General Index

Index of Figures

Index of Tables

1 Introduction

The Schema Evolution Toolsuite (SET) project takes place into the **Pantha Rei** system, a research project for data management under schema evolution, in particular, integrating the Schema Evolution Benchmark (SEB) project, presented in "*Schema Evolution in wikipedia, toward a Web Information System Benchmark"*.
This project has been created with manifold purposes: integrate the SEB providing a framework to easily automate the benchmarking process, and support the evolution process performing statistics over the information system.

1.1 Connection with the Schema Evolution Benchmark project

The Schema Evolution Benchmark project focus its attention on MediaWiki, a data-intensive, open-source, collaborative, web-portal software, originally developed to run Wikipedia, a multilingual, web-based, free-content encyclopedia. It focuses on the problem of DB schema evolution, analyzing the over 170 schema versions, released to the public by the means of CVS or SVN repository during 4 years and 7 months of life of the MediaWiki system.
Along with the analysis of the MediaWiki system, a tool suite has been developed to collect statistical measure and all the data related to the versions installed.
At the end of the paper presenting the SEB project, comes out the need to develop a tool, capable to collect the result of the analysis of several case studies of open-source systems.
The Schema Evolution Tool suite integrate the previous work, providing the tool needed to support the benchmarking procedure. It allows to download, install and collect data from several systems, offering the researcher the possibility to extend the benchmark analysis.
Every component of the suite stores the collected information in a database, named evolution_db.
The MediaWiki analysis has been completed with the definition of some statistics, to better describe the evolution of the system; we can take as example the schema size growth statistic, or the table/colum lifetime or the per-month revision count.
The Schema evolution toll suite enhance this function, providing an extended set of statistics performed over the desired system. All the statistics are performed against the data stored inside the previously defined evolution_db and provides the user/ researcher a quick understanding of the status of the system they're going to work with.
Moreover, the suite produces a report summarizing the result of the statistics performed, and defines a global database evolution rate.

2 Early system architecture

2.1 MySQL information schema overview

The MySQL information_schema is the information database, the place where are stored the informations about all the other databases that the MySQL server maintains. Inside the information_schema there are several read-only tables. They are actually views, not base tables, so there are no files associated with them. It provides access to database metadata.
Metadata is data about the data, such as the name of a database or table, the data type of a column, or access privileges. Other terms that sometimes are used for this information are data dictionary and system catalog.

The MySQL version at the time of this document is the 5.0.76: the table the information_schema is composed of are:

- SCHEMATA: information about the databases;
- TABLES: information about the tables;
- COLUMNS: information about columns in tables;
- STATISTICS: information about indexes;
- USER_PRIVILEGES: informations about global privileges;
- SCHEMA_PRIVILEGES: information about schema (database) privileges;
- TABLE_PRIVILEGES: information about table privileges;
- COLUMN_PRIVILEGES: information about column privileges;
- CHARACTER_SETS: information about available character sets;
- COLLATIONS: information about collations for each character set;
- COLLATION_CHARACTER_SET_APPLICABILITY: indicates what character set is applicable for what collation;
- TABLE_CONSTRAINTS: describes which tables have constraints;
- KEY_COLUMN_USAGE: describes which key columns have constraints;
- ROUTINES: provides information about stored routines (both procedures and functions). The table does not include user-defined functions;
- VIEWS: provides information about views in databases;
- TRIGGERS: provides information about triggers;
- PROFILING: provides statement profiling information

2.2 Evolution database

The system is based on a “evolution database”. The purpose of this database is to store all the informations we collected about the system we're analyzing in order to make statistics over this data in the following phases. It's nothing else than a copy of the information_schema, with a lot more possibilities.
The reasons for creating a sort of copy of the information_schema are different:

- Performance: the information schema is a volatile database, as explained in the previous section; the tables are not real tables, but views; creating a real database will increase the queries performance; moreover the performances of the information_schema are related to

the number of the database installed on the server; the more the number increase, the more the performance decrease. If we wanted to collect the data from hundreds of systems, we'd be unable not only to query the information_schema but also to access it. We have all the tools a database makes available, like indexes or relation, in order to increase the performances.

- Portability: a real database makes the overall system more portable; once the data have been collected, the database can be backed up and transfered everywhere. The use of an in-memory database would draw a line at the use of the system.
- Completeness: we aren't limited to the data the information_schema makes available, but we can enrich our database with data related to the revision like the life interval or other debugging information. We can add tables to the database to store ad-hoc informations and future developments of the system will always use the same database.

The core of the evolution database is the ev_dbs table. This table stores the data the SET system retrieves from the repositories, data about the status of the system, and debugging informations.
The structure of the table is listed below with a brief description of each attribute composing it.

- db_name: the master name used to install the database;
- version: the revision number;
- db_installed: the name of the database installed: it's composed from the master name plus the revision number cleaned from the special character;
- date_from – date_to the interval of date of validity of the revision;
- state: the status of the database, 1 no errors occurred during the installation, 0 otherwise;
- mysql_err_code: if error occurred, the mysql error code returned;
- mysql_err_msg: if any error occurred, the mysql error message returned; since it'd be too expansive to maintain it, it'll be available only the development phase.

