Noname manuscript No.
(will be inserted by the editor)

Schema Evolution and Foreign Keys: a Study on Usage, Heartbeat of Change and Relationship of Foreign Keys to Table Activity

Panos Vassiliadis · Michail-Romanos Kolozoff · Maria Zerva · Apostolos V. Zarras

the date of receipt and acceptance should be inserted later

Abstract In this paper, we study the evolution of foreign keys in the context of schema evolution for relational databases. Specifically, we study the schema histories of a six free, open-source databases that contain foreign keys. Our findings verify previous results that schemata grow in the long run in terms of tables. To our surprise, we discovered that foreign keys appear to be fairly scarce in the projects that we have studied and they do not necessarily grow in sync with table growth. In fact, we have observed different "cultures" for the handling of foreign keys, ranging from treating foreign keys as an indispensable part of the schema, in full sync with the growth of tables, to the unexpected extreme of treating foreign keys as an optional add-on that twice resulted in their full removal from the schema of the database. Apart from the behavior of entire schemata, we have also studied the behavior of individual tables. We model the schema of any version of the history as a graph, with tables being nodes and foreign keys being edges. The union of these graphs is called the Diachronic Graph of the schema and contains all the tables and foreign keys that ever appeared in the schema history. The study of the Total Degree of tables at the Diachronic Graph, reveals several patterns. The population of tables with total degree in the range of [0-2] includes almost all the tables that were eventually removed from the schema, as well as the vast majority of survivor tables. These low-degree tables (especially the dead ones) tend to be mostly with zero or very few internal updates in their entire history. At the same time, the few tables with degree higher than 2 are typically born very early in the life of the schema, overwhelmingly survivors, and, unusually active, typically undergoing medium or high update activity.

P. Vassiliadis, A. Zarras
Dept. of Computer Science & Engineering, Univ. of Ioannina, Greece
E-mail: {pvassil, zarras}@cs.uoi.gr

M. Kolozoff
Advantage FSE, Athens, Greece E-mail: mkolozoff@gmail.com
Work done while in the Univ. of Ioannina

M. Zerva
P&I AG, Ioannina, Greece E-mail: mariazerva92@gmail.com
Work done while in the Univ. of Ioannina

1 Introduction

Software evolution is an inherent part of the lifecycle of software, and schemata, carrying the architecture of a relational database are no exception to the general pattern. *Schema evolution* is necessary for schemata to align the information capacity of a database with user requirements, albeit with a cost: as the schema changes, the surrounding applications are affected both syntactically and semantically. Understanding the fundamental mechanisms and patterns behind schema evolution is of great significance as it can allow us to see problems on how databases are used, predict the change of tables in the future and adapt application development, maintenance and resource management to the forthcoming trends.

Schemata do not contain only tables, but also integrity constraints like primary keys, foreign keys and others. Foreign keys, in particular, are mechanisms that constrain 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. Thus, foreign keys, being integrity constraints for the data of a database, are part of the schema of the database, and as such, they are unavoidably amenable to change, too. The main driver of our research is to answer the question: *how do foreign keys evolve over time?*

To the best of our knowledge, until the present paper, the question has been without any answer. There are several works on the study of schema evolution [18, 4, 12, 27, 17, 19, 3], which mostly focus on the marcroscopic study of how the schema size grows in terms of its tables and how the surrounding code of an application relates to the underlying database (see Section 2). Yet, 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.

In this paper, we study schema evolution by placing the focus on foreign keys, rather than tables. 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 (see Section 3). These data sets were *the only ones containing foreign keys*, out of a larger collection of schema histories of Free Open Source Software (FOSS) projects from different domains with adequately long stories and schema sizes. Subsequently, we studied the characteristics of foreign key evolution.

Our macroscopic study of the evolution of the schemata in their entirety, with respect to foreign keys, is detailed in Section 4. Our findings, include both expected and unexpected phenomena. Schemata grow over time in terms of tables and this 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. 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. Unexpected results came from the data sets of Content Management System (CMS) nature, SlashCode and Zabbix, where foreign keys involved only a small minority of tables (Section 4.3). To our big surprise, foreign keys in these projects, after a period of growth, are *completely removed* from the schema with (a) a slow but constant removal rate in the first case, and, (b) a steep removal in the latter. We also make a detailed discussion on the absence and extinction of foreign keys in specific environments in Section 4.4. Our data indicate that, with the exception of few environments with a strict adherence to the dictations of relational theory, foreign keys are scarce and occasionally unwanted.

Apart from the aforementioned results concerning the entire schema, we also turn our attention to studying individual tables. The main research question concerns the relationship between the topology of a relation within a relational schema (with topology being assessed with respect to the incoming or outgoing foreign keys of a relation) and its evolutionary profile. 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 have computed several graph-metric properties, both in terms of the entire schema and in terms of individual nodes and edges. Given the above data, we then proceeded to answer the question: *is there any inherent property of the graph constructs that forecasts their liability to change?*

Out of all the graph-metric properties we have assessed, in this paper we report on the most discriminating one, in terms of its correlation to evolution, the Total Degree of tables at the Diachronic Graph. We report our findings for the relationship of the Total Degree at the Diachronic Graph with table birth, death (survival, in fact) and update activity (Section 5). At the low end of the degree spectrum, zero-degree tables display two different profiles: whereas in scientific database schemata the percentage of these tables is very small, in the rest of the cases, zero-degree tables constitute the vast majority. In all cases, however, zero-degree and low-degree tables (i.e., with degree lower than 3 in the Diachronic Graph) include almost all the tables that were eventually removed from the schema, as well as the vast majority of survivor tables, and, tend to be mostly rigid, i.e., with zero internal updates in their entire history (especially the dead ones) or quiet, i.e., with very few changes. At the other end of the degree spectrum, high-degree tables (i.e., with degree higher than 2 in the Diachronic Graph) are (a) few in number, (b) typically born very early in the life of the schema – equivalently, there is an absence of births of high-degree tables after the first versions of the schema life, (c) overwhelmingly survivors, and, (d) unusually active, typically being characterized as quiet or active (i.e., of high update activity).

We conclude our deliberations with a summary of our findings, a discussion of the importance of this work, along with threats to validity and open issues in Section 6.

2 Related Work

Research on schema evolution has followed several paths. A first path involves works concerning an algebra of schema evolution operations, that can allow the description of the history of schema changes in a semantically-rich sequence of operations [5], [9]. Another path involves the management of the impact of changes [2], [13]. A fairly novel path involves the identification of profiles and patterns in the usage of relational database access technologies in free, open source projects [7], [6], [15].

2.1 Research towards evolution patterns for the entire schema

As all engineering should be based on well-understood mechanics and laws, the research community has tried to uncover mechanics and patterns that govern schema evolution. Knowing the underlying mechanisms of schema evolution is fundamental in engineering solutions that gracefully handle it: without this knowledge we can easily stray in solutions that have no relationship to real-world problems. Although the body of work is rather small compared to the importance of the matter, one cannot ignore that access to schema histories was practically impossible before the proliferation of Free and Open Source Software (FOSS). Thus, apart from an original study in the early '90s [18], it was only in the late '00s that the problem gained a certain momentum that continues till today [4] [12] [27] [17] [5] [3], basically due to the availability of FOSS projects that contain DDL files in their history.

Back in 1993, [18] by D. Sjoberg was the first (and only) early study to publish findings concerning the changes of a database schema over time and how these changes affect any software built around it. The main findings of the study include the observation of a mostly expanding schema, the quantification of changes in different categories, along with the identification of the problems related to the maintenance of the surrounding code when tables are either added or deleted. Late '00s signaled the slow appearance of a set of works starting in 2008, when C. Curino, H.J. Moon, L. Tanca and C. Zaniolo [4] discuss the evolution of Mediawiki's schema. The paper follows the path of [18] in quantifying the percentages of changes and verifying schema expansion. Also it observes the existence of different traits in the duration of the tables of a schema. In 2009, Lin and Neamtiu [12] published the findings of a study of the co-evolution of applications and databases. The main finding of this study is the absence of total synchronization of the application code with the underlying schema to a surprisingly large extent of the tables and software modules. The results were further supported in 2011 by the work of Wu and Neamtiu [27] who offered tool support to the automatic study of how embedded database schemata evolve. D. Qiu, B. Li and Z. Su [17] study the co-evolution of database schemata and code in ten open-source database applications. The authors report that the main high-level schema changes are transformations, structure refactorings and data quality refactorings, while architectural refactorings took place relatively infrequently. The low-level most frequent atomic change types were: *add table*, *add column* and *change column datatype.* The authors also observed that referential integrity constraints and procedures are rarely used in practice. Finally, the authors confirm previous findings on the lack of synchronization between schema and applications. Cleve et al. [3] discuss the case study of the schema evolution of a Web-based system for medial record management, along with the tool support that allowed them to perform their study. The study verifies the observations of other works concerning the trend of increase in schema size and the reluctance in the deletion of tables.

