A Field Analysis of Relational Database Schemas in Open-source Software

Fabien Coelho, Alexandre Aillos, Samuel Pilot, and Shamil Valeev
CRI, Maths & Systems, MINES ParisTech
35, rue Saint Honoré, 77305 Fontainebleau cedex, France.
fabien.coelho@mines-paristech.fr, *firstname.lastname*@mines-paris.org

***Abstract*—The relational schemas of 407 open-source projects storing their data in MySQL or PostgreSQL databases are investigated by querying the standard *information schema*, looking for various issues. These SQL queries are released as the *Salix* free software. As it is fully relational and relies on standards, it may be installed in any compliant database to help improve schemas. The overall quality of the surveyed schemas is poor: a majority of projects have at least one table without any primary key or unique constraint to identify a tuple; data security features such as referential integrity or transactional back-ends are hardly used; projects that advertise supporting both databases often have missing tables or attributes. PostgreSQL projects have a better quality compared to MySQL projects, and it is even better for projects with PostgreSQL-only support. However, the difference between both databases is mostly due to MySQL-specific issues. An overall predictor of bad database quality is that a project chooses MySQL or PHP, while good design is found with PostgreSQL and Java. The few declared constraints allow to detect latent bugs, that are worth fixing: more declarations would certainly help unveil more bugs. Our survey also suggests some features of MySQL and PostgreSQL as particularly error-prone. This first survey on the quality of relational schemas in open-source software provides a unique insight in the data engineering practice of these projects.**

***Keywords*—open-source software; database quality survey; automatic schema analysis; relational model; SQL.**

I. Introduction

In the beginning of the computer age, software was freely available, and money was derived from hardware only. Then in the 70s it was *unbundled* and sold separately. Stallman initiated the free software movement, which is now quite large [1], to implement his principle of sharing software, Such free software is distributed under a variety of licenses. The common ground is that it must be available as source code to allow its study, change and improvement, as opposed to compiled or obfuscated, hence the expression *open source*. This induces many technical, economical, legal, and philosophical issues. Open-source software (OSS) is a subject of academic studies in psychology, sociology, economics, or software engineering, including quantitative surveys. Developers' motivation [2], organization [3] and profiles [4] are investigated; Quantitative studies exist about code quality in OSS [5][6][7][8] and its dual, static analysis to uncover bugs [9][10]. Database surveys are available about market shares, or server exposure security issues [11]. This study is the first survey on the quality of relational database schemas in OSS. It provides a unique insight in the data engineering practice of these projects.

Codd's relational model [12] is an extension of the set theory to relations (tables) with attributes (columns) in which tuple elements are stored (rows). Elements are identified by keys, which can be used by tuples to reference one another between relations. The relational model is sound, as all questions (in the model) have corresponding practical answers and *vice versa*: the tuple relational calculus describes questions, and the mathematically equivalent relational algebra provides their answers. It is efficiently implemented by many commercial and open-source software such as Oracle, DB2 or SQLite. The *Structured Query Language* (SQL) is available with most relational database systems, although the detailed syntax often differs. The standardization effort also includes the *information schema* [13], which provides meta data about the schemas of databases through relations.

The underlying assumption of our study is that applications store precious transactional user data, thus should be kept consistent, non redundant, and easy to understand. We think that database features such as key declarations, referential integrity and transaction support help achieve these goals. In order to evaluate the use of database features in open-source software, and to detect possible design or implementation errors, we have developed a tool to analyze automatically the database structure of an application by querying its *information schema* and generating a report, and we have applied it to 407 open-source projects. Following MacCabe's metric to measure program complexities [14], several metrics address data models [15][16] or database schemata either in the relational [17][18] or object relational [19] models. These metrics rely on information not necessarily available from the database concrete schemas. We have rather followed the dual and pragmatic approach [20], which is not to try to do an absolute and definite measure of the schema, but rather to uncover issues based on static analyses. Thus the measure is relative to the analyses performed and results change when more are added.

Section II presents the methodology used in this study. We describe our tool, our grading strategy and the statistical validation used on the assertions derived from our analyses. Section III lists the projects by category and technology, and discusses similarities and differences depending on whether they run on MySQL or PostgreSQL. Section IV describes the results of our survey. The overall quality of projects is quite poor, as very few database schemas do not raise error-rated advices. Section V gives our conclusive thoughts.

II. Methodology

Our *Salix* automatic analyzer is based on the *information schema*. We discuss the queries, then describe the available advices, before presenting the statistical validation used.

