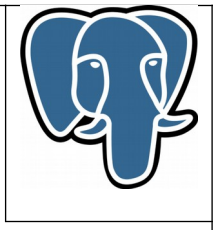


E-Maj

A PostgreSQL extension

Reference Guide

Version: 3.3.0

Last update: 14 March 2020

Summary

1 - Introduction....7
1.1 - Document content....7
1.2 - License....7
1.3 - E-Maj's objectives....7
1.4 - Main components....8
2 - How E-Maj works....9
2.1 - Concepts....9
2.1.1 - Tables Group....9
2.1.2 - Mark....9
2.1.3 - Rollback....9
2.2 - Architecture....10
2.2.1 - Logged SQL statements....10
2.2.2 - Created objects....10
2.2.3 - Schemas....11
2.2.4 - Norm for E-Maj objects naming....12
2.2.5 - Tablespaces....12
3 - How to install E-Maj....13
3.1 - Quick start....13
3.1.1 - Software install....13
3.1.2 - Extension install....13
3.1.3 - Extension use....14
3.2 - Installing the E-Maj software....15
3.2.1 - Downloading sources....15
3.2.2 - Standart installation on Linux....15
3.2.3 - Minimum installation on Linux....16
3.2.4 - Installation on Windows....16
3.2.5 - Alternate location of SQL installation scripts....16
3.3 - E-Maj extension setup....17
3.3.1 - Optional preliminary operation....17
3.3.2 - Standart creation of the emaj EXTENSION....17
3.3.3 - Creating the extension by script....18
3.3.4 - Changes in postgresql.conf configuration file....18
3.3.5 - E-Maj parameters....19
3.4 - Update an existing E-Maj version....20
3.4.1 - General approach....20

3.4.2 - Upgrade by deletion and re-installation....20
3.4.3 - Upgrade from an E-Maj version between 0.11.0 to 1.3.1....22
3.4.4 - E-Maj upgrade from 1.3.1 to a higher version....23
3.4.5 - Upgrade an E-Maj version already installed as an extension....24
3.5 - Uninstalling an E-Maj extension from a database....25
3.6 - PostgreSQL version upgrade....26
3.6.1 - Changing PostgreSQL minor versions....26
3.6.2 - Changing the major PostgreSQL version and the E-Maj version simultaneously....26
3.6.3 - Changing the PostgreSQL major version and keeping the existing E-Maj environment....26
3.6.4 - Post migration adaptation script....26

4 - How to use E-Maj....28
4.1 - Set-up the E-Maj access policy....28
4.1.1 - E-Maj roles....28
4.1.2 - Giving E-Maj rights....28
4.1.3 - Giving rights on application tables and objects....28
4.1.4 - Synthesis....29
4.2 - Creating and dropping tables groups....30
4.2.1 - Tables groups configuration principles....30
4.2.2 - Tables group creation....31
4.2.3 - Drop a tables group....36
4.3 - Main functions....37
4.3.1 - Operations chain....37
4.3.2 - Start a tables group....38
4.3.3 - Set an intermediate mark....39
4.3.4 - Rollback a tables group....40
4.3.5 - Perform a logged rollback of a tables group....41
4.3.6 - Stop a tables group....44
4.4 - Modifying tables groups....46
4.4.1 - General information....46
4.4.2 - The “Dynamic adjustment” method....47
4.4.3 - The « emaj_group_def update » method....50
4.4.4 - Incidence of tables or sequences addition or removal in a group in LOGGING state....51
4.4.5 - Reparing a tables group....53
4.4.6 - Combining dynamic tables groups management and configuration with emaj_group_def....54
4.5 - Other groups management functions....56
4.5.1 - Reset log tables of a group....56
4.5.2 - Comments on groups....56

4.5.3 - Protection of a tables group against rollbacks....56
4.5.4 - Forced stop of a tables group....57
4.5.5 - Forced suppression of a tables group....58
4.5.6 - Logged rollback consolidation....58
4.5.7 - List of “consolidable rollbacks”....59
4.5.8 - Exporting and importing tables groups configurations....60
4.6 - Marks management functions....62
4.6.1 - Comments on marks....62
4.6.2 - Search a mark....63
4.6.3 - Rename a mark....63
4.6.4 - Delete a mark....63
4.6.5 - Delete oldest marks....64
4.6.6 - Protection of a mark against rollbacks....65
4.7 - Statistics functions....67
4.7.1 - Global statistics about logs....67
4.7.2 - Detailed statistics about logs....68
4.7.3 - Estimate the rollback duration....69
4.8 - Data extraction functions....71
4.8.1 - Snap tables of a group....71
4.8.2 - Snap log tables of a group....72
4.8.3 - SQL script generation to replay logged updates....73
4.9 - Other functions....76
4.9.1 - Check the consistency of the E-Maj environment....76
4.9.2 - Exporting and importing parameters configurations....76
4.9.3 - Getting the current log table linked to an application table....78
4.9.4 - Not disabling application triggers at E-Maj rollback time....79
4.9.5 - Monitoring rollback operations....79
4.9.6 - Updating rollback operations state....81
4.9.7 - Deactivating or reactivating event triggers....81
4.10 - Multi-groups functions....83
4.10.1 - General information....83
4.10.2 - Functions list....83
4.10.3 - Syntax for groups array....83
4.10.4 - Other considerations....84
4.11 - Parallel Rollback client....85
4.11.1 - Sessions....85
4.11.2 - Prerequisites....85
4.11.3 - Syntax....86
4.11.4 - Examples....87
4.12 - Rollback monitoring client....88
4.12.1 - Prerequisite....88

4.12.2 - Syntax....88
- Examples....89

5 - Miscellaneous....90
5.1 - Parameters....90
5.2 - Log tables structure....91
5.2.1 - Standart structure....91
5.2.2 - Adding technical columns....92
5.3 - Reliability....92
5.3.1 - Internal checks....92
5.3.2 - Event triggers....93
5.4 - Traces of operations....93
5.5 - Impacts on instance and database administration....96
5.5.1 - Stopping and restarting the instance....96
5.5.2 - Saving and restoring the database....97
5.5.3 - Data load....99
5.5.4 - Tables reorganisation....99
5.5.5 - Using E-Maj with replication....99
5.6 - Sensitivity to system time change....103
5.7 - Performance....104
5.7.1 - Updates recording overhead....104
5.7.2 - E-Maj rollback duration....104
5.7.3 - Optimizing E-Maj operations....104
5.8 - Usage limits....106
5.9 - User's responsibility....107
5.9.1 - Defining tables groups content....107
5.9.2 - Appropriate call of main functions....107
5.9.3 - Management of application triggers....107
5.9.4 - Internal E-Maj table or sequence change....108

6 - Emaj_web....109
6.1 - Overview....109
6.2 - Emaj_web client installation....109
6.2.1 - Prerequisite....109
6.2.2 - Download....109
6.2.3 - Application configuration....109
6.3 - Using Emaj_web....110
6.3.1 - Access to Emaj_web and databases....110
6.3.2 - Tables groups list....111
6.3.3 - Some details about the user interface....112
6.3.4 - Tables group details....113
6.3.5 - Statistics....115

6.3.6 - Tables group content....116
6.3.7 - Schemas and tables groups configuration....117
6.3.8 - Monitoring rollback operations....119
6.3.9 - E-Maj environment state....120

7 - Contribute to the E-Maj development....122
7.1 - Build the E-Maj environment....122
7.1.1 - Clone the E-Maj repository....122
7.1.2 - Description of the E-Maj tree....122
7.1.3 - Setting tools parameters....122
7.2 - Coding....123
7.2.1 - Versionning....123
7.2.2 - Coding rules....123
7.2.3 - Version upgrade script....124
7.3 - Testing....125
7.3.1 - Create PostgreSQL instances....125
7.3.2 - Install software dependancies....125
7.3.3 - Execute non regression tests....125
7.3.4 - Test coverage....128
7.3.5 - Evaluate the performances....128
7.4 - Documenting....130
7.5 - Submitting a patch....130

8 - Appendix....131
8.1 - E-Maj functions list....131
8.1.1 - Tables or sequences level functions....131
8.1.2 - Groups level functions....133
8.1.3 - General purpose functions....135
8.2 - E-Maj distribution content....136
8.3 - PostgreSQL and E-Maj versions compatibility matrix....139

1 INTRODUCTION

1.1 DOCUMENT CONTENT

This document is a reference guide for the E-Maj PostgreSQL extension.

Chapter 2 presents the concepts used by E-Maj and the general architecture of the extension.

Chapter 3 describes E-Maj installation, update and uninstall procedures.

Chapter 4 details how to use E-Maj. It contains a description of each function.

Chapter 5 gives some additional information needed for a good understanding of how the extension works.

Then, chapter 6 presents Emaj_web, a web graphic interfaces that complement the E-Maj extension.

1.2 LICENSE

This extension and its documentation are distributed under GPL license (GNU - General Public License).

1.3 E-MAJ'S OBJECTIVES

E-Maj is the French acronym for « Enregistrement des Mises A Jour », which means « updates recording ».

It meets two main goals:

- E-Maj can be used to **trace updates** performed by application programs on the table's content. Viewing these recorded updates offers an answer to the need for “updates-auditing”,
- By using these recorded updates, E-Maj is able to **logically restore sets of tables into predefined states**, without being obliged to either restore all files of the PostgreSQL instance (cluster) or reload the entire content of the concerned tables.

