Free/Libre and Open Source Software Metrics

Sponsored through Framework Programme Sixth (Call 5) by

The FLOSSMetrics Consortium consists of: Universidad Rey Juan Carlos, University of Maastrich, Wirtschaftsuniversitaet Wien, Aristotle University of Thessaloniki, Conecta s.r.l., Zea Partners and Philips Medical Systems PMS Nederland B.V.

Document Information

Version: 2.0
Date : Oct 10, 2008
revision: 1

Owning Partner: URJC

Author(s):
Santiago Dueñas
Liliana Tovar

Reviewer(s):
Gregorio Robles
Jesus M. Gonzalez-Barahona

To:
PUBLIC

Purpose of distribution:
Final version

Printed on at

Status:

[ ] Draft
[ ] To be reviewed
[ ] Proposal
[x] Final/Released

Confidentiality:

[x]	Public	- Intended for public use
[ ]	Restricted	- Intended for FLOSSMETRICS consortium only
[ ]	Confidential	- Intended for individual partner only

Deliverable ID: D3.1

Title:

Database Specification

Deliverable: D3.1

Title: Database Specification

Executive Summary:

The database with information about projects is a central element of the FLOSSMetrics system. Data retrieved from public repositories via the Retrieval System will be stored in it, and later be used to offer that information via the website, and for more elaborated studies. This report specifies the main aspects of the FLOSSMetrics database, including the main requirements that the design has to have into account, its data architecture in several levels, according to different amounts of processing, and a detailed description of its most relevant tables.

CHANGE LOG

Ver.	Date	Author	Description
0.1	17/07/2007	Santiago Dueñas	Initial version
0.5	20/08/2007	Gregorio Robles	Full review
1.0	03/09/2007	Jesus M. Gonzalez-Barahona	Final revision
1.5	23/09/2008	Liliana Tovar	Included new db schemes
2.0	10/10/2008	Jesus M. Gonzalez-Barahona	Review

APPLICABLE DOCUMENT LIST

Ref.	Title, author, source, date, status	Deliverable Identification

Contents

Chapter 1

Introduction

The database is a central part of the FLOSSMetrics project, as one of the goals, if not the main one, of this project is to provide data from many libre (free, open source) projects publicly for their use by third parties (research groups, companies, public bodies and developers).

This document describes the design of the database of the FLOSSMetrics project. First, the set of requirements that has driven the design of the database is presented and then the design itself is described in detail, especially the three-layer architecture that has been chosen. The description of the database design includes the schema of the tools that already have adapters for the FLOSSMetrics Retrieval System, the tool that provides the database with data from FLOSS projects.

This database will be fed by the Retrieval System (described in another FLOSSMetrics report), which will continuously and incrementally (when feasible) retrieve data from the repositories of the projects. The data will later be used to produce higher level analysis, and published in several formats for comsumption by third parties.

The design in this report corresponds to the second version of the database, that differs from the previous one in the schema of the supported tools. As the project evolves, and more needs are identified, new versions of the design will be produced and implemented.

Chapter 2

Requirements

This chapter shows the main requirements that the database design considered in this report has to meet. Those are:

- Traceability and reproductibility. Data will be extracted and integrated from many projects (in the order of thousands). Each project will usually have several repositories, and each of them will be mined several times. Each retrieval of data from a given repository has to be traced, so that errors can be detected and linked with the repository and the retrieval operation when they were introduced. In addition, the retrieval process has to be reproductable, in the sense that if it is done again, results should be correct and the same.

- Multi-level design. The data, once retrieved from the repositories, is processed in several ways, obtaining new data with a higher level of abstraction. This fact has to be considered in the design:

  - Raw level (tools level). Raw data, as extracted with the extraction tools is stored (first level). Having the original data available allows for the possibility of modifying the process to obtain data for the second and third level. It serves also in case that errors in the process of preparing, linking and enhancing the data are found. At this stage the database schema will be based on the ones used by the extraction tools.
  - Basic processed data level (unification level). This data is prepared, linked and enhanced in order to be ready for research consumption (second level).
  - Higher level (analyses level). A third level should be filled with the result of high-level (for instance, statistical) analysis.

- Independence of the retrieval operations. The data retrieval process is independent of the database and will be handled by the Retrieval System. Therefore the database has to accomodate peculiarities of the retrival processes. In fact, this is tightly related to the existence of the raw level in the data architecture (see above).

- Unique identifiers for projects and repositories. Unique identifiers should be available for projects and repositories at the database level so that queries can be performed in an easy and cost-effective way at the project level.

- Incremental update. The incremental update of the data is a task that is performed by the retrieval system, and by the tools performing higher level analysis. However, this fact has to be taken into consideration when designing the database.

- Concurrent update and use. Several users could be extracting information from the database concurrently with the retrieval system updating it, and maybe some higher level analysis tools using and updating information in different areas of the database. All of this has to be acomplished in a 24x7 environment.

- Use of portable SQL. SQL has been selected as the query language. The database architecture has to have this fact into account. However, first versions of the database will be implemented with MySQL, which could imply for some peculiarities of this database management system to appear. They should be minimized.

- Large amount of data. The data to be stored in the database is well above the range of the millions of records, with several interrelationships between the different tables. This has to be taken into account for performance reasons.