Since our team in the University of Ioannina started working on this topic, in 2013, we have encountered several interesting patterns of schema evolution. Starting with patterns concerning the entire schema, in [19] and [20], we have studied how the schema of a database evolves in terms of its size, growth and activity. Our findings indicate that *schemata grow over time in order to satisfy new requirements, albeit not in a continuous or linear fashion, but rather, with bursts of concentrated effort of growth and/or maintenance interrupting longer periods of calmness.*

2.2 Research towards evolution patterns for individual tables

A different, innovative research path that we have ignited in [24] and [26] concerns the study of profiles and patterns of *tables*, rather than *schemata*. We have tried to correlate static properties of tables, like the point of birth, or their schema size at birth, with characteristics of activity, like the sum of changes the tables have undergone, or their survival (we like to refer to tables removed as "dead" and table that made it to the last known version of the history that we study as "survivors"). We came up with four patterns suggesting that tables with large schemata tend to have long durations, tables with many updates are typically of medium size schema, tables with medium or short durations typically have very small activity, and, that the majority of removed tables have mostly short lives (i.e., old timers are rarely removed). In [23] we have studied how survivors differ from dead tables with respect to the combination of duration and activity profile. The resulting pattern was named *Electrolysis pattern* due to the intense antithesis in the lives of dead and survivor tables: whereas dead tables are attracted to lives of short or medium duration and absence of schema update activity, survivors are mostly located at medium or high durations and the more active they are, the stronger they are attracted towards high durations.

2.3 Research on Foreign Keys

Foreign keys have been part of database design since the very early days [21] till now [11]. Starting from the conceptual model and passing to schema refinement via normalization processes, foreign keys are instrumental in guaranteeing the integrity of the data in a relational database [21]. When foreign keys are absent, or there are semantic discrepancies in the queried databases that practically make them invalid, the extraction of accurate query answers turns to be a very elaborate problem. See for example [8] for the case of aggregate query processing in the presence of inconsistent values. Automatically extracting foreign keys when they are absent is another very important problem that has attracted the attention of the research community, either directly [28, 14], or indirectly via the discovery of hidden functional dependencies [16].

2.4 Comparison to the state of the art

In all previous case studies of schema evolution, the object of study was the schema size as well as the heartbeat of change, and only lately, tables. To the best of our knowledge, this line of research is the first comprehensive effort in the literature to study the evolution of foreign keys. The paper at hand substantially extends a first version published in ER'17 [25] with the entire Section 5, which explores table evolution in the presence of foreign keys and is completely novel. The current section on related work is an extension of the presentation of [22] (whose focus was mainly to touch the issue of lack of evolution in relational schemata).

We kindly refer readers to our website `http://www.cs.uoi.gr/~pvassil/projects/schemaBiographies/index.html` that contains pointers to available data sets, the tools that we have built and material around our published work.

3 Experimental Setup

In this section, we begin with fundamental concepts for our study. Then, we introduce the datasets that we have collected and processed using *Parmenidian Truth*, an open source tool we created for the purpose of this study.

3.1 Datasets and data processing

In the summer of 2013, we collected twenty data sets to support our work on mining evolution patterns in the history of open-source databases [1]. We have followed a typical approach for the mining of software repositories [10] and produced a list of DDL files representing the history of the monitored schemata as the result of our automated collection. The term “data set” refers to the history of the schema of a software project, represented as a sequence of files, sorted in terms of their commit timestamp, with the Data Definition Language commands that create the schema of the project's database. We have worked with the main branch of these projects. Our collected files do not include information on the conceptual model of the monitored schemata and, as they are *schema* declaration files, they do not contain any data. The collection includes two datasets from the biomedical domain (Ensembl and Biosql), 5 data sets from CERN(Atlas, Egee, DQ2, Castor2 and Dirac) and 13 data sets from the CMS domain (Slashcode, Zabbix, Coppermine, Dekiwiki, E107, Joomla 1.5, Mediawiki, Nucleus, phpBB, phpwiki, Tikiwiki, Typo3, Xoops).

When we turned our attention to the study of foreign keys, we came up with a surprising discovery: only two of the 13 CMSs included foreign keys (!) in contrast to all the biomedical and CERN-oriented data sets that came with foreign key usage. In the latter two families, Atlas, Castor and BioSQL are really useful for analysis. Egee is a smaller data set, in number of tables, foreign keys, and in number of revisions. We have retained Egee as a test, proof-of-concept data set. DQ2 comes with 55 versions in its mySQL version, out of which only the first 19 contain foreign keys. The data set starts with 2 foreign keys and ends with 1, only

[1] All our data sets and software are openly available at our group's Github site (`https://github.com/DAINTINESS-Group`)

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	0.03%
BioSQL	47	6 Y, 7 M	21	28	45	33%	17	43	79	153%
Egge	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	0%
Zabbix	160	10 Y, 10 M	15	48	58	220%	10	2	38	-80%

Fig. 1 The main characteristics of the data sets.

to be permanently dropped in the 20th version. DIRAC is quite similar case to Egee, comes with 42 versions over a very small schema, as it starts with 9 tables and 10 foreign keys at first version and ends with 15 tables and 8 foreign keys (less than in the beginning). The only data set that hides a -yet unclear- potential is Ensembl, where we have not yet managed to link the 529 files with table creation statements to the 18 files containing foreign key declarations. Thus, *we ended up with 6 data sets with adequate information on foreign key evolution to study*. The main characteristics of the six data sets that we considered in our study are given in Fig. 1.

Based on our tool, *Parmenidian Truth*[2], we have parsed, internally represented, visualized and measured the evolution of the studied schemata. Given the history of a database, expressed as a sequence of data definition files, and consequently, a sequence of differences between subsequent versions, *our tool models and 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. Then, the tool was also extended with measurement collection capabilities. Thus, all our measurements are also produced by the very same tool.

3.2 Modeling Relational Schemata as Graphs

We treat a relational database schema as a set of relations, along with their foreign key constraints. A relation is characterized by a *name*, *a set of attributes* and *a primary key*. A *foreign key constraint*, is a pair between a set of attributes $\mathcal{S}$ in a relation, R_S, called the *source* of the foreign key, and a set of attributes $\mathcal{T}$ in a relation R_T, called the *target* of the foreign key. The foreign key constraint requires

[2] https://github.com/DAINTINESS-Group/ParmenidianTruth

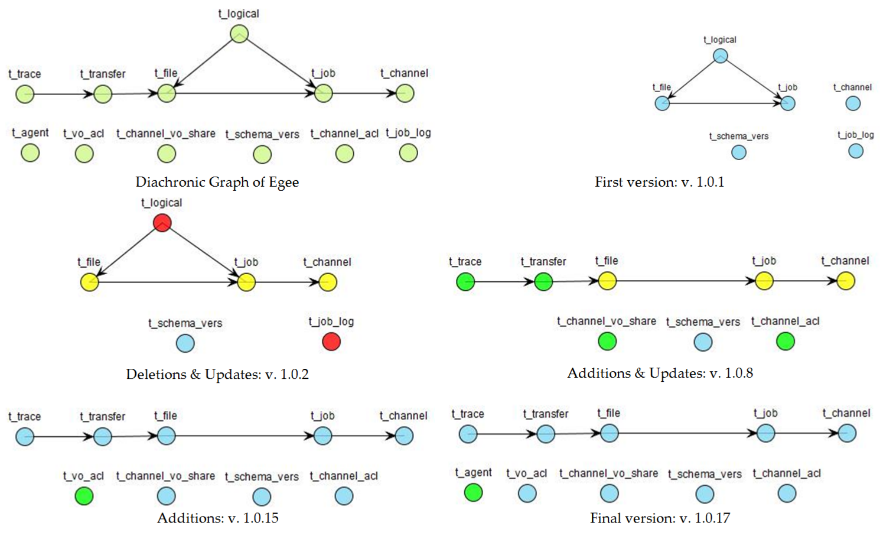


Fig. 2 The Diachronic Graph (top left) and the story of Egee via visualizations generated by Parmenidian Truth. For the readers in color: green nodes are the ones born at the current version, red nodes are the ones to be deleted at the subsequent version, and yellow nodes are the ones being internally updated with new or deleted attributes at the current version.

a 1:1 mapping between $\mathcal{S}$ and $\mathcal{T}$. 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 attributes.

We model a 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. Both nodes and edges are annotated with the respective information mentioned in the previous paragraph. 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*, $H = \{v_1, \ldots, v_n\}$, of a database schema can be thought of as a sequence of versions. Each *version* of the schema v_i is represented by a graph $G_i(V_i,\, E_i)$. A *transition* between two subsequent versions of the history includes a set of changes, involving (a) additions and deletions of relations and foreign keys, and, (b) relation updates in the form of changes of primary keys, modifications of attribute data types, and, attribute additions or deletions.

