Schema Evolution and Foreign Keys: Birth, Eviction, Change and Absence

Panos Vassiliadis[1(✉)], Michail-Romanos Kolozoff[2], Maria Zerva[1],
and Apostolos V. Zarras[1(✉)]

[1] Department of Computer Science and Engineering,
University of Ioannina, Ioannina, Greece
{pvassil,mzerva,zarras}@cs.uoi.gr
[2] Upcom, Athens, Greece
libathos@hotmail.com

Abstract. In this paper, we focus on the study of the evolution of foreign keys in the broader context of schema evolution for relational databases. Specifically, we study the schema histories of a six free, open-source databases that contained foreign keys. Our findings concerning the growth of tables verify previous results that schemata grow in the long run in terms of tables. Moreover, we have come to several surprising, new findings in terms of foreign keys. 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.

Keywords: Schema evolution · Patterns of change · Foreign keys

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. Foreign keys are mechanisms that constrain data entry in relational tables, imposing that the domain of the contents of a table's attribute

M.-R. Kolozoff—Work done while in the Univ. Ioannina.

H.C. Mayr et al. (Eds.): ER 2017, LNCS 10650, pp. 106–119, 2017.
https://doi.org/10.1007/978-3-319-69904-2_9

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 was without any answer. There are several works on the study of schema evolution [1–6,10], which mostly focus on the macroscopic 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 Sect. 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 Sect. 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 findings, detailed in Sect. 4, include both expected and unexpected phenomena. Schemata grow over time in terms of 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. 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. To our big surprise, foreign keys in these projects, after a period of growth, are *completely removed* from the schema with (a) a steep removal in the first case, and, (b) a slow but constant removal rate in the latter. We make a detailed discussion on the absence and extinction of foreign keys in specific environments in Sect. 5. 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.

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

2 Related Work

The related work on schema evolution is not abundant and, in effect, quite recent. Prior to the proliferation of Free Open Source Software (FOSS), researchers were unable to get access to the histories of schemata that they could study. To our knowledge, the only early study that exists is [5], which reports on the growth and change breakdown of the schema of a health system. Late '00s signaled the slow appearance of a set of works [1–4,10] that continued down this road. A consistent finding in all these works is the slow expansion of the size of the schema in terms of tables, albeit with reports of a decreasing growth rate [4]. Frequently, the surrounding application is not in sync with the underlying schema (which, as a side note, signifies the importance of understanding the mechanics of schema evolution) [3,10]. In line with these works, in [6,7], we have assessed whether Lehman's laws of evolution apply to the case of schema evolution, and confirmed the growth of schemata over time, via the alternation of periods of concentrated modifications (mostly table insertions and occasionally including table removals) and periods of calmness with slow, or even zero growth. In [8,9], we report on patterns of how properties of individual tables (rather than of the schema) like duration, number of attributes, or, version of birth relate to the survival or update profile of a table.

In all previous attempts, 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, the current paper is the first comprehensive effort in the literature to study the evolution of foreign keys.

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*[1], an open source tool we created for the purpose of this study.

3.1 Fundamentals

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 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

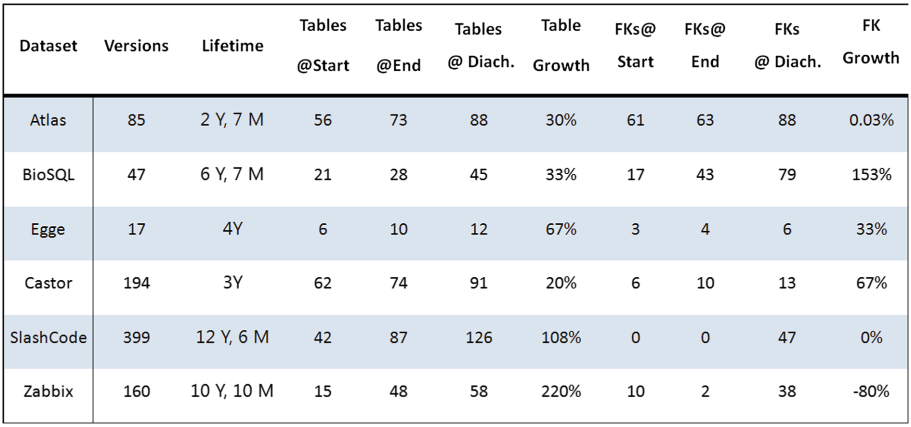
[1] https://github.com/DAINTINESS-Group/ParmenidianTruth.

