PILAR

Usage of a Database to Store Risk Analysis Projects

October 2021

Table of Contents

1 Introduction

Traditionally, PILAR saves the projects in a single file, .MGR. This approach has several benefits:

- files are encrypted (protected by a pass phrase): key management policy is under full control of the analyst
- files are very compact
- file structure is not public, to simplify its design and evolution
- it is very easy to work on a single portable computer
- it is very easy to share, exchanging projects over email

But the approach introduces several limits for working on big organizations:

- it is not easy to collaborate with other tools, in particular
  - inventories
  - service data bases (e.g. ITIL CMDBs)
- it is not easy to exploit results (e.g. scoreboards)
- it is not easy to work in teams, where different people care after different parts of the analysis

The introduction of a database as an external repository of information addresses those concerns.

2 Assumptions

There are many database products available, either freeware, or commercial ones. PILAR should work with any, to the largest extend.

The access to the database will be over the network; so, PILAR may be used on any computer that has access to the server. Nevertheless, for this document, MySQL is used.

The permissions to access data are no longer under control of PILAR but will be transferred to the administrator of the DBMS.

PILAR assumes the database is case-sensitive; it is the responsibility of the DBMS administrator to create the database accordingly. This may differ for different DBMS; the following line crates such a database in MySQL:

```
CREATE DATABASE project DEFAULT CHARACTER SET latin1
COLLATE latin1_general_cs;
```

Up to version 6, one project in PILAR is mapped into one database. If there are N models, there will be N databases. The administrator shall configure the access rights of the users.

From version 6, one database may host several projects.

PILAR will use JDBC to connect to the database. So, the administrator is expected to provide java classes to access the database over the net using JDBC.

JDBC drivers are provided by the manufacturers.

3 User point of view

PILAR does not install or manage the database at all. It is up to your system administrators to set it up, as well as assign users and access control.

Typically, PILAR reads and writes tables that already exist. PILAR may also create, drop, and delete tables. The owner of the tables may have the rights to create and drop, or these rights may be limited to the system administrator.

4 Runtime options

Several options may be fixed at runtime in the configuration file.

`sql.code.length= 32`

PILAR uses VARCHAR(32) columns to store the code of the elements. 32 is the default size. You may change it.

`sql.create_tables= true`

PILAR may create tables that do not exist, or just report an error. The default is to create them.

When PILAR creates the tables on-the-fly, PRIMARY and FOREIGN constraints are ignored.

`sql.name.length= 128`

PILAR uses VARCHAR(128) columns to store the name of the elements (one-line descriptions). 128 is the default size. You may change it.

`sql.prefix=`

PILAR may attach a prefix to every table. This may help the database administration. Default is no prefix.

`sql.single_project= false`

PILAR may use a set of tables for a single project (default) or store several projects using the same tables.

4.1 Multi-project

Many tables have a column named PROJECT. For single-project tables, this column does not exist.

4.2 Update

In order to update a set of tables, the easiest procedure is to translate from the database to a file, restart the database, and restore from the file:

1. start PILAR using the old configuration
2. load the project from the database
3. save the project as a file
4. close PILAR
5. reset the database

6. start PILAR using the new configuration
7. load the project from the file
8. save onto the database
9. close

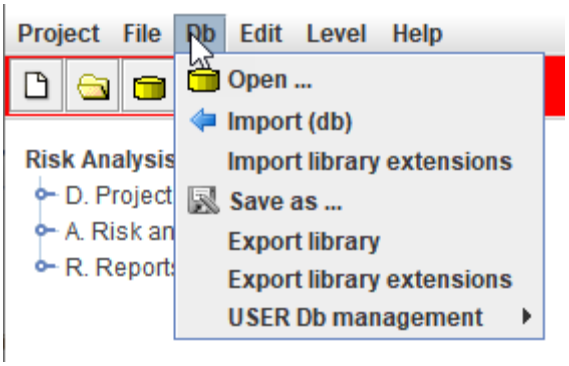
5 Licensing

In order to read / write databases, you need a special license. SQL access is not enabled by default.

6 Graphical user interface

6.1 Main screen

When SQL access is enabled, an additional menu shows up in user interface:



Open (both in the pull-down menu and in the tool bar)

to load a project from the database

Import (db)

to import a project from the data base, overwriting the data of the current project

Save as …

to upload a project to the database

Manage

There are a few operations on the database you may direct from PILAR. Namely:

DELETE

deletes the contents of all the tables related to the project

DROP

drops all the tables

CREATE

creates all the tables needed, either single- or multi-project

script

generates a script to create the tables offline

Import library extensions

to download extensions from the database; that is: new asset classes, new valuation criteria, and new threats

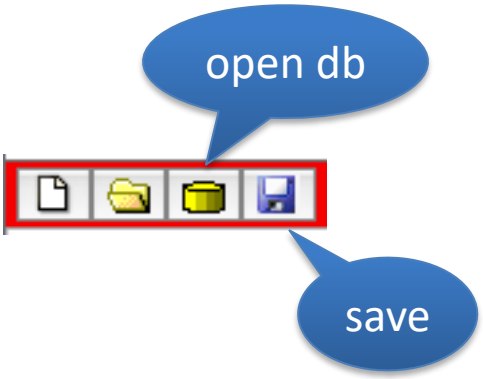
Export library

to upload PILAR library information onto the database

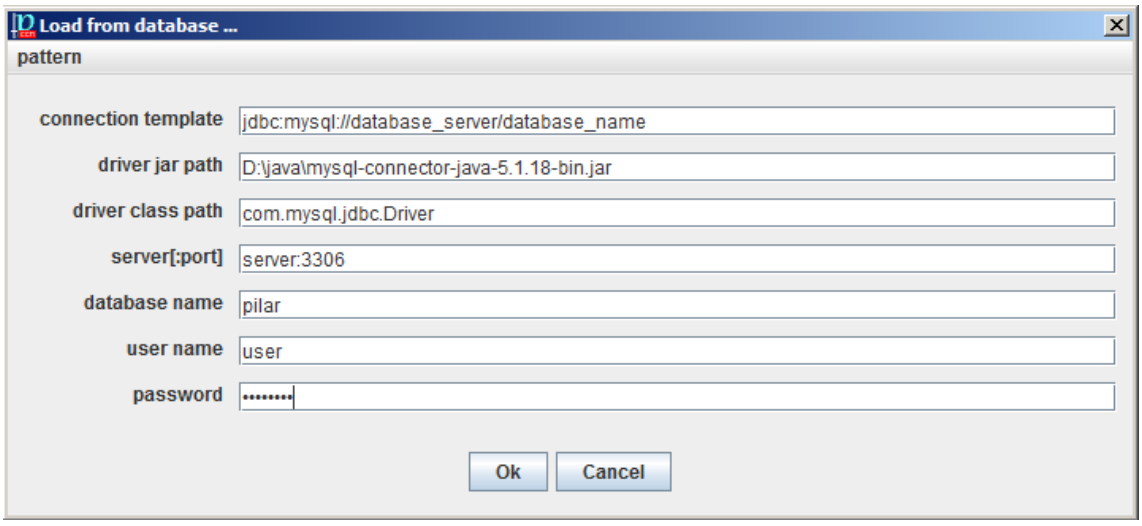
Export library extensions

to upload project extensions onto the database

In the toolbar, there is a button to open the project from the database, and the SAVE button is reused to save either in a file or a database, according to the current source of information:



To connect to the database, a menu is presented to fill in connection data:



pattern (pull-down menu)

PILAR has coded the pattern to build the jdbc url to connect to some typical dbms. It fills the 'connection template' below with a standard pattern.

connection template

The pattern to build the jdbc template. ‘database_server’ and ‘database_name’ are filled with the information below.