To represent the entire history of the schema via a single construct, we introduce the *Diachronic Graph*. The *Diachronic Graph* is the union of all the nodes and edges that ever appeared in the history of the schema. In other words, the Diachronic Graph holds every table and every foreign key that ever existed in the history of the schema.

Definition 1 Assuming a relational schema representing a particular version of a schema history H, say v, the *Schema Graph* of v is defined as the graph $G_v(V_v,$ $E_v)$, with V_v being the set of relations that exist in the schema at the version v and E_v being a set of directed edges, one per foreign key that exists at version v,

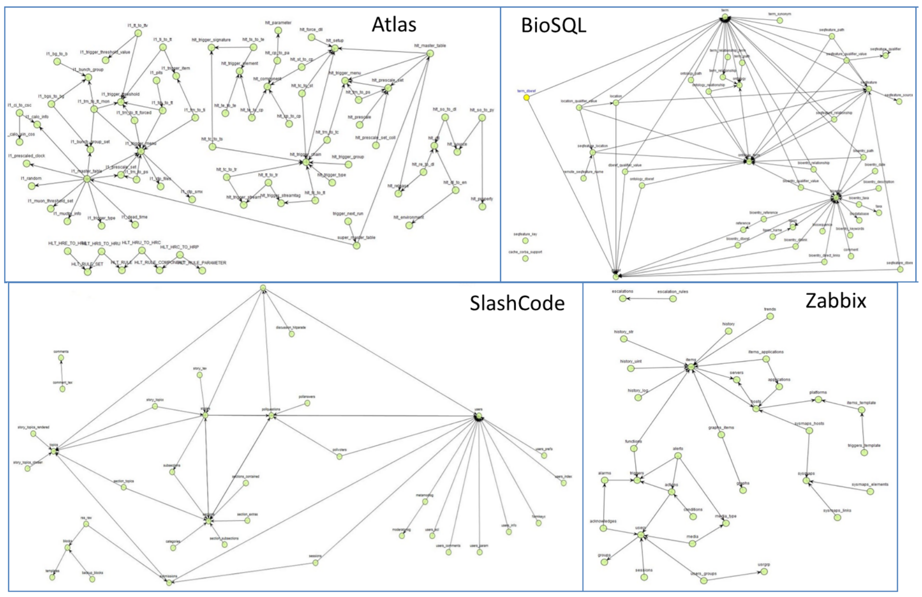


Fig. 3 The Diachronic Graph of 4 datasets

having as source the source table holding the foreign key constraint, and as target the target, referenced, lookup table of the foreign key constraint.

Definition 2 Assuming an evolution history of a schema, $H = \{v_1, \ldots, v_n\}$, with each version v_i being represented by its Schema Graph $G_i(V_i, E_i)$, the *Diachronic Graph* $G_\Omega(V_\Omega, E_\Omega)$ is defined as the union of all nodes and edges in the history, i.e., $V_\Omega = \bigcup V_i$, $E_\Omega = \bigcup E_i$, $i = 1, \ldots, n$.

As an example, the evolution history of the Egee dataset is presented in Fig. 2. The first graph represents Egee's diachronic graph. The following graphs represent versions with deletions and additions that shaped the diachronic graph. In terms of coloring, our tool uses red for deleted nodes, green for added nodes, and yellow for nodes with internal updates (e.g., attribute additions, deletions or data type changes). Fig. 3 shows the Diachronic Graph of 4 datasets under study.

Usage and Opportunities of our Graph Modeling. The concept of Diachronic Graph was born from the necessity to uniquely determine the x,y position of nodes in a 2D canvas for the purpose of telling the story of a schema with the Powerpoint presentations of our tool, Parmenidian Truth. The lesson learned, however, was that we can have a single construct with the entire history of the schema. We have used, thus, the Diachronic Graph to relate the graph properties of the nodes (tables) with their evolution and activity, as discussed in Section 5. What lies outside the scope of this paper, but can be part of future endeavors of the research community is the annotation of the Diachronic Graph with more semantics, and the exploitation of this information for statistical studies, either along the lines of the current paper, or completely novel ones. Such annotations possible include the time of birth and removal of nodes and edges, semantics of the propagation of events via the foreign key edges, (e.g., ON DELETE CASCADE annotations), number of attributes involved, data types, and many others.

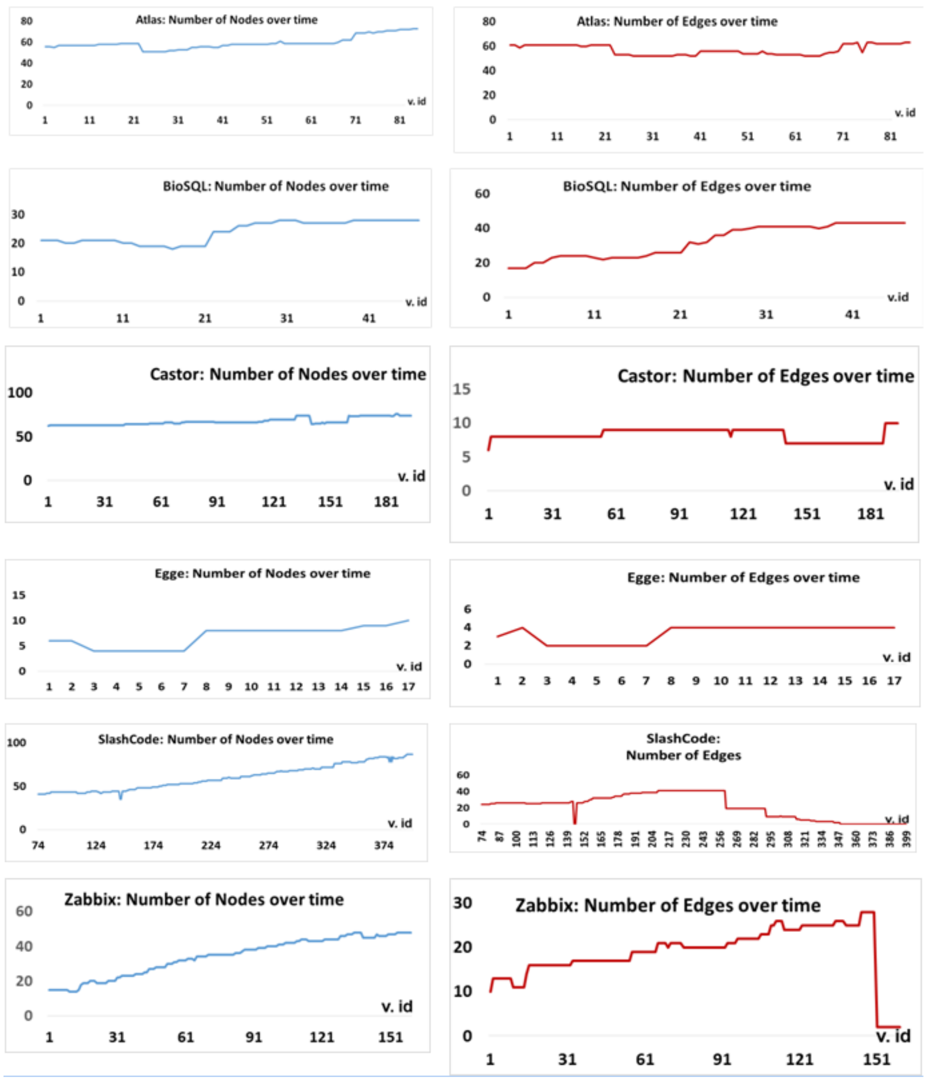


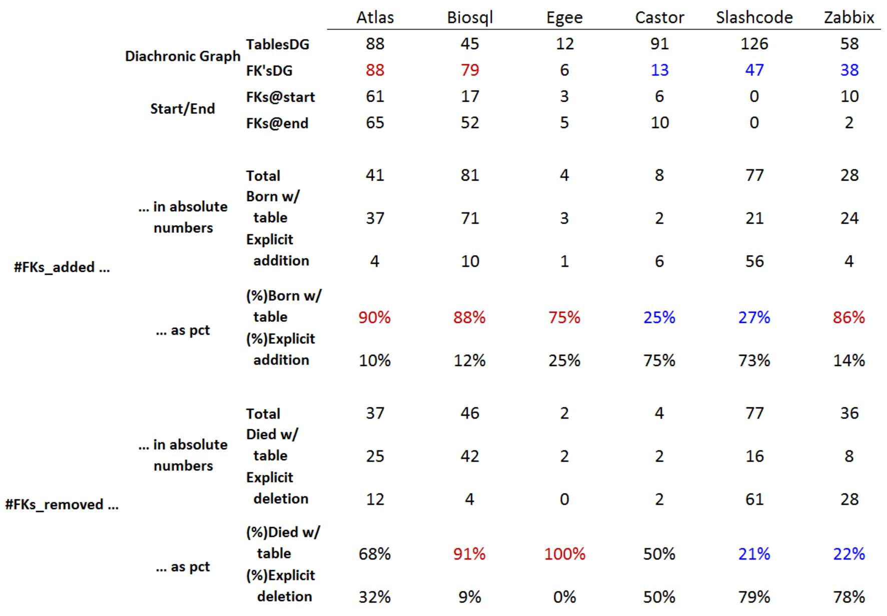
Fig. 4 Number of nodes (tables) and edges (foreign keys) over time (x-axis: version id) for the 6 studied data sets

4 Foreign key evolution at the schema level

4.1 Total number of tables and foreign keys

In this section we quantitatively assess the evolution of the datasets that we study, with respect to the total number of tables and foreign keys throughout their entire lifetime. Fig. 4 depicts the evolution of these two measures.