- Performance. The data has to be readily available for third party users via a web interface. This means that, in addition to less-demanding batch procesing of queries, some of them will be made online via a web interface, which should produce results as soon as possible.

Chapter 3

Database Design

The structure of the FLOSSMetrics database has been designed to cover the various needs of the public, such as researchers or developers. For this reason the database is divided in several levels, according to the various studies that can be performed.

As it can be observed from Figure 3.1, the database will be divided in three levels. The first (lower) level will contain data extracted by the tools integrated into the retrieval system; the second (medium) level will unify the data of the previous level; the third (high) level will contain analyses and statistics.

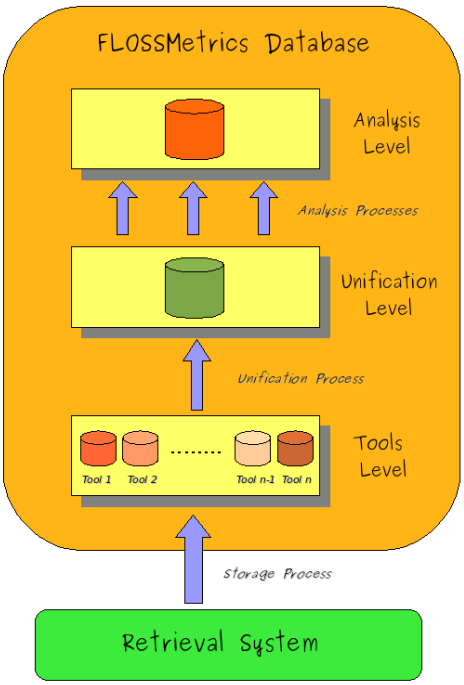

Figure 3.1: FLOSSMetrics database design

The storage of data is simple. Each level will be filled by scripts or programs that take data from the previous level and, after being processed, will be stored in the database. The first level will be filled by the retrieval system, after having executed the tools over the data repositories. Next, a script will take these data, and process it conveniently to identify relationships in the data and will be unified in the second level. Over the second level, a set of tools will take the data from the two lower levels to perform common statistical and other analyses and to store the results in the third level.

The rest of this chapter will explain each level in depth, including the relational diagrams and the description of the tables.

3.1 Tools Level

The tools level is the lower level of the database. As its name indicates, it will store data obtained directly from executing mining or extraction tools on data sources from projects. The retrieval system is in charge of performing this work and managing the insertion of these data into the database.

This level can be considered as a set of small and independent databases. Each tool generates distinct data in its own structure. This implies that the data cannot be stored and unified in the same tables. These data require manipulation to find links and relationships among them. Also, cleaning procedures have to be considered, to aggregate, correct or remove missing, partial or wrong data. But this process is not simple, expensive in time and resources and cannot be performed every time that a tool finish its work.

For this reason, a specific database with any execution of a tool on a specific data source or repository will be created. For example, if the retrieval system executes `CVSAnalY` on the `GNOME` and on the `Apache` subversion repository, two databases with the structure of the `CVSAnalY` database will be created. Hence, we will have a database with the data extracted from the `GNOME` repository and another one with the data from the `Apache` repository. Unless extraction tools are designed to perform incremental updates of the data sources, each time the data is updated another database will be created.

One advantage of this level is that it allows researchers to access specific data related to a specific project without managing large amount of data. For example, they can take the `CVSAnalY` database of the `Apache` project to perform certain analyses about the commiters of the project, without taking the database with the mailing lists of the same project, or without taking other `CVSAnalY` databases of other projects. The researchers will not have to manipulate data of projects that are not valuables for their studies, neither having to search nor filtering data in the database.

3.1.1 Relational Diagram

As described before, this level is composed of a set of databases with data from extraction tools. At this point of time, the supported tools are `CVSAnalY2`, `MailingListStats` and `Bicho`. This section includes the relational diagrams for each tool.