All the functions provided by the system have as prerequisite the availability of the ev_dbs table.
The other tables the evolution database is composed of are a “copy” of the mysql information_schema tables.
In order to perform its task, the evolution database stores the data inside these copies.
As we can note, just reading the name of the tables composing the information_schema, not all of them could contain useful data to our aim. Just to make an example, there are tables containing informations about the privileges of the users, or tables containing informations specific to the MySQL server instance. This is not what we're interested in: we want to support the user during the evolution process gathering useful data about the system, not about the specific server instance.

3 SET modules

The structure of the system counts two main modules:

- data collection;
- statistics.

3.1 Data collection module

The goal of the data collection phase is to allow the user to collect data about the system he's analyzing.
In order to achieve the goal, some other operations must be performed before starting the collection. First of all, the application must download the schema files from the system's repository. Once all the schemas have been downloaded, the application must install them.
Finally the collection can begin: the application reads the information_schema tables chosen in the previous step and starts collecting the data and storing them inside the evolution database.
Once the data have been collected, the user can decide to delete (drop) the database just installed or not; leaving it on the server won't affect the future data collection, but will affect the overall performance of the system.

[…...
install all the versions of the database schema of the considered system to perform analysis and statistics in the following phase.

The integration with the SEB system sees the SET system performing its task in different steps:

- download: the SET retrieves from the repository the number of the revisions and stores them in the evolution database, inside the ev_dbs table.
- installation: the schema file corresponding to each revision is downloaded from the repository and the schema is installed on the server
- collection: the SET collects the metadata of the installed schema from the mysql information_schema and stores them in the evolution database, inside the ev_dbs table.
- Drop: once the data have been collected, the SET system drops all the database installed in order to clean the server and maintain it efficient

….....]

3.1.1 SET schema download phase

During the download step, the system reads the configuration file, gets the informations about the type of repository, gets the root of the repository where the schemas are located, get the authentication information to access the repository and retrieves from it all the revision numbers of the database schema.
The real operation of download of the schema is performed during the installation phase to decouple the operation and allow the user to use his own schema; during this phase the application writes in the evolution database which schemas have been downloaded with the relative revision number and assigns the name will be used in the following phases to install the database.
The application supports both cvs (Cuncurrent Version System) and svn (Subversion) repository
The schemas can be downloaded from the svn or cvs repository supplied by the application's vendor

The user just need to set the right parameters in the configuration file, and set the right type of the repository.

3.1.2 SET schema installation phase

During the schema installation step the application installs all the revisions retrieved before.
The major problem faced during this phase is to avoid the user to look into the schema in order to get the best result during this phase. As reported in [the article presenting the SEB project], some of the MediaWiki schema failed during installation due to errors in the data definition language statements.
The failure affects the analysis phase, resulting in fallen drop in the statistics that measure the evolution of the database in terms of attribute or in terms of table.
An error during the installation can be caused by different reason:

- the use of different version of MySQL: the majority of the systems analyzed [during the development of the project] have been created with MySQL, but not all with the same version. The last stable version released to this day from the MySQL Corporation is the 5.0.76: this imply that during the installation, some schemas could fail due to incompatibility between the different versions of the server.
- different DBMS: not all the system use an open source solution like MySQL. There are societies that uses other DBMS like Oracle, increasing the incompatibility between the two DBMS.
- use of keywords not reserved in the older version of MySQL
- grammatical errors: this type of error can't be resolved. A schema affected by this type of error will always fail during the installation, unless the user does fix the error manually.

The success rate of schema installation is about 90%. This can be considered a good result, but some of the errors could be eliminated through an inspection of the schema, increasing the success rate
One solution could be to leave the burden of fixing the schema to the user; this solution however it's unfeasible because it's too onerous doing it manually.
The solution found in the SET project consist of a cleaning operation: the schema is analyzed and cleaned from all the statements that could cause an error during the installation.
This solution arise from an in-depth analysis of the errors occurred during the installation of the schemas of a sample system;
The results gained during this analysis show how one of the major source of error is due to a change in the way MySQL treats blob and text field from the version 5.0 of the server: in the latest release, blob and default field can't have a default value, while very often the dump tools offered by the Corporation creates default values for text or blob field [… MySQL reference Manual, http://dev.mysql.com/doc/refman/5.0/en/index.html …...].
The second source of error is given by mis-spell.
The result show that the cleaning process, removing all the solvable errors, increases the success rate of schema installation up to 98%
The application offers different level of cleaning going along with the user needs: light cleaning and full cleaning.
Both the two level of cleaning starts from the a common cleaning phase, which we can call *zero cleaning*:

- all the comment and insert statement are removed from the schema, as we aren't interested to the data but only to the metadata

- reserved words are quoted.

The remaining DDL statement isn't modified.

Light cleaning

The light cleaning level cleans the statement removing all the default values for blob and text field

Full cleaning

During the full-cleaning process either the zero-cleaning and light cleaning phase are performed.
The field left into the schema are:

- datatype
- length of the datatype (where present)

All the informations about indexes, keys, or foreign keys are left unchanged.

3.1.3 SET schema data collection phase

During this step the application collects the metadata from the MySQL information_schema and fills the table of the evolution database.
We have analyzed in the paragraph 2.1 the importance of the evolution tables copied from the MySQL information_schema tables.
During this step, the application reads the tables configured by the database administrator; checks if the table exist in the evolution database, otherwise the table is created, and starts filling the table with the metadata read by the information_schema corresponding view.
[. …..... While the creation of the table could be easily overcome with the use of the MySQL metadata, the collection of the data was a tough hurdle to overcome ….....]

