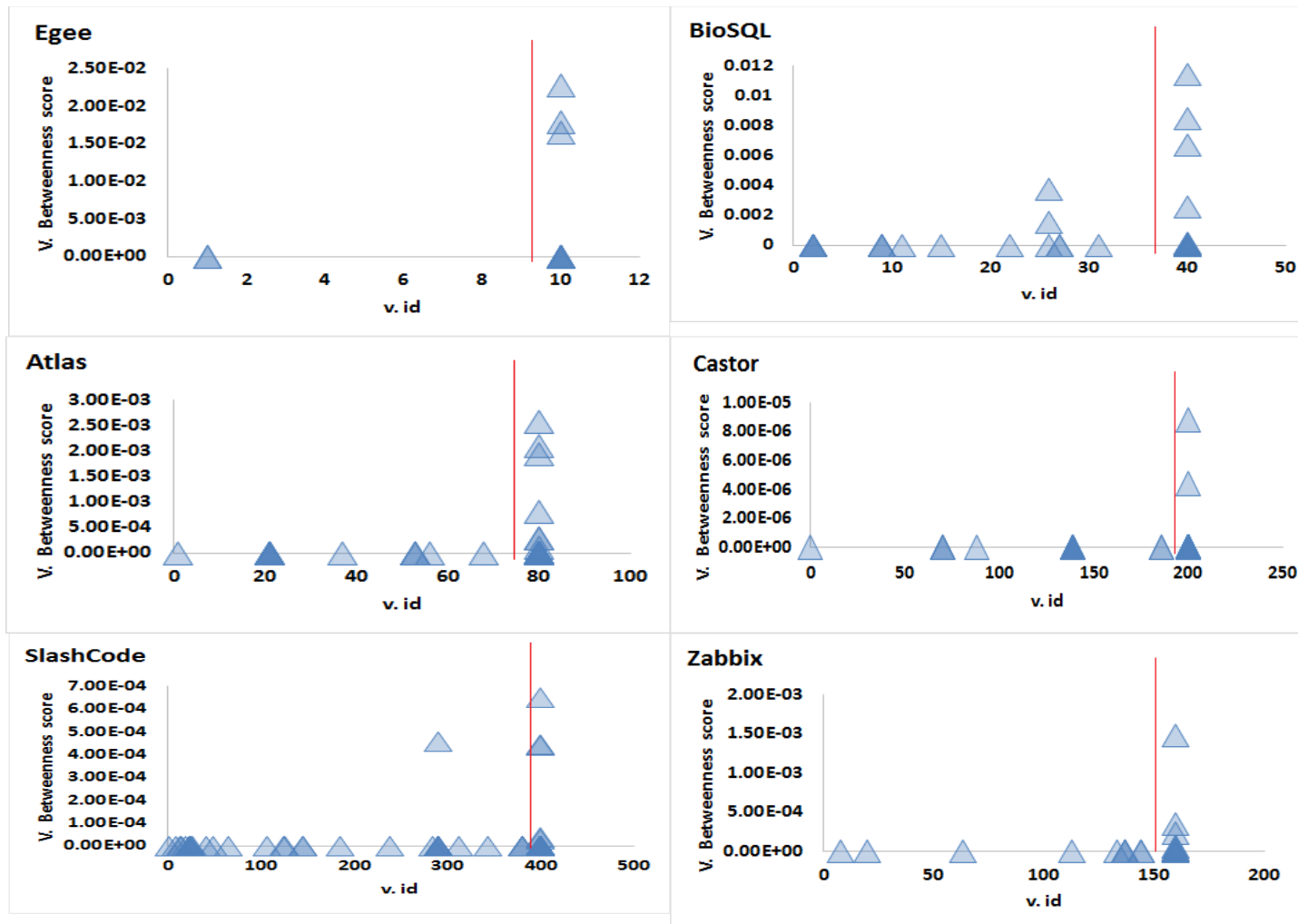


Figure 15 Contrasting Norm. Avg. Vertex Betweenness scores with the last known appearance for the tables of the 6 studied datasets

## 5.4 Edge Betweenness and its relationship to schema evolution

Having a massive graph that is the union of all the tables and edges of the schema evolution of a database, we thought, *what is the most important part of this graph?*

To answer this question, we computed the Average Edge Betweenness Centrality score for each edge and then we kept the edges with the top-2 scores, ties included. We call the resulting subgraph that is produced by these edges, the *2 - Core Component* of the graph.

**Definition:** The *2 - Core Component* is the subgraph that is induced by<sup>3</sup> the top 2 edges of the graph with respect to their Average Edge Betweenness. If two or more edges are tied for the top scores, these edges are as well included in the *2-Core Component*.

As a result, it is reasonable for a top 2-Core Component to include more than 2 edges. Also note, that in four out of six datasets the 2-Core Component was a connected subgraph. This is not a prerequisite condition, rather a fact that seems to occur.

Following a more strict and rigorous study, we have generalized our search to fully test the effect of Edge Betweenness Centrality to the survival of the involved nodes. We proceeded as follows:

- First, we measured the Average Edge Betweenness Centrality for all edges
- We kept the top – 2, including edges that are tied

Interestingly enough we found out that most of the times, the 2 Core Component survives till the end. In Fig., 16 we show the 2 - core component for all six datasets. In each of these figures we depict (a) the 2-Core component over the Diachronic Graph with different coloring, (b) the first version of the graph, and, (c) the last version of the graph. Again, the attempt is to show the survival of the 2-CC in the history of the schema. In 5 out of 6 datasets, the 2-core component is a connected (undirected) subgraph of the Diachronic Graph.

---

<sup>3</sup> Remember that an edge-induced subgraph is a subset of the edges of a graph together with any vertices that are their endpoints (definition by <http://mathworld.wolfram.com/Edge-InducedSubgraph.html>).