CVSAnalY2

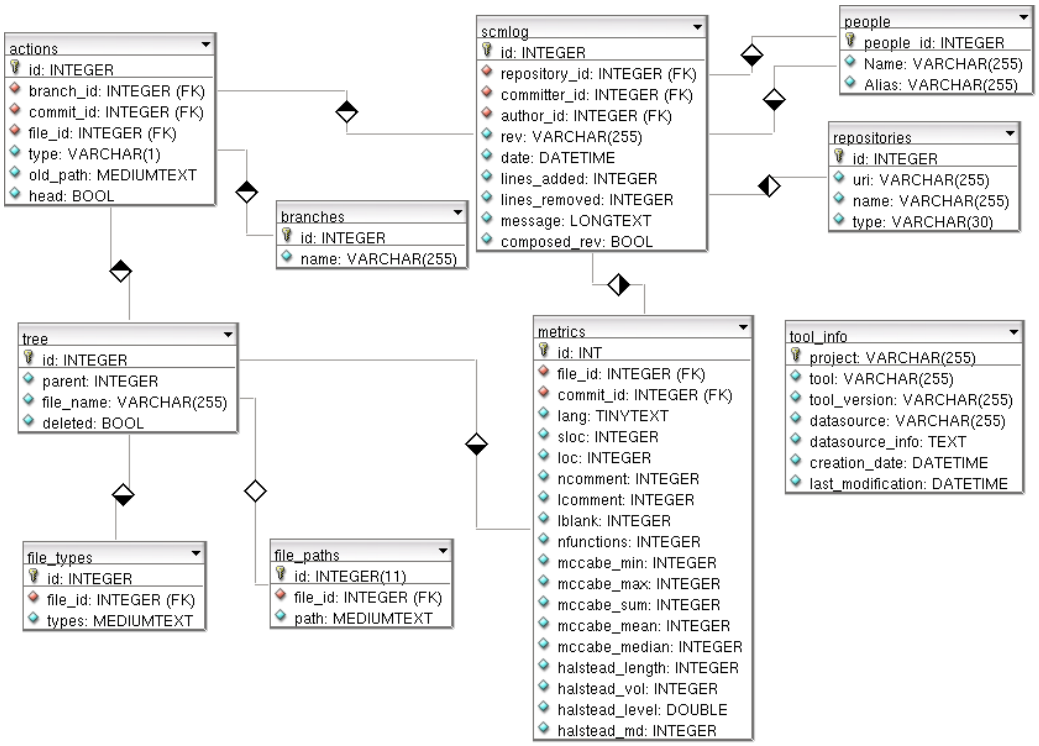

Figure 3.2: CVSAnalY relational diagram

CVSAnalY2 is a tool designed to extract data from Source Code Management Systems (SCM Systems) and insert them into a database. In this second version, this tool supports the mining of Concurrent Versioning System (CVS), Subversion systems (SVN) and Git (the Fast Version Control System).

The tool retrieves data about commiters, actions and files of a project. The data of every commit is stored including who and when the event happen, what was the revision number and the comments attached to the change. The database includes as well when a file was created, the last modification, and if it has been moved, copied or deleted.

Some general metrics for every file and specific ones for C, C++, Python, Java or Perl code can be obtained using external tools.

MailingListStats

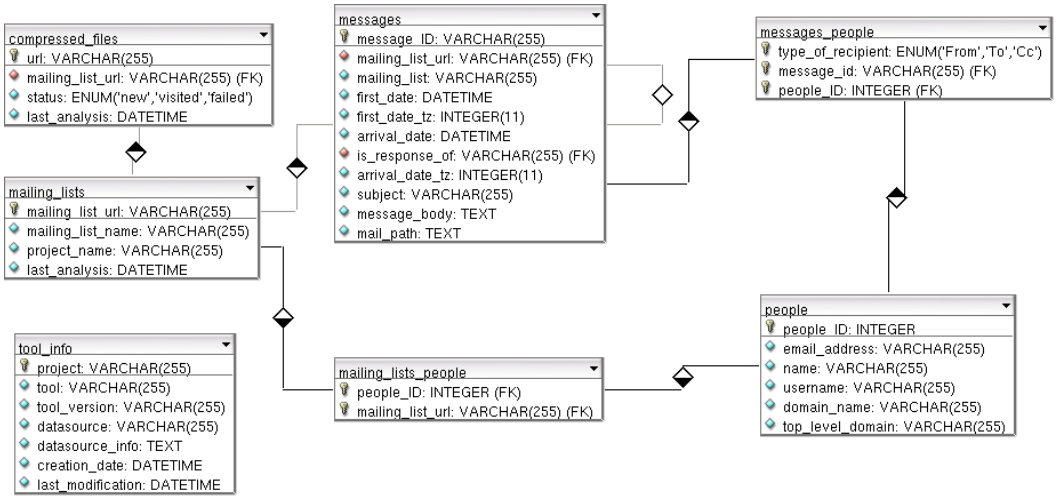

Figure 3.3: MailingListStats relational diagram

MalingListStats is capable of parsing electronic mail messages, with the format described in the RFC 822, from different mboxes and stores the results in a SQL database.

The data stored is basically extracted from the headers of the massages and specifically corresponds to the senders and receivers, the subjects, the send and receive dates and the content of the messages. It also stores information related to the mailing list such as the number of received messages and the people that write to the list.

Bicho

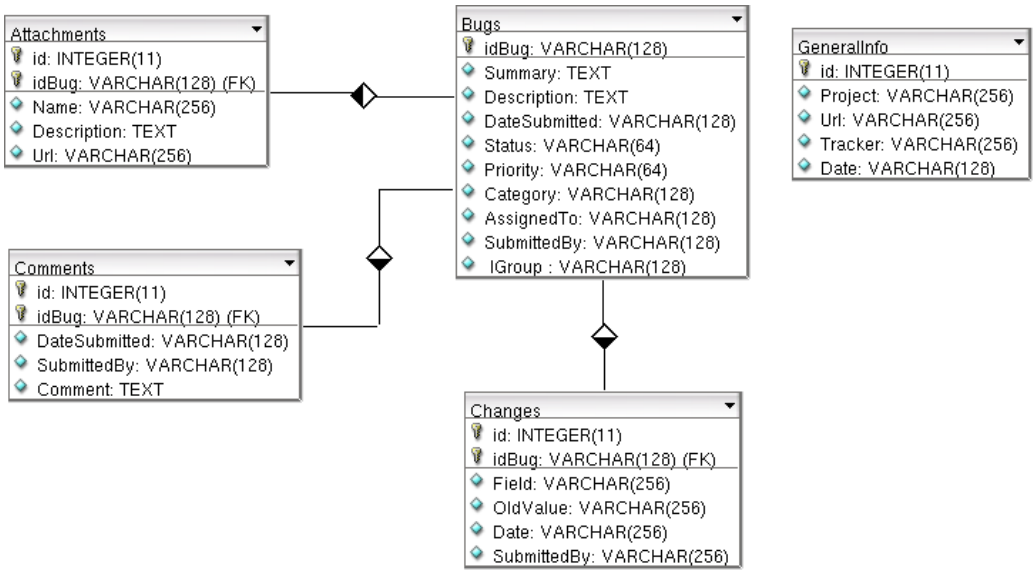

Figure 3.4: Bicho relational diagram

Bicho is a tool designed to extract data from bug tracking systems. So far it is able to retrieve just data from SourceForge and stores them in a database (Sqlite, Postgre or MySQL). The database design can be seen in the above figure.