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 *Diachronic Graph* of the history of a schema is the union of all the nodes and edges that ever appeared in the history of the schema.

The evolution history, $H = \{v^1, \ldots, v^n\}$, of a 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. 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 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.

3.2 Datasets

The main characteristics of the six data sets that we considered in our study are given in Fig. 1. We classify the data sets in three categories as follows.



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.

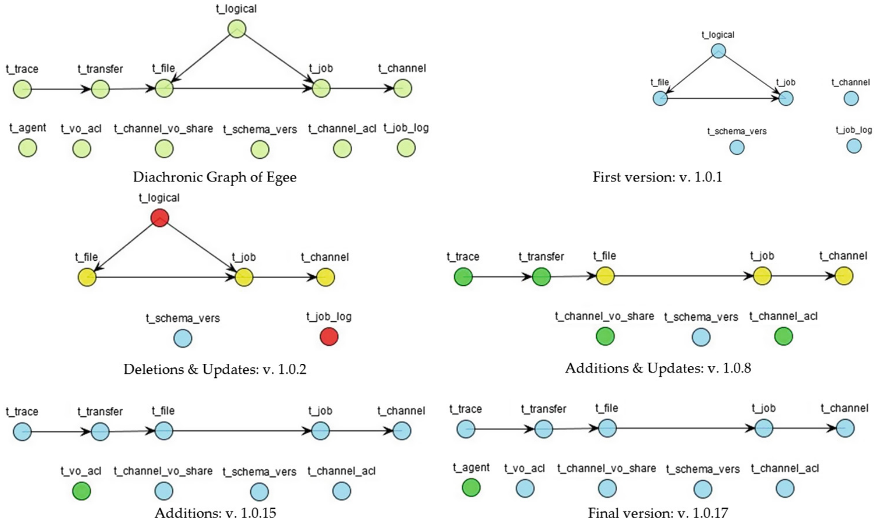
Scientific Applications. *Atlas* is a particle physics experiment at the Large Hadron Collider at CERN, the European Organization for Nuclear Research based on Geneva, Switzerland. Atlas is notably known for its attempt to find the Higgs boson, although its scientific aims are much broader. Trigger is 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. *Biosql* is a generic relational schema that provides unified access to data from various sources, such as GenBank or Swissport that store genomic data like sequences, features, for the BioPerl, BioPython, BioJava, and BioRuby open source toolkits.

Computational Resource Toolkits. *Egee* is a data set from the homonymous EU funded project, whose goal is to provide access to computational grids. Egee is the smallest data set, frequently serving as a testbed, with a small number of releases and a small schema size. *Castor* is a hierarchical storage management (HSM) system developed at CERN, to store physics production files and user files, via command-line tools and APIs.

Content Management Systems. *SlashCode*, a software framework for web sites development; it is widely known for supporting the Slashdot website. *Zabbix* is an open source distributed monitoring solution that can be used for the monitoring of networks, servers and virtual machines. We have used the PostgreSQL version of the schema that includes foreign keys.

3.3 Data Processing

Based on our tool, *Parmenidian Truth*, 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 visualizes each version of the database schema as a graph, with tables as nodes and foreign keys



Diachronic Graph of Egee

First version: v. 1.0.1

Deletions & Updates: v. 1.0.2

Additions & Updates: v. 1.0.8

Additions: v. 1.0.15

Final version: v. 1.0.17

Fig. 2. The story of Egee and its diachronic graph.

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.

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).