Collecting system data

One of the major problems faced during the development of the application has been how to collect-the data from the information_schemas tables.

In order to describe the way the problem has been solved, we'll look inside the structure of the table Statistics. Here follow a description of the table extracted from the MySQL website

INFORMATION_SCHEMA **Name**	SHOW **Name**	**Remarks**
TABLE_CATALOG		NULL
TABLE_SCHEMA		= Database
TABLE_NAME	Table	
NON_UNIQUE	Non_unique	
INDEX_SCHEMA		= Database
INDEX_NAME	Key_name	
SEQ_IN_INDEX	Seq_in_index	
COLUMN_NAME	Column_name	
COLLATION	Collation	
CARDINALITY	Cardinality	
SUB_PART	Sub_part	MySQL extension
PACKED	Packed	MySQL extension

INFORMATION_SCHEMA Name	SHOW Name	Remarks
NULLABLE	Null	MySQL extension
INDEX_TYPE	Index_type	MySQL extension
COMMENT	Comment	MySQL extension

The collection operation reads the tuple of the revision examined and copy them inside the evolution database. The operation is performed once for each installed revision in order to make the system more flexible. This allows the user to execute the collection operation in subsequent moments collecting only the data that haven't been collected before.

The reading of the tuples is made possible by the selection over the attribute containing the name of the schema. As we can notice from the table's structure provided in example here, there are two attributes that represent the name of the database: table_schema and index_schema. This imply the SET application should arbitrarily choose which of the two fields are used to select the tuples.

From the MySQL manual, we learn that there exists an attribute inside the COLUMNS table called ORDINAL_POSITION. It reports the order of the columns inside the table. This attribute could be used to select between the multiple attributes. However this approach is strongly discouraged as it can't be considered reliable and could lead to application bugs whenever the choice of the attribute would select a wrong attribute. Moreover a future change in the MySQL structure of the information_schema tables could lead to a malfunctioning of the entire system.

The solution adopted in the SET application is the use of a configuration file where the user/db administrator sets the name of the table he wants to use, and the name of the attribute over which the selection will be performed. We'll see the details of this file in the configuration chapter.

3.1.4 SET schema dropping phase

The drop step simply allows, once collected the data or in case some errors occurred during the installation, to delete from the server all the database installed.

The drop step takes in input the name of the database we want to delete, selects all the databases with that name and drops them.

The administrator must be very careful to assign a unequivocal name to the database to avoid any accidentally deletion.

3.1.5 SET schema global option

The global function run the SET in “batch mode” , batch processing the following steps:

- drop: clean the server deleting every previous installation of the database in examination;
- download;
- install;
- collect;
- drop.

At the end, the server will be cleaned from every database, and the data collected will be stored inside the evolution database

3.2 Statistics module

In the data collection chapter we explained how the system collects the data into the evolution database.
In this section we'll describe how the data collected will be used to generate statistics to support the database administrator during the evolution process.
As we made clear in the previous chapter the data stored inside the evolution database are a copy of what we would retrieve querying the information_schema.
This data for example report the name of the tables inside each database, the name of the columns of each table, the type of each columns.
The SET uses these informations to represent the story of the database's evolution in the form of charts.
The statistics generated by the system has been grouped in five groups:

- tables
- columns
- index
- constraints
- revisions

Each group contains a list of statistics relative to his group, each preceded by a short introductory description.