The extracted data include important fields like the submitter, priority, category, who fixed or is going to fix the bug, when the event happen, changes, comments and attached files, current status of the bug and other general information.

3.1.2 Tables Descriptions

CVSAnalY2

- **scmlog**

  This table contains general information about the commits.

Name	Data Type	Description	Key
id	INTEGER	Commit unique identifier	PK
repository_id	INTEGER	Repository identifier	
committer_id	INTEGER	Committer identifier	FK
author_id	INTEGER	Developer identifier	FK
rev	VARCHAR(255)	Revision number assigned by the SCM, (CVS, SVN, GIT,...)	
date	DATETIME	Date and time of the commit	
lines_added	INTEGER	Number of lines added	
lines_removed	INTEGER	Number of lines removed	
message	VARCHAR(255)	General comment about the commit	
composed_rev	BOOL	Composed revision	

Table 3.1: Description of the scmlog table

- **filetypes**

  This table contains a register for each kind of file that may be found in the repository, such as documentation source code, images, etc.

Name	Data Type	Description	Key
id	INTEGER	File type unique identifier	PK
file_id	INTEGER	File identifier	FK
types	MEDIUMBLOB	File type: documentation, image, code, etc.	

Table 3.2: Description of the filetypes table

- **actions**

  This table contains the different actions performed in a commit.

Name	Data Type	Description	Key
id	INTEGER	Action unique identifier	PK
branch_id	INTEGER	Branch identifier	FK
commit_id	INTEGER	Commit identifier	FK
file_id	INTEGER	File identifier	FK
type	VARCHAR(1)	Action type (A:Added, M:Modified, D:Deleted, C:copied)	
old_path	VARCHAR(255)	Old path	
head	BOOL	Flag that indicates the last action performed over the commited file	

Table 3.3: Description of the actions table

- **branches**

  This table contains the distinct branches of a repository.

Name	Data Type	Description	Key
id	INTEGER	Branch unique identifier	PK
name	VARCHAR(255)	Branch name	

Table 3.4: Description of the branches table

- **file_paths**

  This table contains a list of file paths.

Name	Data Type	Description	Key
id	INTEGER	File unique identifier	PK
file_id	INTEGER	File identifier	FK
path	VARCHAR(255)	Path of the file commited	

Table 3.5: Description of the file_paths table

- **metrics**

  This table contains distinct metrics obtained from a file.

Name	Data Type	Description	Key
id	INT	Metric unique identifier	PK
file_id	INTEGER	File identifier	FK
commit_id	INTEGER	Commit identifier	FK
lang	TINYTEXT	Programming language	
sloc	INTEGER	Number of lines of code	
loc	INTEGER	Number of lines	
ncomment	INTEGER	Number of comments	
lcomment	INTEGER	Number of lines of comments	
lblank	INTEGER	Number of blank lines	
nfunctions	INTEGER	Number of fuctions	
mccabe_min	INTEGER	Minimun Mccabe complexity of the existing functions in the file	
mccabe_max	INTEGER	Maximun Mccabe complexity of the existing functions in the file	
mccabe_sum	INTEGER	Sum Mccabe complexity of the existing functions in the file	
mccabe_mean	INTEGER	Mean Mccabe complexity of the existing functions in the file	
mccabe_median	INTEGER	Median Mccabe complexity of the existing functions in the file	
halstead_length	INTEGER	Halstead length in the file	
halstead_vol	INTEGER	Halstead volumen in the file	
halstead_level	DOUBLE	Halstead level in the file	
halstead_md	INTEGER	Halstead mental discrimination	

Table 3.6: Description of the metrics table

- **people**

  This table contains registers about people have worked in the repository.

Name	Data Type	Description	Key
people_id	INTEGER	People unique identifier	PK
Name	VARCHAR(255)	Name	
Alias	VARCHAR(255)	Alias	

Table 3.7: Description of the people table

- **repositories**

  This table contains uris to the analysed repositories.

Name	Data Type	Description	Key
id	INTEGER	Repository unique identifier	PK
uri	VARCHAR(255)	URI of the repository	
name	VARCHAR(255)	Repository name	
type	VARCHAR(255)	Repository type (cvs, svn,git)	

Table 3.8: Description of the repositories table

- **tool_info**

  Table to store information about the retrieval process such as the tool executed, its version and the creation date.

Name	Data Type	Description	Key
project	VARCHAR(255)	Project name	PK
tool	VARCHAR(255)	Tool name	
tool_version	VARCHAR(255)	Tool version	
datasource	VARCHAR(255)	Path to the datasource	
datasource_info	TEXT	Info of the datasource	
creation_date	DATETIME	Creation date	
last_modification	DATETIME	Last modification date	

Table 3.9: Description of the tool_info table

- **tree**

  This table contains a register for each file stored in the version control repository.