In other words, E-Maj is a PostgreSQL extension which enables fine-grained write logging and time travel on subsets of the database.

It provides a good solution to :

- define save points at precise time on a set of tables,
- restore, if needed, this table set into a stable state, without stopping the instance,
- manage several save points, each of them being usable at any time as a restore point.

So, in a **production** environment, E-Maj may simplify the technical architecture, by offering a smooth and efficient alternative to time and/or disk consuming intermediate saves (*pg_dump*, *mirror disks*,...). E-Maj may also bring a help to the debugging by giving a way to precisely analyse how suspicious programs update application tables.

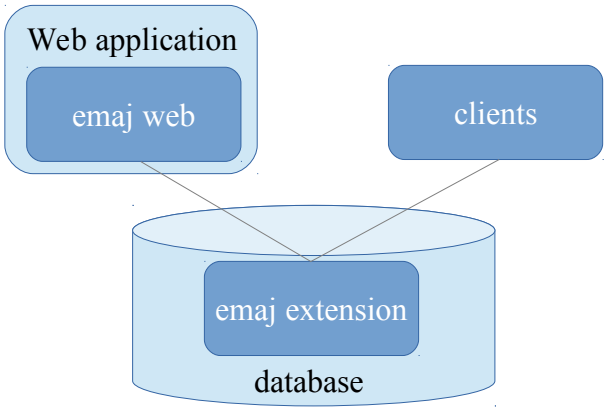
In a **test** environment, E-Maj also brings smoothness into operations. It is possible to very easily restore database subsets into predefined stable states, so that tests can be replayed as many times as needed.

1.4 MAIN COMPONENTS

E-Maj actually groups several components:

- a PostgreSQL *extension* object created into each database, named *emaj* and holding some tables, functions, sequences, ...
- a set of external clients working in command line interface,
- a web application,, *Emaj_web*.

The external clients and the GUI call the functions of the *emaj* extension.



All these components are discribed in this documentation.

2 HOW E-MAJ WORKS

2.1 CONCEPTS

E-Maj is built on three main concepts.

2.1.1 Tables Group

The « *tables group* » represents a set of application tables that live at the same rhythm, meaning that their content can be restored as a whole if needed. Typically, it deals with all tables of a database that are updated by one or more sets of programs. Each tables group is defined by a name which must be unique inside its database. By extent, a tables group can also contain partitions of partitionned tables and sequences. Tables (including partitions) and sequences that constitute a tables group can belong to different schemas of the database.

At a given time, a tables group is either in a « logging » state or in a « *idle* » state. The logging state means that all updates applied on the tables of the group are recorded.

A tables group can be either "*rollback-able*", which is the standard case, or "*audit_only*". In this latter case, it is not possible to rollback the group. But using this type of group allows to record tables updates for auditing purposes, even with tables that have no explicitely created primary key or with tables of type *UNLOGGED* or *WITH OIDS*.

2.1.2 Mark

A « *mark* » is a particular point in the life of a tables group, corresponding to a stable point for all tables and sequences of the group. A mark is explicitly set by a user operation. It is defined by a name that must be unique for the tables group.

2.1.3 Rollback

The « *rollback* » operation consists of resetting all tables and sequences of a group in the state they had when a mark was set.

There are two rollback types:

- with a « *unlogged rollback* », no trace of updates that are cancelled by the rollback operation are kept,
- with « *logged rollback* », update cancellations are recorded in log tables, so that they can be later cancelled: the rollback operation can be … rolled back.

Note that this concept of E-Maj rollback is different from the usual concept of “transactions rollback” managed by PostgreSQL.

2.2 ARCHITECTURE

In order to be able to perform a rollback operation without having previously kept a physical image of the PostgreSQL instance's files, all updates applied on application tables must be recorded, so that they can be cancelled.

With E-Maj, this updates recording takes the following form.

2.2.1 Logged SQL statements

The recorded update operations concerns the following SQL verbs:

- rows insertions:
  - *INSERT*, either elementary (*INSERT … VALUES*) or set oriented (*INSERT … SELECT*)
  - *COPY … FROM*
- rows updates:
  - *UPDATE*
- rows deletions:
  - *DELETE*
- tables truncations
  - *TRUNCATE*

For statements that process several rows, each creation, update or deletion is individually recorded. For instance, if a “*DELETE FROM <table>*” is performed against a table having 1 million rows, 1 million row deletion events are recorded.

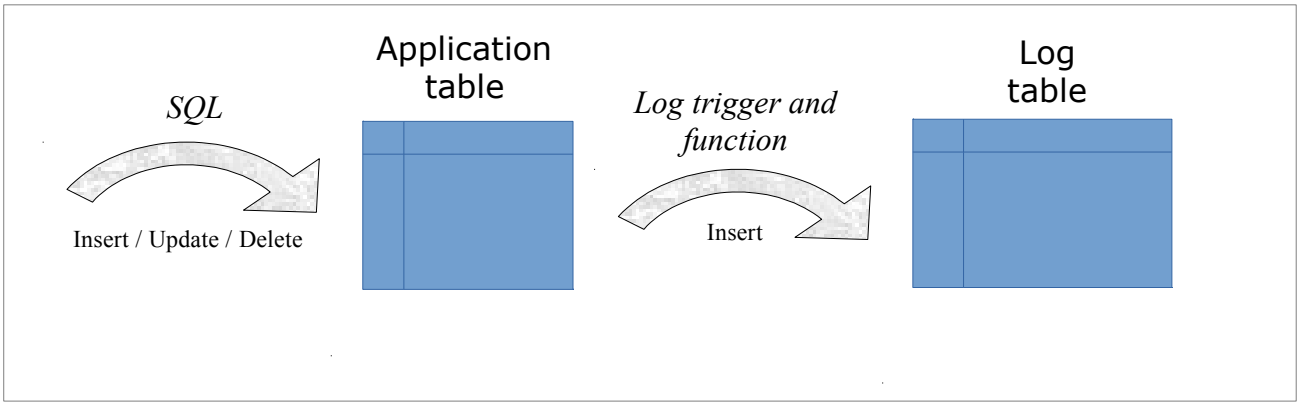
The case of *TRUNCATE* SQL verbs is specific. As no “*FOR EACH ROW*” trigger can be fired for this verb, the consequences of a *TRUNCATE* cannot be cancelled by E-Maj. Therefore, its execution is forbidden for “*rollbackable*” tables groups in “*logging”* state. In contrast, *TRUNCATE* is always permitted for “*audit_only*” tables groups. In such a case, only its execution is recorded.

2.2.2 Created objects

For each application table, the following objects are created:

- a dedicated log table, containing data corresponding to the updates applied on the application table,
- a trigger and a specific function, that, for each row creation (*INSERT, COPY*), change (*UPDATE*) or suppression (*DELETE*), record into the log table all data needed to potentially cancel later this elementary action,
- another trigger, that either blocks any execution of a *TRUNCATE* SQL verb for “*rollbackable*” tables groups or records the execution of a *TRUNCATE* SQL verb for “*audit_only*” tables groups,

- a sequence used to quickly count the number of updates recorded in log tables between 2 marks.



A log table has the same structure as its corresponding application table. However, it contains some additional technical columns, described at §5.2.1.

To let E-Maj work, some other technical objects are also created at extension installation time:

- 17 tables,
- 8 composite and 3 enum types,
- 1 view,
- 2 triggers,
- more than 150 functions, more than 70 of them being directly callable by users (cf §8.1),
- 1 sequence named *emaj_global_seq* used to assign to every update recorded in any log table of the database a unique identifier with an increasing value over time,
- 1 specific schema, named *emaj*, that contains all these relational objects,
- 2 roles acting as groups (NOLOGIN): *emaj_adm* to manage E-Maj components, and *emaj_viewer* to only look at E-Maj components
- 3 event triggers.

Technical tables, whose structure is interesting to know, are described in the coming chapters (*emaj_group_def* is described in §4.2.2.2, *emaj_param* is described in §5.1 and *emaj_hist* is described in §5.4).

2.2.3 Schemas

Almost all technical objects created at E-Maj installation are located into the schema named *emaj*. The only exception is the event trigger « *emaj_protection_trg* » that belongs to the *public* schema.

All objects linked to application tables are stored into schemas named :

emaj_<schema>

where <schema> is the schema name of the application tables.

The creation and the suppression of log schemas are only managed by E-Maj functions. They should NOT contain any other objects than those created by the extension.

2.2.4 Norm for E-Maj objects naming

For an application table, the log objects name is prefixed with the table name. More precisely, for an application table:

- the name of the log table is:
  <table.name>_log

- the name of the log function is:
  <table.name>_log_fnct

- the name of the sequence associated to the log table is:
  <table.name>_log_seq

For application tables whose name is very long (over 50 characters), the prefix used to build the log objects name is generated so it respects the PostgreSQL naming rules and avoids name conflict.

A log table name may contain a suffix like "_1", "_2", etc. In such a case, it deals with an old log table that has been renamed.

Other E-Maj function names are also normalised:
- function names that begin with 'emaj_' are functions that are callable by users,
- function names that begin with '_' are internal functions that should not be called directly.