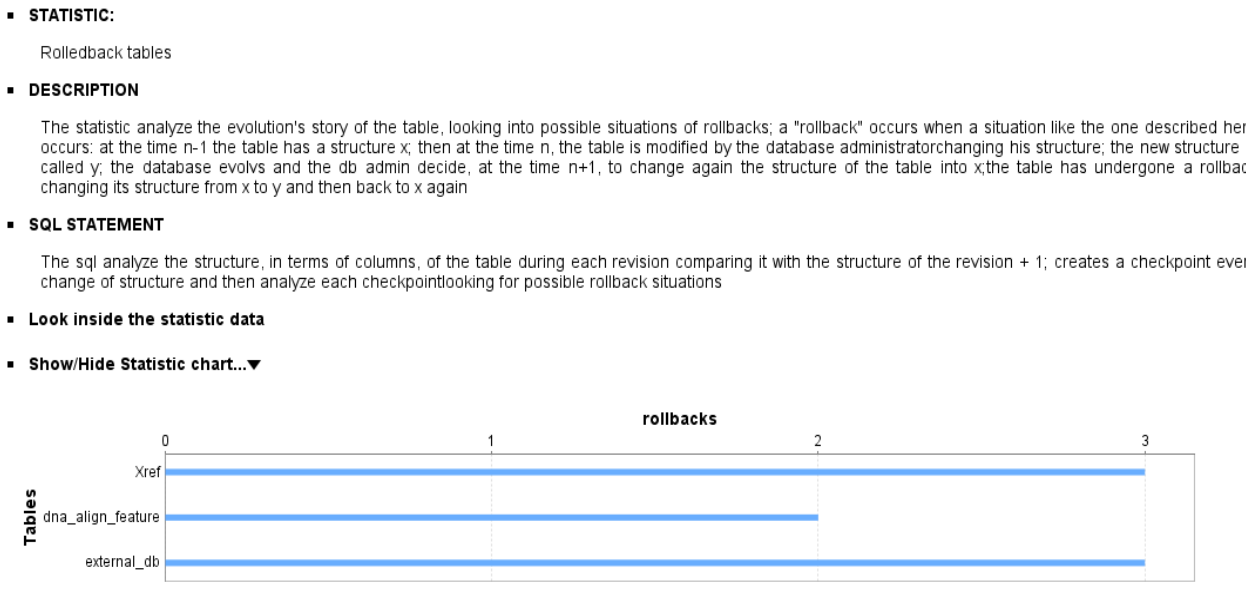

Figure 3.1: Statistic description sample

The description contains:

- a short explanation of the statistic, and if it's necessary, an explanation of some terms present in the chart or used to compute the statistic;
- if available, the SQL statement used to generate the statistic. In the case it wasn't available, the statement is substituted with an explanation of the steps performed to obtain the result;
- a link to the statistic data. The axis of the chart as often as not, uses a compressed scale in order to fit all the data inside the chart; if the user wanted to read or check through the data

that generated the chart, could look at the linked CSV file, or use this file to perform other analysis not foreseen during the design phase.

In addition to the statistics available in each of the five groups, the SET generates a report of the statistics executed and defines an evolution's rate for the entire database.
The evolution's rate will be useful to the user to understand better the complexity of the database he's going to work with.
We'll start first and foremost by looking into the different groups and explaining which statistics are available in each group and what they mean.

3.2.1 Tables statistic group

This group contains the statistics related to the table. At the time of the first deploy of the project in the group tables are assumed the following statistics:

- tables count
- tables added and deleted
- rolledback tables
- tables lifetime

There are some statistics that appear in almost every group, like for instance the *count of* statistic, or the *lifetime* statistic. These will be described only in the first group. In the remaining groups, we'll assume the knowledge of them by the user.

3.2.1.1 Tables count statistic

Count the number of table of each revision and report this evolution in a line chart, ordering the revision by date; the first revision to appear in the chart is the oldest one.
On the x-axis there are the revision number, on the y-axis the number of tables; the revision's number are progressively reported on axis, the CSV file contains the link between the progressive number and the revision number.

3.2.1.2 Tables added and deleted statistic

Analyze the evolution of the database with respect to the operation performed over the table.
The statistics counts how many tables have been added or deleted between two successive revision's release. This is simply performed by retrieving the name of the tables available in the two revisions and comparing them.
The result try to give an overall idea of the evolution of the system, it doesn't take into account every single change could happen between two revision; it doesn't reflect for example if a table has been renamed or if a table has been splitted or joined.

3.2.1.3 Rolled-back tables statistic

In order to let the reader better understand the statistic, we must define the meaning of the word “rollback”. A rollback occurs when the table present again a structure that already had in the past.

This can be better explained with an example:

- let assume the structure of the table in the revision *x* with the name *a;*
- in the revision *x+1* the table is modified: it's structure is now *b;*
- in the revision *x+2* the table's structure has changed again, brought back to *a:* the table has been rolled back*;*

The statistic rolled-back tables analyze the table's structure during its lifetime looking for possible situation of roll back. The inspection is made checking only the columns belonging to the table, not the data-type of the columns or other factors like indexes or primary keys. This can be improved with future release of the suite, adding other factors like the ones mentioned.

3.2.1.4 Tables lifetime statistic

The statistic count the number of revisions where the table appears. The choice of a bar chart to represent the results can be a bit controversial: the statistic doesn't check if the table wasn't present for an undefined number of revision, it does only count the number of revisions where the table was present. If there wasn't an interruption during the "life" of the table the chart doesn't show it. This is another aspect that could be improved with the development or future releases of the project, using a broken line to represent the life of a table.

3.2.2 Columns statistics group

This group contains the statistics related to the columns. At the time of the first deploy of the project in the group columns are assumed the following statistic:

- columns count
- columns lifetime

3.2.3 Index statistic group

This group contains the statistics related to the indexes. At the time of the first deploy of the project in the group index are assumed the following statistic:

- index count
- index changes

3.2.3.1 Index count statistic

The statistic index count is the same as the other statistics encountered up to this time of the report We just want highlight an aspect related to the way we consider the index inside the evolution database.
MySQL has different way to create an index: with the keyword KEY or with UNIQUE KEY; the difference between the two declarations is that in the second one the columns belonging to the index cannot contain duplicated values, while this is possible with the use of the KEY statement.
The statistic count both the type of indexes, either unique or non-unique

3.2.3.2 Index changes statistic

The assumption made for this statistic are valid both for the primary keys changes statistic and for the foreign keys changes statistics described in the constraints section
The system analyze the evolution's story of an index looking for possible changes either in the column's name or in the column's number.
The first situation imply that a real change occurred to the index; a column has been added to it, maybe to increase the performance of the queries issued against the table. The second situation reflects more a change in the table, if a change in the name of one of the columns belonging to the index occurred, this reflects a change happened in one of the columns of the table.
The two types of changes are represented in the chart with bar of different colors.