Name	Data Type	Description	Key
id	INTEGER	File unique identifier	PK
parent	INTEGER	Parent directory	
file_name	VARCHAR(255)	Filename	
deleted	BOOL	Indicates if the file was deleted (1:deleted)	

Table 3.10: Description of the tree table

MailingListStats

- **compressed_files**

  This table contains a register for each archive file that has been downloaded, or tried to download.

Name	Data Type	Description	Key
url	VARCHAR	URL of the file	PK
status	ENUM	Either visited, new of failed	
last_analysis	DATETIME	Date and time of the last analysis of this file	
mailing_list_url	VARCHAR	URL of the web archives of the mailing list where this file belongs to	FK

Table 3.11: Description of the compressed_files table

- **mailing_lists**

  This table contains a register for each different mailing list analysed.

Name	Data Type	Description	Key
mailing_list_url	VARCHAR(255)	URL of the archives web page.	PK
mailing_list_name	VARCHAR(255)	Name of the mailing list, as it appears in the headers of the messages	
project_name	VARCHAR(255)	Name of the software project were this list belongs to. Taken from the email address of the mailing list	
last_analysis	DATETIME	Date and time of the last analysis performed on this mailing list	

Table 3.12: Description of the mailing_lists table

- **mailing_lists_people**

  This table joins the table mailing lists and people, making possible to search for people grouping by different mailing lists.

Name	Value	Description	Key
people_ID	INTEGER	People unique identifier	PK
mailing_list_url	VARCHAR(255)	URL of the mailing list archives web page	FK

Table 3.13: Description of the mailing_lists_people table

- **messages**

  This table contains a register for each message in the mailing list archives. It contains all the information in the headers plus the message itself.

Name	Value	Description	Key
message_id	VARCHAR(255)	Unique identifier assigned by the mailing list manager	PK
mailing_list_url	VARCHAR(255)	URL of the archives web page of the mailing list	FK
mailing_list	VARCHAR(255)	Name and address of the mailing list	
first_date	DATETIME	Local date written in the message by the original sender	
first_date_tz	INTEGER	Time zone of the above date	
arrival_date	DATETIME	Local time of the server that received the message	
arrival_date_tz	INTEGER	Time zone of the above date	
subject	VARCHAR(255)	Subject of the message	
message_body	TEXT	Main text of the message	
mail_path	TEXT	Mail path	
is_response_of	VARCHAR(255)	If this message is a reply of another, this is the id of the original message	FK

Table 3.14: Description of the messages table

- **messages_people**

  This is a table establish the relationship between email addresses and messages.

Name	Value	Description	Key
type_of_recipient	ENUM	Either To, Cc or Bcc	
message_id	VARCHAR(255)	Id of the message where that person appears	FK
people_ID	VARCHAR(255)	People unique identifier	FK

Table 3.15: Description of the messages_people table

- **people**

  This table contains a register for each one of the people who has written a message to the mailing list, or at least appears as destination in a message that has been sent to the mailing list.

Name	Value	Description	Key
people_ID	INTEGER	People unique identifier	PK
email_address	VARCHAR(255)	Email address of the person	PK
name	VARCHAR(255)	Name (if appears in the header)	
username	VARCHAR(255)	The first part (before the @) of the email address.	
domain_name	VARCHAR(255)	The second part (after the @) of the email address	
top_level_domain	VARCHAR(255)	Top level domain of the email address (.com, .org, .es, etc)	

Table 3.16: Description of the people table

- **tool_info**

  Table to store information about the retrieval process such as the tool executed, its version and the creation date.

Name	Value	Description	Key
project	VARCHAR(255)	Project name	PK
tool	VARCHAR(255)	Name of the tool	
tool_version	VARCHAR(255)	Tool version	
datasource	VARCHAR(255)	Location of the data sources	
datasource_info	TEXT	Access parameters to the data sources	
created_date	DATETIME	Date of creation of the database	
last_modification	TIMESTAMP	Date of the last modification of the database	

Table 3.17: Description of the cvsanal_modrequest table

Bicho

- **Attachments**

  This table contains general information about the file list attached.

Name	Data Type	Description	Key
id	INTEGER(11)	Attachments unique identifier	PK
idBug	VARCHAR(128)	Bug identifier from the web site	
Name	VARCHAR(256)	Attach name	
Description	TEXT	Attach description	
Url	VARCHAR(256)	Url where the file is	

Table 3.18: Description of the attachments table

- **Bugs**

  This table contains general information about the list of bugs found into the tracker.

Name	Data Type	Description	Key
id	INTEGER(11)	Bug unique identifier	
idBug	VARCHAR(128)	Bug unique identifier obtained from the web site	
Summary	TEXT	Summary of the bug	
Description	TEXT	Description of the bug	
DateSubmitted	VARCHAR(128)	Date submitted	
Status	VARCHAR(64)	Status of the bug (opened, closed, reopened, confirmed, deleted)	
Priority	VARCHAR(64)	Priority go from 9 to 1 where 9 is maximum and 1 minimum priority	
Category	VARCHAR(128)	Category of the bug	
AssignedTo	VARCHAR(128)	Name of the person who fixed the bug	
SubmittedBy	VARCHAR(128)	Name and user of the submitter	
IGroup	VARCHAR(128)	Group of the bug	

Table 3.19: Description of the bugs table

- **Changes**

  This table contains general information about the list of changes performed over the bugs.