Triggers created on application tables have the same name:
- *emaj_log_trg* for the log triggers,
- *emaj_trunc_trg* for the triggers that manage TRUNCATE verbs.

The name of event triggers starts with "*emaj_*" and ends with "*_trg*".

2.2.5 Tablespaces

When the extension is installed, the E-Maj technical tables are stored into the default tablespace set at instance or database level or explicitely set for the current session.

The same rule applies for log tables and index. But using tables group parameters (Refer to §Erreur : source de la référence non trouvée), it is also possible to store log tables and/ or their index into specific tablespaces.

3 HOW TO INSTALL E-MAJ

In this chapter, we will describe how to download and install or upgrade the E-Maj extension. Uninstallation is also discussed in this chapter.

3.1 QUICK START

The E-Maj installation is described in detail in the coming chapter. But the few following commands allow to quicky install and use E-Maj under Linux.

3.1.1 Software install

To install E-Maj, log on your *postgres* (or another) account, download E-Maj from PGXN (https://pgxn.org/dist/e-maj/) and type:

```
unzip e-maj-<version>.zip

cd e-maj-<version>/

sudo cp emaj.control $(pg_config --sharedir)/extension/.

sudo cp sql/emaj--* $(pg_config --sharedir)/extension/.
```

The chapter §3.2 presents other ways to download and install the software.

3.1.2 Extension install

To install the *emaj* extension into a database, log on the target database, using a *superuser* role and execute:

```
create extension emaj cascade;

grant emaj_adm to <role>;
```

For PostgreSQL versions prior 9.6, see §3.3.2.

With the latest statement, you give E-Maj administration grants to a particular role. Then, this role can be used to execute all E-Maj operations, avoiding the use of *superuser* role.

3.1.3 Extension use

You can now log on the database with the role having the E-Maj administration rights.

Then, an empty (here *ROLLBACKABLE*) tables group must be created:

```
SELECT emaj.emaj_create_group('my_group', true, true);
```

The tables group can now be populated with tables and sequences, using statements like:

```
SELECT emaj.emaj_assign_table('my_schema', 'my_table', 'my_group');
```

to add a table into the group, or, to add all tables and sequences of a given schema

```
SELECT emaj.emaj_assign_tables('my_schema', '.*', '', 'my_group');
SELECT emaj.emaj_assign_sequences('my_schema', '.*', '', 'my_group');
```

Note that only tables having a primary key will be effectively assigned to a *ROLLBACKABLE* group.

Then the typical commands sequence:

```
SELECT emaj.emaj_start_group('my_group', 'Mark-1');
[INSERT/UPDATE/DELETE on tables]
SELECT emaj.emaj_set_mark_group('my_group','Mark-2');
[INSERT/UPDATE/DELETE on tables]
SELECT emaj.emaj_set_mark_group('my_group','Mark-3');
[INSERT/UPDATE/DELETE on tables]
SELECT emaj.emaj_rollback_group('my_group','Mark-2');
SELECT emaj.emaj_stop_group('my_group');
SELECT emaj.emaj_drop_group('my_group');
```

will start the tables group, log updates and set several intermediate marks, go back to one of them, stop the recording and finally drop the group.

Additionally, the Emaj_web can also be installed and used (see §6).

3.2 INSTALLING THE E-MAJ SOFTWARE

3.2.1 Downloading sources

E-Maj is available for download on the Internet site PGXN, the PostgreSQL Extension Network (https://pgxn.org/dist/e-maj/).

E-Maj and its add-ons are also available on the github.org Internet site:

- source components (https://github.com/dalibo/emaj)
- documentation (https://github.com/beaud76/emaj_doc)
- Emaj_web GUI (https://github.com/dalibo/emaj_web)

For the records, an old repository (https://github.com/beaud76/emaj_ppa_plugin) contains a plugin for *phpPgAdmin*. It is not maintained since E-Maj 3.0.

3.2.2 Standart installation on Linux

Download the latest E-Maj version by any convenient way. If the *pgxn* client is installed, just execute the command:

```
pgxn download E-Maj
```

Then decompress the downloaded archive file with the commands::

```
unzip e-maj-<version>.zip

cd e-maj-<version>/
```

Identify the precise location of the *SHAREDIR* directory. Depending on the PostgreSQL installation, the *pg_config --sharedir* shell command may directly report this directory name. Otherwise, look at typical locations like:

- */usr/share/postgresql/<pg_version>* for Debian or Ubuntu
- */usr/pgsql-<pg_version>/share* for RedHat or CentOS

Copy some files to the extension directory of the postgresql version you want to use. As a super-user or pre-pended with *sudo*, type:

```
cp emaj.control <SHAREDIR_directory>/extension/.

cp sql/emaj--* <SHAREDIR_directory>/extension/.
```

The latest E-Maj version is now installed and referenced by PostgreSQL. The e-maj-<version> directory contains the file tree described at §8.2.

3.2.3 Minimum installation on Linux

On some environments (like DBaaS clouds for instance), it is not allowed to add extensions into the *SHAREDIR* directory. For these cases, a minimum installation is possible.

Download the latest E-Maj version by any convenient way and decompress it.

If the *pgxn* client is installed, just execute the commands:

```
pgxn download E-Maj

unzip e-maj-<version>.zip
```

The e-maj-<version> directory contains the file tree described at §8.2.

The extension creation will be a little bit different (§3.3).

3.2.4 Installation on Windows

To install E-Maj on Windows:

- Download the extension from the pgxn.org site,
- Extract the file tree from the downloaded zip file,
- Copy the files *emaj.control* and *sql/emaj--** into the *share\extension* folder of the PostgreSQL installation folder (typically *c:\Program_Files\PostgreSQL\<postgres_version>*)

3.2.5 Alternate location of SQL installation scripts

The *emaj.control* file located in the *SHAREDIR/extension* directory of the PostgreSQL version, may contain a directive that defines the directory where SQL installation scripts are located.

So it is possible to only put the *emaj.control* file into this *SHAREDIR/extension* directory, by creating a pointer towards the script directory.

To setup this, just:

- copy the emaj.contol file from the root directory of the decompressed structure into the *SHAREDIR/extension* directory,

- adjust the *directory* parameter of the *emaj.control* file to reflect the actual location of the E-Maj SQL scripts.

3.3 E-MAJ EXTENSION SETUP

If an extension already exists in the database, but in an old E-Maj version, please go on with §3.4.

The standart way to install E-Maj consists in creating an *EXTENSION* object (in the PostgreSQL terminology). To achieve this task, the user must be logged to the database as *superuser*.

In environments for which this is not possible (cases of minimum installation – see §3.2.3), a *psql* script can be executed.

3.3.1 Optional preliminary operation

The technical tables of the E-Maj extension are created into the default tablespace. If the E-Maj administrator wants to store them into a dedicated tablespace, he can create it if needed and define it as the default tablespace for the session:

```
SET default_tablespace = <tablespace.name>;
```

3.3.2 Standart creation of the emaj EXTENSION

3.3.2.1 PostgreSQL version 9.6 and above

The E-Maj extension can now be created into the database, by executing the SQL command:.

```
CREATE EXTENSION emaj CASCADE;
```

To start with, the script verifies that the PostgreSQL version is at least 9.5, and that the current user has the *superuser* attribute.

Then the script creates the *emaj* schema with its technical tables, types and functions.

The emaj schema must only contain E-Maj related objects.

If they are not already present, both *emaj_adm* and *emaj_viewer* roles are created.

Finally, the installation script looks at the instance configuration and may display a warning message regarding the *-max-prepared-statements* parameter (see §4.11.2).

3.3.2.2 PostgreSQL version 9.5

With PostgreSQL version prior 9.6, the *CASCADE* clause doesn't exist. In that case, the required extensions must be explicitely created, if needed, before emaj.

```
CREATE EXTENSION IF NOT EXISTS dblink;
CREATE EXTENSION IF NOT EXISTS btree_gist;
CREATE EXTENSION emaj;
```

3.3.3 Creating the extension by script

When creating the emaj *EXTENSION* is not possible, a *psql* script can be used instead.

```
\i <emaj_directory>/sql/emaj-<version>.sql
```

where <emaj_directory> is the directory generated by the E-Maj installation (see §3.2.3) and <version> the current E-Maj version.

It is not mandatory to execute the installation script as superuser. But if it is not the case, the role used for this installation will need the rights to create triggers on the application tables of the future tables groups.

3.3.4 Changes in postgresql.conf configuration file

Main E-Maj functions set a lock on each table of a processed tables group. If some groups contains a large number of tables, it may be necessary to increase the value of the ***max_locks_per_transaction*** parameter in the *postgresql.conf* configuration file. This parameter is used by PostgreSQL to compute the size of the "*shared lock table*" that tracks locks for the whole instance. Its default value equals 64. It can be increased if an E-Maj operation fails with a message indicating that all entries of the "*shared lock table*" have been used.

Furthermore, if the parallel rollback tool may be used (see § 4.11), it will be probably necessary to adjust the ***max_prepared_transaction*** parameter.

3.3.5 E-Maj parameters

Several parameters have an influence on the E-Maj behaviour. They are presented in details in chapter §5.1.

The parameters setting step is optional. E-Maj works well with the default parameter values.