driver jar path

The JAR file holding the JDBC connector. It is provided by the vendor.

driver class path

The java class. It is vender specific. Consult vendor information for product and version.

server[:port]

The name of the machine hosting the database. It maybe “localhost”, or any reachable host.

database name

The name of the database. One database is expected per project.

user name

The login name. Established by the DB administration.

password

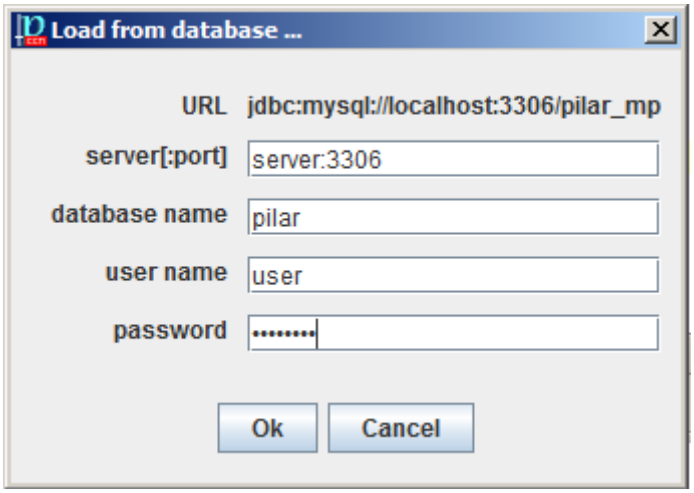
The password to log in. Established by the DB administration.

6.2 Reopen

After accessing the database, the information is stored under the RECENT PROJECTS menu. For instance, the following URL shows up

```
jdbc:mysql://localhost/test
```

When you select it, PILAR tries to access the database, but still a password is required



6.3 From files to database tables

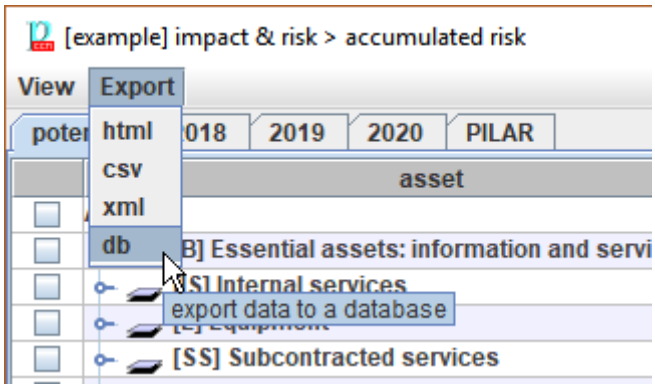
You may open a project from a .mgr file and store it in the database tables.

6.4 From database tables to files

You may download the project from the database and save it in a .mgr file.

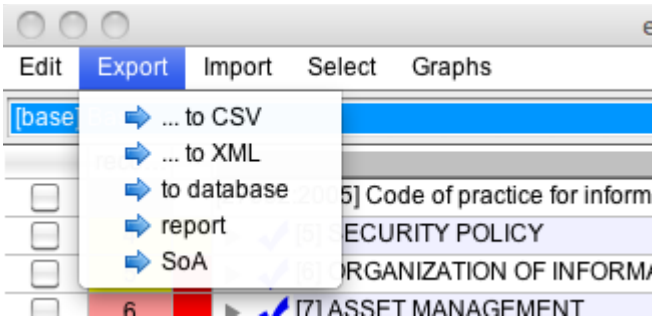
6.5 How to save analysis results?

In the screens displaying impact and risk analysis results, there is a menu option to export to a database:



6.6 Security profiles (EVL)

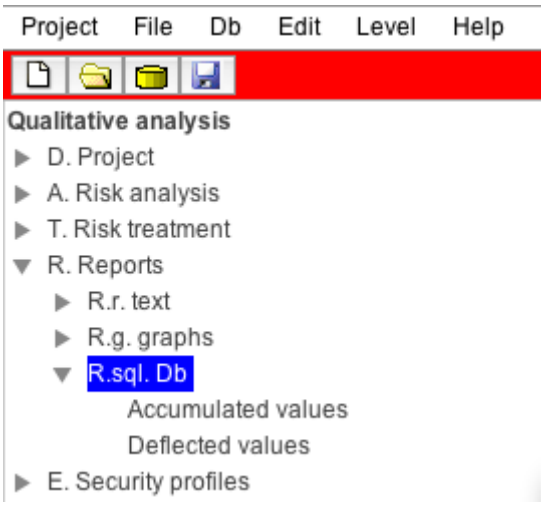
In the screens displaying evaluation profiles, a pull-down menu in the top toolbar offers an option to export to a database



Select to database to upload the evaluation information profile on the corresponding tables.

6.7 Reports

In the main project screen, under Reports, you will find options to upload accumulated and deflected values



7 Batch mode

To read from and write to a database, PILAR uses a JDBC connector that must be provided:

```
<jar> absolute path to .jar connector </jar>
<class> Driver class in jar </class>
```

Example:

```
<jar> /java/mysql-connector-java-5.1.8-bin.jar </jar>
<class> com.mysql.jdbc.Driver </class>
```

7.1 To download a project from the database:

```
<db user="..." password="..."> jdbc url </db>
```

Example:

```
<db user="john" password= "password">
  jdbc:mysql://localhost/risk_model
</db>
```

7.2 To upload the project onto the database:

```
<output db="url" user="..." password="..." />
```

Example:

```
<output db="jdbc:mysql://localhost/risk_model"
        user="john"
        password= "password
/>
```

7.3 Reports

The following tables may be generated from a batch plan:

Risk analysis: impact and risk

```
riskdown1
riskdown2
riskup1
riskup2
```

Business continuity: impact and risk

```
bcmdown1
bcmdown2
bcmup1
bcmup2
```

Security profiles

```
evl_attr
evl_domain
```

7.3.1 Accumulated impact and risk

Format:

```
<report what="risk_down"
    format="sql" user="..." password="..."
    [ domain="comma-separated list of domain codes" ]
    [ phase="comma-separated list of phase codes" ]
>
  jdbc url
</report>
```

If no "domain" is specified, all the domains are used.

If no "phase" is specified, all the phases are used. The potential values are reported under phase "null".

7.3.2 Deflected impact and risk

Format:

```
<report what="risk_up"
    format="sql" user="..." password="..."
    [ domain="comma-separated list of domain codes" ]
    [ phase="comma-separated list of phase codes" ]
>
  jdbc url
</report>
```

If no "domain" is specified, all the domains are used.

If no "phase" is specified, all the phases are used. The potential values are reported under phase "null".

7.3.3 Security profile (.evl)

Format:

```
<report evl="code"
    format="sql" user="..." password="..."
    [ domain="comma-separated list of domain codes" ]
    [ phase="comma-separated list of phase codes" ]
>
 jdbc url
```

```
</report>
```

If no “domain” is specified, all the domains are used.

If no “phase” is specified, all the phases are used.

For instance, for the ISO 27002 profile, PILAR will load the tables

```
evl_attr
evl_domain
```

8 Databases

The version is stored at

```
mysql> select * from modelattr where mykey = 'sql_tables_version';
+--------------------+---------+
| mykey              | val     |
+--------------------+---------+
| sql_tables_version | 2021.1  |
+--------------------+---------+
```

The following table presents the history of tables used by PILAR.

- The first column is the name of the table.
- Second column is the version where the table is first used (that is, in previous versions the table was not used). *.* means that is has been used since first PILAR version.
- Third column is the version where the table disappears (that is, in this version the table is not used any more). *.* means that the table is currently used.
- In green, tables that are currently used.
- In red, tables that are removed in this version.