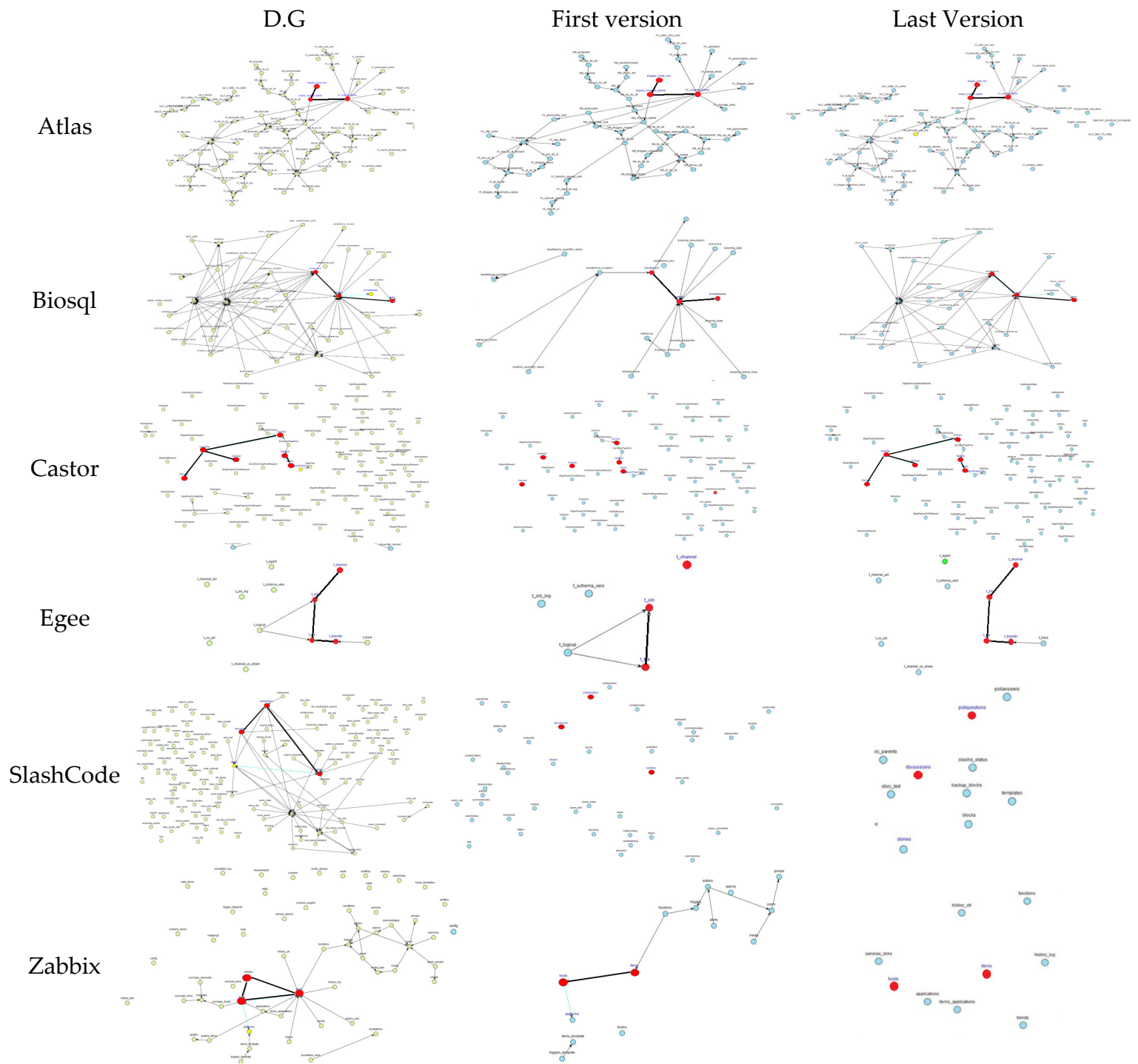


Figure 16 Evolution of the 2-Core Components for the 6 studied datasets (figures are partially cropped to fit)

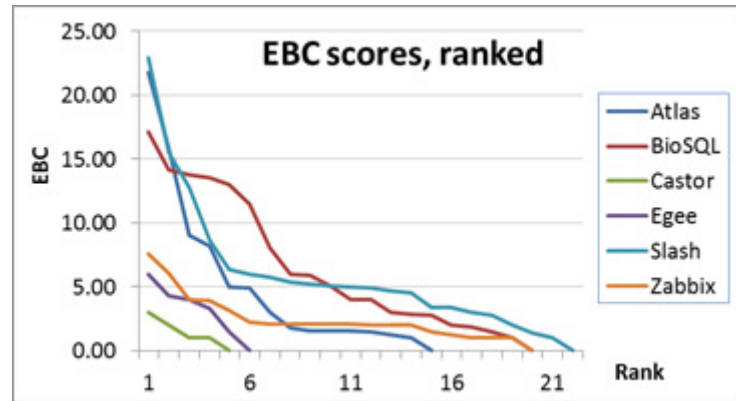


Figure 17 Average Edge Betweenness scores, ordered decreasingly, for all 6 data sets.

				#tables with EBC >1			#tables with EBC = 1			#tables with EBC= 0			Probability to survive			
	Survivors	Dead	Total	Survivors	Dead	Total	Survivors	Dead	Total	Survivors	Dead	Total	Total	EBC>1	EBC=1	EBC=0
Atlas	73	15	88	33	1	34	30	13	43	10	1	11	83%	97%	70%	91%
BioSQL	28	17	45	26	13	39	2	2	4	0	2	2	62%	67%	50%	0%
Castor	74	17	91	7	0	7	6	3	9	61	14	75	81%	100%	67%	81%
Egee	10	2	12	5	1	6	0	0	0	5	1	6	83%	83%	-	83%
Slashcode	87	39	126	14	9	23	11	2	13	62	28	90	69%	61%	85%	69%
Zabbix	48	10	58	21	3	24	9	2	11	18	5	23	83%	88%	82%	78%

Table 19 Breakdown of tables per category of Average EBC score and relationship to survival.

The distribution of values for the Average Edge Betweenness is depicted in Figure 17. The vertical axis depicts the individual Average Edge Betweenness scores appearing in a data set and the horizontal axis its rank, in increasing order. The scores start in many cases from very high values (above 15) and quickly drop in middle range values (around 5) with heavy tails between 2 and 1.