4 Growth and Heartbeat of Foreign Key Evolution

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. Figure 3 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 foreign 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.

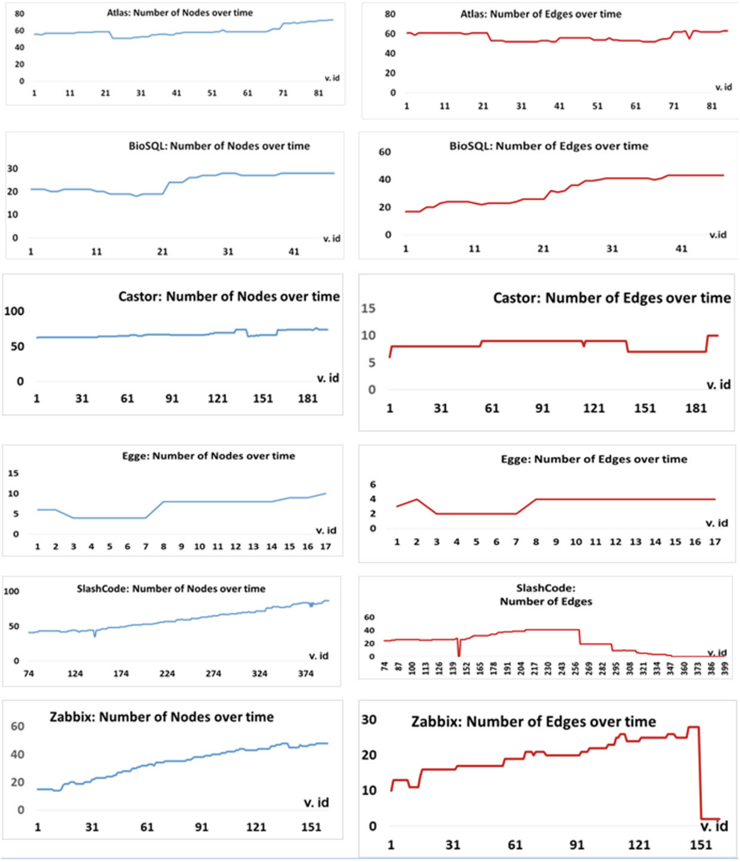
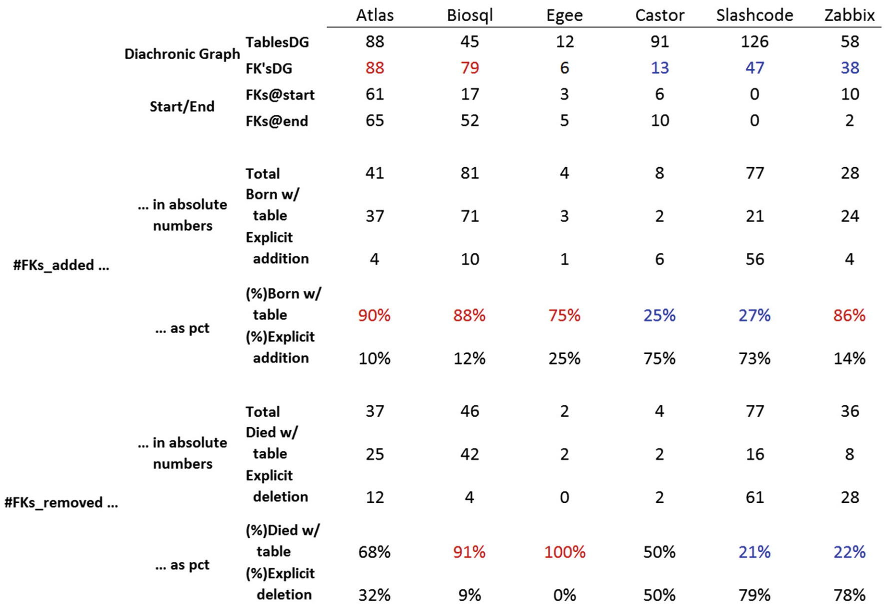


Fig. 3. Number of nodes (tables) and edges (foreign keys) over time (x-axis: version id) for the 6 studied data sets

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 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. 4, 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.