3.2.4 Constraints statistic group

The constraint group contains the statistics for the two main types of constraints available in a database: primary key and foreign key.
The statistics available in this group are:

- primary keys count
- foreign keys count
- primary keys changes
- foreign keys changes

As we already mentioned in the previous paragraph, the statistics *primary keys changes* and *foreign key changes* are created following the idea described for the statistic index changes

3.2.5 Schema's version group statistic

The group contains the following statistics:

- monthly revisions
- revision's lifetime

3.2.5.1 Monthly revisions statistics

The statistic count the number of revisions released per month. The information are derived from the repository data stored during the download phase.
The month is considered to be the minimum level of aggregation for large scale projects like the ones considered during the development of the MediaWiki project

3.3 Statistic summary report

As we just said in the introduction of the chapter, the system generates a summary of the result of

the statistics executed; now we'll have a look inside this summary and we'll explain in details what is reported inside it.

Table Name	Colomns evolution	Colomns data type changes	Number of roll back	Primary key changes	Foreign key changes	Index changes	Table risk factor	Table Name
analysis	9	Show/Hide DataTypes...	0	0	0	0	5%	analysis
analysis_history	15		0	0	0	0	8%	analysis_history
clone	21		0	2	0	0	29%	clone
contig	36	Show/Hide DataTypes...	0	0	0	0	19%	contig
contig_equiv	13		0	0	0	0	6%	contig_equiv
dna	15	Show/Hide DataTypes...	0	0	0	0	8%	dna
exon	52	Show/Hide DataTypes...	0	4	0	0	65%	exon
exon_feature	15		0	0	0	0	8%	exon_feature
exon_transcript	10		0	0	0	0	5%	exon_transcript
feature	22	Show/Hide DataTypes...	0	0	0	0	12%	feature
fset	13	Show/Hide DataTypes...	0	0	0	0	7%	fset
fset_feature	5	Show/Hide DataTypes...	0	0	0	0	3%	fset_feature

Figure 3.2: Statistics summary report

The summary contains aggregated information about the statistics performed against the database tables.
The statistics are the ones outlined in the previous paragraph plus a *column's evolution* statistic and a *columns data type change* statistic; for each of the statistic present in the table a numeric result is shown. The column that is well rendered to the reader's eye is the table evolution's rate: we'll talk about it later int the report. Before all let's have a look trough the columns of the summary table

- Table name: the name of the table (reported on the both side of the table);
- Columns evolution;
- Data type change;
- Number of rollbacks;
- Primary key changes;
- Foreign key changes;
- Index changes.

The columns the reader must pay more attention to are columns evolution and data type change, the new statistics we're going to explain, the other columns are a sum up of what explained in the previous sections.

3.3.1 Columns evolution statistic

This statistic analyze the evolution story of every single table of the database, particularly looking to the table's columns.
For every single table's revision, the statistic retrieve the columns belonging to the table, then check against the successive revision if any column has been added or deleted.
The number in the table indicates the total number of operation performed against the table; it takes into account both the columns added and deleted along all the table's story. Has already happened in other statistics presented in this paper, the result shown by the columns evolution statistic must be considered purely qualitative; the result doesn't differentiate between a column's renaming and a column's deletion; both these event will be considered by the system as a column has been added and one has been deleted. Further release of the system could improve this aspect trying to understand if an operation is ambiguous or not.

The example below explain better the behavior of this statistics.

Let's consider the table analysis at the revision 116; the columns of the table are:

- id;
- db;
- db_version;
- program;
- program_version;
- gff_source;
- gff_feature;

in the revision 119 the table's structure change, the new columns are:

- analysis_id;
- created;
- logic_name;
- db;
- db_version;
- db_file;
- program;
- program_version;
- program_file;
- parameters;
- module;
- module_version;
- gff_source;
- gff_feature;

The number of the columns is considerably changed; the result returned by the system would be

- columns added: 8
- columns deleted:1

total columns evolution 9.

What really happens to the table is that the column analysis has been renamed to analysis_id; this can't be find out by the system that will sign this operation as a column added plus a column deleted.

3.3.2 Data type change statistic

The statistic analyze every database's table, looking for possible data-type change in the evolution's story of the table. The column's data-type is compared revision by revision and every change is reported in the system's summary.

3.3.3 Table evolution's rate

The table's *evolution's rate* is a number computed by the system to represent the risk associated to the table.
The systems considered during the development of the SET are almost all large scale, open-source systems. It's assumed that the system evolves during the years, in order to accomplish the user's needs or to include some new developed functions. In addition, everybody knows that developing an information system means to modify, if necessary, the system according to the changes made to the underlaying database.
The *evolution's rate* tries to define an indicator of how much the table has been modified during its life. It has been computed considering several case studies and several observations for each case study, as a weighted sum of different variables.

Variables and coefficients

The variables considered during the definition of the evolution's rate are the ones presented at the beginning of the paragraph: columns evolution, data type change, number of rollbacks, primary key changes, foreign key changes and index changes. Each variable contributes to the table's evolution's rate with a different weigh.