The situation looks quite different, however, if one takes the population size that pertains to each EBC. As already mentioned in the commentary of vertex degrees, a very large percentage of tables have zero incident edges, and quite a few of them have exactly one incident edge. This separates these two particular values for the rest of the (very broad) range of values. We performed a statistical analysis of the respective values, which we depict in Table 19, and –much to our amazement– the results that we found are as follows:

- *Large Average EBC scores (greater than 1 that is) do not guarantee survival more than the average value of the data set.* Note that although in the scientific databases (castor excluded) the percentage of this group is large, it is only in 2 out of 4 cases that survival is very high compared to average.
- The most surprising fact for us was that *the group with Average EBC equal to 1 has higher chances to die than average!* This appears counter intuitive in a sense, as even a single relationship seems a good as to give second thoughts to the DBA before a removal of a table.
- The behavior of the group with EBC equal to 0 is quite close to the overall average, which we attribute to its vast size that gravitates the overall average towards its behavior. Expectedly, survival rate is lower than the one of the group with EBC >1 in all but one occasions.



Rank	Atlas	BioSQL	Castor	Egee	Slashcode	Zabbix	count	survival pct >90%	Pct of >90% survivors
1	100%	100%	100%	100%	50%	50%	6	4	67%
2	100%	100%	100%	100%	100%	100%	6	6	100%
3	100%	100%	100%	100%	100%	100%	6	6	100%
4	100%	100%	67%	100%	0%	0%	6	3	50%
5	100%	0%	81%	0%	100%	100%	6	3	50%
6	100%	0%		83%	0%	100%	5	2	40%
7	94%	100%			0%	100%	4	3	75%
8	100%	100%			100%	100%	4	4	100%
9	100%	0%			100%	100%	4	3	75%
10	100%	0%			0%	100%	4	2	50%
11	100%	100%			0%	100%	4	3	75%
12	100%	100%			100%	100%	4	4	100%
13	100%	75%			0%	100%	4	2	50%
14	70%	100%			0%	100%	4	2	50%
15	91%	100%			100%	100%	4	4	100%
16		42%			0%	100%	3	1	33%
17		0%			100%	100%	3	2	67%
18		100%			100%	0%	3	2	67%
19		50%			100%	82%	3	1	33%
20		0%			100%	78%	3	1	33%
21					85%		1	0	0%
22					69%		1	0	0%

Table 20 Percentage of survivor tables per rank for all the studied data sets.

We decided to elaborate more on the correlation of ranking in terms of Average Edge Betweenness and survival and used the *k-Core Component* analysis to this end.

So, we generalized the definition for larger values of the rank, and thus we define the general version of the Core Component, the *k-Core Component*.

**Definition:** The  $k$  - Core Component, is the subgraph that is induced by the top  $k$  edges of the graph with respect to their Edge Betweenness in the Diachronic Graph.

In Table 20, the first column depicts the rank of an Average EBC score (not shown here), the next 6 columns correspond to the 6 datasets, and the last 3 columns count (a) how many of the datasets actually have even one table for the rank of the respective row ("rank") (b) the number of datasets where the survival rate exceeds 90% ("survival percentage >90%") and (c) the percentage of case (b) over case (a)

The ranks with the higher probability of death are the two last (which are the values 1 and 0 for EBC, demarcated with a mauve background). For the rest of the ranks, in all data sets except for BioSQL and SlashCode, which are the two data sets with higher deletion rates among the ones that we study (both with survival rate lower than 70%), one can easily observe that the value 100% survival is overwhelmingly present, with very few exceptions.

Apparently, to a large extent, the update profile of a schema is responsible for the prediction accuracy of the EBC score and this is why, our two deletion-prone data sets seem to have many "holes" in the otherwise perfect survival rate of higher scores of Table 20 at ranks higher than the two last.

Moreover, we observe that for the first 3 rows of Table 20 (i.e. for the 3 - Core Component), we can get very good results in terms of predictor power for the survival of the tables. In fact, with the exception of Zabbix and SlashCode, the two CMS's, the top scores would achieve perfect survival.

## 5.5 Summary of Findings

In this chapter we have focused on metrics that describe the relationship between nodes and how they are structured inside the graph, aiming to find any hidden survival mechanisms that reside within. In other words, we focus on metrics regarding the individual nodes rather than the whole graph. To this end, we correlate the results of these metrics with evolution-specific events -or in our case, the death of a node.

- *In/out/total Degree.* After calculating the in/out degree for each version of each dataset we discover that from the 6 studied datasets, only BioSQL and Atlas have a low number of nodes that are not incident to an edge. Egee and Zabbix have the vast majority of tables without edges, and in the cases of Castor and SlashCode tables without edges are the largest group. Next, we correlate the in and out degree with each nodes' survivability, and highlight two observations:
  - With few exceptions, the higher the degree of a node is, the more probable is for the node to survive
  - In all 6 datasets, there is an inverse-gamma pattern in the correlation of degree and survival. The pattern suggests that low degrees carry a non-negligible probability of removal for the corresponding tables; at the same time, high degrees carry a significantly lower probability of removal.
- *Clustering Coefficient.* After calculating the Average Clustering Coefficient for each node in each dataset we discovered that the vast majority of the nodes has clustering coefficient score equal to zero, and only a few in the BioSQL dataset have score higher than 0.1. Noteworthy is the case of BioSQL which contains 5 nodes with clustering coefficient equal to 1 and Egee, Atlas, and, Zabbix that contain only one node with clustering coefficient score equal to 1. These results point that the nodes of these datasets are very weakly coupled, and therefore their respective structure is loose. Next, we tested if the clustering coefficient is correlated to survivability, and our conclusions are:
  - This method cannot predict very well the survivability of a node. However, it is important to highlight that in cases where the database's schema is a graph rich in edges, like the Atlas or BioSQL dataset, this method performs much better as opposed to the other datasets.
  - The datasets with a satisfactory number of edges tend to follow the aforementioned inverse-gamma pattern.