8.1 Database tables

table name	min version	max version
apps	*.*	6.0
assetattr	5.2	*.*
assetbackup	5.2	*.*
assetcve	7.2	*.*
assetdomains	5.2	6.0
assetecm	5.3	2021.1
assetinfo	*.*	5.2
assetinfo5	5.2	*.*
assetinfo7	7.3	*.*
assetkb	1.1	2021.1
assetphase	5.2	6.0
assetprofiles	6.0	2021.1
assets	*.*	*.*
assetsafeguards	*.*	2021.1
assetsources	*.*	6.0
bcmdown1	*.*	*.*
bcmdown2	*.*	*.*
bcmup1	*.*	*.*

bcmup2	*.*	*.*
borderassets	6.2	*.*
borderprotection	6.2	*.*
borders	6.2	*.*
comments	*.*	5.2
comments5	5.2	*.*
compensatory	7.4	*.*
cpenames	5.2	6.0
cvss	7.2	*.*
domainattr	5.2	*.*
domainecm	5.3	*.*
domainkb	1.1	2021.1
domains	*.*	*.*
domainsafeguards	*.*	*.*
domainvaluebc	*.*	5.0
domainvaluera	*.*	5.0
eav_layers	5.2	*.*
eav_sources	5.2	*.*
evl_attr	*.*	*.*
evl_domain	*.*	*.*
extended_classes	*.*	*.*
extended_criteria	*.*	*.*
extended_threats	*.*	*.*
history	*.*	*.*
incidents	7.3	*.*
layers	*.*	*.*
links	*.*	*.*
log	*.*	*.*
logattr	*.*	*.*
maturity	*.*	*.*
metrics	5.2	6.0
model	*.*	*.*
modelattr	5.2	*.*
modelinfo	*.*	5.0

modelinfo5	5.2	*.*
modelinfo7	7.3	*.*
msactionattr	7.1	*.*
msactioncontrol	7.1	*.*
msactions	7.1	*.*
paths	6.2	*.*
perimeters	5.4	*.*
phaseattr	5.2	*.*
phases	*.*	*.*
pilar_classes	*.*	*.*
pilar_criteria	*.*	*.*
pilar_refs	6.0	*.*
pilar_safeguards	*.*	*.*
pilar_threatmodifiers	*.*	*.*
pilar_threats	*.*	*.*
preferences	*.*	*.*
profiles	*.*	*.*
referenceattr	6.0	*.*
referencesource	5.2	6.0
riskdown1	*.*	*.*
riskdown2	*.*	*.*
riskup1	*.*	*.*
riskup2	*.*	*.*
safeguards	*.*	*.*
scenarioattr	7.1	*.*
scenariocontrol	7.1	*.*
scenariophase	7.1	*.*
scenarios	7.1	*.*
sources	*.*	*.*
steps	5.2	*.*
supplement	2021.2	*.*
threatdegradation	*.*	*.*
threats	*.*	*.*
timing	6.2	*.*

valuebc	*.*	*.*
valuera	*.*	*.*
valuescale	*.*	5.4
vulnerabilities	5.2	7.1
vulnerability	5.2	7.1
vulnerablesoftwarelist	5.2	*.*
xor_selection	5.3	*.*
zones	6.2	*.*

8.2 Encoding

As a general rule,

- codes are VARCHAR(32),
- names are VARCHAR(128), and
- long texts are TEXT.

You may change text length using the following entries in the configuration file (CAR):

command	default	meaning
sql.code.length= 32	32	characters for codes
sql.name.length= 128	128	characters for names

For each table, it is specified what makes a row unique. Typically, it is a column working as primary key; sometimes there are several combinations of values that are unique, working as a composite primary key.

Dimension is encoded using the following codes:

dimension	language independent	English	Spanish
for availability	D	en.A	es.D
for integrity	I	en.I	es.I
for confidentiality	C	en.C	es.C
for authenticity	A	en.Auth	es.A
for accountability	T	en.Acc	es.T
for value	V	en.V	es.V
for personal data	DP	en.PD	es.DP

9 Input data

This section describes the tables used by PILAR in current version. The syntax is for MySQL. If you need a script to create the tables before the user saves a project, please use

```
PILAR / Db /manage / script
```

to generate for the needed vendor.

9.1 Model (a.k.a. project)

```
CREATE TABLE model (
    project VARCHAR(32) PRIMARY KEY NOT NULL,
    code VARCHAR(32),
    name VARCHAR(128),
    description TEXT,
    lib VARCHAR(32)
)
```

For single-project databases, there is a single row in this table. For multi-project databases there are as many entries as projects in the database.

9.1.1 Model attributes

```
CREATE TABLE modelattr (
    project VARCHAR(32),
    mykey VARCHAR(32),
    val VARCHAR(128),
    FOREIGN KEY (project) REFERENCES model(project)
)
```

Used to store some information about the project context. Currently, it uses the following keys:

sql_tables_version	version of the tables
date	date and time of saving
tool_name	the name of the tool (one of PILAR family)
tool_version	version of the tool
library	the library used for creation
tsv_name	the name of the .TSV file used
tsv_path	the path of the .TSV file used
tsv_sign	the signature (hash value) of the .TSV file used

9.1.2 Model information

```
CREATE TABLE modelinfo5 (
    project VARCHAR(32),
    mykey VARCHAR(32),
```

```
    name VARCHAR(32),
    val VARCHAR(128),
    position INT,
    FOREIGN KEY (project) REFERENCES model(project)
)
```

POSITION is a relative ordering of the rows.

There shall be as many rows as administrative lines in the project.

Since version 7.3, a new table is used to save long text values:

```
CREATE TABLE modelinfo7 (
    project TEXT(32),
    mykey TEXT(32),
    val TEXT,
    FOREIGN KEY (project)
        REFERENCES model(project)
)
```

9.1.3 Preferences

```
CREATE TABLE preferences (
    project VARCHAR(32) NOT NULL,
    mykey VARCHAR(32) NOT NULL,
    val VARCHAR(128),
    FOREIGN KEY (project) REFERENCES model(project)
)
```

There are several rows to record the elections of the user under Edit / Options.

9.1.4 History

It is used to keep track of users. When a user loads the contents of the database, it is recorded. If another user tries to open the same database simultaneously, it is prevented, and an explanatory message is shown. When the user closes PILAR, the table is cleared.

```
CREATE TABLE history (
    project VARCHAR(32),
    mgr_when TIMESTAMP,
    mgr_what VARCHAR(32),
    mgr_who VARCHAR(128),
    FOREIGN KEY (project) REFERENCES model(project)
)
```

"mgr_what" is set to "LOCK" when the user starts, and changes to "DONE" when the user leaves.

"mgr_who" identifies the user.

9.2 Sources of information

```
CREATE TABLE sources (
    project VARCHAR(32) NOT NULL,
```

```
    code VARCHAR(32) NOT NULL,
    name VARCHAR(128),
    description TEXT,
    position INT,
    PRIMARY KEY (project, code),
    FOREIGN KEY (project) REFERENCES model(project)
)
```

There shall be as many rows as sources.

POSITION is a relative ordering of the rows.

9.2.1 Information source attributes

Other information for sources

```
CREATE TABLE eav_sources (
    project VARCHAR(32),
    source VARCHAR(32),
    attr VARCHAR(32),
    value VARCHAR(128),
    FOREIGN KEY (project)
        REFERENCES model(project),
    FOREIGN KEY (project, source)
        REFERENCES sources(project, code)
)
```

Usage

source	attr	value
source code	“password”	password associated to the source
source code	"assets_tree"	the source is associated to the assets tree.

.