A. Information schema queries

Our analyses are performed automatically by SQL queries on the databases meta data using the standard *information schema*. This relational schema stores information about the databases structure, including catalogs, schemas, tables, attributes, types, constraints, roles, permissions... The set of SQL queries used for this study are released as the *Salix* free software. It is based on `pg-advisor` [21], a PostgreSQL-specific proof of concept prototype developed in 2004. Some checks are inspired by [22][23][24] or similar to [25] others. Our tool creates a specific table for every advice by querying the *information schema*, and then aggregates the results in summary tables. It is fully relational in its conception: there is no programming other than SQL queries. The development of *Salix* uncovered multiple issues with both implementations of the *information schema*.

B. Advice classification and project grading

The 47 issues derived by our SQL queries on the standard *information schema* are named ***advices***, as the user is free to ignore them. Although the performed checks are basic and syntactic, we think that they reflect the quality of the schemas. Each advice has a category (19 design, 13 style, 6 consistency, 4 version, 5 system), a severity (7 errors, 21 warnings, 14 notices, 5 informations), and a level (1 raised per database, 10 per schema, 27 per relation, 7 per attribute, 2 per role). The severity classification is arbitrary and must be evaluated critically by the recipient: most of them should be dealt with, but in some cases they may be justifiable. Moreover, detected errors do not imply that the application is not functional.

The 19 **design** advices focus on detecting design errors. Obviously, semantic error, say an attribute is in the wrong relation, cannot be guessed without understanding the application and thus are out of reach of our automatic analysis. We rather focus on primary and foreign key declarations, or warn if they are missing. The rate of non-null attributes is also checked, with the underlying assumption from our experience that most data are mandatory in a relation. We also check the number of attributes so as to detect a possible insufficient conception effort.

The 13 **style** advices focus on relation and attribute names. Whether a name is significant in the context cannot be checked, so we simply look at their length. Short names are discouraged as they would rather be used as aliases in queries, with the exception of `id` and `pk`. We also check that the same name does not represent differently typed data, to avoid confusing the user.

The 6 **consistency** advices checks for type and schema consistency in a project, such as type mismatches between a foreign key and the referenced key. As databases may also implements some of these checks, it is possible that some cases cannot arise.

The 4 **version** advices focus on database-specific checks, such as capabilities and transaction support, as well as homogeneous choices of back-end engines in a project. This category could also check the actual version of a database used looking for known bugs or obsolescence. Only MySQL-specific version advices are currently implemented.

Finally, the 5 **system** advices, some of which PostgreSQL-specific, check for weak passwords, and key and index issues.

These advices aim at helping the schema developer to improve its relational design. We also use them in our survey to grade projects with a mark from 0 to 10, by removing points each time an advice is raised, taking more points if the severity is high. The grading process is normalized using the number of possible occurrences, so that larger projects do not receive lower marks just because of the likelihood of having more issues for their size. Also, points are not removed twice for the same issue: for instance, if a project does not have a single foreign key, the same issue will not be raised again on every tables. Advices not relevant to our open-source database schema survey, *e.g.*, weak password checks, were deactivated.

C. Survey statistical validation

The data collected suggest the influence of some parameters on others. These results deal with general facts about the projects (say foreign keys are more often used with PostgreSQL) or about their grading (say MySQL projects get lower marks). In order to determine significant influences, we applied Pearson's chi-square tests to compute probabilistic degrees of certainty. Each checked assertion is labeled with an expression indicating the degree of certainty of the influence of one parameter on another:

very sure The probability is 1% or less to get a result as or more remote from the average. Thus we conclude that there is an influence, with a very high degree of certainty.

rather sure The probability of getting such a result is between 1% and 5% (the usual statistical threshold). Thus there is an influence, with a high degree of certainty.

marginally sure The probability is between 5% and 25%: Such a result may have been obtained even if there is no influence. The statement must be taken with a pinch of salt.

not sure The probability is over 25%, or there is not enough available data to compute it. The test cannot asserts that there is a significant influence.

The rational for choosing Pearson's chi-square test is that it does not make any assumption about the distribution of values. However, it is crude, and possibly interesting and somehow true results may not be validated. Moreover, the test requires a minimal population, which is not easily reached on our small data set especially when criteria are crossed. Finally, it needs to define distinct populations: for grades or sizes, these populations are cut at the median value in order to perform the test on balanced partitions.

We also computed a correlation matrix to look for possible inter-parameter influence. The result suggested that the parameters are pretty independent beyond the obvious links (say the use of a non-transactional back-end is correlated with isolated tables), and did no help uncover significant new facts.