- *Vertex Betweenness.* We performed the same procedure as before, for the Average vertex betweenness score this time. In the datasets of Egee and Zabbix every node that possessed average vertex betweenness above zero managed to survive, while this probability for the SlashCode dataset drops to 83%. It is noteworthy, that the survival probability for the BioSQL and Atlas datasets is 37% and 43% respectively. Lastly, the Probability of survival due to Average Vertex Betweenness for Castor is zero. This is expected as the dataset has a really low number of edges.
- *The k-Core Component.* Lastly, we calculated the Edge betweenness for every edge of each of the 6 studied datasets and kept the graph induced by the edges with the top-k scores. We named the induced subgraph as *k - Core Component*. Our observations:
  - In cases of Atlas, BioSQL, Egee, Zabbix and SlashCode the induced subgraph of the 2 – Core Component is connected, except for the cases of Castor. This is very important since it points to the fact that Important/Centralized nodes tend to be inter-connected in the database’s structure, thus creating the Core of the graph.
  - Nodes connected in the Core Component have a very high survival probability. Another observation that strengthens the importance of the core component –The core is important, thus survives till the end.

## CHAPTER 6.

### SOFTWARE ARCHITECTURE AND DESIGN –

### PARMENIDIAN TRUTH

---

6.1 Package diagram

6.2 The core package

6.3 The export package

6.4 The model package

6.5 The gui package

6.6 The model.Loader package

---

#### 6.1 Package Diagram

We start with the high-level architecture of the tool, expressed as a package diagram that is depicted in Figure 18.

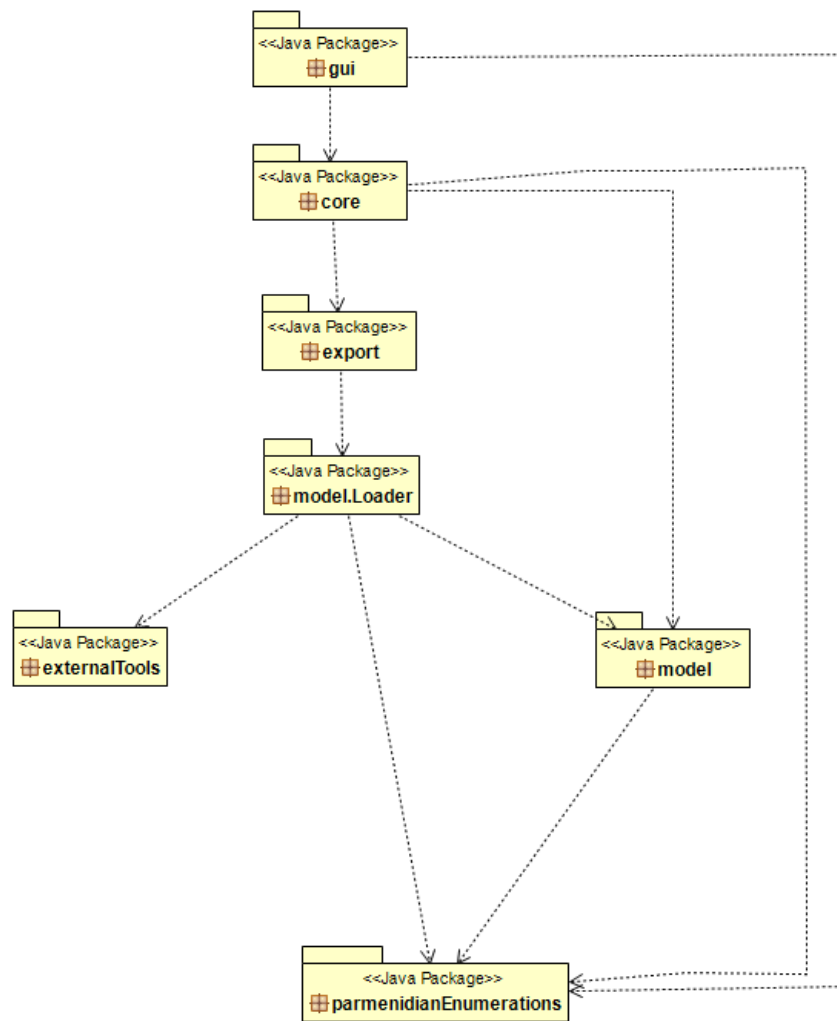


Figure 18 Package Diagram for Parmenidian Truth

- The package `core` consists of the main manager classes that orchestrate every use-case provided by Parmenidian truth
- The `export` package consists of classes concerning the exported files of our tool (.csv, .jpg, .ppt)
- The `model` package consists of classes that basically organize the data in memory in a structured form
- The `gui` package includes all classes that refer to graphical interface notions.
- The `model.Loader` package includes classes that, with the help of the `externalTools` package, are responsible for parsing data files and organizing the created objects into memory

- The externalTools package are the Hecate main engine files that are responsible for parsing the SQL files as well as creating the XML transition files. It is the deepest level as far as parsing goes.
- The parmenidianEnumerations package consists of just enumerations. Their purpose is to make easier the communication between classes. Those enumerations exist for the sake of easy maintenance and code readability.

## 6.2 The core package



Figure 19 Class Diagram for the core package

The core package consists of the main manager classes that orchestrate every use-case provided by Parmenidian Truth.

These are the use cases that are provided by our tool:

- The user can choose his own workspace, a personal folder where every project produced by our tool will be archived
- The user can create a new Parmenidian project, for the creation of such a project the user needs to provide the following input:
  - A folder with all the SQL schema versions of the database under study
  - An XML file that contains the transition changes between two consecutive schemata
  - An output folder where the result files will be exported from our tool
- The user can edit a Parmenidian project
- The user can load a Parmenidian project. When a project is loaded, our tool makes a graphical representation of the database as a graph, with tables as nodes and foreign key relations as edges between nodes. The graph is automatically laid out, but it is in the user's ability to improve it
- When the user is done making graph changes he can save the layout. A graphml file is then produced by our tool for persisting changes
- The user can batch produce a png file for each separate version of the database's schemata, and then combine them in a PowerPoint presentation and even in a video file. Those actions can be taken either individually, or sequentially per user's choice
- The user has the ability to create a transition file for a whole dataset of SQL schema versions. This action can be taken either during the creation of a Parmenidian project or at any other time per user's choice.
- Finally, the user can batch compute a collection of graph metrics for the purpose of studying the schema evolution of the database. The result will be saved in csv files that will be exported in the user's selected folder for output. Those metrics can be separated in the following categories:
  - Node-wise
  - Edge-wise
  - Graph-wise

Each of the above uses-cases can be selected from the user through our graphical interface. Then the core manager `ParmenidianTruthManager` will be responsible for calling the corresponding methods in either the `ModelManager` or the `ExportManager`.