However, if the E-Maj administrator wishes to take benefit from the rollback operations monitoring capabilities, it is necessary to insert a row into the *emaj_param* table to setup the value of the "*dblink_user_password*" parameter (see §4.9.5.1)

Test and demonstration

It is possible to check whether the E-Maj installation works fine, and discover its main features by executing a demonstration script. Under psql, just execute the *emaj_demo.sql* script that is supplied with the extension.

```
\i <emaj_directory>/sql/demo.sql
```

If no error is encountered, the script displays this final message:

This ends the E-Maj demo. Thank You for using E-Maj and have fun!

Examining the messages generated by the script execution, allows to discover most E-Maj features. Once the script execution is completed, the demonstration environment is left as is, so that it remains possible to examine it or to play with it. To suppress it, execute the cleaning function that the script has created.

```
SELECT emaj.emaj_demo_cleanup();
```

This drops the *emaj_demo_app_schema* schema and both *emaj demo group 1* and *emaj demo group 2*' tables groups.

3.4 UPDATE AN EXISTING E-MAJ VERSION

3.4.1 General approach

The first step consists in installing the new version of the E-Maj software, using the procedure described in §3.2. Keep the old E-Maj version directory at least until the end of the upgrade. Some files may be needed.

It is also necessary to check whether some preliminary operations as described in §3.3.1 must be executed.

Then, the process to upgrade an E-Maj extension in a database depends on the already installed E-Maj version.

For E-Maj versions prior 0.11.0, there is no particular update procedure. A simple E-Maj deletion and then re-installation has to be done, as described in §3.4.2. This approach can also be used for any E-Maj version, even though it has a drawback: all log contents are deleted, resulting in no further way to rollback or look at the recorded updates.

For installed E-Maj version 0.11.0 and later, it is possible to perform an upgrade without E-Maj deletion.

The upgrade procedure for an existing E-Maj version between 0.11.0 and 1.3.1 is described in §3.4.3.
The upgrade from version 1.3.1 to a higher version is described in §3.4.4.
The upgrade from a version greater or equal 2.0.0 is described in §3.4.5.

Starting from version 2.2.0, E-Maj no longer supports PostgreSQL versions prior 9.2. Starting from version 3.0.0, E-Maj no longer supports PostgreSQL versions prior 9.5. If an older PostgreSQL version is used, it must be updated before migrating E-Maj to a higher version.

3.4.2 Upgrade by deletion and re-installation

For this upgrade path, it is not necessary to use the full un-installation procedure described in §3.5. In particular, the tablespace and both roles can remain as is. However, it may be judicious to save some useful pieces of information. Here is a suggested procedure.

3.4.2.1 Stop tables groups

If some tables groups are in *LOGGING* state, they must be stopped, using the *emaj_stop_group()* function (see §4.3.6) (or the *emaj_force_stop_group()* function if *emaj_stop_group()* (see §4.5.4) returns an error).

3.4.2.2 Save user data

The full existing tables groups configuration, as well as E-Maj parameters, can be saved on flat files, using:

```
SELECT emaj.emaj_export_groups_configuration('<file.path.1>');

SELECT emaj.emaj_export_parameters_configuration
      ('<file.path.2>');
```

If the installed E-Maj version is prior 3.3.0, these functions are not available. Thus, the *emaj_group_def* and *emaj_param* terchnical tables must be manually saved. For instance, one can download each table on file with a *copy* command, or duplicate them ouside the *emaj* schema.

Il the current E-Maj version contains the *emaj_sync_def_group()* function (3.2+), it is recommended to use it to previously synchronize the *emaj_group_def* table with the current groups structure.

If the installed E-Maj version is 3.1.0 or higher, and if the E-Maj administrator has registered application triggers as “not to be automatically disabled at E-Maj rollback time”, the *emaj_ignored_app_trigger* table must also be saved.

```
SELECT group_name, emaj.emaj_sync_def_group(group_name)
      FROM emaj.emaj_group;

CREATE TABLE public.sav_group_def AS
      SELECT * FROM emaj.emaj_group_def;

CREATE TABLE public.sav_ignored_app_trigger AS
      SELECT * FROM emaj.emaj_ignored_app_trigger;

CREATE TABLE public.sav_param AS
      SELECT * FROM emaj.emaj_param WHERE param_key <> 'emaj_version';
```

3.4.2.3 E-Maj deletion and re-installation

Once connected as super-user, just chain the execution of the uninstall.sql script, of the current version and then the extension creation.

```
\i <old_emaj_directory>/sql/uninstall.sql

CREATE EXTENSION emaj;
```

NB: starting from E-Maj 2.0.0, the uninstall script is named *emaj_uninstall.sql*.

3.4.2.4 Restore user data

If the tables groups and the parameters configurations have been exported into files using the *emaj_export_parameters_configuration()* and *emaj_export_groups_configuration()* functions, they can be reloaded with:

```
SELECT emaj.emaj_import_parameters_configuration
      ('<file.path.2>', TRUE);

SELECT emaj.emaj_import_groups_configuration('<file.path.1>');
```

It the configurations have been previously saved by downloading or copying the technical tables, they can be reloaded for instance with INSERT SELECT statements::

```
INSERT INTO emaj.emaj_group_def
      SELECT grpdef_group, grpdef_schema, grpdef_tblseq,
             grpdef_priority, grpdef_log_dat_tsp, grpdef_log_idx_tsp
      FROM public.sav_group_def;

INSERT INTO emaj.emaj_ignored_app_trigger
      SELECT * FROM public.sav_ignored_app_trigger;

INSERT INTO emaj.emaj_param
      SELECT * FROM public.sav_param;
```

Once data are copied, temporary tables or files can be deleted.

3.4.3 Upgrade from an E-Maj version between 0.11.0 to 1.3.1

For installed version between 0.11.0 and 1.3.1, psql upgrade scripts are supplied. They allow to upgrade from one version to the next one.

Each step can be performed without impact on existing tables groups. They may even remain in *LOGGING* state during the upgrade operations. This means in particular that:

- updates on application tables can continue to be recorded during and after this version change,
- a « *rollback* » on a mark set before the version change can also be performed after the migration.

Source version	Target version	psql script	Duration	Concurrent updates (1)
0.11.0	0.11.1	emaj-0.11.0-to-0.11.1.sql	Very quick	Yes
0.11.1	1.0.0	emaj-0.11.1-to-1.0.0.sql	Very quick	Yes
1.0.0	1.0.1	emaj-1.0.0-to-1.0.1.sql	Very quick	Yes
1.0.1	1.0.2	emaj-1.0.1-to-1.0.2.sql	Very quick	Yes
1.0.2	1.1.0	emaj-1.0.2-to-1.1.0.sql	Variable	No (2)
1.1.0	1.2.0	emaj-1.1.0-to-1.2.0.sql	Very quick	Yes
1.2.0	1.3.0	emaj-1.2.0-to-1.3.0 .sql	Quick	Yes (3)
1.3.0	1.3.1	emaj-1.3.0-to-1.3.1.sql	Very quick	Yes

(1) The last column indicates whether the E-Maj upgrade can be executed while application tables handled by E-Maj are accessed in update mode. Note that any other E-Maj operation executed during the upgrade operation would wait until the end of the upgrade.
(2) When upgrading into 1.1.0, log tables structure changes. As a consequence:

- eventhough tables groups may remain in *LOGGING* state, the upgrade can only be executed during a time period when application tables are not updated by any application processing,
- the operation duration will mostly depends on the volume of data stored into the log tables.

Note also that E-Maj statistics collected during previous rollback operations are not kept (due to large differences in the way rollbacks are performed, the old statistics are not pertinent any more).
(3) It is advisable to perform the upgrade into 1.3.0 in a period of low database activity. This is due to *Access Exclusive* locks that are set on application tables while the E-Maj triggers are renamed.

At the end of each upgrade step, the script displays the following message :

>>> E-Maj successfully migrated to <new_version>

3.4.4 E-Maj upgrade from 1.3.1 to a higher version

The upgrade from the 1.3.1 version is specific as it must handle the installation mode change, moving from a *psql* script to an *extension*.

Concretely, the operation is performed with a single SQL statement:

```
CREATE EXTENSION emaj FROM unpackaged;
```

The PostgreSQL extension manager determines the scripts to execute depending on the E-Maj version identifier found in the *emaj.control* file.

But this upgrade is not able to process cases when at least one tables group has been created with a PostgreSQL version prior 8.4. In such a case, these old tables groups must be dropped before the upgrade and recreated after.

3.4.5 Upgrade an E-Maj version already installed as an extension

An existing version already installed as an extension can be upgraded using the SQL statement:

```
ALTER EXTENSION emaj UPDATE;
```

The PostgreSQL extension manager determines the scripts to execute depending on the current installed E-Maj version and the version found in the *emaj.control* file.

The operation is very quick et does not alter tables groups. They may remain in *LOGGING* state during the upgrade. As for previous upgrades, this means that:

- updates on application tables can continue to be recorded during and after this version change,
- a « *rollback* » on a mark set before the version change can also be performed after the migration.