The different categories of schemata expose very different behaviors with respect to their growth, and especially with respect to the growth in terms of for-



			Atlas	Biosql	Egee	Castor	Slashcode	Zabbix
Diachronic Graph		TablesDG	88	45	12	91	126	58
		FK'sDG	88	79	6	13	47	38
Start/End		FKs@start	61	17	3	6	0	10
		FKs@end	65	52	5	10	0	2
#FKs_added ...	... in absolute numbers	Total	41	81	4	8	77	28
		Born w/ table	37	71	3	2	21	24
		Explicit addition	4	10	1	6	56	4
	... as pct	(%)Born w/ table	90%	88%	75%	25%	27%	86%
		(%)Explicit addition	10%	12%	25%	75%	73%	14%
#FKs_removed ...	... in absolute numbers	Total	37	46	2	4	77	36
		Died w/ table	25	42	2	2	16	8
		Explicit deletion	12	4	0	2	61	28
	... as pct	(%)Died w/ table	68%	91%	100%	50%	21%	22%
		(%)Explicit deletion	32%	9%	0%	50%	79%	78%

Fig. 5 Statistical breakdown on the creation and removal of Foreign Keys

eign keys. The first group of schemata, involving scientific databases, like Atlas and Biosql expose *growth that has expansion periods, shrinkage actions, and periods of calmness* in terms of both tables and foreign keys. The schema is of moderate size for Atlas (from 56 at start, to 73 tables at end) and of small size for Biosql (starting with 21 and ending with 28 tables), and the growth of nodes and edges is practically in sync.

Concerning the category of computational resource toolkits, Egee is very small in size and history and mostly serves as a demo example. Castor, on the other hand, has a very large percentage of nodes without edges (observe the difference of values in the y-axis). The number of tables grows from 62 to 74 tables (with the occasional removals and periods of calmness), whereas the number of foreign keys is relatively stable (from 6 to 10).

Concerning the case of the two CMS's (SlashCode and Zabbix), both CMS's go through a clear trend of expansion. Slashcode started without foreign keys at all and obtained its first set of foreign keys in version 74. *Both CMS's end up with zero foreign keys*, however! For Slashcode there is a clear phase of progressive removal, whereas for Zabbix, there is an abrupt removal of almost the entire set of foreign keys in a single transition. *The fact that developers can resort in full removal of foreign keys at some point in the lifetime of a schema is a real surprise.* We devote a dedicated discussion on this in the sequel of the paper.

4.2 Heartbeat of changes

How do foreign keys germinate and die? A first question that we wanted to explore is how does the generation and removal of foreign keys takes place. We

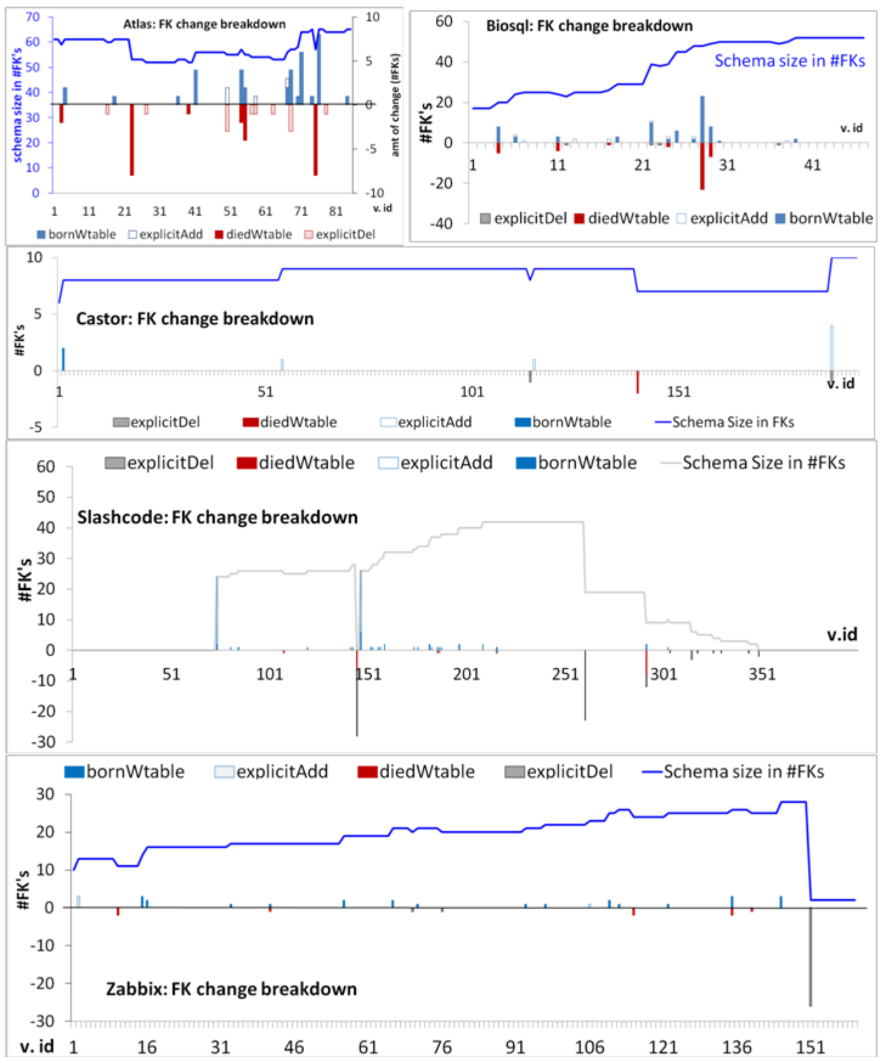


Fig. 6 Heartbeat of Foreign key creation and removal

have classified births and deaths of foreign keys in four categories. An addition of a foreign key is considered as *born with table*, when either the source or the target table is born along with the foreign key, while an *explicit addition* happens, when a foreign key is added to two existing tables. Respectively, in the case of deletions, a deletion of a foreign key is considered as *died with table*, when either the source or the target table is removed along with the foreign key, while an *explicit deletion* takes place when neither of the source or target tables gets deleted and only the foreign key is removed. In Fig. 5, we present the statistical breakdown of the creation and removal of foreign keys and we can see that different cultures for handling foreign keys exist.

	Total # transitions	Total # transitions with FK change	Pct. of transitions with FK change
Atlas	85	25	29%
BioSQL	46	19	41%
Egee	16	3	19%
Castor	191	6	3%
Slashcode	398	34	9%
Zabbix	159	22	14%

Fig. 7 Percentage of transitions containing schema change of foreign keys

The scientific data sets, Atlas and BioSQL, deal with foreign keys as a regular part of the schema. Thus, foreign keys are overwhelmingly born along with tables, and are very rarely added explicitly to existing tables (the latter percentage ranges between 10%-12%), while, at the same time, they are mainly removed when one of the involved tables is removed too. Egee, coming from the category of computational resource toolkits behaves similarly.

Castor and Slashcode deal with foreign keys as an ad-hoc add on. In these two data sets, a very large part of the schema is without foreign keys (compare the first two data rows of Fig. 5). In Slashcode, foreign keys are introduced in v. 74. In both cases, foreign keys are added to existing tables three times more often than they are created with new tables. The death of foreign keys is also taking part without the removal of the tables: in the very few such occasions in Slashcode, the two removal methods are evenly split, but in Slashcode, explicit removals are 4:1 over removals along with table death. Remember, of course, that Slashcode is a data set were eventually all foreign keys were removed.

Zabbix is a mixture of the above behavior with a sudden *change of style*. It is clear that Zabbix started by dealing with foreign keys as a regular part of the schema: foreign keys were present at the beginning, they were mostly born with the birth of new tables and additions to existing tables were rare. Towards the end of the schema's history, however, between version 1.150 and 1.151 *all* foreign keys are explicitly commented out and never restored back. This means that an intentional decision of treating foreign keys as a disposable add-on to the schema has been taken.