The `ModelManager` interacts with components that reside in the `model` package. Basically, it is responsible for visualizing the schema as a graph, editing and saving the changes performed by the user, as well as, computing graph metrics.

The `ExportManager` interacts with the `export` package, and is basically, responsible for all the output files of our tool.

### 6.3 The export package

As already mentioned, the `export` package consists of classes concerning the files that are exported as output from our tool. Specifically:

The `HecateScript` class, given the path of a folder containing SQL files with the versions of the schema of the database, produces an XML file with the changes between two consecutive schema versions. The `SQLFileFilter` is an auxiliary class for the `HecateScript` to ignore everything but SQL files.

The `PowerPointGenerator` is the class responsible for producing a PowerPoint presentation of the database's schema evolution. For the production it requires a folder containing each schema's graphical representation, which then binds into a PowerPoint slide.

Respectively, the `VideoGenerator` class is responsible for the production of the video of the database's schema evolution. It requires a PowerPoint presentation file, which then decomposes, rendering each slide as a png and then combines every extracted png into a video stream. The `FilenameSorter`, `FilenameSorter2` and `ImageFileFilter` are auxiliary subclasses for sorting the produced png files in the appropriate way.

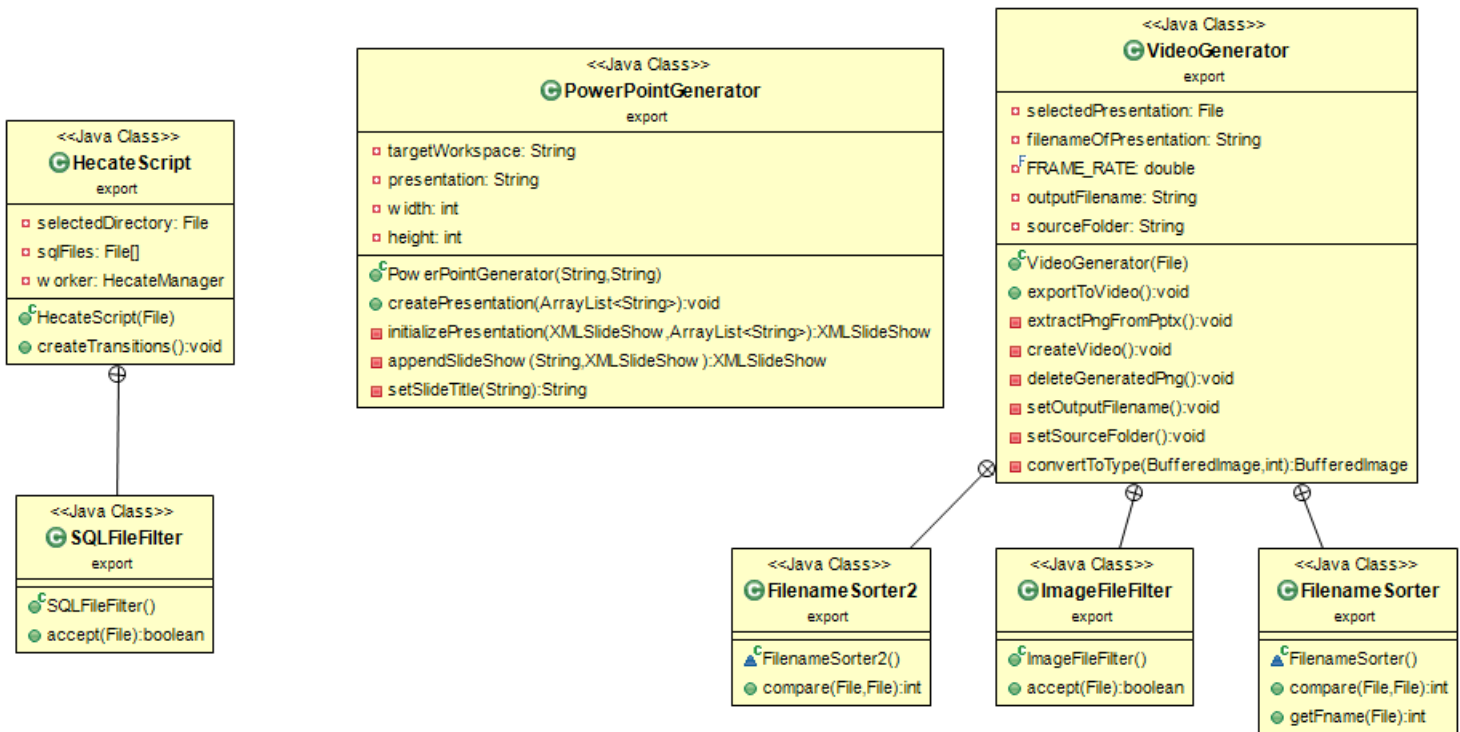


Figure 20 Class Diagram for the export package



The `model` package consists of classes that basically organize the data in memory in a structured form.

The `DiachronicGraph` class is the main class in the `model` package. It is responsible for micromanaging the individual parts of the schema's evolution. At this point we would like to remind the reader that as a notion the *diachronic graph* is the union of table and edges of the whole database's lifetime. So, in design level it contains a collection for tables and versions respectively. All operations regarding manipulating the graph, or computing metrics are done through this class.

The `DiachronicGraphVisualRepresentation` is an auxiliary class introduced through the *Single Responsibility Principle*. It is a class that is responsible for visualizing to the user the *diachronic graph* of the database, as well as, rendering the *diachronic graph* into png format. As so, it contains collections for all the nodes and edges of the schema's evolution and all the improvements that were done from the user layout-wise.