Version specific details:

- The procedure that upgrades a version 2.0.1 into 2.1.0, may modify the *emaj_group_def* table in order to reflect the fact that the *tspemaj* tablespace is not automatically considered as a default tablespace anymore. If *tspemaj* was effectively used as default tablespace for created tables groups, the related *grpdef_log_dat_tsp* and *grpdef_log_idx_tsp* columns content of the *emaj_group_def* table is automatically adjusted so that a future drop and recreate operation would store the log tables and indexes in the same tablespace. The administrator may review these changes to be sure this corresponds to his expectations. (Cf §4.2.2.2)
- The procedure that upgrades a version 2.2.2 into 2.2.3 checks the recorded log sequences values. In some cases, it may ask for a preliminary reset of some tables groups.
- The procedure that upgrades a version 2.3.1 into 3.0.0 changes the structure of log tables: both *emaj_client_ip* and *emaj_client_port* columns are not created anymore. Existing log tables are not modified. Only the new log tables are impacted. But the administrator can add these columns, by using the '*alter_log_tables*' parameter (Cf §5.2.2).
- The procedure that upgrades a version 3.0.0 into 3.1.0 renames existing log objects. This leads to locking the application tables, which may generate conflicts with the parallel use of these tables. This procedure also issues a warning message indicating that the changes in E-Maj rollback functions regarding the application triggers processing may require changes in user's procedures.

3.5 UNINSTALLING AN E-MAJ EXTENSION FROM A DATABASE

To uninstall E-Maj from a database, the user must log on this database with psql, as a superuser.

If the drop of the *emaj_adm* and *emaj_viewer* roles is desirable, rights on them given to other roles must be previously deleted, using *REVOKE* SQL verbs.

```
REVOKE emaj_adm FROM <role.or.roles.list>;
REVOKE emaj_viewer FROM <role.or.roles.list>;
```

If these *emaj_adm* and *emaj_viewer* roles own access rights on other application objects, these rights must be suppressed too, before starting the uninstall operation.

Allthough E-Maj is usualy installed with a CREATE EXTENSION statement, it cannot be uninstalled with a simple *DROP EXTENSION* statement. An event trigger blocks such a statement.

To uninstall E-Maj, just execute the *emaj_uninstall.sql* delivered script.

```
\i <emaj_directory>/emaj_uninstall.sql
```

This script performs the following steps:

- it executes the cleaning functions created by demo or test scripts, if they exist,
- it stops the tables groups in *LOGGING* state, if any,
- it drops the tables groups, removing in particular the triggers on application tables,
- it drops the extension and the main *emaj* schema,
- it drops roles *emaj_adm* and *emaj_viewer* if they are not linked to any objects in the current database or in other databases of the instance.

The uninstallation script execution displays:

```
$ psql ... -f sql/emaj_uninstall.sql
>>> Starting E-Maj uninstallation procedure...
SET
psql:sql/emaj_uninstall.sql:203: WARNING:  emaj_uninstall:
emaj_adm and emaj_viewer roles have been dropped.
DO
SET
>>> E-maj successfully uninstalled from this database
```

3.6 POSTGRESQL VERSION UPGRADE

3.6.1 Changing PostgreSQL minor versions

As changing the minor PostgreSQL version only consists in replacing the binary files of the software, there is no particular constraint regarding E-Maj.

3.6.2 Changing the major PostgreSQL version and the E-Maj version simultaneously

A PostgreSQL major version change may be the opportunity to also change the E-Maj version. But in this case, the E-Maj environment has to be recreated from scratch and old objects (tables groups, logs, marks,…) cannot be reused.

3.6.3 Changing the PostgreSQL major version and keeping the existing E-Maj environment

Nevertheless, it is possible to keep the existing E-Maj components (tables groups, logs, marks,…) while changing the PostgreSQL major version. And the tables groups may event stay in logging state during the operation.

But 2 conditions must be met:

- the old and new instances must share the same E-Maj version,
- a post migration script must be executed, before any E-Maj use.

Of course, it is possible to upgrade the E-Maj version before or after the PostgreSQL version change.

If the PostgreSQL version upgrade is performed using a database dump and restore, and if the tables groups may be stopped, a log tables purge, using the *emaj_reset_group()* function, may reduce the volume of data to manipulate, thus reducing the time needed for the operation.

3.6.4 Post migration adaptation script

It may happen that a PostgreSQL major version change has an impact on the E-Maj extension content. Thus, a script is supplied to handle such cases.

After each PostgreSQL major version upgrade, a *psql* script must be executed as superuser:

```
\i <emaj_directory>/sql/emaj_upgrade_after_postgres_upgrade.sql
```

For E-Maj versions 2.0.0 and later, the script only creates the event triggers that may be missing:

- those that appear in version 9.3 and protect against the drop of the extension itself and the drop of E-Maj objects (log tables, functions,...),
- those that appear in version 9.5 and protect against log table structure changes.

The script may be executed several times on the same version, only the first execution modifying the environment.

4 HOW TO USE E-MAJ

4.1 SET-UP THE E-MAJ ACCESS POLICY

A bad usage of E-Maj can break the database integrity. So it is advisable to only authorise its use to specific skilled users.

4.1.1 E-Maj roles

To use E-Maj, it is possible to log on as superuser. But for safety reasons, it is preferable to take advantage of both roles created by the installation script:

- *emaj_adm* is used as the administration role ; it can execute all functions and access to all E-Maj tables, with reading and writing rights,
- *emaj_viewer* is used for read only purpose ; it can only execute statistics functions and can only read E-Maj tables.

All rights given to *emaj_viewer* are also given to *emaj_adm*.

When created, these roles have no connection capability (no defined password and *NOLOGIN* option). It is recommended NOT to give them any connection capability. Instead, it is sufficient to give the rights they own to other roles, with *GRANT* SQL verbs.

4.1.2 Giving E-Maj rights

Once logged on as superuser in order to have the sufficient rights, execute one of the following commands to give a role all rights associated to one of both *emaj_adm* or *emaj_viewer* roles:

```
GRANT emaj_adm TO <my.emaj.administrator.role>;
GRANT emaj_viewer TO <my.emaj.viewer.role>;
```

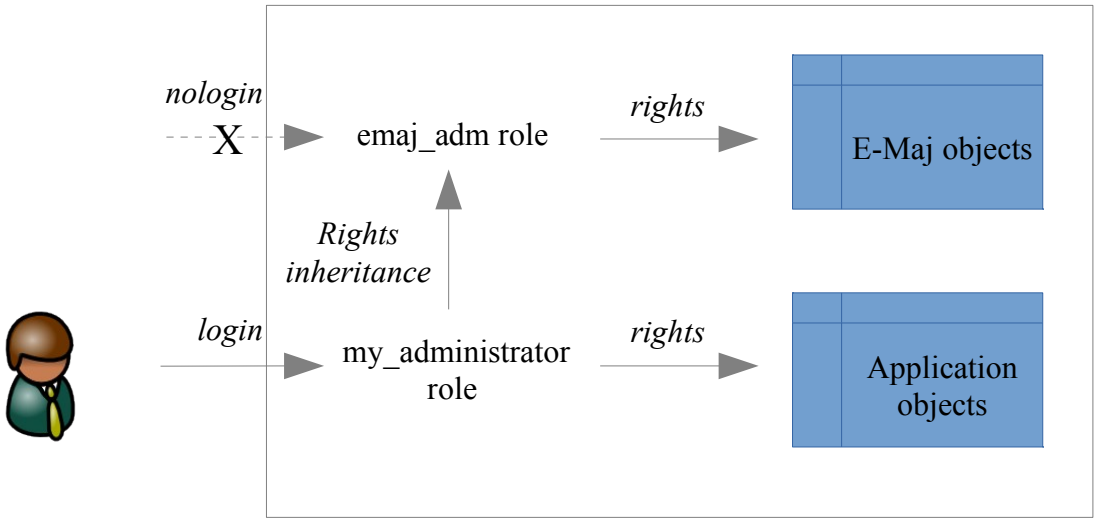
Of course, *emaj_adm* or *emaj_viewer* rights can be given to several roles.

4.1.3 Giving rights on application tables and objects

To let an E-Maj administrator also access application tables or other application objects (schemas, sequences, views, functions,...), it is possible to give rights on these objects to *emaj_adm* or *emaj_viewer* roles. But it is preferable to only give these rights to the roles which are also given *emaj_adm* or *emaj_viewer* rights, so that the E-Maj roles only directly own rights on E-Maj tables and objects.

4.1.4 Synthesis

The following schema represents the recommended rights organisation for an E-Maj administrator.



Of course the schema also applies to *emaj_viewer* role.

Except when explicitly noticed, the operations presented later can be indifferently executed by a superuser or by a role belonging to the *emaj_adm* group.

4.2 CREATING AND DROPPING TABLES GROUPS

4.2.1 Tables groups configuration principles

Configuring a tables group consists in:

- defining the tables group characteristics,
- defining the tables and sequences to assign to the group,
- optionnaly, defining some specific properties for each table.

4.2.1.1 The tables group

A tables group is identified by its **name**. Thus, the name must be unique withing the database. A tables group name contains at least 1 character. It may contain spaces and/or any punctuation characters. But it is advisable to avoid commas, single or double quotes.

At creation time, the "*ROLLBACKABLE*" or "*AUDIT_ONLY*" groups property (Cf §2.1.1) must be set. Note that this property cannot be modified once the tables group is created. If it needs to be changed, the tables group must be dropped and then recreated.