Category	Total	%	My	%	Pg	%	both	%	tabs	atts
CMS	70	17.2	61	20.2	1	3.7	8	10.3	39.2	6.6
System	36	8.8	17	5.6	1	3.7	18	23.1	28.1	11.9
Blog	24	5.9	20	6.6	0	0.0	4	5.1	28.2	7.1
Market	21	5.2	20	6.6	0	0.0	1	1.3	55.0	7.6
Project	20	4.9	11	3.6	3	11.1	6	7.7	29.7	7.2
Forum	19	4.7	17	5.6	0	0.0	2	2.6	23.1	8.3
Accounting	16	3.9	9	3.0	6	22.2	1	1.3	93.1	8.4

TABLE I
MAIN CATEGORIES OF PROJECTS, WITH COUNTS, DATABASE SUPPORT AND SIZES

Technology	Total	%	My	%	Pg	%	both	%	tabs	atts
PHP	311	76.4	262	86.8	7	25.9	42	53.8	32.2	7.3
C	34	8.4	9	3.0	5	18.5	20	25.6	22.4	11.7
Java	18	4.4	7	2.3	5	18.5	6	7.7	57.3	9.4
Perl	18	4.4	9	3.0	4	14.8	5	6.4	50.5	7.1

TABLE II
MAIN TECHNOLOGIES OF PROJECTS, WITH COUNTS, DATABASE SUPPORT AND SIZES

III. PROJECTS

We discuss the projects considered in this study, grouped by categories, technologies, sizes and release dates. We first present how projects were selected, and then an overview.

A. *Project selection*

We have downloaded 407 open-source projects starting in the first semester of 2008, adding to our comparison about every project that uses either MySQL or PostgreSQL that we could find and install with reasonable time and effort. The database schemas included in this study are derived from a dump of the database after installation, or from the creation statements when found in the sources. These projects were discovered from various sources: lists and comparisons of software on Wikipedia and other sites; package dependencies from Linux distributions requiring databases; security advisories mentioning SQL; searches on SourceForge.

Some projects were fixed manually because of various issues, such as: the handling of double-dash comments by MySQL, attribute names (*e.g.*, out) rejected by MySQL, bad foreign key declarations or other incompatibilities detected when the projects were forced to use the InnoDB back-end instead of MyISAM, or even some PostgreSQL table definitions including a MySQL specific syntax that were clearly never tested. A particular pitfall of PostgreSQL is that by default syntax errors in statements from an SQL script are ignored and the interpreter simply jumps to the next statement. When installing a project, the flow of warnings often hides these errors. Turning off this feature requires modifying the script, as no command option disables it. More than a dozen PostgreSQL projects contained this kind of issues, which resulted in missing tables or ignored constraint declarations.

B. *Overview of projects*

We have studied the relational schemas of 407 (see [26] for the full list) open-source projects based on databases: 380 of these run with MySQL, 105 with PostgreSQL, including 78 on both. A project supporting PostgreSQL is very likely to support also MySQL (74%), although the reverse is not true (only 20%) *(very sure)*, outlining the relative popularity of these tools. Only 27 projects are PostgreSQL specific. Although there is no deliberate bias in the selection process described in the previous section, where we aimed at completeness, some implicit bias remains nevertheless: for instance, as we can speak mostly English and French, we found mostly international projects advertised in these tongues; Table I shows main project categories, from the personal mundane (game, homepage) to the professional serious (health-care, accounting, system). Table II shows the same for project technologies. Projects in rare categories or using rare technologies do not appear in these cut-off tables. The result is heavily slanted towards PHP web applications (76%), which seems to reflect the current trend of open-source programming as far as the number of projects is concerned, without indication of popularity or quality. The ratio of PHP projects increases from PostgreSQL only support (25%) to both database support (53%) *(rather sure)* to MySQL only support (86%) *(very sure)*: PHP users tend to choose specifically MySQL.

The survey covers 16104 tables (MySQL 11303, PostgreSQL 4801) containing 139092 attributes (MySQL 93960, PostgreSQL 45132). The project sizes in tables average at 33.2, median 17 (from 1 to 607 tables), with 2 to 10979 attributes. MySQL projects average at 30 tables, median 16 (from 1 to 466), with 247 attributes (from 2 to 9725), while PostgreSQL projects average 46 tables, median 20 (from 1 to 607), with 430 attributes (from 6 to 10979 attributes). The largest MySQL project is OSCARMCMASTER, and the largest PostgreSQL project is ADEMPIERE. Detailed table counts raise from projects with MySQL only support (average 28.3, median 16), to both databases (average 34.5, median 18) or PostgreSQL only (average 81.1, median 31). MySQL-only projects are smaller than other projects *(marginally sure)*: more ambitious projects seem to use feature-full but maybe less easy to administrate PostgreSQL. However obvious this assertion would seem, the statistical validation is weak because of the small number of projects with PostgreSQL. MySQL projects that use the InnoDB back-end are much larger that