The `DBVersion` class is a memory representation of a physical schema. It contains a collection of tables and foreign keys.

The `DBVersionVisualRepresentation` class is the equivalent of the `DiachronicGraphVisualRepresentation` but for `DBVersion` objects. As such, it contains information about each version's visualization process, and is also responsible for rendering each version in png format.

The `GraphMetrics` class is responsible for computing the selected graph metrics. Inside this class resides the implementation of each metric. `DiachronicGraph` and `DBVersion` both contain an aggregation of this class as a field. The `DiachronicGraph` triggers the computation of a, user selected, metric and then it is performed for each object individually.

The `Table` class is a mapping for the input data that concern schema tables. Other than the schema table's name the `Table` class contains information about the table's coordinates as well as the table's status, something that changes from version to version.

Finally, the `ForeignKey` class is equivalently a mapping for the input data that refer to schema foreign keys. `ForeignKey` objects are visually represented as graph edges.

## 6.5 The gui package

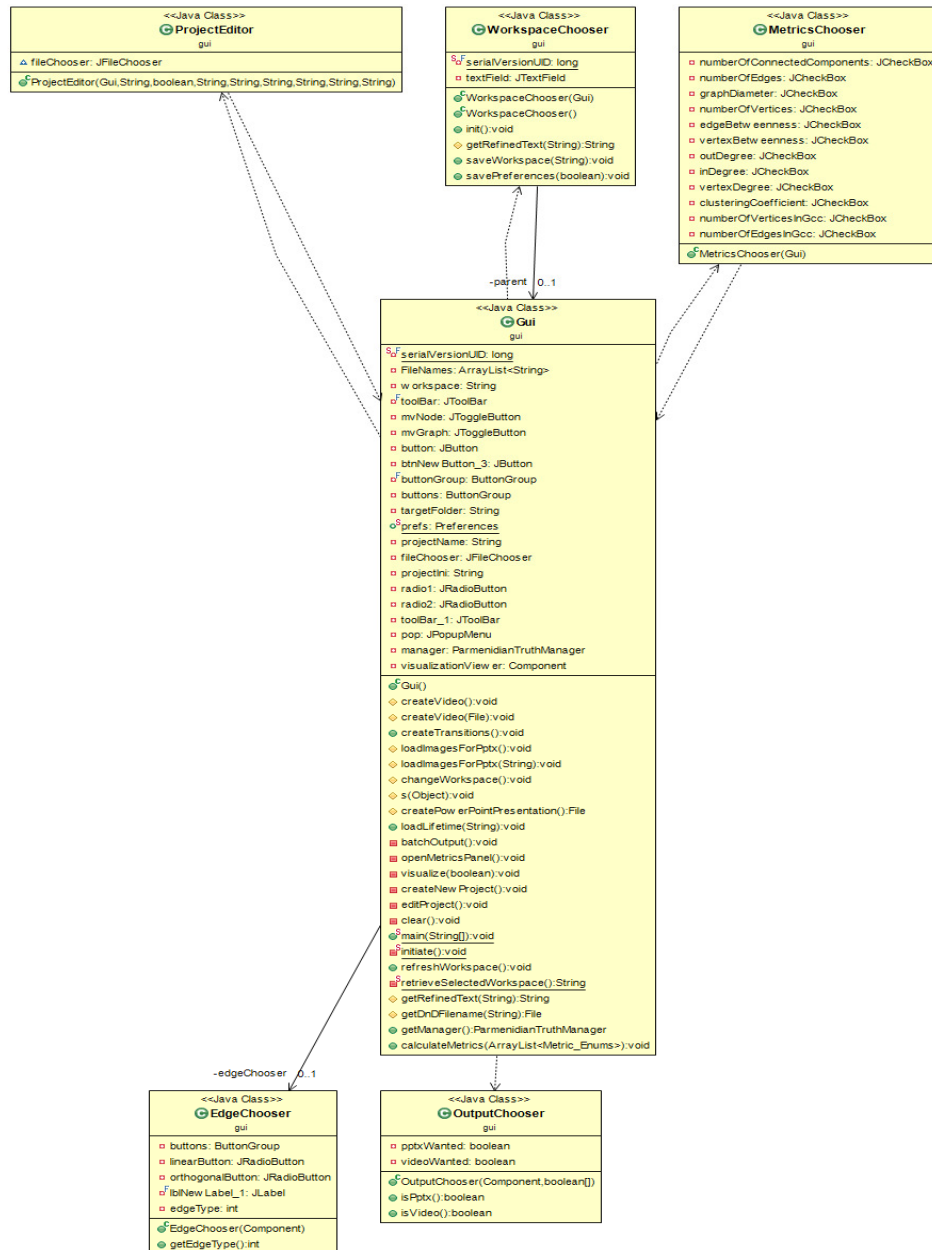


Figure 22 Class Diagram for the gui package

The gui package includes all classes that refer to graphical user interface of Parmenidian truth.

The `gui` class is the main graphical interface. Every use-case that our tool provides, is shown to user through this class. Ergo, when an action is selected, this class triggers the corresponding call in the `ParmenidianTruthManager` which in turn triggers the corresponding call in either the `ModelManager` or the `ExportManager`.

The `workspaceChooser` class, is the dialog that appears when Parmenidian Truth is executed. This class enables the user to choose a physical folder for Parmenidian Truth's output to be stored.

The `ProjectEditor` class, is the dialog that appears when the user chooses project-related actions, like creating or editing the metadata of an already existing project. It enables the user deal with the input and output of his/her project and it also enables him/her create the transition file of a database's schema evolution.

The `EdgeChooser` class, is the dialog that appears just before a project is loaded. It contains information about the type of edge that should be used when the schema is visualized as well as when the png files are rendered. Edge type varies from linear to orthogonal.

The `MetricsChooser` class, is the dialog that appears when the user chooses to produce graph metrics for the database schema under study. It contains a list of checkboxes each of which correspond to a specific metric. After the user is done selecting the `MetricsChooser` informs the `gui` which of the metrics were chosen. This information sharing, between those two classes is done with the help of enumeration for the sake of easy maintenance and code readability.