4.2.1.2 The tables and sequences to assign

A tables group can contain tables and/or sequences belonging to one or several schemas.

All tables of a schema are not necessarily member of the same group. Some of them can belong to another group. Some others can belong to any group.

But at a given time, **a table or a sequence** cannot be assigned to more than **one tables group**.

To guarantee the integrity of tables managed by E-Maj, it is essential to take a particular attention to the tables groups content definition. If a table were missing, its content would be out of synchronisation with other tables it is related to, after an E-Maj rollback operation. In particular, when application tables are created or suppressed, it is important to always maintain an up-to-date groups configuration.

All tables assigned to a "*ROLLBACKABLE*" group must have an explicit **primary key** (*PRIMARY KEY* clause in *CREATE TABLE* or *ALTER TABLE*).

E-Maj can process elementary partitions of partitionned tables created with the declarative DDL (with PostgreSQL 10+). They are processed as any other tables. However, as there is no need to protect mother tables, which remain empty, E-Maj refuses to include them in tables groups. All partitions of a partitionned table do not need to belong to a tables group. Partitions of a partitionned table can be assigned to different tables groups.

By their nature, *TEMPORARY TABLE* are not supported by E-Maj. *UNLOGGED* tables and tables created as *WITH OIDS* can only be members of "*audit_only"* tables groups.

If a sequence is associated to an application table, it is advisable to assign it into the same group as its table, so that, in case of E-maj rollback, the sequence can be reset to its state at the set mark time. If it were not the case, an E-Maj rollback would simply generate a hole in the sequence values.

E-Maj log tables and sequences should NOT be assigned in a tables group.

4.2.1.3 Specific tables properties

Three properties are associated to tables assigned to tables group:

- priority level,
- tablespace for log data,
- tablespace for log index.

The **priority** level is of type *INTEGER.* It is NULL by default. It defines a priority order in E-Maj tables processing. This can be especialy useful at table lock time. Indeed, by locking tables in the same order as what is typically done by applications, it may reduce the risk of deadlock. E-Maj functions process tables in priority ascending order, NULL being processed last. For a same priority level, tables are processed in alphabetic order of schema name and table name.

To optimize performances of E-Maj installations having a large number of tables, it may be useful to spread log tables and their index on several tablespaces. Two properties are available to specify:

- the name of the **tablespace** to use for the **log table** of an application table,
- the name of the **tablespace** to use for the **index** of the log table.

By default, these properties have a NULL value, meaning that the default tablespace of the current session at tables group creation is used.

4.2.2 Tables group creation

There are two main ways to handle the structure of tables groups:

- dynamically manage the tables and sequences assignment into tables groups, using dedicated functions,
- describe the list of tables groups and the tables and sequences they contain into the *emaj_group_def* configuration table.

Eventhough these methods are very different, they can be combined. (Cf §4.4.6)

4.2.2.1 The "dynamic configuration" method

Creating a tables group requires several steps. The first one consists in creating an empty tables group. Then one populates it with tables and sequences.

Creating the empty tables group

To create en empty tables group, just execute the following SQL statement:

```
SELECT emaj.emaj_create_group('<group.name>',<is_rollbackable>,
<is_empty>);
```

The second parameter, of type boolean, indicates whether the group's type is "*rollbackable*" (with value *TRUE*) or "*audit_only*" (with value *FALSE*) group. If this second parameter is not supplied, the group is considered "*rollbackable*".

Note that the third parameter must be explicitely set to *TRUE*.

If the group is also referenced into the *emaj_group_def* table, the *emaj_group_def* table's content is simply ignored.

The function returns the number of tables and sequences contained by the group.

Assigning tables and sequences into the tables group

Six functions allow to dynamically assign one or several tables or sequences to a group.

To add one or several tables into a tables group:

```
SELECT emaj.emaj_assign_table('<schema>','<table>',
      '<groupe.name>' [,'<properties>' [,'<mark>']]);
```

or

```
SELECT emaj.emaj_assign_tables('<schema>','<tables.array>',
      '<group.name>' [,'<properties>' [,'<mark>']] );
```

or

```
SELECT emaj.emaj_assign_tables('<schema>',
      '<tables.to.include.filter>','<tables.to.exclude.filter>',
      '<group.name>' [,'<properties>' [,'<mark>']] );
```

To add one or several sequences into a tables group:

```
SELECT emaj.emaj_assign_sequence('<schema>','<sequence>',
      '<group.name>' [,'<mark>'] );
```

or

```
SELECT emaj.emaj_assign_sequences('<schema>',
     '<sequences.array>', '<group.name>'
     [,'<mark>'] );
```

or

```
SELECT emaj.emaj_assign_sequences('<schema>',
     '<sequences.to.include.filter>','<sequences.to.exclude.filter>',
     '<group.name>' [,'<mark>'] );
```

For functions processing several tables or sequences in a single operation, the list of tables or sequences to process is:

- either provided by a parameter of type *TEXT* array,
- or built with two regular expressions provided as parameters.

A *TEXT* array is typically expressed with a syntax like:
ARRAY['element_1',' element_2', ...]

Both regular expressions follow the *POSIX* rules. Refer to the PostgreSQL documentation for more details. The first one defines a filter that selects the tables of the schema. The second one defines an exclusion filter applied on the selected tables. For instance:

- selects all tables or sequences of the schema my_schema
  `'my_schema', '.*', ''`
- select all tables of this schema and whose name start with 'tbl'
  `'my_schema', '^tbl.*', ''`
- select all tables of this schema and whose name start with 'tbl', except those who end with '_sav'
  `'my_schema', '^tbl.*', '_sav$'`

The functions assigning tables or sequences to tables groups that build their selection with regular expressions take into account the context of the tables or sequences. Are not selected for instance: tables or sequences already assigned, or tables without primary key for *rollbackable* groups, or tables declared *UNLOGGED*.

The <properties> parameter of functions that assign tables to a group allows to set values to some properties for the table or tables. Of type *JSONB,* its value can be set like this:

```
'{ "priority" : <n> , "log_data_tablespace" : "<xxx>" ,
"log_index_tablespace" : "<yyy>" }'
```

where:

- <n> is the priority level for the table or tables
- <xxx> is the name of the tablespace to handle log tables
- <yyy> is the name of the tablespace to handle log indexes

If one of these properties is not set, its value is considered NULL.

If specific tablespaces are referenced for any log table or log index, these tablespaces must exist before the function's execution.

For all these functions, an exclusive lock is set on each table of the concerned table groups, so that the groups stability can be guaranted during these operations.

All these functions return the number of assigned tables or sequences.

The tables assignment functions create all the needed log tables, the log functions and triggers, as well as the triggers that blocks the execution of *TRUNCATE* SQL statements. They also create the log schemas if needed.

4.2.2.2 The "configuration table" method

This is the method that was initialy available with E-Maj.

Creating a tables group needs 2 steps:

- register the groups configuration into the *emaj_group_def* table,
- create the tables group by itself.

Populating the *emaj_group_def* table

The content of tables groups to create has to be defined by populating the *emaj.emaj_group_def* table. One row has to be inserted into this table for each application table or sequence to include into a tables group. This *emaj.emaj_group_def* table has the following structure:

Column	**Type**	**Description**
grpdef_group	TEXT	tables group name
grpdef_schema	TEXT	name of the schema containing the application table or sequence
grpdef_tblseq	TEXT	application table or sequence name
grpdef_priority	INT	priority level for the table in E-Maj processing (optional)
grpdef_log_dat_tsp	TEXT	name of the tablespace containing the log table (optional)
grpdef_log_idx_tsp	TEXT	name of the tablespace containing the index of the log table (optional)

The administrator can populate this table by any usual mean: *INSERT* SQL verb, *COPY* SQL verb, *\copy* psql command, graphic tool, etc.

The content of the *emaj_group_def* table is case sensitive. Schema names, table names and sequence names must reflect the way PostgreSQL registers them in its catalogue. These names are mostly in lower case. But if a name is encapsulated by double quotes in SQL statements because it contains any upper case characters or spaces, then it must be registered into the *emaj_group_def* table with the same upper case characters or spaces.

Caution: the *emaj_group_def* table content is modified by the *emaj_import_groups_configuration()* functions.

Creating the tables group

Once the content of the tables group is defined, E-Maj can create the group. To do this, there is only one SQL statement to execute:

```
SELECT emaj.emaj_create_group('<group.name>',<is_rollbackable>);
```

or in an abbreviated form:

```
SELECT emaj.emaj_create_group('<group.name>');
```

The second parameter, of type boolean, indicates whether the group is a "*rollbackable*" (with value true) or an "*audit_only*" (with value false) group. If this second parameter is not supplied, the group is considered "*rollbackable*".

The function returns the number of tables and sequences contained by the group.

For each table of the group, this function creates the associated log table, the log function and trigger, as well as the trigger that blocks the execution of *TRUNCATE* SQL statements.

The function also creates the log schemas if needed.

On the contrary, if specific tablespaces are referenced for any log table or log index, these tablespaces must exist before the function's execution.

The *emaj_create_group()* function also checks the existence of application triggers on any tables of the group. If a trigger exists on a table of the group, a message is returned, suggesting the user to check the impact of this trigger on E-Maj rollbacks.