Advice	*Lvl.*	*Cat.*	*Sev.*	MySQL				PostgreSQL			
				Proj	*%*	*Adv*	*%*	*Proj*	*%*	*Adv*	*%*
Schema without any FK	sch.	design	error	339	89	339	89	58	55	58	55
Tables without PK nor Unique	table	design	error	218	57	1272	11	67	63	917	19
FK type mismatch	table	consist.	error	2	0	17	0	10	9	153	3
Backend engine inconsistency	sch.	version	error	26	6	26	6	0	0	0	0
FK length mismatch	table	consist.	error	3	0	5	0	1	0	3	0
Integer PK but no other key	table	design	warn	345	90	6119	54	89	84	2062	42
Homonymous heterogeneous attributes	att.	style	warn	242	63	1998	2	66	62	477	1
Unsafe backend engine used in schema	sch.	version	warn	338	88	338	88	0	0	0	0
Isolated Tables	table	design	warn	25	6	857	7	35	33	1120	23
Tables without PK but with Unique	table	design	warn	98	25	341	3	15	14	40	0
Unique nullable attributes	att.	design	warn	58	15	226	0	22	20	166	0
Nullable attribute rate over 80%	sch.	design	warn	24	6	24	6	21	20	21	20
Redundant indexes	table	system	warn	0	0	0	0	22	20	192	3
Attribute name length too short	att.	style	warn	22	5	65	0	15	14	46	0
Large PK referenced by a FK	table	design	warn	9	2	99	0	15	14	178	3
Composite Foreign Key	table	design	warn	5	1	19	0	8	7	26	0
Table name length too short	table	style	warn	9	2	10	0	6	5	15	0
FK not referencing a PK	table	design	warn	1	0	12	0	7	6	23	0
Redundant FK	table	system	warn	1	0	1	0	2	1	6	0
Non-integer Primary Key	table	design	note	214	56	1950	17	66	62	1565	32
Attribute count per table over 20	table	design	note	188	49	535	4	54	51	356	7
MySQL is used	base	version	note	380	100	380	100	0	0	0	0
Tables with Composite PK	table	design	note	164	43	1583	14	54	51	636	13
Attribute name length quite short	att.	style	note	159	41	609	0	42	40	198	0
Attribute named after its table	att.	style	note	100	26	1473	1	35	33	4767	10
Table without index	table	system	note	0	0	0	0	53	50	660	13
Nullable attribute rate in 50-80%	sch.	design	note	62	16	62	16	24	22	24	22
Table name length quite short	table	style	note	56	14	81	0	24	22	47	0
Table with a single attribute	table	design	note	59	15	360	3	22	20	48	0
Mixed attribute name styles	table	style	note	89	23	752	6	1	0	37	0
Mixed table name styles	sch.	style	note	31	8	100	26	3	2	4	3
Attribute name length short	att.	style	info	267	70	2240	2	69	65	618	1
Unsafe backend engine used on table	table	version	info	338	88	8746	77	0	0	0	0
Nullable attribute rate in 20-50%	sch.	design	info	107	28	107	28	39	37	39	37
Table name length short	table	style	info	99	26	197	1	32	30	70	1

TABLE III
LIST OF RAISED ADVICES AND DETAILED COUNTS ABOUT THE 407 PROJECTS

their MyISAM counterpart *(very sure)* and are comparable to projects based on PostgreSQL, with 58 tables on average. The number of attributes per table is comparable although smaller for MySQL (average 8.3 – median 7.0) with respect to PostgreSQL (average 9.4 – median 7.0).

The per-category tables *(tabs)* and attributes-per-table *(atts)* counts shows that *accounting*, *health-care* and *market* projects are more ambitious than other categories *(marginally sure)*. The per-technology analysis counts suggests that *Perl*, *Python* and *Java* projects are larger than those based on other technologies *(rather sure)*.

These projects are mostly recent, taking their status at an arbitrary common reference date chosen as March 31, 2009: 257 (63%) were updated in the last year, including 141 (34%) in the last six months, and the others are either obsolete or very stable. The rate of recent projects raises from MySQL-only projects (57%) to projects with both support (79%) *(very sure)* or with PostgreSQL support at (81%) *(very sure)*. However there is no significant difference on the recent maintenance figure between projects that are PostgreSQL-only and projects with both databases support. Projects that include PostgreSQL support were updated more recently that those based on MySQL only.

IV. SURVEY RESULTS

We now analyze the open-source projects of our survey by commenting actual results on MySQL and PostgreSQL, before comparing them. Table III summarizes the advices raised for MySQL and PostgreSQL applications. The first four columns give the advice title, level, category and severity. Then four columns for each database list the results. The first two columns hold the number of projects (*i.e.* schema) tagged and the overall rate. The last two columns give the actual number of advices and rate, which varies depending on the level. A per-project aggregate is also available online [26].