The `outputChooser` class, is the dialog that appears when the user chooses to produce either a PowerPoint presentation, or a video file. As a class collects the information and triggers on the `gui` class the selected action/actions.

## 6.6 The model.Loader package

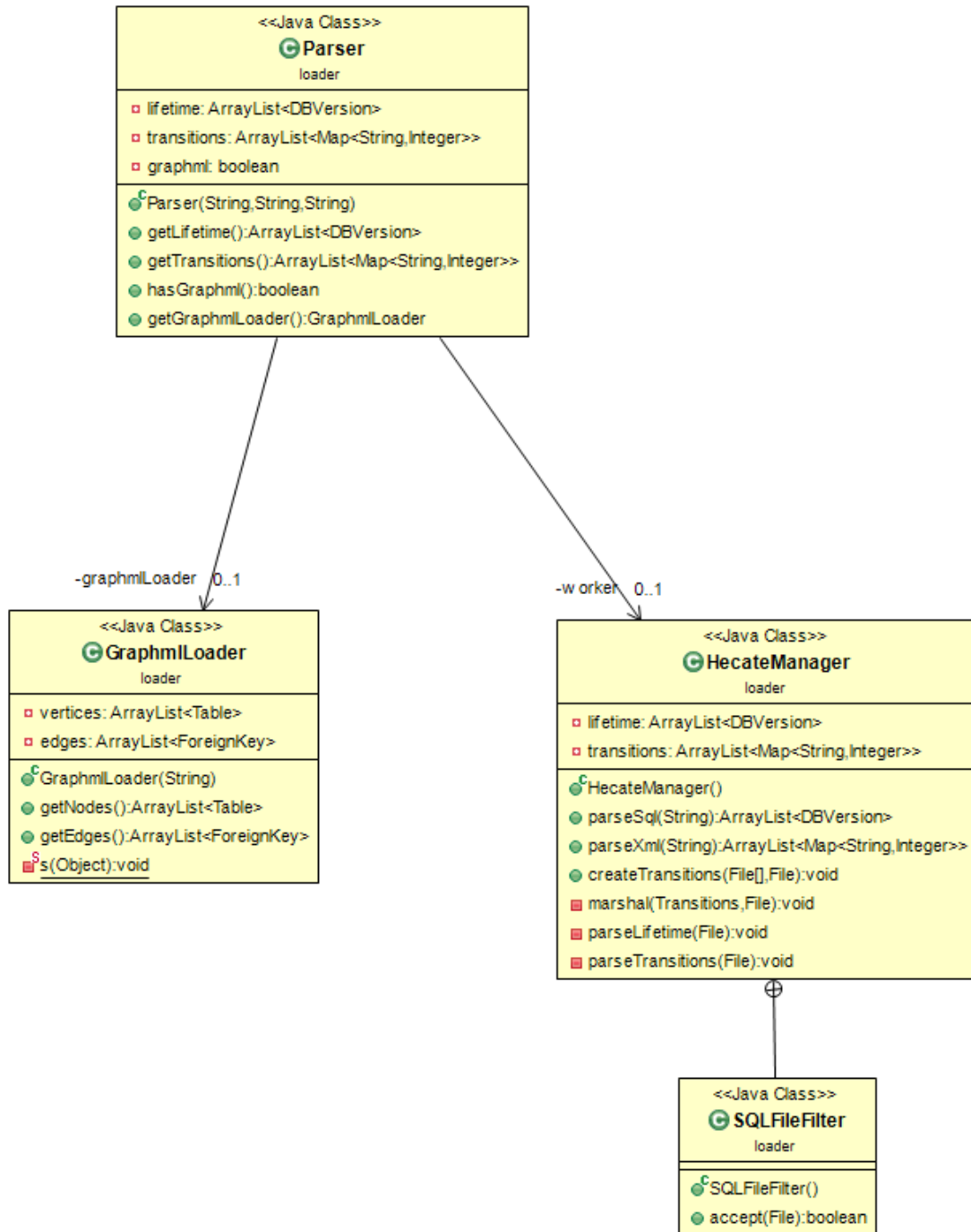


Figure 23 Class Diagram for the model.Loader package

The loader package includes classes that, with the help of the externalTools package, are responsible for parsing data files and organizing the created objects into memory

The Parser class, is responsible for parsing the SQL files, the XML files, and the graphml files, if present, and then map this information into memory through the objects provided in the model package.

The GraphmlLoader class is responsible for parsing the graphml files of the project. Graphml files contain information for a node’s coordinates. After the information is retrieved from the physical files it is stored in the field of a Table object.

The HecateManager class is an abstraction that separates Parmenidian Truth from Hecate. As said at the beginning of this chapter, externalTools is the deepest and final level of parsing physical files. The main entry point and connection between those two tools is this class.