Definition of the variable weigh

The definition of the variables' coefficient has followed a reversed path: we started from the hypothesis of the evolution's rate leaving the computation of the coefficients to a statistical software (STAT). The software takes in input a simple xls file, where in the row there are the observations and in the columns the variables.
We are able, analyzing a system, to tell whether or not the database has undergone complex changes in relation to its life and how much this changes could compromise the development of the information system. Looking for example at the risk variables previously defined, we could say, quite precisely, that the *primary key changes* is more important, in a table's evolution, than the *columns evolution*. The addition of a column is inbuilt with a database: the user's requirements can change and to satisfy their requirements, the database must change. However, other changes, as the one relative to the primary key, are more dangerous and complex; they force the developer to change the code, rewrite some queries or review some statements, in order to support the database's evolution.
However its difficult for us to assign a precise, objective coefficient to these changes, the variables that contribute to the definition of the evolution's rate. We can estimate, thanks to our experience and to our intuition, a table's evolution's rate, and then compute the variables' coefficients starting from it.
In order to compute analytically these value we followed these steps:

- execution of the SET statistic module over multiple sample systems
- collection of the results
- definition of the evolution's rate associated to the table
- computation of the variables' coefficient via a statistical software

Execution of the statistic module

The table below, show a sample of summary generated by the SET statistics module during one of the case studies considered.

Table Name	Colomns count evolution	Data Type change	Number of roll back	Primary key changes	Foreign key changes	Index changes
analysis	9	3	0	0	0	0
analysis_history	15	0	0	0	0	0
clone	21	0	0	2	0	0
contig	36	3	0	0	0	0
contig_equiv	13	0	0	0	0	0

Table 3.1: statistic summary detail

The row's header contains the name of the table, the column's header the name of the statistics executed. Looking at the table we can say that the *analysis* table had seven column's changes and three data-type changes.

Definition of the table evolution's rate

Looking at the result in Table 3.1 we define, for each table, the associated evolution's rate. It is computed taking into account the number of revision of the system; the same number of changes will produce a different result according to the total number of version of the system. This will produce a better estimation of the coefficient from the statistic software.
The evolution's rate is visualized as a percentage, the maximum value is 1.

Table Name	Colomns count evo-lution	Data Type change	Number of roll back	Primary key changes	Foreign key changes	Index changes	Table risk factor
analysis	9	3	0	0	0	0	0,01
analysis_history	15	0	0	0	0	0	0,02
clone	21	0	0	2	0	0	0,1
contig	36	3	0	0	0	0	0,05
contig_equiv	13	0	0	0	0	0	0,02

Table 3.2: table evolution's rate

Computation of the variables' coefficient

Once collected the data from several systems, using the regression tool analysis provided with the software we compute the coefficient of each variable.
We report below the result obtained running the software with some sample data: all the coefficients, apart from the data type change, have a high level of significance, this means that they can be considered reliable; the r square is close to 1, it express how well the model explain the observed data.
The *foreign key changes* hasn't been computed by the software because all the data reported a zero value for this variables. Further regression analysis will use a wider sample, a more characteristic population in order to compute all the desired coefficients.

Variables	coefficients	std. Error	sig.
Colomns count evolution	0,005	0,001	0,000
Datatype Changes	0,002	0,002	0,323
Number of roll back	0,082	0,004	0,000
Primary key changes	0,094	0,004	0,000
Index changes	0,020	0,003	0,000

Table 3.3: coefficient computation

4 Configuration and testing of the system

In this paragraphs will see how to configure the system. All the system is fully customizable via an xml file released with the application. The configuration file is called config.xml and contains a tag for every parameter that can be changed by the user.
The structure of the file is defined following the rule defined in the library CaConfigurator developed by Ing. Giorgio Orsi. Each tag has the following structure:

<param name="parameter_name">parameter_value</param>

The complete file is reported ..

```
<?xml version="1.0" encoding="UTF-8"?>
<parameters>
	<param name="repositoryType">cvs</param>
	<param name="pathtoschema">./output/schemas/ensembl_schema/</param>
	<param
name="svnurl">https://nucleuscms.svn.sourceforge.net/svnroot/nucleuscms/trunk/nucleus/install.sql</param>
	<param name="svnuser">anonymous</param>
	<param name="svnpwd">anonymous</param>
	<param name="cvsModule">SRM2/srm/db/srm_oracle_create.sql</param>-->
	<param name="cvsModule">zabbix/create/mysql/schema.sql</param>  -->
	<param name="cvsRoot">:pserver:anonymous@cvs.sanger.ac.uk:/cvsroot/ensembl</param>
	<param name="cvsModule">ensembl/sql/table.sql</param>
	<param name="cvsSaveSchemaPath">./output/schemas/ensembl_schema/</param>
	<param name="cvsLocalPath">/home/fangar/</param>
	<param name="mysqlhost">localhost</param>
	<param name="mysqlport">3306</param>
	<param name="dbbasename">ensembl</param>
	<param name="dburi">jdbc:mysql://localhost:3306/</param>
	<param name="user">root</param>
	<param name="pass">roberta</param>
	<param name="dbevolution">ham_bmevo</param>
	<param name="drop">false</param>
	<param name="download">false</param>
	<param name="install">false</param>
	<param name="collect">false</param>
	<param name="global">false</param>
	<param name="statistics">true</param>
	<param name="headerEvolutionTables">ev_</param>
	<param name="cleaningSchema">full</param>
	<param name="queryfile">./config/collectionquery.xml</param>
	<param name="statisticsFile">./config/statistics.xml</param>
</parameters>
```

Now we'll see in detail the different section the configuration file is composed of. Where the value the parameter can assume is restricted among some options, these will be indicated.