9.3 Domains

```
CREATE TABLE domains (
    project VARCHAR(32) NOT NULL,
    code VARCHAR(32) NOT NULL,
    next VARCHAR(32),
    name VARCHAR(128),
    qualifiers TEXT,
    description TEXT,
    position INT,
    PRIMARY KEY (project, code),
    FOREIGN KEY (project) REFERENCES model(project)
)
```

There shall be as many rows as domains.

POSITION is a relative ordering of the rows.

QUALIFIERS were filled in this table up to version 5.3. See “domainattr”.

9.3.1 Domain attributes

Since version 5.2, there is an additional table that expands the qualifiers, one per row, and adds some additional information about the domain

```
CREATE TABLE domainattr (
    project VARCHAR(32),
    domain VARCHAR(32),
    attr VARCHAR(32),
    value TEXT,
    FOREIGN KEY (project)
        REFERENCES model(project),
    FOREIGN KEY (project, domain)
        REFERENCES domains(project, code)
)
```

domain	attr	value
code	“qualifier”	vulnerability code (1 per row)
code	“tsv_path”	path of the .TSV file
code	“tsv_name”	name of the .TSV file
code	“tsv_sign”	signature (hash) of the .TSV file

9.4 Domain valuation

Domain valuation was used previously to version 5. PILAR is no longer filling this table.

PILAR still reads this information and generates essential assets to assign the valuation to.

9.5 Layers

```
CREATE TABLE layers (
    project VARCHAR(32) NOT NULL,
    code VARCHAR(32) NOT NULL,
    name VARCHAR(128),
    description TEXT,
    position INT,
    PRIMARY KEY (project, code),
    FOREIGN KEY (project) REFERENCES model(project)
)
```

There shall be as many rows as layers.

POSITION is a relative ordering of the layers. Layers with a lower position shall appear before (in a previous row in PILAR GUI) than those with a higher row. It is a loose ordering of rows.

9.5.1 Layer attributes (version 5.3)

Other information for asset layers.

```
CREATE TABLE eav_layers (
    project VARCHAR(32),
    layer VARCHAR(32),
    attr VARCHAR(32),
    value VARCHAR(128),
    FOREIGN KEY (project)
        REFERENCES model(project),
    FOREIGN KEY (project, layer)
        REFERENCES layers(project, code)
)
```

9.6 Assets

```
CREATE TABLE assets (
    project VARCHAR(32) NOT NULL,
    code VARCHAR(32) NOT NULL,
    name VARCHAR(128),
    description TEXT,
    domain VARCHAR(32),
    layer VARCHAR(32),
    father VARCHAR(32),
    position INT,
    classes TEXT,
    extra TEXT,
    PRIMARY KEY (project, code),
    FOREIGN KEY (project)
        REFERENCES model(project),
    FOREIGN KEY (project, domain)
        REFERENCES domains(project, code),
    FOREIGN KEY (project, layer)
        REFERENCES layers(project, code)
)
```

There shall be as many rows as assets, either groups or real assets.

Assets have a FATHER if they belong to a group.

POSITION is a relative ordering of the assets. Assets with a lower position shall appear before (in a previous row in PILAR GUI) than those with a higher row. It is a loose ordering of rows.

CLASSES were filled in this table up to version 5.3. See “assetattr”.

EXTRA were filled in this table up to version 5.3. See “assetattr”.

.

9.6.1 Asset attributes

Collects the information previously in ‘extra’ column of “assets”

```
CREATE TABLE assetattr (
    project VARCHAR(32),
    asset VARCHAR(32),
    attr VARCHAR(32),
    val VARCHAR(128),
    FOREIGN KEY (project)
        REFERENCES model(project),
    FOREIGN KEY (project, asset)
        REFERENCES assets(project, code)
)
```

where “code” identifies uniquely the asset, “attr” determines the attribute being set, and “val” is the value of the attribute. The following attributes are currently defined:

attr	**val**
available	boolean “true” if the asset is available default is “true”
class	class code classes that qualify the asset, one per row
domain	code of a security domain domains to which the [essential] asset is associated the domain of the asset is not included here
eval1	phase code for this phase, the asset has independent valuation of safeguards and controls, overriding those of the domain
evl	evl code (security profile) specific security profile for this asset
exists”	boolean “true” if the asset is exists default is “true”
group	boolean “true” if the asset is a group default is “false”
mix	threat code this threat is manually valuated for the asset
source	source code the source is associated to the asset

attr	val
threats	boolean “true” if the asset has threats default is “true”
visible	boolean “true” if the asset is visible default is “true”

9.6.2 Asset information

```
CREATE TABLE assetinfo5 (
    project VARCHAR(32),
    asset VARCHAR(32),
    mykey VARCHAR(32),
    name VARCHAR(32),
    val VARCHAR(128),
    position INT,
    FOREIGN KEY (project)
        REFERENCES model(project),
    FOREIGN KEY (project, asset)
        REFERENCES assets(project, code)
)
```

There shall be as many rows as lines of information.

POSITION is a relative ordering of the rows.

Since version 7.3, a new table is used to save long text values:

```
CREATE TABLE assetinfo7 (
    project TEXT(32),
    asset TEXT(32),
    mykey TEXT(32),
    val TEXT,
    FOREIGN KEY (project)
        REFERENCES model(project),
    FOREIGN KEY (project, asset)
        REFERENCES assets(project, code)
)
```

9.6.3 Asset dependencies

```
CREATE TABLE links (
    project VARCHAR(32),
    asset VARCHAR(32),
    below VARCHAR(32),
    expression TEXT,
    comment TEXT,
    FOREIGN KEY (project)
        REFERENCES model(project),
    FOREIGN KEY (project, asset)
```

```
        REFERENCES assets(project, code),
    FOREIGN KEY (project, below)
        REFERENCES assets(project, code)
)
```

There shall be as many rows as connections between two assets.

The EXPRESSION is encoded as in PILAR GUI:

encoding	meaning
	100% over all dimensions
50%	50% over all dimensions
C:10%/I:100%	10% for [C], 100% for [I], 0% for those not mentioned.

9.6.4 Asset valuation

There are two tables involved:

- valuera for normal risk analysis
- valuebc for business continuity

```
CREATE TABLE valuera (
    project VARCHAR(32) NOT NULL,
    asset VARCHAR(32) NOT NULL,
    dimension VARCHAR(8) NOT NULL,
    level INT,
    value FLOAT,
    criteria TEXT,
    comment TEXT,
    PRIMARY KEY (project, asset, dimension),
    FOREIGN KEY (project)
        REFERENCES model(project),
    FOREIGN KEY (project, asset)
        REFERENCES assets(project, code)
)
```

```
CREATE TABLE valuebc (
    project VARCHAR(32) NOT NULL,
    asset VARCHAR(32) NOT NULL,
    step INT NOT NULL,
    level INT,
    value FLOAT,
    criteria TEXT,
    comment TEXT,
    PRIMARY KEY (project, asset, step),
    FOREIGN KEY (project)
        REFERENCES model(project),
    FOREIGN KEY (project, asset)
        REFERENCES assets(project, code)
)
```

There shall be as many rows as value entries.

CRITERIA is a comma-separated list of criteria codes.

When the user only specifies a level, the value will be -1. When the user only specifies a value, the level will be -1. When the user sets a dimension as n.a., the level will be 1000.