If a sequence of the group is associated either to a *SERIAL* or *BIGSERIAL* column or to a column created with a *GENERATED AS IDENTITY* clause, and the table that owns this column does not belong to the same tables group, the function also issues a *WARNING* message.

A specific version of the function allows to create an empty tables group, i.e. without any table or sequence at creation time:

```
SELECT emaj.emaj_create_group('<group.name>',<is_rollbackable>,
<is_empty>);
```

The third parameter is false by default. If it is set to true, and the group is referenced in the *emaj_group_def* table, the *emaj_group_def* table's content is ignored. Once created, an empty group can be then populated using the *emaj_alter_group*() function (see §4.4) or dynamic tables groups adjusment functions (see §Erreur : source de la référence non trouvée).

All actions that are chained by the *emaj_create_group()* function are executed on behalf of a unique transaction. As a consequence, if an error occurs during the operation, all tables, functions and triggers already created by the function are cancelled.

By registering the group's composition in an internal table (*emaj_relation)*, the *emaj_create_group()* function freezes its definition for the other E-Maj functions, even if the content of the *emaj_group_def* table is modified later.

A tables group can be altered by the *emaj_alter_group()* function (see §4.4).

4.2.3 Drop a tables group

To drop a tables group previously created by the *emaj_create_group()* function, this group must be already in idle state. If it is not the case, the *emaj_stop_group()* function has to be used first (see § 4.3.6).

Then, just execute the SQL command:

```
SELECT emaj.emaj_drop_group('<group.name>');
```

The function returns the number of tables and sequences contained in the group.

For this tables group, the *emaj_drop_group()* function drops all the objects that have been created by the assignment functions or by the *emaj_create_group()* function: log tables, sequences, functions and triggers.

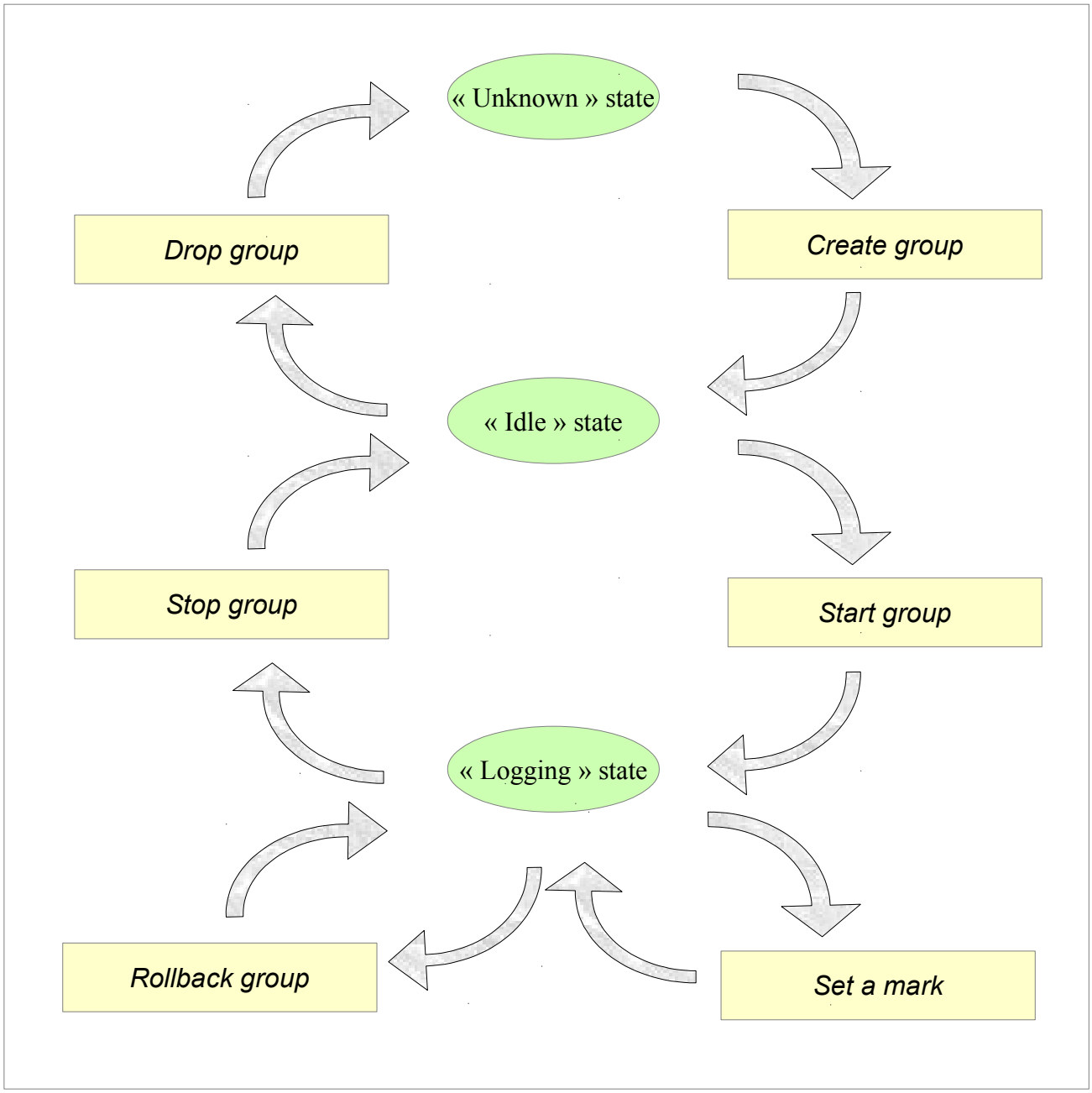
The function also drops all log schemas that are now useless.
The locks set by this operation can lead to *deadlock*. If the *deadlock* processing impacts the execution of the E-Maj function, the error is trapped and the lock operation is repeated, with a maximum of 5 attempts.

4.3 MAIN FUNCTIONS

Before describing each main E-Maj function, it is interesting to have a global view on the typical operations chain.

4.3.1 Operations chain

The possible chaining of operations for a tables group can be materialised by this schema.



4.3.2 Start a tables group

Starting a tables group consists in activating the recording of updates for all tables of the group. To achieve this, the following command must be executed:

```
SELECT emaj.emaj_start_group('<group.name>'[,
'<mark.name>'[,<delete.old.logs?>]]);
```

The group must be first in *IDLE* state.

When a tables group is started, a first mark is created.

If specified, the initial mark name may contain a generic '%' character. Then this character is replaced by the current time, with the pattern “hh.mn.ss.mmmm“,

If the parameter representing the mark is not specified, or is empty or *NULL*, a name is automatically generated: “START_*%”*, where the '%' character represents the current time with a “hh.mn.ss.mmmm” pattern.

The <are.old.logs.to.be.deleted?> parameter is an optional boolean. By default, its value is true, meaning that all log tables of the tables group are purged before the trigger activation. If the value is explicitly set to false, all rows from log tables are kept as is. The old marks are also preserved, even-though they are not usable for a rollback any more, (unlogged updates may have occurred while the tables group was stopped).

The function returns the number of tables and sequences contained by the group.

To be sure that no transaction implying any table of the group is currently running, the *emaj_start_group()* function explicitly sets a *SHARE ROW EXCLUSIVE* lock on each table of the group. If transactions accessing these tables are running, this can lead to *deadlock*. If the *deadlock* processing impacts the execution of the E-Maj function, the error is trapped and the lock operation is repeated, with a maximum of 5 attempts.

The function also performs a purge of the oldest events in the *emaj_hist* technical table (see §5.4).

When a group is started, its state becomes “*LOGGING*”.

Using the *emaj_start_groups()* function, several groups can be started at once:

```
SELECT emaj.emaj_start_groups('<group.names.array>'[,
'<mark.name>'[,<delete.old.logs?>]]);
```

The chapter §4.10.3 explains how to describe the group names array.

4.3.3 Set an intermediate mark

When all tables and sequences of a group are considered as being in a stable state that can be used for a potential rollback, a mark can be set. This is done with the following SQL statement:

```
SELECT emaj.emaj_set_mark_group('<group.name>', '<mark.name>');
```

The tables group must be in *LOGGING* state.

A mark having the same name can not already exist for this tables group.

The mark name may contain a generic '%' character. Then this character is replaced by the current time, with the pattern “hh.mn.ss.mmmm”,

If the parameter representing the mark is not specified or is empty or *NULL*, a name is automatically generated: “*MARK_%”*, where the '%' character represents the current time with a “hh.mn.ss.mmmm” pattern.

The function returns the number of tables and sequences contained in the group.

The *emaj_set_mark_group()* function records the identity of the new mark, with the state of the application sequences belonging to the group, as well as the state of the log sequences associated to each table of the group. The application sequences are processed first, to record their state as earlier as possible after the beginning of the transaction, these sequences not being protected against updates from concurrent transactions by any locking mechanism.

It is possible to set two consecutive marks without any update on any table between these marks.

The *emaj_set_mark_group()* function sets *ROW EXCLUSIVE* locks on each table of the group in order to be sure that no transaction having already performed updates on any table of the group is running. However, this does not guarantee that a transaction having already read one or several tables before the mark set, updates tables after the mark set. In such a case, these updates would be candidate for a potential rollback to this mark.

Using the *emaj_set_mark_groups()* function, a mark can be set on several groups at once:

```
SELECT emaj.emaj_set_mark_groups('<group.names.array>',
'<mark.name>');
```

The chapter §4.10.3 explains how to describe the group names array.