What are the characteristics of the heartbeat of change of the foreign keys? In Fig. 6, (a) the number of foreign keys in each version of the schema history is depicted as a solid line, and, (b) the number of foreign key births and deaths is depicted via the respective bars. The bars belong to the aforementioned four categories of change.

A common theme in all the data sets is the consistent scarcity of foreign key changes (Fig. 7). Apart from the scientific data sets, where the number of foreign key changes is high, the rest of the datasets demonstrate a small percentage of transitions with foreign key change. As already mentioned, scientific data sets treat foreign keys as an integral part of the schema and table births and deaths come along with the respective changes. Thus the frequency of change is high. In the rest of the datasets, the additions are too few and explicit (see Fig. 5). In the case of Slashcode, if the phase of mass deletions was not part of the history, the activity would be even less.

In terms of time spread, *in most of the data sets, the events are proportionally spread in time*. Atlas is an exception to this pattern. We occasionally see (a) do-undo actions (in Atlas, Slashcode and Castor), where an update of the schema is undone in the following commit and, (b) restructuring due to table renamings (4 times in Biosql, and twice in Zabbix). *The volume of change is also low*: most changes do not exceed one foreign key, with the exceptions of explicit mass additions and deletions, as well as do-undo actions.

4.3 The strange case of the disappearing foreign keys

Slashcode and Zabbix are our two CMS's that displayed the phenomenon of eventually losing all their foreign keys. Whereas for Zabbix, all our efforts for retrieving any documented reasons for the removal have been fruitless, Slashcode has an abundance of records on the removals of foreign keys. In the sequel, we report on this story.

In the first occurrence of massive foreign key removals (at version rev_1.120), 22 foreign keys were deleted. This massive removal took place due to a problem with the compatibility of the attribute types that the foreign keys referred to. The Data Definition file contains an explanatory comment for this removal:

> "Commented-out foreign keys are ones which currently cannot be used because they refer to a primary key which is NOT NULL AUTO_INCREMENT and the child's key either has a default value which would be invalid for an aut_increment field, typically NOT NULL DEFAULT '0'. Or, in some cases, the primary key is e.g. VARCHAR(20) NOT NULL and the child's key will be VARCHAR(20). The possibility of NULLs negates the ability to add a foreign key. ⇐ That's my current theory, but it doesn't explain why discussions.topic SMALLINT UNSIGNED NOT NULL DEFAULT '0' is able to be foreign-keyed to topics.tid SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT"

In the second deletion (at version rev_1.151), 10 foreign keys were removed, because some tables changed their storage engine to InnoDB from MyISAM. There was also an explanatory comment inside the corresponding sql file:

> "Stories is now InnoDB and these other tables are still MyISAM, so no foreign keys between them."

The rest of the deletions happened because the foreign keys caused too many problems to the system that could not debugged, resulting in the decision to leave the schema without any foreign keys. We have retrieved several comments for these removals. At version re_1.174, where 3 foreigns keys were deleted the following comment was found:

> "This doesn't work, makes createStory die. These don't work, should check why..."

At version's rev_1.189 file the comments mention :

> "This doesn't work, since in the install pollquestions is populated before users, alphabetically"

Finally, at version rev_1.201 the following comment was found:

> "This doesn't work, since discussion may be 0."

At the end of this process, the schema is left with zero foreign keys. Interestingly enough, the schema also contained no foreign keys at its start. Quite importantly, Slashcode's behavior holds both foreign key additions and deletions mostly happening explicitly (i.e., without the addition or removal of the involved tables). In other words, it appears that *foreign keys are treated as a disposable add-on that was removed when problems occurred*.

What do we make out of these removals? *The main problem seems to be the difficulty that developers had to face with fine details in the tuning and handling of the foreign keys. Practically, it appears that the easiest way out of this kind of problems is to comment out the respective foreign key.* We acknowledge the difficulties that occur e.g., in the case of different storage engines and the performance constraints that can drive such decisions. However, the fact that the removals of foreign keys went on as a regular practice, instead of attempting to fix the problems (some of which can be considered fairly easy fixes, like for example changing the order of table population) simply states that the essence of the contribution of foreign keys in the consistency of the schema does not seem to outweigh the need to quickly get things done.

4.4 Are foreign keys unwelcome in CMS's (and elsewhere)?

One could easily suggest that the removal of foreign keys from our two CMS's is just a coincidence. Although this can certainly be the case -and we cannot verify the problem unless more studies by independent groups are performed- there is evidence to suggest that the case of CMS's suffers from an "unfriendly" attitude towards foreign keys: as already mentioned in Section 3.1, out of 13 CMS's that we had collected, only 2 contained foreign keys, and both eventually dropped them.

We do not insist that this is a CMS-specific phenomenon, but rather attribute it to the combination of two factors, that, whenever present, support the avoidance of foreign keys. First, in a CMS environment, the population of the table columns where foreign keys ought to be present, mostly comes from drop-down listboxes with values. This creates the -dangerous, in our opinion- impression to the developers that the data consistency is attainable via the application. Of course, this opinion overlooks the possibility of integrity violation due to the actions of a DBA independently of the surrounding application, as well as the possibility of a bug in the population of the drop-down listboxes. Second, compared to alternatives like ISAM (frequently used in CMS's), foreign keys impose a time lag in terms of efficiency, along with a penalty on size and simplicity, which developers decide not to pay, especially if they operate under the aforementioned impression of data consistency. Again, we note that these phenomena (albeit frequent in CMS's) are not CMS-specific, but can be encountered in other occasions too.

5 How does the evolution of tables correlate with foreign key metrics

In this section, we discuss how evolution is related to the graph metrics of each relation in a schema. *The main goal of our effort is to relate the change activity and the survival of individual tables to their graph-theoretic properties.*

One of the first problems that we had to address in our study is the choice of graph-theoretic metric that we should use to characterize tables, out of the multitude of the measures involved in the study of the evolution of foreign keys. Out of the vast number of options (including in-, out-, and total degree, betweenness centrality, and others, and for each of these metrics, the respective value at the Diachronic Graph, the birth and the last known version for each node, along with the average/max/min value of the metric throughout the lifetime of the node), our tests have indicated that *the Total Degree of a table at the Diachronic Graph, DegreeDG, is both a simple measure*, and, at the same time, *the most promising measure to research on its relationship with survival and activity.* Thus, the rest of our deliberations are devoted to describing our findings concerning *DegreeDG* and its relationship to survival and activity of tables.

5.1 Total Degree and its relationship to Survival

Research Question: is there a relationship between the Total Degree at the Diachronic Graph of a table and its survival?

We have studied the relationship of Last Known Version of a table (which is the version of death for non-survivors or the last version of the schema for survivors) with its Degree at the Diachronic Graph. Fig. 8 left, and Fig. 9 indicate that *the few tables with high Total Degrees at the Diachronic Graph systematically tend to survive and reach the last version of the schema*.

Fig. 8, in its leftmost column, graphically depicts how total degree relates to survival. The clear observation from the figure is that the relatively few tables found at high degrees are almost certainly survivors. Fig. 9 confirms with concrete numbers the observation that total degree higher than 2 is almost guaranteeing survival. Even Biosql which is a deletion-prone data set, with 62% survival ratio and the lowest survival rate for high degrees, reaches a survival rate of 84%.

Given this data, *we can methodically investigate whether evolutionary behavior is independent from graph properties.* For each data set, we created a 2x2 contingency table over survival or death and totalDegreeDG. For the latter, we grouped tables in two groups (a) degrees larger than 2 and (b) degrees lower or equal to 2. Given this contingency table, we can compare the observed to the expected values and see whether the differences are significant enough. This is the essence of the widely used Chi-square test. Due to is somewhat restrictive assumptions, we also employ the Fischer test in our study, which is more tailored to small or sparse data sets. See [1] for a very illuminating presentation of both tests.