A. Primary keys

A majority of MySQL projects (218 – 57%) have at least one table without neither a primary key nor a unique constraint, and this is even worse with PostgreSQL projects (67 – 63%). The certainty of the observation *(marginally sure)* on MySQL-only vs PostgreSQL-only is low because of the small number of projects using the later. As 11% of all MySQL tables and 19% of all PostgreSQL tables do not have any key, the view of relations as sets is hindered as tuples are not identified, and data may be replicated without noticing.

A further analysis gives some more insight. For MySQL, 42% of tables without key do have some `KEY` option for indexes, but without the `UNIQUE` or `PRIMARY` keyword that makes it a key. Having `KEY` not always declaring a key was clearly a bad design choice. A little 2% of tables without key have an *auto increment* attribute, which suggest uniqueness in practice, but is not enforced. Also, the missing key declaration often seems to be composite. Some tables without key declarations are intended as one tuple only, say to check for the version of the schema or configuration of the application. Similarly, 28% of PostgreSQL tables without key have an index declared. Moreover, 19% have a `SERIAL` (auto incremented) attribute: Many designers seem to assume wrongly that `SERIAL` implies a key. A comment found in the SQLGREY project source suggests that some keys are not declared because of MySQL key size limits.

A simple integer primary key is provided on 60% of tables, with a significantly decreasing rate from MySQL-only (65%) to both database support (59%) *(very sure)* down to PostgreSQL-only support (39%) *(very sure)*. If these primary keys were non-semantic numbers to identify tuples, one would expect at least one other key declared on each table to identify the underlying semantic key. However it is not the case: most (84%) of these tables do not have any other key. When a non simple primary key is available, it is either based on another type or a composite key. The composite keys are hardly referenced, but as the foreign keys are rarely declared one cannot be sure, as shown in the next section.

B. Referential integrity

Foreign keys are important for ensuring the data consistency in a relational database. They are supported by PostgreSQL, and by MySQL but with some back-end engines only. In particular, the default MyISAM back-end does not support foreign keys, and this feature was deemed noxious in previous documentations: Version *3.23* includes a *Reasons NOT to Use Foreign Keys constraints* Section arguing that they are only useful to display diagrams, hard to implement and terrible for performance. Foreign key constraints are introduced with the InnoDB engine starting with *MySQL 3.23.44* in January 2001. Although the constraints are ignored by the default MyISAM engine, the syntax is parsed, and triggers the creation of indexes. Version *5.1* documentation has a *Foreign Keys* Section praising the feature, as it *offers benefits*, although it slows down the application. Caveats describe the inconsistencies that may result from *not* using transactions and referential integrity. From a pedagogical perspective, this is a progress.

Foreign key constraints have long been a missing or avoided feature in MySQL and this seems to have retained momentum in many projects, as it is not supported by the default engine: few MySQL projects (41 – 10% of all projects, 60% of those with InnoDB) use foreign key constraints. The foreign key usage rate is significantly higher (22%) when considering projects supporting both databases *(marginally sure)*.

Among MySQL projects, 312 (82%) use only the default MyISAM back-end engine, thus do not have any foreign key checks enabled. In the remainder, 42 (11%) use only InnoDB, and 26 (6%) use a combination of both. More projects (20 – 25%) rely on InnoDB among those supporting both MySQL and PostgreSQL *(rather sure)*. A third of InnoDB projects (26 – 38%) are not consistent in their engine choice: 35% of tables use MyISAM among the 68 InnoDB projects. A legitimate reason for using MyISAM tables in an InnoDB project is that `FULLTEXT` indexes are only available with the former engine. However, this only applies to 11 tables in 6 projects, all other 1403 MyISAM tables in InnoDB projects are not justified by this argument. A project may decide to store transient data in an unsafe engine (*e.g.*, memory) for performance reason and possibly without any risk of losing data, but this optimization is beyond our tool and is reported as an error. This case is rare, as it represents only 13 tables in 6 projects. About 26% of tables use MyISAM as a default implicit choice in InnoDB projects, similar to 26% when considering all MySQL projects. Some engine inconsistencies seems due to forgotten declarations falling back to the default MyISAM engine.

We have forced the InnoDB back-end engine for all MySQL projects: 22 additional projects declare 92 new foreign key constraints previously ignored. These new foreign keys are very partial, targeting only some tables. They allow to uncover about two dozen issues, either because the foreign key declaration were failing (say from type errors detected by MySQL) or thanks to analyses from our tool. Additional checks based on foreign keys cannot be raised on schemas that do not declare any of them. Thus *isolated tables* warnings must be compared to the number of projects that do use referential constraints: 25 – 60% of these seem to have forgotten at least some foreign keys, and it is actually the case by checking some of these projects manually.

The foreign key usage is better with PostgreSQL projects, although it is still a minority (47 projects – 44%). This rate is close to the foreign key usage of MySQL projects when considering InnoDB projects only. It gives a better opportunity for additional advices to be checked. The foreign key usage rate raises significantly to 78% when considering PostgreSQL-only projects vs dual support projects *(very sure)*.