9.7 Steps

Describes the steps used for continuity valuation:

```
CREATE TABLE steps (
    project VARCHAR(32) NOT NULL,
    step INT NOT NULL,
    visible BOOLEAN,
    PRIMARY KEY (project, step),
    FOREIGN KEY (project) REFERENCES model(project)
)
```

9.8 Threats

```
CREATE TABLE threats (
    project VARCHAR(32) NOT NULL,
    asset VARCHAR(32) NOT NULL,
    threat VARCHAR(32) NOT NULL,
    freq FLOAT,
    step INT,
    comment TEXT,
    PRIMARY KEY (project, asset, threat),
    FOREIGN KEY (project)
        REFERENCES model(project),
    FOREIGN KEY (project, asset)
        REFERENCES assets(project, code)
)
```

```
CREATE TABLE threatdegradation (
    project VARCHAR(32) NOT NULL,
    asset VARCHAR(32) NOT NULL,
    threat VARCHAR(32) NOT NULL,
    dimension VARCHAR(8) NOT NULL,
    deg INT,
    PRIMARY KEY (project, asset, threat, dimension),
    FOREIGN KEY (project)
        REFERENCES model(project),
    FOREIGN KEY (project, asset)
        REFERENCES assets(project, code)
)
```

9.9 Vulnerabilities DB (version 5)

Two tables are used to store information about vulnerabilities. Refer to

`http://cve.mitre.org/`

for further information on these data.

```
CREATE TABLE vulnerabilities (
    project VARCHAR(32) NOT NULL,
    id VARCHAR(32) NOT NULL,
    accessvector VARCHAR(32),
    accesscomplexity VARCHAR(32),
    authentication VARCHAR(32),
    confidentiality VARCHAR(32),
    integrity VARCHAR(32),
    availability VARCHAR(32),
    exploitability VARCHAR(32),
    remediationlevel VARCHAR(32),
    reportconfidence VARCHAR(32),
    cwe VARCHAR(32),
    summary TEXT,
    PRIMARY KEY (project, id),
    FOREIGN KEY (project) REFERENCES model(project)
)
```

```
CREATE TABLE vulnerablesoftwarelist (
    project VARCHAR(32),
    cveid VARCHAR(32),
    cpeid VARCHAR(128),
    FOREIGN KEY (project) REFERENCES model(project)
)
```

9.10 Vulnerabilities per asset (version 5)

This table associates a vulnerability to an asset. By default, PILAR uses the values from the CVE database (see previous section). When there is a change, this table supersedes the previous ones.

```
CREATE TABLE vulnerability (
    project VARCHAR(32) NOT NULL,
    asset VARCHAR(32) NOT NULL,
    cveid VARCHAR(32) NOT NULL,
    accessvector VARCHAR(32),
    accesscomplexity VARCHAR(32),
    authentication VARCHAR(32),
    confidentiality VARCHAR(32),
    integrity VARCHAR(32),
    availability VARCHAR(32),
    exploitability VARCHAR(32),
    remediationlevel VARCHAR(32),
    reportconfidence VARCHAR(32),
    PRIMARY KEY (project, asset, cveid),
    FOREIGN KEY (project)
        REFERENCES model(project),
    FOREIGN KEY (project, asset)
        REFERENCES assets(project, code)
)
```

9.11 CVE (version 7)

In version 7.2, tables vulnerability and vulnerabilities are replaced by assetcve and cvss

```
CREATE TABLE assetcve (
    project VARCHAR(32) NOT NULL,
    asset VARCHAR(32) NOT NULL,
    cveid VARCHAR(32) NOT NULL
);
```

```
CREATE TABLE cvss (
    project VARCHAR(32) NOT NULL,
    id VARCHAR(32) NOT NULL,
    vector VARCHAR(128),
    cwe VARCHAR(32),
    summary TEXT
);
```

Let's show an example

[PC] Work positions	
CVE-2011-0346	AV:N/AC:L/Au:N/C:C/I:C/A:C/RL:OF
CVE-2011-0347	AV:N/AC:M/Au:N/C:C/I:C/A:C/RL:OF
[SRV] Server	
CVE-2010-4398	AV:L/AC:L/Au:N/C:C/I:C/A:C/RL:OF
CVE-2010-4669	AV:N/AC:L/Au:N/C:N/I:N/A:C/RL:OF
CVE-2010-4701	AV:N/AC:H/Au:N/C:N/I:N/A:N/RL:W

SELECT * FROM assetcve

project	asset	cveid
example	SRV	CVE-2010-4398
example	SRV	CVE-2010-4669
example	SRV	CVE-2010-4701
example	PC	CVE-2011-0346
example	PC	CVE-2011-0347

SELECT * FROM cvss

project	id	vector	cwe	summary
example	CVE-2010-4398	AV:L/AC:L/Au:N/C:C/I:C/A:C/RL:OF	CWE-0	Stack-based buffer overflow in the RtlQu
example	CVE-2010-4669	AV:N/AC:L/Au:N/C:N/I:N/A:C/RL:OF	CWE-0	The Neighbor Discovery (ND) protocol ir
example	CVE-2010-4701	AV:N/AC:H/Au:N/C:N/I:N/A:N/RL:W	CWE-0	Heap-based buffer overflow in the CDra
example	CVE-2011-0346	AV:N/AC:L/Au:N/C:C/I:C/A:C/RL:OF	CWE-0	Use-after-free vulnerability in the Releas
example	CVE-2011-0347	AV:N/AC:M/Au:N/C:C/I:C/A:C/RL:OF	CWE-0	Microsoft Internet Explorer on Windows

9.12 Incidents

```
CREATE TABLE incidents (
    project VARCHAR(32) NOT NULL,
    position INT,
    asset VARCHAR(32) NOT NULL,
    threat VARCHAR(32) NOT NULL,
    date VARCHAR(32) NOT NULL,
    PRIMARY KEY (project, code),
    FOREIGN KEY (project) REFERENCES model(project),
    FOREIGN KEY (project, asset)
        REFERENCES assets(project, code)
);
```

POSITION is a relative ordering of the rows.

9.13 Phases

```
CREATE TABLE phases (
    project VARCHAR(32) NOT NULL,
    code VARCHAR(32) NOT NULL,
    prev VARCHAR(32),
    name VARCHAR(128),
    description TEXT,
    position INT,
    PRIMARY KEY (project, code),
    FOREIGN KEY (project) REFERENCES model(project)
)
```

POSITION is a relative ordering of the rows.

9.14 Phase attributes

There is an additional table to store more info for safeguards.

```
CREATE TABLE phaseattr (
    project VARCHAR(32),
    phase VARCHAR(32),
    attr VARCHAR(32),
    val TEXT,
    FOREIGN KEY (project)
        REFERENCES model(project),
    FOREIGN KEY (project, phase)
        REFERENCES phases(project, code)
)
```

“val” is the value of the attribute. The following attributes are currently defined:

attr	val
“source”	source code the source is associated to the phase

9.15 Safeguards

9.15.1 Maturity

Maturity is encoded as follows

char(1)	maturity
0	L0
1	L1
2	L2
3	L3
4	L4
5	L5
n	n.a.
?	...?
-	no info

```
CREATE TABLE maturity (
    project VARCHAR(32) NOT NULL,
    reference INT NOT NULL,
    phase VARCHAR(32) NOT NULL,
    maturity CHAR(1),
    PRIMARY KEY (project, reference, phase),
    FOREIGN KEY (project)
        REFERENCES model(project),
    FOREIGN KEY (project, reference)
        REFERENCES pilar_refs(project, reference)
)
```

REFERENCE: see References.

9.15.2 Flags

In this section:

- applies is a boolean: TRUE if applies (default is true)
- active is a boolean: TRUE if it is on (default is true)
- doubts is a boolean: TRUE if there are doubts (default is false)

```
CREATE TABLE referenceattr (
    project VARCHAR(32),
    reference INT,
    attr VARCHAR(32),
    val TEXT,
    FOREIGN KEY (project)
        REFERENCES model(project),
    FOREIGN KEY (project, reference)
        REFERENCES pilar_refs(project, reference)
)
```

REFERENCE: see References.