The *statistical evidence* in terms of Chi-square and Fisher tests *is not very strong.* With the exception of BioSQL (with a large population of tables with degree higher than 2) the rest of the data sets have very small numbers of tables in the category of total degree > 2. Due to this very low number of high degree tables, the results come with high p-values between 28% and 45% (even 100% for castor with only 2 tables of degree higher than 2). Thus, these high p-values prohibit us from

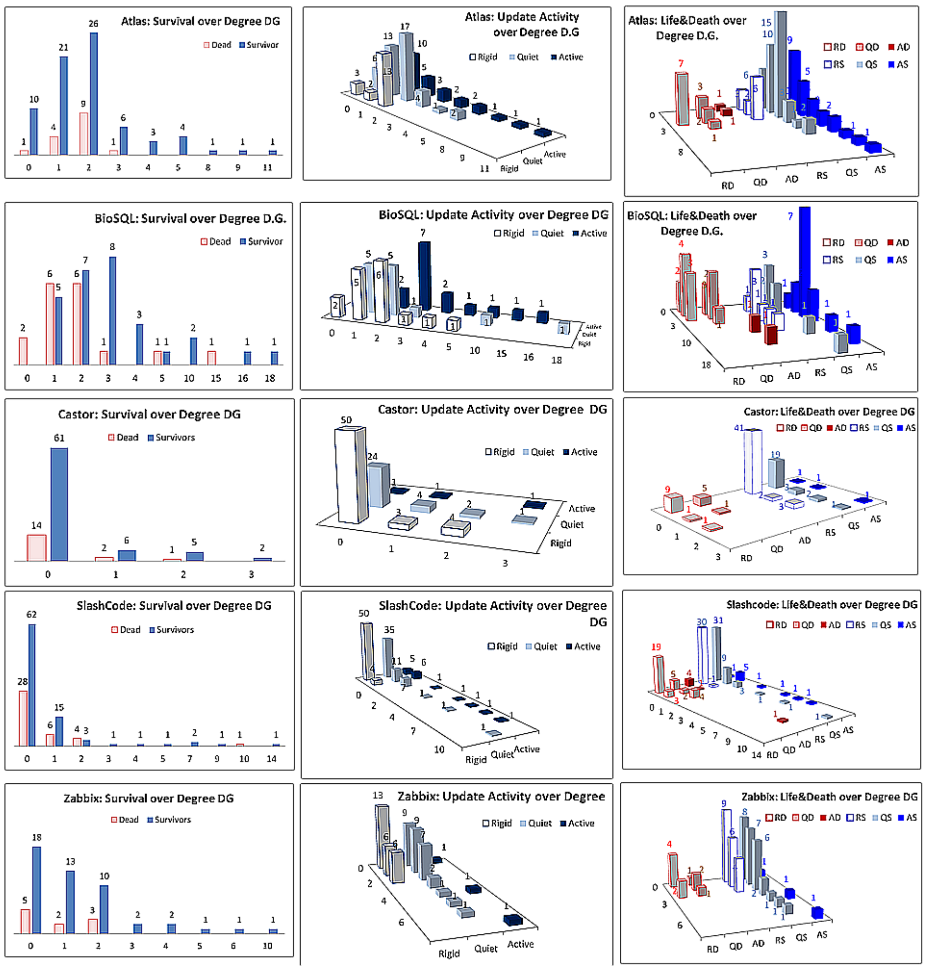
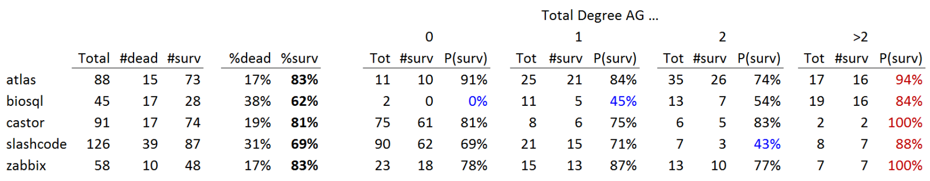


Fig. 8 Breakdown of tables with respect to their survival(left), activity profile (center) and their combination (right) over the Total Degree at the Diachronic Graph

unequivocally declaring a clear pattern for the relationship of survival and high total degree, despite the fact that in all the data sets that we have studied, the increase in the survival rate of high-degree tables compared to the average ranged between 11% and 24%.

5.2 Total Degree and its relationship to Update Activity

We classify tables in three groups: (a) *rigid*, with no updates at all in their lifetime, (b) quiet, if they have sustained less than 5 updates or an Average Transitional Update (ATU) rate less than 10% (by dividing the table's total number of updates by its duration, counted in number of versions), and (c) *active,* if they have more than five updates and ATU higher than 10% [24, 26].



						Total Degree AG ...											
						0			1			2			>2		
	Total	#dead	#surv	%dead	%surv	Tot	#surv	P(surv)	Tot	#surv	P(surv)	Tot	#surv	P(surv)	Tot	#surv	P(surv)
atlas	88	15	73	17%	**83%**	11	10	91%	25	21	84%	35	26	74%	17	16	94%
biosql	45	17	28	38%	**62%**	2	0	0%	11	5	45%	13	7	54%	19	16	84%
castor	91	17	74	19%	**81%**	75	61	81%	8	6	75%	6	5	83%	2	2	100%
slashcode	126	39	87	31%	**69%**	90	62	69%	21	15	71%	7	3	43%	8	7	88%
zabbix	58	10	48	17%	**83%**	23	18	78%	15	13	87%	13	10	77%	7	7	100%

Fig. 9 Probability of survival per degree contrasted to the average probability of survival for the entire data set. The coloring is as follows: blue, if lower than the average probability by 10% or more, red if higher than 10% or with 100% survival.

Research Question: is there a relationship between the Total Degree at the Diachronic Graph of a table and its update activity?

The relationship of activity with total degree at the Diachronic Graph shows both some expected behaviors and some unexpected biases in the breakdown of tables. Fig. 8 depicts the situation graphically in its middle column.

With the exception of high-degrees, which avoid rigidity, for the rest of the tables, their breakdown in activity categories follows the breakdown of tables of the entire population – thus, the answer to the research question is negative for most tables and positive for the high-degree tables. Observe for example, (a) the two scientific data sets at Fig. 8, and see the "battleship" pattern with a small number of tables having zero total degree and (b) the "monotonically decreasing" pattern for the rest of the data sets. However, it is indeed striking how the tables with very high degrees avoid rigidity, and frequently, even quiet lives.

Rigid tables span a very short range of degrees compared to the other categories. This is quite unexpected: one would actually expect tables being strongly correlated with other tables to be unaffected by changes and tables with zero degree would attract most of the changes. Yet this is not the case, and tables of large degree are by no means rigid, while small or zero degrees are more prone to rigidity.

Quiet tables span longer than the rigid ones and equal or less than the active ones. Quiet and rigid tables share the position of the most populous group in our data sets. Still, in all our data sets, we will see quiet tables reaching much higher ranges of degrees than the rigid ones.

Finally, active tables are by far the smallest group in all our data sets except for Biosql. Yet, they span a broad (in half of the data sets, the broadest) domain of values for their total degree. Similarly, not only in terms of range, but also in terms of population, in half the cases, the high degrees are mostly populated by active tables. *Thus, there is an unexpected relationship between activity and high degrees that is both (a) against what one would originally expect from the statistics (the very few active tables do not follow the global trend of clustering in small degrees) and (b) counter-intuitive with respect to the expectancy that high level of interdependency would gravitate tables towards rigidity.* Gravitation to rigidity would have us expect that the more interconnected a table is, the more reluctant would the developers be to modify it, because changes might affect other tables too. However, it appears that these high-degree tables are in the core of the schema, and thus attract maintenance attention and expansion, while, at the same time, they are mostly internally modified, without the changes (most of the times) affecting the relationship to other tables. An in-depth study of these tables, found in the supplementary material of the paper, suggests that *the reason for the high update rate of the high-degree, active tables is an unusual combination of information capacity expansion and perfective maintenance, the latter being expressed via data type updates as well as do-undo actions*.

The statistical evidence in favor of the different behavior of tables, depending on their degree is very strong. The Chi-square and Fisher tests that we performed returned p-values ranging between 2.56E-07 for Slashcode (the lowest) and 0.037 for Atlas (the highest). In other words, all these low p-values indicate that different degree ranges are correlated with different activity patterns.

5.3 Total Degree and its relationship to the combination of Survival and Activity (LifeAndDeath Class)

Research Question: is there a relationship between the Total Degree at the Diachronic Graph of a table and its combination of survival and activity?

The concise answer to the above question is that *whereas the set of tables with DegreeDG lower than 3 includes almost all the dead tables and the larger part of the survivors with a strong tendency to rigidity (especially for the dead) or a quiet life (depending on the data set), the few tables with degree higher than 2 are overwhelmingly survivors, and, unusually active, typically undergoing medium or high update activity*. In the rest of this subsection, we detail how we can support these claims with concrete numbers from our findings.

To address the combination of survival and activity we introduce the *LifeAnd-Death (LAD)* class produced by the Cartesian product of *survival* $\times$ *activity*, i.e., by the combination of {rigid, quiet, active}$\times${dead, survivor}. Concerning total degree at DG, we group tables in (a) 0 degree, (b) degree within 1 and 2, and (c) degree higher than 2.

The graphical representation of the data that we will use to answer the research question is shown in Fig. 8 (right). We see two families of data sets: (a) the scientific ones, Atlas and BioSQL with a large number of high degree tables, a large number of active survivors, and a fairly small number of tables with zero degree, and, (b) the rest of the data sets, with a fairly large number of zero-degree nodes, and rather small numbers for (i) high-degree nodes and (ii) active tables.