On the very few projects with partial foreign key declarations, several of these declaration reveal latent bugs, including type mismatch, typically `CHAR` targeting a `VARCHAR` or vice-versa, or different integers, and type length mismatch, usually non matching `VARCHAR` sizes. There are 22 such bugs found out of the small 1387 declared MySQL attribute constraints, and 156 among the 3861 PostgreSQL constraints. There are also 130 important warnings related to foreign keys raised for MySQL, and 227 for PostgreSQL. If this ratio of errors is projected on a the number of tables involved, hundreds additional latent bugs could be detected if the developers were to declare the referential constraints.

C. Miscellaneous issues

More issues were found about style, attribute constraints and by comparing projects with dual database support.

There is 10679 noticeable style issues raised from our analyses (5088 for MySQL, 5591 for PostgreSQL), relating

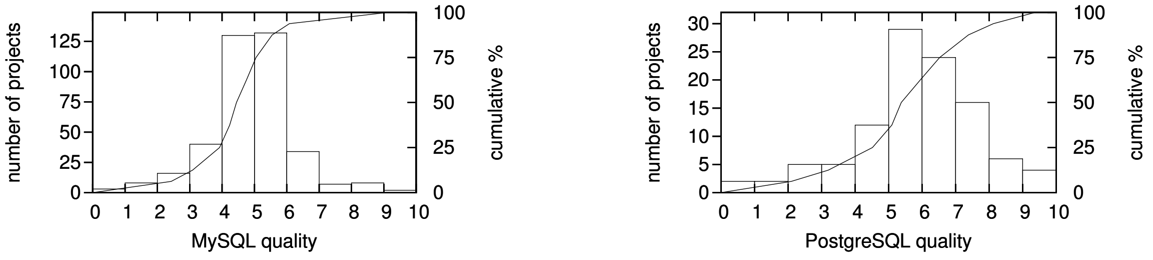

TABLE IV
QUALITY PER DECILE

MySQL projects

Size	nb	avg σ	min	med	max
small	134	4.6 ± 1.3	0.5	4.5	9.1
medium	132	4.3 ± 1.2	0.0	4.4	8.7
large	114	4.4 ± 1.3	0.0	4.5	8.2

PostgreSQL projects

Size	nb	avg σ	min	med	max
small	37	5.1 ± 1.9	0.0	5.2	9.4
medium	27	5.7 ± 1.6	2.0	5.3	9.3
large	41	5.4 ± 2.0	0.0	5.7	9.1

TABLE V
QUALITY PER SIZE

MySQL projects

Category	nb	avg σ	min	med	max
project	17	4.5 ± 1.1	1.4	4.8	6.2
system	35	4.4 ± 1.4	0.0	4.7	7.0
blog	24	4.5 ± 0.9	2.5	4.5	7.2
forum	19	4.3 ± 0.9	2.4	4.4	5.7
cms	69	4.3 ± 0.9	0.5	4.3	6.3
market	21	4.0 ± 1.4	1.9	4.3	8.2

PostgreSQL projects

Category	nb	avg σ	min	med	max
accounting	7	6.1 ± 2.3	2.0	6.5	9.1
cms	9	6.4 ± 1.3	4.1	6.2	8.1
irc	7	5.4 ± 1.7	2.0	5.7	7.4
project	9	5.8 ± 1.5	4.4	5.3	9.3
system	19	4.9 ± 1.7	2.0	5.0	9.0
mail	8	4.9 ± 1.6	3.0	4.8	7.5

TABLE VI
QUALITY PER PROJECT MAIN CATEGORIES

MySQL projects

Techno.	nb	avg σ	min	med	max
java	13	4.7 ± 2.4	0.0	5.2	7.8
c	29	4.7 ± 1.5	2.0	4.7	8.4
perl	14	4.1 ± 2.1	0.5	4.6	8.7
php	304	4.4 ± 1.1	0.0	4.4	8.4

PostgreSQL projects

Techno.	nb	avg σ	min	med	max
java	11	6.0 ± 2.6	0.0	6.5	9.3
perl	9	5.8 ± 1.8	2.0	6.1	8.1
php	49	5.2 ± 1.7	0.0	5.4	8.2
c	25	5.0 ± 1.7	2.0	5.1	9.0

TABLE VII
QUALITY PER PROJECT MAIN TECHNOLOGIES

MySQL projects

Date	nb	avg σ	min	med	max
recent	125	4.4 ± 1.2	1.3	4.4	8.4
older	255	4.4 ± 1.3	0.0	4.4	9.1

PostgreSQL projects

Date	nb	avg σ	min	med	max
recent	52	5.3 ± 1.6	0.0	5.3	9.3
older	53	5.5 ± 2.0	0.0	5.7	9.4

TABLE VIII
QUALITY PER PROJECT RELEASE DATE

to table or attribute names, including a number of one-letter attribute names or two-letters table names. The *id* attribute name is used in the SLASH project with up to 6 different types, mixing various integers and fixed or variable length text types. In PHPETITION, a *date* attribute has types `DATE`, `DATETIME` or `VARCHAR`. 80% of MySQL projects and 79% of PostgreSQL have such style issues.