9.15.3 XOR Selection

Selects one child in a XOR branch of the safeguards tree.

```
CREATE TABLE xor_selection (
    project VARCHAR(32) NOT NULL,
    reference INT NOT NULL,
    phase VARCHAR(32) NOT NULL,
    child VARCHAR(32),
    PRIMARY KEY (project, reference, phase),
    FOREIGN KEY (project)
        REFERENCES model(project),
    FOREIGN KEY (project, phase)
        REFERENCES phases(project, code),
    FOREIGN KEY (project, reference)
        REFERENCES pilar_refs(project, reference)
)
```

In version 7.2 a weakness is repaired. In old versions, the child was the internal safeguard code. In new versions, it is a reference.

REFERENCE: see References.

9.15.4 Comments

```
CREATE TABLE comments5 (
    project VARCHAR(32) NOT NULL,
    reference INT NOT NULL,
    phase VARCHAR(32) NOT NULL,
    comment TEXT,
    PRIMARY KEY (project, reference, phase),
    FOREIGN KEY (project)
        REFERENCES model(project),
    FOREIGN KEY (project, reference)
        REFERENCES pilar_refs(project, reference)
)
```

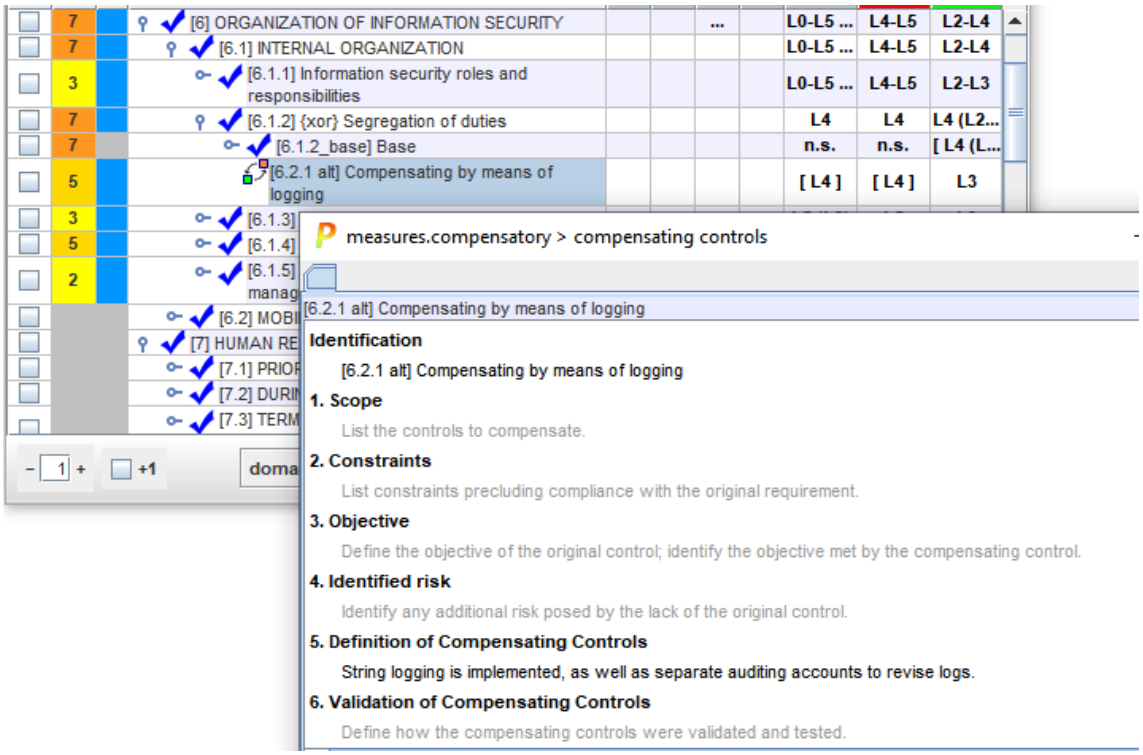
REFERENCE: see References.

9.15.5 Compensatory controls

This table saves information about compensatory controls introduced to compensate regular controls in security profiles (evl).

```
CREATE TABLE compensatory (
    project VARCHAR(32) NOT NULL,
    profile VARCHAR(32) NOT NULL,
    control VARCHAR(32) NOT NULL,
    attr TEXT NOT NULL,
    val TEXT,
    FOREIGN KEY (project)
        REFERENCES model(project)
)
```

Example



Saved as

	project	profile	control	attr	val
	Fi...	Filter	Fi...	Filter	Filter
1	example	27002:2013	6.2.1 alt	hook	6.1.2
2	example	27002:2013	6.2.1 alt	compensating.measures.id	[6.2.1 alt] Compensating by means of logging
3	example	27002:2013	6.2.1 alt	compensating.measures.selection	String logging is implemented, as well as separate auditing ...

9.15.6 Perimeters

This table stores information about expansion perimeters in safeguards and security profiles. The table saves those nodes of the tree that are expanded under the named label.

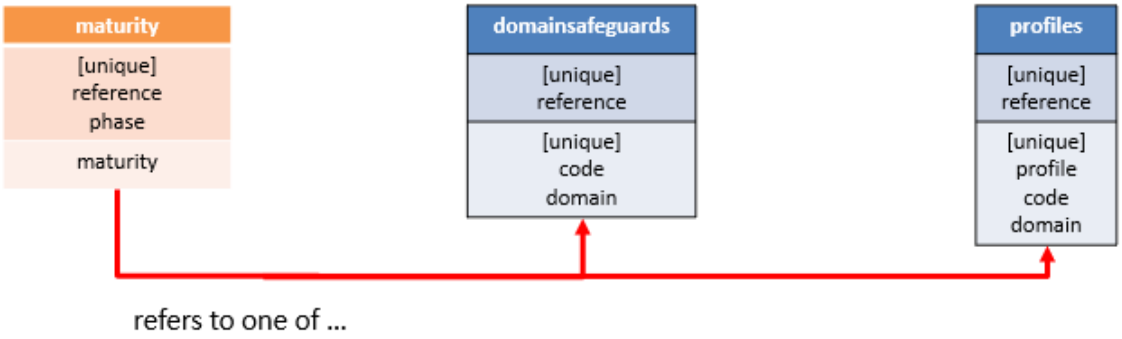
```
CREATE TABLE perimeters (
    project VARCHAR(32),
    profile VARCHAR(32),
    label VARCHAR(32),
    code VARCHAR(32),
    FOREIGN KEY (project) REFERENCES model(project)
)
```

The “label” is the user name for the perimeter. “profile” is empty for safeguard perimeters, but it saves the code of the security profile for EVL perimeters. Lastly, “code” is the code of the safeguard or of the control.

9.15.7 References

The references in previous sections are established as follows. Reference ids are unique in the database, referencing either a safeguard, or a control in a profile (EVL).

maturity.reference	domainsafeguards safeguards valuated for a domain
	profiles controls and questions (from evl) valuated for a domain



Safeguards per domain:

```
CREATE TABLE domainsafeguards (
    project VARCHAR(32) NOT NULL,
    code VARCHAR(32) NOT NULL,
    domain VARCHAR(32) NOT NULL,
    reference INT,
    PRIMARY KEY (project, code, domain),
    FOREIGN KEY (project)
        REFERENCES model(project),
    FOREIGN KEY (project, domain)
        REFERENCES domains(project, code),
    FOREIGN KEY (project, reference)
        REFERENCES pilar_refs(project, reference)
)
```

CODE stands for safeguard code (users code).