### 6.7 The parmenidianEnumeration package

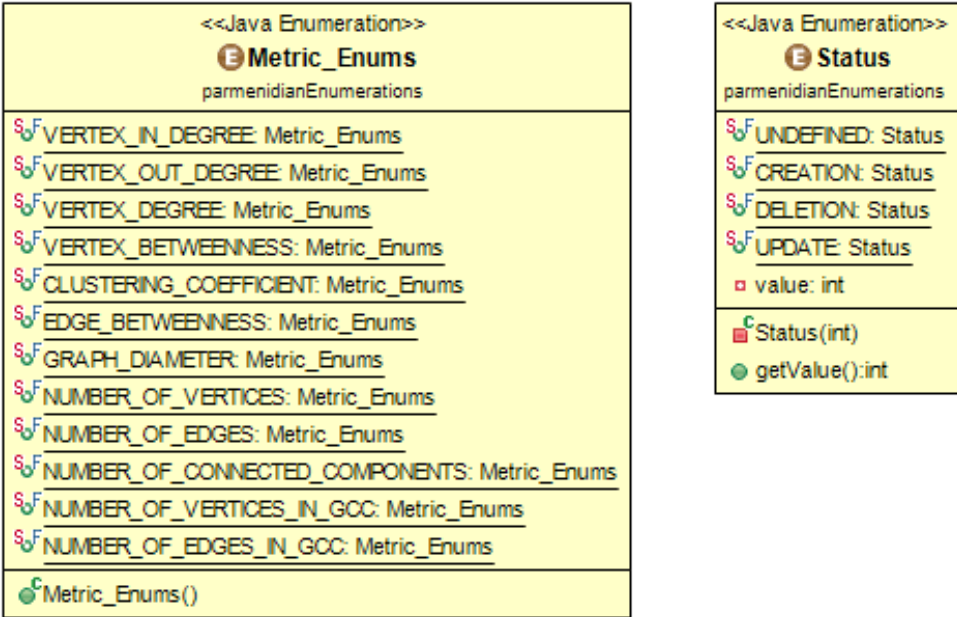


Figure 24 Class Diagram for the ParmenidianEnumeration package

The Status is just an enumeration for mapping integer values [-1.0.1.2] to meaningful notions [UNDEFINED, CREATION, DELETION, UPDATE]. Those values are used for rendering each node with the appropriate color, [Light Green, Green, Red, and Yellow].



Finally, `Metric_Enums` as we mentioned before is an enumeration that makes the code a little more intelligible. Every metric is corresponded to a field in this enumeration, making the communication between classes a little more transparent.



## CHAPTER 7.

### CONCLUSIONS AND FUTURE WORK

---

#### 7.1 Summary

#### 7.2 Future work

---

#### 7.1 Summary

In this Thesis, We have studied the schema histories of a six free, open-source databases that contained foreign keys. To facilitate a quantitative study, we model each version of the schema as a graph, with tables as nodes and foreign keys as directed edges (stemming from the referencing table to the referenced one). Our findings concerning the growth of nodes verify previous results that schemata slowly grow over time in terms of tables. Moreover, we have come to several surprising, new findings in terms of the schema edges (foreign keys). Foreign keys appear to be fairly scarce in the projects that we have studied and they do not necessarily grow in synch with table growth. In fact, we have observed different “cultures” for the handling of foreign keys, ranging from full sync with the growth of nodes to the unexpected extreme of full removal of foreign keys from the schema of the database. Node degrees and survival are related with an inverse gamma pattern: the few nodes with high degrees stand higher chances of survival than average. Similarly, nodes with inciting edges with high values for edge betweenness centrality frequently (but not always) stand higher chances to survive compared to the nodes with a single or zero inciting edges, which have significantly higher chances of removal.

## 7.2 Future Work

Future work can continue in many ways. Here, we mention a few of them that seem more important.

First, one could attempt to identify *patterns of change* over the different graphs. This involves graph mining techniques that look for similar patterns of change over time, either within the same schema's history, or throughout the set of histories that we have collected.

Second, one can attempt to construct a *predictor module* for change. Can we predict when and how (and with what amount of change) a table will change, given its graph-metric properties?

Third, we could try to extract "all-star transitions" – i.e., transitions with the largest possible effect on the structure and properties of the graph. Then, we can make good use of these transitions when we want to pick a set of characteristic transitions in order to concisely visualize the evolution history of the database schema or study transitions with potentially large effect.

## REFERENCES

- [Brand01] Ulrik Brandes. A Faster Algorithm for Betweenness Centrality. *Journal of Mathematical Sociology* 25(2):163-177, 2001.
- [CMTZ08] C. Curino, H. J. Moon, L. Tanca, and C. Zaniolo. Schema Evolution in Wikipedia: Toward a Web Information System Benchmark. In *Proceedings of 10th International Conference on Enterprise Information Systems (ICEIS)*, 2008.
- [Kara01] A. Karahasanovic. Identifying impacts of database schema changes on applications. *Proceedings of the 8th Doctoral Consortium at the CAiSE*, pp.93-104, 2001.
- [LiNe09] Dien-Yen Lin and Iulian Neamtii. Collateral Evolution of Applications and Databases. In *Proceedings of the Joint International and Annual ERCIM Workshops on Principles of Software Evolution and Software Evolution Workshops (IWPSE)*, pages 31-40, 2009.
- [Newm03] Mark EJ Newman. The structure and function of complex networks. *SIAM review* 45.2, pp: 167-256, 2003.
- [PVSV12] G. Papastefanatos, P. Vassiliadis, A. Simitsis, and Y. Vassiliou. Metrics for the Prediction of Evolution Impact in ETL Ecosystems: A Case Study. *Journal on Data Semantics*, 1(2):75-97, 2012.
- [QiLS13] Dong Qiu, Bixin Li, and Zhendong Su. An Empirical Analysis of the Co- evolution of Schema and Code in Database Applications. In *Proceedings of the 9th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE)*, pages 125-135, 2013.
- [Sjob93] D. Sjøberg. Quantifying Schema Evolution. *Information and Software Technology*, 35(1):35-44, 1993.

- [SkVZ14] I. Skoulis, P. Vassiliadis, A. Zarras Open-Source Databases: Within, Outside, or Beyond Lehman's Laws of Software Evolution. In Proceedings of 26<sup>th</sup> International Conference on Advanced Information Systems Engineering (CAiSE), pp. 379-393, 2014
- [WhSm03] S. White and P. Smyth. Algorithms for estimating relative importance in networks. In Proceedings of the 9<sup>th</sup> ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 266-275, 2003.
- [WuNe11] Shengfeng Wu and Iulian Neamtii. Schema evolution analysis for embedded databases. In Proceedings of the 27th IEEE International Conference on Data Engineering Workshops (ICDEW), pages 151-156, 2011.

## SHORT CV

---

Michail - Romanos Kolozoff was born in Kavala, Greece in 1988. In 2013, he received his Diploma in Computer Science from University of Ioannina. In the following months, he started a 6 month internship concerning Android development, in a small startup in Kavala. By 2014, he had re-joined University of Ioannina, which was now renamed to Department of Computer Science and Engineering, University of Ioannina, for acquiring his Master's Degree, while working in parallel as a Senior Android developer, in another startup in Ioannina for 2 years. After successfully completing every course within his Master's curriculum, he joined the army for fulfilling his military duty. In January 2017 he return for his thesis presentation and the completion of his Master's Degree.