			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. 4. Statistical breakdown on the creation and removal 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. 4). 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 revision 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. 5, (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. 6). Apart from the scientific data sets, where the number of foreign

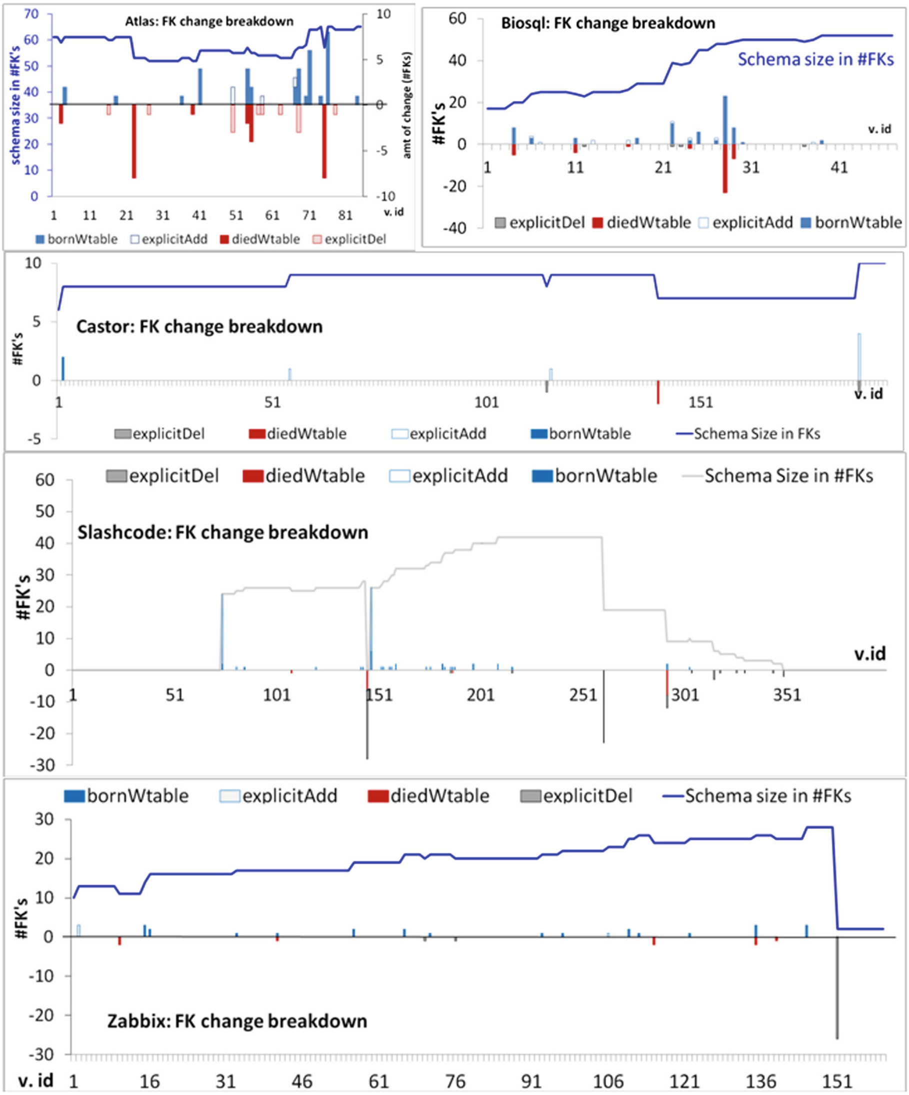
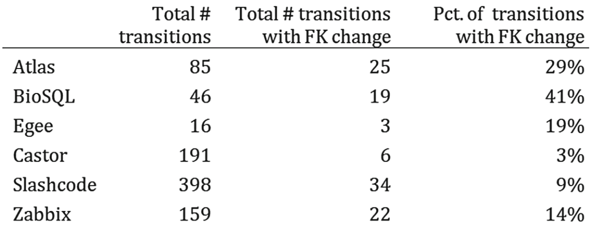


Fig. 5. Heartbeat of foreign key creation and removal



	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. 6. Percentage of transitions containing schema change of foreign keys

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. 4). 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 a revision 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.