Security profiles (EVL):

Per domain

```
CREATE TABLE profiles (
    project VARCHAR(32) NOT NULL,
    profile VARCHAR(32) NOT NULL,
    code VARCHAR(32) NOT NULL,
    domain VARCHAR(32) NOT NULL,
    reference INT,
    PRIMARY KEY (project, profile, code, domain),
    FOREIGN KEY (project)
        REFERENCES model(project),
    FOREIGN KEY (project, domain)
        REFERENCES domains(project, code),
    FOREIGN KEY (project, reference)
        REFERENCES pilar_refs(project, reference)
)
```

Extended countermeasures:

Per domain:

```
CREATE TABLE domainecm (
    project VARCHAR(32) NOT NULL,
    domain VARCHAR(32) NOT NULL,
    evl VARCHAR(32) NOT NULL,
    code VARCHAR(32) NOT NULL,
    reference INT,
    PRIMARY KEY (project, domain, evl, code),
    FOREIGN KEY (project)
        REFERENCES model(project),
    FOREIGN KEY (project, domain)
        REFERENCES domains(project, code),
    FOREIGN KEY (project, reference)
        REFERENCES pilar_refs(project, reference)
)
```

Reference uniqueness:

In order to impose the constraint on uniqueness of REFERENCES, PILAR may use an additional table:

```
CREATE TABLE pilar_refs (
    project VARCHAR(32) NOT NULL,
    reference INT NOT NULL,
    PRIMARY KEY (project, reference),
    FOREIGN KEY (project) REFERENCES model(project)
)
```

This table is NOT created by default. It needs to be created either by a script, or by the CREATE management option. PILAR fills the table if it exists.

9.16 Safeguard codes

Safeguards are presented with dynamically generated codes. Pilar uses an internal code for long-term recovery. The following table establishes the relation. Most external users may safely ignore it.

```
CREATE TABLE safeguards (
    project VARCHAR(32) NOT NULL,
    code VARCHAR(32) NOT NULL,
    icode VARCHAR(128),
    PRIMARY KEY (project, code),
    FOREIGN KEY (project) REFERENCES model(project)
)
```

CODE stands for safeguard code (users code).

ICODE stands for internal safeguard code (internal code).

9.17 Backup equipment.

```
CREATE TABLE assetbackup (
    project VARCHAR(32) NOT NULL,
    asset VARCHAR(32) NOT NULL,
    phase VARCHAR(32) NOT NULL,
    step INT,
    maturity CHAR(1),
    cost FLOAT,
    criteria TEXT,
    comment TEXT,
    PRIMARY KEY (project, asset, phase),
    FOREIGN KEY (project) REFERENCES model(project),
    FOREIGN KEY (project, asset)
        REFERENCES assets(project, code),
    FOREIGN KEY (project, phase)
        REFERENCES phases(project, code)
)
```

9.18 Borders

```
CREATE TABLE zones (
    project TEXT(32),
    zone TEXT,
    attr TEXT(32),
    value TEXT,
    FOREIGN KEY (project) REFERENCES model(project)
);
```

```
CREATE TABLE borders (
    project TEXT(32) NOT NULL,
    reference TEXT(32) NOT NULL,
    from_ TEXT(32),
    to_ TEXT(32),
    PRIMARY KEY (project, reference),
    FOREIGN KEY (project) REFERENCES model(project)
);
```

```
CREATE TABLE borderassets (
    project TEXT(32),
    reference TEXT(32),
    asset TEXT(32),
    FOREIGN KEY (project) REFERENCES model(project),
    FOREIGN KEY (project, reference)
        REFERENCES borders(project, reference)
);
```

```
CREATE TABLE borderprotection (
    project TEXT(32),
    reference TEXT(32),
    phase TEXT(32),
    class TEXT(32),
    FOREIGN KEY (project) REFERENCES model(project),
    FOREIGN KEY (project, reference)
        REFERENCES borders(project, reference)
);
```

```
CREATE TABLE paths (
    project TEXT(32) NOT NULL,
    reference INTEGER NOT NULL,
    type TEXT(32),
    origin TEXT(32),
    before_ INTEGER,
    threat TEXT(32),
    zone TEXT(32),
    PRIMARY KEY (project, reference)
);
```

```
CREATE TABLE timing (
    project TEXT(32),
    reference INTEGER,
    phase TEXT(32),
    attack TEXT(32),
    detect TEXT(32),
    react TEXT(32)
);
```

9.19 Management System

```
CREATE TABLE msactions (
    project TEXT(32),
    position INTEGER,
    id TEXT(32),
    start INTEGER,
    end INTEGER,
    status TEXT(32),
    domain TEXT(32),
    description TEXT,
    comment TEXT,
    resources TEXT,
    FOREIGN KEY (project) REFERENCES model(project)
);
```

```
CREATE TABLE msactionattr (
    project TEXT(32),
    position INTEGER,
    attr TEXT(32),
    val TEXT(32),
    FOREIGN KEY (project) REFERENCES model(project)
);
```

```
CREATE TABLE msactioncontrol (
    project TEXT(32),
    position INTEGER,
    profile TEXT(32),
    code TEXT(32),
    FOREIGN KEY (project) REFERENCES model(project)
);
```

9.20 Scenarios

```
CREATE TABLE scenarios (
    project TEXT(32),
    position INTEGER,
    id TEXT(32),
    dim TEXT(32),
    freq REAL,
    description TEXT,
    auto INTEGER,
    FOREIGN KEY (project) REFERENCES model(project)
);
```

```
CREATE TABLE scenarioattr (
    project TEXT(32),
    position INTEGER,
    attr TEXT(32),
    val TEXT(32),
    FOREIGN KEY (project) REFERENCES model(project)
);
```

```
CREATE TABLE scenariocontrol (
    project TEXT(32),
    position INTEGER,
    profile TEXT(32),
```

```
    code TEXT(32),
    FOREIGN KEY (project) REFERENCES model(project)
);
```

```
CREATE TABLE scenariophase (
    project TEXT(32),
    position INTEGER,
    phase TEXT(32),
    impact TEXT(32),
    freq TEXT(32),
    risk TEXT(32),
    FOREIGN KEY (project) REFERENCES model(project)
);
```

10 Calculated values

10.1 Risk analysis: Impact and risk

10.1.1 Accumulated values

```
CREATE TABLE riskdown1 (
    project VARCHAR(32) NOT NULL,
    asset VARCHAR(32) NOT NULL,
    threat VARCHAR(32) NOT NULL,
    dimension VARCHAR(8) NOT NULL,
    level INT,
    value FLOAT,
    deg INT,
    freq FLOAT,
    reference INT,
    PRIMARY KEY (project, asset, threat, dimension),
    FOREIGN KEY (project)
        REFERENCES model(project)
)
```

```
CREATE TABLE riskdown2 (
    project VARCHAR(32) NOT NULL,
    reference INT NOT NULL,
    phase VARCHAR(32) NOT NULL,
    impact FLOAT,
    risk FLOAT,
    PRIMARY KEY (project, reference, phase),
    FOREIGN KEY (project)
        REFERENCES model(project),
    FOREIGN KEY (project, phase)
        REFERENCES phases(project, code)
)
```

REFERENCE in riskdown1 is linked to REFERENCE in riskdown2.

10.1.2 Deflected values

```
CREATE TABLE riskup1 (
    project VARCHAR(32) NOT NULL,
    above VARCHAR(32) NOT NULL,
    below VARCHAR(32) NOT NULL,
    threat VARCHAR(32) NOT NULL,
    dimension VARCHAR(8) NOT NULL,
    dimensionBelow VARCHAR(8) NOT NULL,
    level INT,
    value FLOAT,
    deg INT,
    freq FLOAT,
    reference INT,
    PRIMARY KEY (project, above, below,
        threat, dimension, dimensionBelow),
    FOREIGN KEY (project)
        REFERENCES model(project),
    FOREIGN KEY (project, above)
        REFERENCES assets(project, code),
    FOREIGN KEY (project, below)
        REFERENCES assets(project, code)
)
```

```
CREATE TABLE riskup2 (
    project VARCHAR(32) NOT NULL,
    reference INT NOT NULL,
    phase VARCHAR(32) NOT NULL,
    impact FLOAT,
    risk FLOAT,
    PRIMARY KEY (project, reference, phase),
    FOREIGN KEY (project)
        REFERENCES model(project),
    FOREIGN KEY (project, phase)
        REFERENCES phases(project, code)
)
```

REFERENCE in riskup1 is linked to REFERENCE in riskup2.