Many projects does not bother with `NOT NULL` attribute declarations: 86 MySQL projects (22%) and 45 PostgreSQL projects (42%) have over half of their attributes null-able. This does not reflect the overall use of constraints: for MySQL, the average number of key-related constraints per table is 1.06 (from KANNEL 0.00 to OPENMRS 3.54), while for PostgreSQL it is 1.20 (from ANDROMEDA 0.00 to ADEMPIERE 4.25). Project ANDROMEDA is astonishing: there is not a single constraint declared (no primary key, no foreign key, no unique, no not null) on the 180 tables, although there are a number of non-unique indexes and of sequences.

It is interesting to compare the schemas of the 78 projects available with both databases. This dual support must not be taken at face value: PostgreSQL support is often an afterthought and is not necessarily functional, including project such as ELGG, TAGADASH, QUICKTEAM or TIKIWIKI where some PostgreSQL table declarations use an incompatible MySQL syntax; 27 (34%) projects have missing tables or attributes between the MySQL and PostgreSQL versions: 190 tables and 173 individual attributes are missing or misspelled one side or another. Among the missing tables, 73 look like some kind of sequence, and thus might be possibly legitimate, although why the *auto increment* feature was not satisfactory is unclear. At the minimum, the functionalities are not the same between MySQL and PostgreSQL versions for those projects.

D. Overall quality

We have computed a synthetic project quality evaluation ranging from 10 (good) to 0 (bad) by removing points based on advice severity (error, warning, notice), level (schema, table, attribute) and project size. The MySQL projects quality average is 4.4 ± 1.3 (from 9.1 GENOVAWEB to 0.0 OSCARMC-MASTER), significantly lower than PostgreSQL 5.4 ± 1.8 (from 9.4 COMICS to 0.0 NURPAWIKI) *(very sure)*. This does not come as a surprise: most MySQL projects choose the default data-unsafe MyISAM engine, hence incur a penalty. Also, the multiplicity of MySQL back-ends allows the user to mix them unintentionally, what is not possible with PostgreSQL. When all MySQL-specific advices are removed, the quality measure is about the same for both databases. However, as PostgreSQL schemas provide more information about referential integrity constraints, they are also penalized as more advices can be raised based on the provided additional information.

Table IV shows the projects per quality decile. The PostgreSQL project quality is more spread than MySQL projects *(marginally sure)*. Table V compares the quality of projects according to size, where small is up to 9 tables, medium up to 29, and large otherwise. The quality is quite evenly distributed among sizes, which suggests that our effort to devise a size-neutral grading succeeded. Table VI compares quality based on the project categories. The number of projects in each category is too small to draw deep conclusions. Table VII addresses the technology used in the project: Java leads while PHP is near bottom. PHP projects take less care of their relational design *(very sure)*. Finally, Table VIII shows that quality evaluation is basically the same for recent and older projects.

V. Conclusion

This is the first survey on the quality of relational schemas in open-source software. The overall quality results are worse than envisioned at the beginning of the study. Although we did not expect a lot of perfect projects, having so few key declarations and referential integrity constraints came as a surprise. We must acknowledge that our assumption that data are precious, and that the database should help preserve its consistency by enforcing integrity constraints and implementing transactions, is not shared by most open-source projects, especially when based on MySQL and PHP. This is illustrated by bug report 15441 about missing keys on tables in MEDIAWIKI: it had no visible effect after two years.

The first author contributed both to the best PostgreSQL project (COMICS), and to one of the worst MySQL project (SLXBBL), which is *Salix* executed on its own schema! This deserves an explanation: COMICS is a small database used for teaching SQL. The normalized schema emphasizes clarity and cleanliness with a pedagogic goal in mind. Even so, the two raised warnings deserve to be fixed, although one would require an additional attribute. SLXBBL tables generate a lot of errors, because they are views materialized for performance issues. Also, they rely on MyISAM because some SQL create table statements must be compatible with both MySQL and PostgreSQL to ease the tool portability. Nevertheless, the comparison of schemas allowed to find one bug: an attribute had a different name, possibly because of a bad copy-paste.

We have released our *Salix* tool as a free software. As it is fully relational and relies on standards, it may be installed in any compliant database to help improve schemas.

Acknowledgement – Thanks to Pierre Jouvelot.