5 Where Did the Foreign Keys Go?

5.1 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 mass 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.

5.2 Are Foreign Keys Unwelcome in CMS's?

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.

In summer of 2013, we collected twenty data sets to support our work on mining evolution patterns in the history of open-source databases. 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. 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, as already mentioned. 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 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.

In terms of the CMSs, we believe that the absence of foreign keys in the schema declaration of their database is systematic. Even the two of the 13 CMS's that adopted foreign keys at some point, eventually dropped them. We attribute the phenomenon to the combination of two factors. 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, foreign keys impose a time lag in terms of efficiency, which developers decide not to pay, especially if they operate under the aforementioned impression of data consistency.

6 Conclusions

Summary. Our findings 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 [7]. 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 CMS's 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 CMS's 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).

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 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.

Importance of This Work. To the best of our knowledge, this is the first time that the study of the evolution of foreign keys is performed, and, quite importantly, at a large scale, in terms of data sets. Apart from the increase of our understanding of how schemata evolve over time with solid evidence, the study noteworthily reveals unexpected results. Although it is important not to over-generalize our findings outside the area of Free, Open Source Software that defines the scope of the study, we have now significant evidence that, unless specifically curated, foreign keys in a FOSS database 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 (a) in terms of making systems easier at handling foreign keys and their implications, especially at the deep technical details, as well as, (b) in terms of better educating developers on the benefits and necessities behind the usage of foreign keys in their databases.*

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. The particularities of unusual behaviors concerning foreign keys need to be further investigated too. Mining patterns of graph evolution in the graph of foreign keys is also another path for future work.

References

1. Cleve, A., Gobert, M., Meurice, L., Maes, J., Weber, J.H.: Understanding database schema evolution: a case study. Sci. Comput. Program. **97**, 113–121 (2015)
2. Curino, C., Moon, H.J., Tanca, L., Zaniolo, C.: Schema evolution in Wikipedia: toward a web information system benchmark. In: Proceedings of ICEIS 2008. Citeseer (2008)
3. Lin, D.Y., Neamtiu, I.: Collateral evolution of applications and databases. In: Proceedings of Joint International and Annual ERCIM Workshops on Principles of Software Evolution (IWPSE) and Software Evolution (Evol) Workshops, IWPSE-Evol 2009, pp. 31–40 (2009)
4. Qiu, D., Li, B., Su, Z.: An empirical analysis of the co-evolution of schema and code in database applications. In: Proceedings of 2013 9th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2013, pp. 125–135 (2013)
5. Sjøberg, D.: Quantifying schema evolution. Inf. Softw. Technol. **35**(1), 35–44 (1993)
6. Skoulis, I., Vassiliadis, P., Zarras, A.: Open-source databases: within, outside, or beyond Lehman's laws of software evolution? In: Proceedings of 26th International Conference on Advanced Information Systems Engineering - CAiSE 2014 (2014)
7. Skoulis, I., Vassiliadis, P., Zarras, A.V.: Growing up with stability: how open-source relational databases evolve. Inf. Syst. **53**, 363–385 (2015)
8. Vassiliadis, P., Zarras, A., Skoulis, I.: Gravitating to rigidity: patterns of schema evolution - and its absence - in the lives of tables. Inf. Syst. **63**, 24–46 (2017)
9. Vassiliadis, P., Zarras, A.V., Skoulis, I.: 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, 19–22 October 2015, pp. 453–466 (2015)
10. Wu, S., Neamtiu, I.: Schema evolution analysis for embedded databases. In: Proceedings of 2011 IEEE 27th International Conference on Data Engineering Workshops, ICDEW 2011, pp. 151–156 (2011)