Name	Data Type	Description	Key
id	INTEGER(11)	Change unique identifier	PK
idBug	VARCHAR(128)	Bug unique identifier obtained from the web site	
Field	VARCHAR(256)	Changed field	
OldValue	VARCHAR(256)	Old value	
Date	VARCHAR(256)	Creation date	
SubmittedBy	VARCHAR(256)	Name of the person who did the change	

Table 3.20: Description of the changes table

- **Comments**

  This table contains general information about the comments of the bugs.

Name	Data Type	Description	Key
id	INTEGER(11)	Comment unique identifier	PK
idBug	VARCHAR(128)	Bug unique identifier obtained from the web site	
DateSubmitted	VARCHAR(128)	Submision date	
SubmittedBy	VARCHAR(128)	Submitter	
Comment	TEXT	Comment	

Table 3.21: Description of the comments table

- **GeneralInfo**

This table contains general information about the retrieved tracker.

Name	Data Type	Description	Key
id	INTEGER(11)	Bug unique identifier	PK
Project	VARCHAR(256)	Project name	
Url	VARCHAR(256)	URL of the tracker	
Tracker	VARCHAR(256)	Tracker	
Date	VARCHAR(128)	Creation date	

Table 3.22: Description of the general info table

3.2 Unification Level

The main intention of this level is to establish relationships between the data stored in the `tools level`. These relationships can be done among data of the same project. For example, it is interesting to know which messages a commiter writes to the mailing lists of its project; or to know if she is the author of some files or only approves the changes and summits them to the repository. It could be also possible to locate the bugs that are being reported and if reporters write on the project mailing lists and so on.

One of the main difficulties for researchers is to obtain information about the relationships and the cooperation among projects. There are different levels of relationships: from developers that work in several projects, or write giving ideas to the mailing list or fixing bugs; to duplicated or reused code. Maybe, the most important characteristic of this level, is that it tries to find these correlations, and provide them to researchers and developers.

To carry out this work of unification, a script will be developed. This script will take the data from the databases of the `tools level`. Next, it will mix the data, and in a last step, after removing irrelevant, duplicated or incorrect data, the data will be stored into the database of this level. This work is expensive and difficult in terms of time and resources, as the amount of data to be manipulated is large. Due to this, its execution will require to be scheduled to avoid overloading the machine and having maximum availability.

3.2.1 Relational Diagram

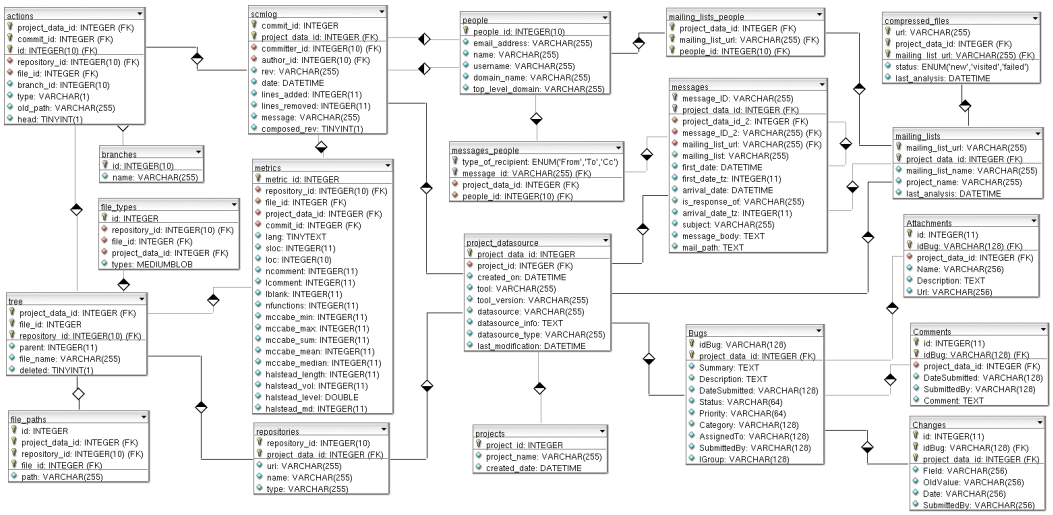

Figure 3.5: Unified relational diagram

This diagram includes the unification of the tools supported at the moment: `CVSAnalY2`, `MailingListStats` and `Bicho`. As it has been already described in the previous section, when other tools will be supported by the Retrieval System, they will be included in this diagram.

3.2.2 Tables Descriptions

The description of the tables are almost identical to ones described in the `Tools Level` section. The main differences are following:

- The diagram includes two new tables named `projects` and `projects_datasource`.

  – **projects**
  This table contains the projects stored in the unified database.

Name	Value	Description	Key
project_id	INTEGER	Unique identifier	PK
project	VARCHAR	Name of the project	
date	TIMESTAMP	Date of its first insertion in the database	

Table 3.23: Description of the projects table

  – **project_datasource**
  This table establishes the relationships between a project and its distinct datasources. This includes the parameters used to access to sources and the dates when the retrieval process was performed.