When the dimension of the deflected risk is different from the dimension on which the threats impacts, both dimensions are reported as “above/below”.

10.2 Business continuity: Impact and risk

10.2.1 Accumulated values

```
CREATE TABLE bcmdown1 (
    project VARCHAR(32) NOT NULL,
    asset VARCHAR(32) NOT NULL,
    threat VARCHAR(32) NOT NULL,
    reference INT,
    PRIMARY KEY (project, asset, threat),
    FOREIGN KEY (project)
        REFERENCES model(project)
)
```

```
CREATE TABLE bcmdown2 (
    project VARCHAR(32) NOT NULL,
    reference INT NOT NULL,
    phase VARCHAR(32) NOT NULL,
    step INT,
    freq FLOAT,
    impact FLOAT,
    risk FLOAT,
    PRIMARY KEY (project, reference, phase),
    FOREIGN KEY (project)
        REFERENCES model(project),
    FOREIGN KEY (project, phase)
        REFERENCES phases(project, code)
)
```

REFERENCE in bcmdown1 is linked to REFERENCE in bcmdown2.

10.2.2 Deflected values

```
CREATE TABLE bcmup1 (
    project VARCHAR(32) NOT NULL,
    above VARCHAR(32) NOT NULL,
    below VARCHAR(32) NOT NULL,
    threat VARCHAR(32) NOT NULL,
    reference INT,
    PRIMARY KEY (project, above, below, threat),
    FOREIGN KEY (project)
        REFERENCES model(project),
    FOREIGN KEY (project, above)
        REFERENCES assets(project, code),
    FOREIGN KEY (project, below)
        REFERENCES assets(project, code)
)
```

```
CREATE TABLE bcmup2 (
    project VARCHAR(32) NOT NULL,
    reference INT NOT NULL,
    phase VARCHAR(32) NOT NULL,
    step INT,
    freq FLOAT,
    impact FLOAT,
    risk FLOAT,
    PRIMARY KEY (project, reference, phase),
    FOREIGN KEY (project)
        REFERENCES model(project),
    FOREIGN KEY (project, phase)
        REFERENCES phases(project, code)
)
```

REFERENCE in bcmup1 is linked to REFERENCE in bcmup2.

11 Security profiles (EVL)

```
CREATE TABLE evl_asset (
    project VARCHAR(32),
    evl VARCHAR(32),
    control VARCHAR(32),
    asset VARCHAR(32),
    phase VARCHAR(32),
    value VARCHAR(16),
    comment TEXT,
    FOREIGN KEY (project, asset)
        REFERENCES assets(project, code),
    FOREIGN KEY (project, phase)
        REFERENCES phases(project, code)
)
```

```
CREATE TABLE evl_attr (
    project VARCHAR(32),
    evl VARCHAR(32),
    control VARCHAR(32),
    domain VARCHAR(32),
    attr VARCHAR(32),
    val VARCHAR(128),
    FOREIGN KEY (project)
        REFERENCES model(project),
    FOREIGN KEY (project, domain)
        REFERENCES domains(project, code)
)
```

```
CREATE TABLE evl_domain (
    project VARCHAR(32),
    evl VARCHAR(32),
    control VARCHAR(32),
    domain VARCHAR(32),
    phase VARCHAR(32),
    value VARCHAR(16),
    comment TEXT,
    FOREIGN KEY (project)
        REFERENCES model(project),
    FOREIGN KEY (project, domain)
        REFERENCES domains(project, code),
    FOREIGN KEY (project, phase)
        REFERENCES phases(project, code)
)
```

APPLIES encodes the applicability of the control:

- yes
- no
- mandatory

VALUE encodes the evaluation of the control:

- a maturity level (L0 – L5) if the control has further detail (tree expansion into sub-controls or safeguards)

- n.a. if it does not apply
- empty string if unknown

12 Library

Library elements go to another database to be shared between different projects.

The full library may be exported for report generation. This part cannot be imported back into PILAR.

The extensions may be exported and may be later imported on start-up.

12.1 Core elements

There are 5 tables:

```
mysql> show tables;
+-----------------------+
| Tables_in_bgr         |
+-----------------------+
| pilar_classes         |
| pilar_criteria        |
| pilar_safeguards      |
| pilar_threatmodifiers |
| pilar_threats         |
+-----------------------+
```

Each of them has the same structure:

```
CREATE TABLE pilar_classes (
    code VARCHAR(32) PRIMARY KEY NOT NULL,
    father VARCHAR(32),
    position INT,
    name TEXT
)
```

```
CREATE TABLE pilar_criteria (
    code VARCHAR(32) PRIMARY KEY NOT NULL,
    father VARCHAR(32),
    position INT,
    name TEXT
)
```

```
CREATE TABLE pilar_safeguards (
    code VARCHAR(32) PRIMARY KEY NOT NULL,
    father VARCHAR(32),
    position INT,
    name TEXT
)
```

```
CREATE TABLE pilar_threatmodifiers (
    code VARCHAR(32) PRIMARY KEY NOT NULL,
    father VARCHAR(32),
    position INT,
    name TEXT
)
```

```
CREATE TABLE pilar_threats (
    code VARCHAR(32) PRIMARY KEY NOT NULL,
    father VARCHAR(32),
    position INT,
    name TEXT
)
```

POSITION provides a relative index between the children of the same mother.

12.2 Extensions

There are 3 tables:

```
mysql> show tables;
+--------------------------+
| Tables_in_bgr            |
+--------------------------+
| extended_classes         |
| extended_criteria        |
| extended_threats         |
+--------------------------+
```

Each of them has the same structure:

```
CREATE TABLE extended_classes (
    code VARCHAR(32) PRIMARY KEY NOT NULL,
    father VARCHAR(32),
    position INT,
    name TEXT
)
```

```
CREATE TABLE extended_criteria (
    code VARCHAR(32) PRIMARY KEY NOT NULL,
    father VARCHAR(32),
    position INT,
    name TEXT
)
```

```
CREATE TABLE extended_threats (
    code VARCHAR(32) PRIMARY KEY NOT NULL,
    father VARCHAR(32),
    position INT,
    name TEXT
)
```

POSITION provides a relative index between the children of the same mother.

13 Mapping between databases

Tables in this document are described using MySQL syntax.

Column names and data types need to be adjusted to different vendors.

When column names are reserved words for some vendor, an underscore is added to the columns name

level → level_

Data types are mapped as following table:

DB2	MySQL	Oracle	PostgreSQL	SQL Server	SQLite
CHAR(1)	BOOLEAN	NUMBER(1)	CHAR(1)	TINYINT	INTEGER
CHAR(n)	CHAR(n)	CHAR(n)	CHAR(n)	NCHAR(n)	TEXT
DOUBLE	FLOAT	FLOAT	FLOAT(n)	FLOAT(n)	REAL
INTEGER	INT	NUMBER(n, 0)	INT	INT	INTEGER
CLOB(65535)	TEXT	VARCHAR2	TEXT	NVARCHAR(MAX)	TEXT
TIMESTAMP	TIMESTAMP	DATE	TIMESTAMP	DATETIME	INTEGER
VARCHAR(n)	VARCHAR(n)	VARCHAR2(n)	VARCHAR(n)	NVARCHAR(n)	TEXT