To statistically test whether different degrees demonstrate a different breakdown in terms of Survival and Activity, we have computed the contingency matrices for LAD values and degree ranges, for each data set. For lack of space, we can only present the averaged values of these contingency tables, (Figure 10). The statistical evidence in favor of the different behavior of tables depending on their different degree is fairly strong. For the data sets without a column with zero expected frequency, the Chi-square and Fisher tests that we performed returned p-values ranging between 5.8E-07 for Slashcode (the lowest) and 0.046 for Atlas (the highest). For the two data sets that include an Active Dead column with all zero's (Castor and Zabbix), the modified chi-square test returned p-values 0.022 and 0.035 respectively. tand the Fischer test, 0.107 and 0.068, respectively. *Overall, all these low p-values make us quite confident to suggest that different degree ranges are correlated with different Survival* $\times$ *Activity patterns*.

In the sequel, we list out findings based on the data of all the aforementioned figures.

Dead tables. We know from previous research that dead tables are typically fewer (often, way fewer) than survivors [24, 26] and that they typically gravitate to rigidity [23]. This is confirmed by the data of all our data sets. Here, we also see the following extra patterns: *In all occasions, rigid dead are more than quiet dead, who are more than (the actually too few) active dead.* Moreover, in all but 5 tables in all the studied schemata, *dead tables do not exceed the barrier of 2 in their total degree* (as already mentioned, the probability of survival for high degrees is higher or equal to 84% in all our datasets) and *there is not even one dead table that is both rigid and with degree higher than 2.*

Survivor tables. Survivors are significantly more than the dead tables and they span significantly higher ranges of degrees (Fig. 8 right). Much like dead tables, there is almost no survivor table of high degree that is rigid (except for BioSQL). This confirms *the abstinence of high degree tables from rigidity independently of their survival*. At the same time, these *high-degree tables tend to be survivors to a very large extent, with these survivor high-degree tables being mostly quiet or active*. In fact, with the exception of Zabbix, active high-degree survivors are more than both the quiet and the rigid high-degree survivors: in other words, *high-degree tables are more likely to be active(!)*.

Finally, *the majority of survivors typically lies in the space defined by (a) degree within 0 and 2 and (b) rigid or quiet survivors. Depending on the overall tendency of the data set the gravitation to a category changes*. Specifically, in scientific data sets and in Zabbix, quiet survivors of degree 1-2 are the most populous category, followed by rigid survivors, also of degree 1-2. In the case of castor and Slashcode, we have almost the inverse situation.

5.4 Relationship of the Total Degree of a table with its Birth

Research question: how does the total Degree at the Diachronic Graph of a table, relate to the time of its birth?

We have studied how birth version and DegreeDG are related and, we have observed that there is a different behavior between small and high degrees in terms of their birth:

- *Small degrees (less than 3) are born anytime*
- High degrees are only born very early, or better said, *there is an absence of tables with high degrees being born at medium or late stages of the schema life* (thus, the upper right triangular part of the chart is empty).

The pattern is practically consistent in all data sets – with few exceptions in BioSQL. Fig. 11 gives a statistical description of how infrequent it is to observe high degree tables being born after the original version of the database history.

The statistical support for the difference in the birth behavior of tables with degree higher than two and the rest is very strong. With the exception of Castor that has only 2 such tables, the p-values for the Chi-square and the Fisher tests for the rest of the data sets range from 3.82E-04 (slashcode) to 4.17E-02 (biosql). The contingency tables constructed separated tables as born at v0 of the database history or not and in having degree in [0-2] or higher. Overall, it is clear that, with the exception of BioSQL that has exactly the opposite behavior, *there is*

Average for all five data sets

	RD	QD	AD	RS	QS	AS	Σ
0	7%	2%	1%	18%	13%	1%	*42%*
1-2	6%	6%	0%	8%	15%	6%	*41%*
>2	0%	1%	1%	1%	5%	9%	*16%*
Σ	*14%*	*8%*	*2%*	*27%*	*33%*	*15%*	*100%*

Atlas and BioSQL: average

	RD	QD	AD	RS	QS	AS	Σ
0	2%	0%	1%	2%	3%	1%	*8%*
1-2	12%	8%	1%	9%	20%	11%	*61%*
>2	0%	2%	2%	3%	6%	18%	*31%*
Σ	*14%*	*10%*	*3%*	*14%*	*29%*	*30%*	*100%*

Castor, Slashcode and Zabbix: average

	RD	QD	AD	RS	QS	AS	Σ
0	11%	4%	1%	28%	19%	1%	*65%*
1-2	3%	4%	0%	8%	12%	2%	*29%*
>2	0%	0%	0%	0%	4%	3%	*7%*
Σ	*13%*	*7%*	*2%*	*36%*	*36%*	*5%*	*100%*

Fig. 10 Average values for the breakdown of tables with respect to the combination of (a) their LAD class (i.e., the combination of activity and survival) and (b) their Total Degree at the Diachronic Graph, with respect to (top) all 5 data sets, (b) Atlas and Biosql, and (c) Castor, Slashcode and Zabbix. The 6 labels for the LAD class correspond to the combination of {Rigid, Quiet, Active}x{Dead, Survivor}

Probability of a table being born at v0, if its DegreeDG lies in the range of ...

	#tables	0 - max	0-2	>=3	Tables @ v0 with DegDG >=3
Atlas	88	64%	56%	94%	15 out of 16
Biosql	45	47%	62%	26%	5 of 19 (9 with deg = 3)
Castor	91	68%	67%	100%	2 out of 2
Slashcode	126	33%	29%	100%	8 out of 8
Zabbix	58	26%	18%	86%	6 out of 7

Fig. 11 Probability of a table being born in the original version of the schema history; the column 0 – max denotes the overall probability, independently of degree

strong evidence that high degree tables are not born after the originating version of the database.

Our in-depth study has revealed that the phenomenon is not attributed to the birth of these tables with small degrees and their subsequent "scale-up" in terms of degree. Overall, we can say that few table reach high degrees and these are "*important*" tables created early. In the non-scientific data sets, these high-degree tables are distinguished mostly by their high in-degree, sometimes too large, which signifies that they are "lookup" tables frequently referenced by others. In the two scientific data sets, apart from this type of high-degree tables, there also exist "fact" nodes, referencing others, and "crossroad" nodes that combine both a high in-degree and a high out-degree (yet, with out-degrees not far from the value of 3). We believe this is another demonstration of gravitation to rigidity: as time progresses the tendency to build such tables diminishes. So, (with the clear exception of BioSQL) we see a pattern: *after the originating version of the schema history, the probability of seeing the birth of tables with DegreeDG higher or equal to 3 is very small.*

6 Discussion of results and open roads

Summary. Our findings for the study of foreign key evolution at the schema level, can be summarized as follows. For all the studied data sets, *schemata grow in the long run* in terms of tables. The usual pattern of alternation between periods of slow growth, calmness periods, spikes of extension, and occasional cleanups of the schema is present [20]. In some cases, mainly *in projects of scientific nature, foreign keys are treated as an integral part of the system*, and they are born and evicted along with table birth and eviction. At the same time, we have observed cases where *foreign keys are treated as a second-class add-on*. In these cases, there is a small subset of the tables involved in foreign keys, while birth and eviction of foreign keys is rarely performed in synch with the respective table events. In the case of CMSs we have seen *a disinclination towards having foreign keys as part of the schema*. In the data sets that we have collected, the mere existence of foreign keys is too scarce. Moreover, in the case of the two CMSs that had foreign keys in their lifetime, both *ended-up with their complete removal*. To the best of our understanding this removal was chosen due to difficulty of managing technical issues with foreign keys, that discouraged developers from trying to solve the encountered problems. The *heartbeat of foreign key change is mostly rare and small in volume*: changes of foreign keys are not really frequent and they are typically small in volume (with the exception of do-undo pairs of commits and the aforementioned massive removals).

Our findings for the study of foreign key evolution at the table level, can be summarized as follows. With the strict exception of *high-degree tables, which avoid rigidity*, for the rest of the tables, their breakdown in activity categories follows the breakdown of tables of the entire population. *Dead tables are strongly biased towards being of zero or low degree and rigid*. The majority of survivors typically lies in the space defined by (a) degree within 0 and 2 and (b) rigid or quiet survivors. High-degree tables tend to be survivors to a very large extent. These survivor high-degree tables are mostly quiet or active. With the exception of one dataset, active high-degree survivors are more than half of the high-degree survivors: in other words, *high-degree tables are more likely to be active!* In all occasions, active survivors of zero degree are too few, even in the data sets with overwhelming percentages of zero degree tables. Finally, we also observe that *small degrees (less than 3) are born anytime*, whereas high degrees are only born very early, or better said, *there is an absence of tables with high degrees being born at medium or late stages of the schema life*.