Name	Value	Description	Key
project_data_id	INTEGER	Unique identifier	PK
project_id	INTEGER	Identifier of the project assigned to this datasource	FK (projects.project_id)
tool	VARCHAR	Name of the tool	
tool_version	VARCHAR	Tool version	
datasource	VARCHAR	Location of the data sources	
datasource_info	TEXT	Access parameters to the data sources	
created_on	DATETIME	Date of creation of the database	

Table 3.24: Description of the project_datasource

- The rest of the tables includes a new field named `project_data_id`.

3.3 Analyses Level

The `analyses level` tries to minimized the time used by the researchers during their investigations. It will store a large set of distinct analyses, including common results and those who need long-time of process. If these analyses are available for the researches, they will not waste the time recalculating them.

This level will be fed by the analysis applications that will perform their work over the data taken from the `tools level` and the `unification level`.

For these version, only a few of these will be available and related with the data obtained from the available tools: `CVSAnalY2`, `MailingListStats` and `Bicho`.

3.3.1 Relational Diagram

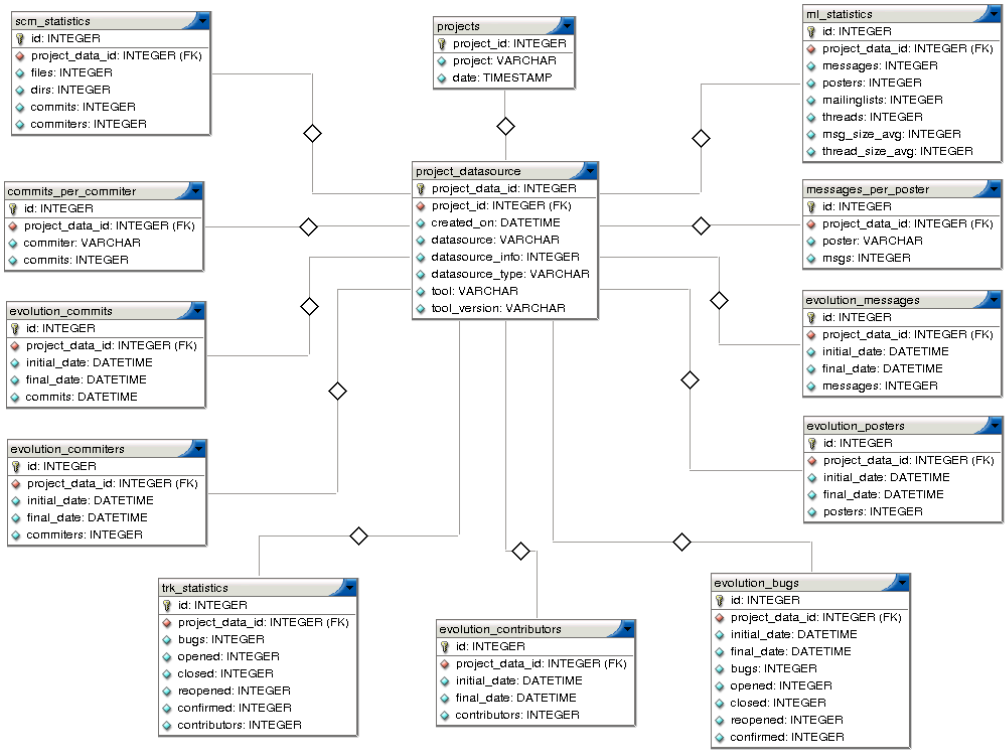

Figure 3.6: Analyses diagram

This diagram includes the tables that will store all the analyses performed over the data stored in `tools level` and `unified level`.

3.3.2 Tables Descriptions

- **projects**

  This table contains the projects stored in the analyses level.

Name	Value	Description	Key
project_id	INTEGER	Unique identifier	PK
project	VARCHAR	Name of the project	
date	TIMESTAMP	Date of its first insertion in the database	

Table 3.25: Description of the projects table

- **project_datasource**

  This table establishes the relationships between a project and its distinct datasources. This includes the parameters used to access to sources and

the dates when the retrieval process was performed.

Name	Value	Description	Key
project_data_id	INTEGER	Unique identifier	PK
project_id	INTEGER	Identifier of the project assigned to this datasource	FK (projects.project_id)
tool	VARCHAR	Name of the tool	
tool_version	VARCHAR	Tool version	
datasource	VARCHAR	Location of the data sources	
datasource_info	TEXT	Access parameters to the data sources	
created_on	DATETIME	Date of creation of the database	

Table 3.26: Description of the project_datasource

- **scm_statistics**

  This table stores basic statistics about a concrete snapshot of the scms of each project.

Name	Value	Description	Key
id	INTEGER	Unique identifier	PK
project_data_id	INTEGER	Identifier of the project's snapshot from the results has been obtained	FK (project_datasource.project_data_id)
files	INTEGER	Number of files	
dirs	INTEGER	Number of directories	
commits	INTERGER	Number of commits	
commiters	INTEGER	Number of commiters	

Table 3.27: Description of the scm_statistics

- **ml_statistics**

  This table stores basic statistics about a concrete snapshot of the mailing lists of each project.