4.1 Repository configuration

These are the configuration from both the repositories, the first parameter set the type of the repository and will be used by the system to understand which protocol to use

repositoryType: cvs or svn, the type of the repository will be used to retrieve the schema

4.1.1 SVN configuration

- svnurl: the url of the svn repository
- svnuser: the user name to access the svn repository
- svnpwd: the password to access the svn repository

If svnuser or svnpwd are left blank, the system will use anonymous access

4.1.2 CVS configuration

- cvsRoot: the root of the cvs repository
- cvsModule: the module to download the schema from the repository (usually the relative location of the schema from the cvs root)
- cvsLocalPath: a local path on your system (used by the cvs protocol to start the download)
- pathtoschema: the local path where the application saves the schema downloaded from the repository; is is recommended to use a path with the same name of the database considered; if for example the database name is xxx, create a folder on your system with the name xxx_schema, just to avoid multiple database to download the schema in the same location

4.2 MySQL server configuration

This are the parameters to grant the SET system a root privileged access to the MySQL server instance available on the machine

- mysqlhost: the address of the mysql host
- mysqlport:the mysql port
- dburi: the uri to locate the mysql server (for example if you have the server installed on your machine you can set dbrui to jdbc:mysql://localhost/, otherwise use the ip address of your network dbms)
- user: the username to access mysql services
- pass: the password to access mysql services
- dbbasename: the name will be used to install the revision on the server

4.3 Data collection configuration

The following parameters allow the user to set which operation he wants to perform: in order to perform the operation the associated parameter must be set to true;

- download: downloads the schema from the repository and store them inside the *pathtoschema* location
- install: install the revision

- filling: performs the data collection
- global: performs the drop, download, install, collect and drop in batch mode
- dropping: performs the drop operation.
- CleaningSchema: light or full, set the desired level of cleaning.

4.4 Other configuration file

In this section are listed the parameters used to set the location of the others configuration file used by the system

- statList: the path to the file containing the built-in queries
- statisticsFile: the statistics configuration file
- queryFile: support file to the database's table creation

4.4.1 Statistic list file

The file contains a list of built-in statistic. It allows the user to define his own queries, within the limit of inserting a SQL statement inside a configuration file. The only type of queries that can be defined by the user are single SQL statements, with one parameter, the name of the database, and with two output parameters, a string and an integer.
The structure of the xml file is reported below, with the description for each element of the file.

```
<statistic>
        <stat-name>the statistic name</stat-name>
        <stat-description>the description of the statistic </stat-description>
        <stat-sql>the statistic sql statement</stat-sql>
        <stat-title-page></stat-title-page>
        <stat-chart-range>the range of the chart</stat-chart-range>
        <stat-chart-domain>the domain of the chart</stat-chart-domain>
        <stat-chart-type>the type of the chart, bar or line</stat-chart-type>
        <stat-chart-title>the chart's title</stat-chart-title>
        <stat-chart-alt>the chart's alternative text</stat-chart-alt>
        <stat-chart-orientation>the chart's orientation, horizontal or vertical</stat-chart-orientation>
        <stat-group>the group the stat belongs to</stat-group>
</statistic>
```

Inside the file is defined also the list of built-in statistics provided within the application. It is possible to enable or disable the generation of a single statistic by setting the corresponding parameter value to true or false.

```
<statistics_built_in>
        <statistic>
                <statistic_name>the name of the statistic: example, statRevisionTransition</statistic_name>
                <statistic_exe>true or false whether the statistic must be executed or not</statistic_exe>
        </statistic>
</statistics_built_in>
```

4.4.2 Statistics file

The structure of the statistics file is reported below along with the description of each single tag/parameter that can be customized by the user

```
<?xml version="1.0" encoding="UTF-8"?>
<parameters>
        <param name="pathPage"></param>
        <param name="statisticsList"></param>
        <param name="groups">/param>
        <param name="templateDir"></param>
        <param name="templateStatPage"></param>
        <param name="templateStatHome"></param>
        <param name="introPage"></param>
        <param name="columnsCount"></param>
        <param name="columnsTypeChange"></param>
        <param name="rollbackCount"></param>
        <param name="primaryKeyChanges"></param>
        <param name="foreignKeyChanges"></param>
        <param name="indexChanges"></param>
</parameters>
```

tag's description

- PathPage: the directory where the system store the html page resulting from the statistics execution
- statisticsList: the path to the list of statistics (the file described in the previous section)
- groups: the path to the groups file, the file containing the definition of the groups
- TemplateStatHome: the velocity home's template
- templateStatPage: the velocity page's template
- introPage: the name of the summary page
- columnsCount, columnsTypeChange, rollbackCount, primaryKeyChanges, foreignKeyChanges, indexChanges: the value of the evolution's rate's coefficient

4.4.3 Groups file

The groups file contains the description of the groups of statistics:

```
<groups>
        <group>
                <group-name>The group name</group-name>
                <group-title>the group title</group-title>
                <group-subtitle>the grouop subtitle</group-subtitle>
                <group-page>the group html page name</group-page>
        </group>
</groups>
```

4.5 Schema Evolution Toolsuite test

The suite has been released in a jar file and can be tested and executed via the command line.
To run the suite performs the following task:

- configure the file listed in the previous paragraph.
- create a directory in the file system with the following structure:
  - *<application_dir>*
    - *lib*
    - *output*
      - *statistics*
        - *templateHome.vm*
        - *templatePage.vml*
        - *stat.css*
      - *schemas*
    - *config*
      - *config.xml*
      - *statistics.xml*
      - *statList.xml*
      - *collectionQuery.xml*
      - *groups.xml*

- put the jar file inside the *application_dir*
- put all the configuration file inside the config directory (for the list of the necessary configuration files check the paragraph)
- check if all the library have been downloaded and store inside the lib directory (for the list of the necessary libraries check the paragraph)
- check the templates file in the statistics directory
- start the application with the following syntax (standard java syntax)
  - *java -jar SchemaEvolutionToolsuite.jar*

4.5.1 Configuration files

These are the configuration files that must be present inside the *config* directory in order to run correctly the application: (the description of each single file can be found in chapter 4

- config.xml
- statList.xml
- collectionQuery.xml
- groups.xml
- statistics.xml

4.5.2 Libraries

These are the library that must be copied inside the lib directory in order to avoid runtime errors

during the execution of the application.

- lib/CAConfigurator.jar
- lib/jargs.jar
- lib/saxon8.jar
- lib/svnkit.jar
- lib/mysql-connector-java-5.1.6-bin.jar
- lib/org-netbeans-lib-cvsclient.jar
- lib/velocity-1.5.jar
- lib/jfreechart-1.0.1.
- jar lib/jcommon-1.0.0.jar (released with jfreechart)
- lib/commons-collections-3.1.jar (released with jfreechart)
- lib/commons-lang-2.1.jar (released with jfreechart)

4.5.2.1 JfreeChart library & associated libraries

JFreeChart (http://www.jfree.org/) is a free 100% Java chart library that makes it easy for developers to display professional quality charts in their applications. JFreeChart's extensive feature set includes:

- a consistent and well-documented API, supporting a wide range of chart types;
- a flexible design that is easy to extend, and targets both server-side and client-side applications;
- support for many output types, including Swing components, image files (including PNG and JPEG), and vector graphics file formats (including PDF, EPS and SVG);
- JFreeChart is "open source" or, more specifically, free software. It is distributed under the terms of the GNU Lesser General Public Licence (LGPL), which permits use in proprietary applications.

4.5.2.2 Velocity library

Velocity (http://velocity.apache.org/) is a Java-based template engine. It permits anyone to use a simple yet powerful template language to reference objects defined in Java code.
When Velocity is used for web development, Web designers can work in parallel with Java programmers to develop web sites according to the Model-View-Controller (MVC) model, meaning that web page designers can focus solely on creating a site that looks good, and programmers can focus solely on writing top-notch code. Velocity separates Java code from the web pages, making the web site more maintainable over its lifespan and providing a viable alternative to Java Server Pages (JSPs) or PHP .
Velocity's capabilities reach well beyond the realm of the web; for example, it can be used to generate SQL, PostScript and XML from templates. It can be used either as a standalone utility for generating source code and reports, or as an integrated component of other systems.
In the Schema Evolution Toolsuite Velocity allows to create dynamic web pages showing the result of the statistics without the need to deploy the application in an application server like Tomcat, using a simple template.
The file templateHome and templatePage located in the statistics directory contains the HTML source code used to generate the statistics pages. The code, relative to the look of the page, can be freely modified by the user, provided that the Velocity syntax is respected and the general structure of the file is preserved. The look of the page is totally customizable by the CSS file stat.css

4.5.2.3 SvnKit library

SVNKit (http://svnkit.com/) brings Subversion closer to the Java world! SVNKit is a pure Java toolkit - it implements all Subversion features and provides APIs to work with Subversion working copies, access and manipulate Subversion repositories - everything within your Java application. It is written in Java and does not require any additional binaries or native applications. It is portable and there is no need for OS specific code. It is compatible with the latest version of Subversion.

4.5.2.4 NetBeans cvs client

Netbeans cvsClient (http://javacvs.netbeans.org/library/) is a framework for processing CVS commands in a java environment

4.5.2.5 Jargs & CaConfigurator library

This jargs library provides a convenient, compact, pre-packaged and comprehensively documented suite of command line option parsers for the use of Java programmers.
The CaConfigurator library provides an extension to the jargs library creating a standard customization framework for the projects developed in the database's group.

5 Future development

The development of the Schema evolution tool suite arose the possibility of several future work.
In this paragraph we list some of this ideas

1. Oracle support: up to now the suite support the MySQL DBMS; all the schema data are collected from the MySQL information_schema and stored inside the evolution database. Integrating the suite with the Oracle support will increase the range of systems analysable
2. Repository spider: the majority of the systems analyzed has a public SVN or CVS repository; the address of the repository can be easily retrieved from the system official site; however digging inside the repository looking for the schema SQL file is not only an hard work, but sometimes the research fails. The idea is developing a repository spider that goes through the repository entries looking for a text file containing a schema declaration or some SQL data definition's statements. The spider could show a list of the result to the user if one or more file are retrieved from the repository
3. SMO reverse engineering: given a couple of schema version, try to understand which are the schema modification operators used to perform the revision's passage. This work could be carried on as a possible thesis work