Threats to validity. The *scope* of our study is restricted to databases that are part of FOSS projects (and not closed ones) that have even moderate amounts of versions published on-line and also pay the price for data consistency via foreign keys. The reader should avoid over-generalizing findings to closed projects, or projects with a strict management plan. The *external validity* of our results is, of course restricted within the scope of the study. Whenever we report an observed pattern, we make clear whether it is ubiquitous in our data sets, or to what subset of the data sets it applies. We believe, however, that our study is representative of the reported scope. We have a set of data sets from different domains (occasionally with characteristics that are domain-dependent and which we comment upon) with adequately long stories and schema sizes. Thus, we believe that patterns that appear to be either omni-present or strictly characteristic to a domain can indeed be generalized. In terms of *measurement validity*, we have tested our tools with black box testing and we have fixed any identified problems during their operation. Any processing of the input data is reported above. Although one can never exclude the possibility of occasional errors, we are confident with our results in terms of their measurement validity. We are also very sensitive to the fact that this is the first -to our knowledge- study of its kind, and consequently, it is strictly of exploratory nature. Internal validity concerns are covered by the fact that we restrain ourselves to the retrieved evidence and common knowledge. Still, more targeted experiments are needed to increase our confidence.

All our data sets and software are openly available at our group's site at Github (`https://github.com/DAINTINESS-Group`) for the research community to reuse.

Messages for our community. Even within the limited scope of this study, we have now significant evidence that, unless specifically curated, foreign keys can potentially be unwelcome (and thus, rare) or even completely removed by the developers. *This is a clear warning that we, as a community, need to do better in two battles: (a) in improving the ease of use of DBMS's, and, (b) in teaching. The former goal refers to making DBMS's easier at handling both primary and foreign keys and their implications, especially at the deep technical details. The latter aims at pointing out that the importance of foreign keys and normalization in the effort to maintain data integrity is not adequately passed to new developers. Apart from the theoretical difficulties, the difficulty of handling their practical aspects only reinforces the tendency to avoid foreign keys in certain schema designs. We propose that special emphasis is placed to demonstrate the hazards, and teach why the enforcement of referential integrity outside the schema (e.g., via a client application) can be disastrous, as, unless integrity is enforced at the schema level, there is always a plethora of alternative ways to populate the database with offending data by circumventing the external client checks*.

Follow up. Future work can continue in many directions. More studies, preferably by other groups, over other data sets, need to be performed in an attempt to be able to establish common patterns of evolution. This concerns both conducting broader and deeper studies in the the FOSS domain (to support or disproved the results reported here) and, ideally, in the proprietary domain too (for which, due to lack of data, we have only anecdotal evidence). The particularities of unusual behaviors concerning foreign keys need to be further investigated too. Mining patterns of graph evolution is also another path for future work.

Acknowledgments. We thank the reviewers of both the current and previous papers for their comments that helped us clarify better key points of interest.

References

1. Boslaugh S (2012) Statistics in a nutshell - a desktop quick reference. O'Reilly
2. Cleve A, Brogneaux A, Hainaut J (2010) A conceptual approach to database applications evolution. In: 29th International Conference on Conceptual Modeling (ER 2010), Vancouver, BC, Canada, November 1-4, 2010, pp 132–145
3. Cleve A, Gobert M, Meurice L, Maes J, Weber JH (2015) Understanding database schema evolution: A case study. Sci Comput Program 97:113–121
4. Curino C, Moon HJ, Tanca L, Zaniolo C (2008) Schema evolution in wikipedia: toward a web information system benchmark. In: Proceedings of ICEIS 2008, Citeseer
5. Curino C, Moon HJ, Deutsch A, Zaniolo C (2013) Automating the database schema evolution process. VLDB J 22(1):73–98
6. Decan A, Goeminne M, Mens T (2015) On the interaction of relational database access technologies in open source java projects. In: Post-proceedings of the 8th Seminar on Advanced Techniques and Tools for Software Evolution, Mons, Belgium, July 6-8, 2015., pp 26–35
7. Decan A, Goeminne M, Mens T (2017) On the interaction of relational database access technologies in open source java projects. CoRR abs/1701.00416, URL `http://arxiv.org/abs/1701.00416`
8. García-García J, Ordonez C (2010) Extended aggregations for databases with referential integrity issues. Data Knowl Eng 69(1):73–95
9. Herrmann K, Voigt H, Behrend A, Lehner W (2015) Codel - A relationally complete language for database evolution. In: 19th East European Conference on Advances in Databases and Information Systems (ADBIS 2015), Poitiers, France, September 8-11, 2015., pp 63–76
10. Kagdi HH, Collard ML, Maletic JI (2007) A survey and taxonomy of approaches for mining software repositories in the context of software evolution. Journal of Software Maintenance 19(2):77–131
11. Köhler H, Link S (2018) SQL schema design: foundations, normal forms, and normalization. Inf Syst 76:88–113
12. Lin DY, Neamtiu I (2009) Collateral evolution of applications and databases. In: Proceedings of the Joint International and Annual ERCIM Workshops on Principles of Software Evolution (IWPSE) and Software Evolution (Evol) Workshops, IWPSE-Evol '09, pp 31–40
13. Manousis P, Vassiliadis P, Papastefanatos G (2013) Automating the adaptation of evolving data-intensive ecosystems. In: Proceedings of 32th International Conference on Conceptual Modeling (ER 2013), Hong-Kong, China, November 11-13, 2013., pp 182–196
14. Meurice L, Ruiz FJB, Weber JH, Cleve A (2014) Establishing referential integrity in legacy information systems - reality bites! In: 30th IEEE International Conference on Software Maintenance and Evolution, Victoria, BC, Canada, September 29 - October 3, 2014, pp 461–465

15. Meurice L, Nagy C, Cleve A (2016) Static analysis of dynamic database usage in java systems. In: Advanced Information Systems Engineering - 28th International Conference, CAiSE 2016, Ljubljana, Slovenia, June 13-17, 2016. Proceedings, pp 491–506
16. Papenbrock T, Ehrlich J, Marten J, Neubert T, Rudolph J, Schönberg M, Zwiener J, Naumann F (2015) Functional dependency discovery: An experimental evaluation of seven algorithms. PVLDB 8(10):1082–1093
17. Qiu D, Li B, Su Z (2013) An empirical analysis of the co-evolution of schema and code in database applications. In: Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2013, pp 125–135
18. Sjøberg D (1993) Quantifying schema evolution. Information and Software Technology 35(1):35–44
19. Skoulis I, Vassiliadis P, Zarras A (2014) Open-source databases: Within, outside, or beyond Lehman's laws of software evolution? In: 26th International Conference on Advanced Information Systems Engineering (CAiSE 2014), Thessaloniki, Greece, June 16-20, 2014.
20. Skoulis I, Vassiliadis P, Zarras AV (2015) Growing up with stability: How open-source relational databases evolve. Information Systems 53:363–385
21. Teorey TJ, Yang D, Fry JP (1986) A logical design methodology for relational databases using the extended entity-relationship model. ACM Comput Surv 18(2):197–222
22. Vassiliadis P (2017) Schema evolution and gravitation to rigidity: A tale of calmness in the lives of structured data. In: Proceedings of 7th International Conference Model and Data Engineering - MEDI 2017, Barcelona, Spain, October 4-6, 2017, pp 18–23
23. Vassiliadis P, Zarras AV (2017) Survival in schema evolution: Putting the lives of survivor and dead tables in counterpoint. In: 29th International Conference on Advanced Information Systems Engineering (CAiSE 2017), Essen, Germany, June 12-16, 2017, pp 333–347
24. Vassiliadis P, Zarras AV, Skoulis I (2015) How is life for a table in an evolving relational schema? birth, death and everything in between. In: Proceedings of 34th International Conference on Conceptual Modeling (ER 2015), Stockholm, Sweden, October 19-22, 2015, pp 453–466
25. Vassiliadis P, Kolozoff M, Zerva M, Zarras AV (2017) Schema evolution and foreign keys: Birth, eviction, change and absence. In: Proceedings of 36th International Conference on Conceptual Modeling (ER 2017), Valencia, Spain, November 6-9, 2017, pp 106–119
26. Vassiliadis P, Zarras AV, Skoulis I (2017) Gravitating to rigidity: Patterns of schema evolution - and its absence - in the lives of tables. Information Systems 63:24–46
27. Wu S, Neamtiu I (2011) Schema evolution analysis for embedded databases. In: Proceedings of the 2011 IEEE 27th International Conference on Data Engineering Workshops, ICDEW '11, pp 151–156
28. Zhang M, Hadjieleftheriou M, Ooi BC, Procopiuc CM, Srivastava D (2010) On multi-column foreign key discovery. PVLDB 3(1):805–814