Name	Value	Description	Key
id	INTEGER	Unique identifier	PK
project_data_id	INTEGER	Identifier of the project's snapshot from the results has been obtained	FK (project_datasource.project_data_id)
messages	INTEGER	Number of messages	
posters	INTEGER	Number of posters	
mailinglists	INTEGER	Number of mailing lists	
threads	INTEGER	Number of threads	
msg_size_avg	INTEGER	Message size average	
thread_size_avg	INTEGER	Thread size average (num of messages)	

Table 3.28: Description of the scm_statistics

- **trk_statistics**

  This table stores basic statistics about a concrete snapshot of tracking (bug-tracking) systems of each project.

Name	Value	Description	Key
id	INTEGER	Unique identifier	PK
project_data_id	INTEGER	Identifier of the project's snapshot from the results has been obtained	FK (project_datasource.project_data_id)
bugs	INTEGER	Number of bugs	
opened	INTEGER	Number of bugs opened	
closed	INTEGER	Number of bugs closed	
reopened	INTEGER	Number of bugs reopened	
confirmed	INTEGER	Number of bugs confirmed	
contributors	INTEGER	Number of contributors	

Table 3.29: Description of the trk_statistics

- **commits_per_commiter**

  This table stores the number of commits per commiter of a concrete project snapshot.

Name	Value	Description	Key
id	INTEGER	Unique identifier	PK
project_data_id	INTEGER	Identifier of the project's snapshot from the results has been obtained	FK (project_datasource.project_data_id)
commiter	VARCHAR	Identifier of the commiter	
commits	INTEGER	Number of commits	

Table 3.30: Description of the commits_per_commiter

- **messages_per_poster**

This table stores the number of messages per poster of a concrete project snapshot.

Name	Value	Description	Key
id	INTEGER	Unique identifier	PK
project_data_id	INTEGER	Identifier of the project's snapshot from the results has been obtained	FK (project_datasource.project_data_id)
poster	VARCHAR	Identifier of the poster	
messages	INTEGER	Number of messages	

Table 3.31: Description of the messages_per_poster

- **evolution_commits**

  This table stores the evolution of commits of a certain interval of time.

Name	Value	Description	Key
id	INTEGER	Unique identifier	PK
project_data_id	INTEGER	Identifier of the project's snapshot from the results has been obtained	FK (project_datasource.project_data_id)
initial_date	DATETIME	Initial date of the interval	
final_date	DATETIME	Final date of the interval	
commits	INTEGER	Number of commits	

Table 3.32: Description of the evolution_commits

- **evolution_commiters**

  This table stores the evolution of commiters of a certain interval of time.

Name	Value	Description	Key
id	INTEGER	Unique identifier	PK
project_data_id	INTEGER	Identifier of the project's snapshot from the results has been obtained	FK (project_datasource.project_data_id)
initial_date	DATETIME	Initial date of the interval	
final_date	DATETIME	Final date of the interval	
commiters	INTEGER	Number of commiters	

Table 3.33: Description of the evolution_commiters

- **evolution_commiters**

  This table stores the evolution of commiters of a certain interval of time.

Name	Value	Description	Key
id	INTEGER	Unique identifier	PK
project_data_id	INTEGER	Identifier of the project's snapshot from the results has been obtained	FK (project_datasource.project_data_id)
initial_date	DATETIME	Initial date of the interval	
final_date	DATETIME	Final date of the interval	
commiters	INTEGER	Number of commiters	

Table 3.34: Description of the evolution_commiters

- **evolution_messages**

  This table stores the evolution of messages of a certain interval of time.

Name	Value	Description	Key
id	INTEGER	Unique identifier	PK
project_data_id	INTEGER	Identifier of the project's snapshot from the results has been obtained	FK (project_datasource.project_data_id)
initial_date	DATETIME	Initial date of the interval	
final_date	DATETIME	Final date of the interval	
messages	INTEGER	Number of messages	

Table 3.35: Description of the evolution_messages

- **evolution_posters**

  This table stores the evolution of posters of a certain interval of time.

Name	Value	Description	Key
id	INTEGER	Unique identifier	PK
project_data_id	INTEGER	Identifier of the project's snapshot from the results has been obtained	FK (project_datasource.project_data_id)
initial_date	DATETIME	Initial date of the interval	
final_date	DATETIME	Final date of the interval	
posters	INTEGER	Number of posters	

Table 3.36: Description of the evolution_posters

- **evolution_contributors**

  This table stores the evolution of contributors in tracking systems in a certain interval of time.

Name	Value	Description	Key
id	INTEGER	Unique identifier	PK
project_data_id	INTEGER	Identifier of the project's snapshot from the results has been obtained	FK (project_datasource.project_data_id)
initial_date	DATETIME	Initial date of the interval	
final_date	DATETIME	Final date of the interval	
contributors	INTEGER	Number of contributors	

Table 3.37: Description of the evolution_contributors

- **evolution_bugs**

  This table stores the evolution of bugs in tracking systems in a certain interval of time.

Name	Value	Description	Key
id	INTEGER	Unique identifier	PK
project_data_id	INTEGER	Identifier of the project's snapshot from the results has been obtained	FK (project_datasource.project_data_id)
initial_date	DATETIME	Initial date of the interval	
final_date	DATETIME	Final date of the interval	
bugs	INTEGER	Number of bugs	
opened	INTEGER	Number of bugs opened	
closed	INTEGER	Number of bugs closed	
reopened	INTEGER	Number of bugs reopened	
confirmed	INTEGER	Number of bugs confirmed	

Table 3.38: Description of the evolution_contributors