References

[1] A. Deshpande and D. Riehle, "The Total Growth of Open Source," in *4th Conference on Open Source Systems (OSS)*. Springer Verlag, 2008, pp. 197–209.

[2] A. Hars, "Working for free? motivations for participating in open-source projects," *International Journal of Electronic Commerce*, vol. 6, pp. 25–39, 2002, also IEEE 34th Hawaii International Conference on System Sciences 2001.

[3] K. Crowston and H. Annabi, "Effective work practices for software engineering: Free/libre open source software development," in *in Proc. of WISER*. ACM Press, 2004, pp. 18–26.

[4] D. M. Nichols and M. B. Twidale, "The usability of open source software," *First Monday*, vol. 8, 2003.

[5] B. Mishra, A. Prasad, and S. Raghunathan, "Quality and Profits Under Open Source Versus Closed Source," in *ICIS*, no. 32, 2002.

[6] I. Stamelos, L. Angelis, A. Oikonomou, and G. L. Bleris, "Code quality analysis in open-source software development," *Information Systems Journal, 2nd Special Issue on Open-Source*, vol. 12, no. 1, pp. 43–60, Feb. 2002, blackwell Science.

[7] E. Capra, C. Francalanci, and F. Merlo, "En Empirical Study on the Relationship among Software Design Quality, Development Effort and Governance in Open Source Projects," *IEEE Software Engineering*, vol. 34, no. 6, pp. 765–782, nov-dec 2008.

[8] R. Gobeille, "The FOSSology Project," in *Working Conference on Mining Software Repositories*, no. 5, Leipzig, Germany, May 2008.

[9] Coverty, "Coverty scan open source report," Coverty, White Paper, 2009.

[10] Veracode, Inc, "State of security report," White paper, Mar. 2010.

[11] D. Litchfield, "The Database Exposure Survey 2007," NGSSoftware Insight Security Research (NISR), Nov. 2007.

[12] E. F. Codd, "A relational model for large shared databanks," *Communications of the ACM*, vol. 13, no. 6, pp. 377–387, Jun. 1970.

[13] ISO/IEC, Ed., *9075-11:2003: Information and Definition Schemas (SQL/Schemata)*. ISO/IEC, 2003.

[14] T. J. MacCabe, "A Complexity Measure," *IEEE Software Engineering*, vol. SE-2, no. 4, pp. 308–320, Dec. 1976.

[15] M. Piattini, M. Genero, C. Calero, and G. Alarcos, "Data model metrics," in *In Handbook of Software Engineering and Knowledge Engineering: Emerging Technologies, World Scientific*, 2002.

[16] M. Genero, "A survey of Metrics for UML Class Diagrams," *Journal of Object Technology*, vol. 4, pp. 59–92, Nov. 2005.

[17] H. M. Sneed and O. Foshag, "Measuring legacy database structures," in *European Software Measurement Conference (FESMA'98)*, Hooft and Peeters, Eds., 1998.

[18] M. Piattini, C. Calero, and M. Genero, "Table Oriented Metrics for Relational Databases," *Software Quality Journal*, vol. 9, no. 2, pp. 79–97, 2001.

[19] A. L. Baroni, C. Calero, F. Ruiz, and F. Brito e Abreu, "Formalizing object-relational structural metrics," in *Conference of APSI, Lisbon*, no. 5, Nov. 2004.

[20] A. Bessey, K. Block, B. Chelf, A. Chou, B. Fulton, S. Hallem, C. Henri-Gros, A. Kamsky, S. McPeak, and D. Engler, "A Few Billion Lines of Code Later: Using Static Analysis to Find Bugs in the Real World," *Communication of the ACM*, vol. 53, no. 2, pp. 66–75, Feb. 2010.

[21] F. Coelho, "PG-Advisor: proof of concept SQL script," Mailed to `pgsql-hackers`, Mar. 2004.

[22] J. Currier, "SchemaSpy: Graphical database schema metadata browser," Source Forge, Aug. 2005.

[23] B. Schwartz and D. Nichter, "Maatkit," Google Code, 2007, see *duplicate-key-checker* and *schema-advisor*.

[24] J. Berkus, "Ten ways to wreck your database," O'Reilly Webcast, Jul. 2009.

[25] A. M. Boehm, M. Wetzka, A. Sickmann, and D. Seipel, "A Tool for Analyzing and Tuning Relational Database Applications: SQL Query Analyzer and Schema EnHancer (SQUASH)," in *Workshop über Grundlagen von Datenbanken*, Jun. 2006, pp. 45–49.

[26] F. Coelho, "Database quality survey projects and results," Nov. 2010, detailed list of projects considered in *A Field Analysis of Relational Database Schemas in Open Source Software*, report A/423/CRI. [Online]. Available: http://www.coelho.net/salix/projects.html