4.3.4 Rollback a tables group

If it is necessary to reset tables and sequences of a group in the state they were when a mark was set, a rollback must be performed. To perform a simple ("*unlogged*") rollback, the following SQL statement can be executed:

```
SELECT * FROM emaj.emaj_rollback_group('<group.name>',
'<mark.name>', <is_alter_group_allowed>);
```

The tables group must be in *LOGGING* state and the supplied mark must be usable for a rollback, i.e. it cannot be logically deleted.

The *'EMAJ_LAST_MARK'* keyword can be used as mark name, meaning the last set mark.

The third parameter is a boolean that indicates whether the rollback operation may target a mark set before an alter group operation (function *emaj_alter_group()* - see §4.4). Depending on their nature, changes performed on tables groups in *LOGGING* state can be automatically cancelled or not. In some cases, this cancellation can be partial. By default, this parameter is set to *FALSE*.

The function returns a set of rows with a severity level set to either "*Notice*" or "*Warning*" values, and a textual message. The function returns a "*Notice*" row indicating the number of tables and sequences that have been **effectively** modified by the rollback operation. Other messages of type "*Warning*" may also be reported when the rollback operation has processed tables group changes.

To be sure that no concurrent transaction updates any table of the group during the rollback operation, the *emaj_rollback_group()* function explicitly sets an *EXCLUSIVE* lock on each table of the group. If transactions updating these tables are running, this can lead to *deadlock*. If the *deadlock* processing impacts the execution of the E-Maj function, the error is trapped and the lock operation is repeated, with a maximum of 5 attempts. But tables of the group remain accessible for read only transactions during the operation.

If tables belonging to the group to rollback have triggers, these triggers are by default temporarily disabled during the operation. Using the *emaj_ignore_app_trigger()* function, it is possible to record triggers as 'not to be automatically disabled at rollback' (see §4.9.4 and §5.9.3).

If a table impacted by the rollback owns a *foreign key* or is referenced by a *foreign key* from another table, then this *foreign key* is taken into account by the rollback operation. If the check of the keys created or modified by the rollback cannot be deferred at the end of the operation (constraint not declared as *DEFERRABLE*), then this *foreign key* is dropped at the beginning of the rollback and recreated at the end.

When the volume of updates to cancel is high and the rollback operation is therefore long, it is possible to monitor the operation using the *emaj_rollback_activity()* function (§4.9.5.2) or the *emajRollbackMonitor.php* client (§4.12).

When the rollback operation is completed, the following are deleted:

- all log tables rows corresponding to the rolled back updates,
- all marks later than the mark referenced in the rollback operation.

The history of executed rollback operations is maintained into the *emaj_rlbk* table. The final state of the operation is accessible from the *rlbk_status* and *rlbk_msg* columns of this *emaj_rlbk* table.

Then, it is possible to continue updating processes, to set other marks, and if needed, to perform another rollback at any mark.

By their nature, the reset of sequences is not “cancellable” in case of abort and rollback of the transaction that executes the *emaj_rollback_group()* function. That is the reason why the processing of application sequences is always performed after the processing of application tables. However, even-though the time needed to rollback a sequence is very short, a problem may occur during this last phase. Rerunning immediately the *emaj_rollback_group()* function would not break database integrity. But any other database access before the second execution may lead to wrong values for some sequences.

Using the *emaj_rollback_groups()* function, several groups can be rolled back at once:

```
SELECT * FROM emaj.emaj_rollback_groups('<group.names.array>',
'<mark.name>', <is_alter_group_allowed>);
```

The chapter §4.10.3 explains how to describe the group names array.

The supplied mark must correspond to the same point in time for all groups. In other words, this mark must have been set by the same *emaj_set_mark_group()* function call.

An old version of these functions had only 2 input parameters and just returned an integer representing the number of effectively processed tables and sequences.

```
SELECT emaj.emaj_rollback_group('<group.name>', '<mark.name>');
```

```
SELECT emaj.emaj_rollback_groups('<group.names.array>',
'<mark.name>');
```

Both functions are deprecated and are subject to be deleted in a future E-Maj version.

4.3.5 Perform a logged rollback of a tables group

Another function executes a "*logged*" rollback. In this case, log triggers on application tables are not disabled during the rollback operation. As a consequence, the updates on application tables are also recorded into log tables, so that it is possible to cancel a rollback. In other words, it is possible to rollback … a rollback.

To execute a "*logged*" rollback, the following SQL statement can be executed:

```
SELECT * FROM emaj.emaj_logged_rollback_group('<group.name>',
'<mark.name>', <is_alter_group_allowed>);
```

The usage rules are the same as with *emaj_rollback_group()* function.

The tables group must be in *LOGGING* state and the supplied mark must be usable for a rollback, i.e. it cannot be logically deleted.

The *'EMAJ_LAST_MARK'* keyword can be used as mark name, meaning the last set mark.

The third parameter is a boolean that indicates whether the rollback operation may target a mark set before an alter group operation (function *emaj_alter_group()* - see §4.4). Depending on their nature, changes performed on tables groups in *LOGGING* state can be automatically cancelled or not. In some cases, this cancellation can be partial. By default, this parameter is set to *FALSE*.

The function returns a set of rows with a severity level set to either "*Notice*" or "*Warning*" values, and a textual message. The function returns a "*Notice*" row indicating the number of tables and sequences that have been **effectively** modified by the rollback operation. Other messages of type "*Warning*" may also be reported when the rollback operation has processed tables group changes.

To be sure that no concurrent transaction updates any table of the group during the rollback operation, the *emaj_logged_rollback_group()* function explicitly sets an *EXCLUSIVE* lock on each table of the group. If transactions updating these tables are running, this can lead to *deadlock*. If the *deadlock* processing impacts the execution of the E-Maj function, the error is trapped and the lock operation is repeated, with a maximum of 5 attempts. But tables of the group remain accessible for read only transactions during the operation.

If tables belonging to the group to rollback have triggers, these triggers are by default temporarily disabled during the operation. Using the *emaj_ignore_app_trigger()* function, it is possible to record triggers as 'not to be automatically disabled at rollback' (see §4.9.4 and §5.9.3).

If a table impacted the rollback owns a *foreign key* or is referenced by a *foreign key* from another table, then this *foreign key* is taken into account by the rollback operation. If the check of the keys created or modified by the rollback cannot be deferred at the end of the operation (constraint not declared as *DEFERRABLE*), then this *foreign key* is dropped at the beginning of the rollback and recreated at the end.

Unlike with *emaj_rollback_group()* function, at the end of the operation, the log tables content as well as the marks following the rollback mark remain.
At the beginning and at the end of the operation, the function automatically sets on the group two marks named:

- 'RLBK_<rollback.mark>_<rollback.time>_START'
- 'RLBK_<rollback.mark>_<rollback.time>_DONE'

where rollback.time represents the start time of the transaction performing the rollback, expressed as “hours.minutes.seconds.milliseconds”.

When the volume of updates to cancel is high and the rollback operation is therefore long, it is possible to monitor the operation using the *emaj_rollback_activity()* function (§4.9.5.2) or the *emajRollbackMonitor.php* client (§4.12).

The history of executed rollback operations is maintained into the *emaj_rlbk* table. The final state of the operation is accessible from the *rlbk_status* and *rlbk_msg* columns of this *emaj_rlbk* table.

Following the rollback operation, it is possible to resume updating the database, to set other marks, and if needed to perform another rollback at any mark, including the mark set at the beginning of the rollback, to cancel it, or even delete an old mark that was set after the mark used for the rollback.

Rollback from different types (*logged*/*unlogged*) may be executed in sequence. For instance, it is possible to chain the following steps:

Set Mark M1
…
Set Mark M2
…
Logged Rollback to M1,
generating RLBK_M1_<time>_STRT,
and RLBK_M1_<time>_DONE
…
Rollback to RLBK_M1_<time>_DONE
(to cancel the updates performed after the first rollback)
…
Rollback to RLBK_M1_<time>_STRT
(to finally cancel the first rollback)

A “*consolidation*” function for “*ogged rollback*“ allows to transform a logged rollback into a simple unlogged rollback (see §4.5.6).

Using the *emaj_logged_rollback_groups()* function, several groups can be rolled back at once:

```
SELECT * FROM emaj.emaj_logged_rollback_groups
('<group.names.array>', '<mark.name>', <is_alter_group_allowed>);
```

The chapter §4.10.3 explains how to describe the group names array.

The supplied mark must correspond to the same point in time for all groups. In other words, this mark must have been set by the same *emaj_set_mark_group()* function call.

An old version of these functions had only 2 input parameters and just returned an integer representing the number of effectively processed tables and sequences.

```
SELECT emaj.emaj_logged_rollback_group('<group.name>',
'<mark.name>');
```

```
SELECT emaj.emaj_logged_rollback_groups('<group.names.array>',
'<mark.name>');
```

Both functions are deprecated and are subject to be deleted in a future E-Maj version.

4.3.6 Stop a tables group

When one wishes to stop the updates recording for tables of a group, it is possible to deactivate the logging mechanism, using the command:

```
SELECT emaj.emaj_stop_group('<group.name>'[, '<mark.name>')];
```

The function returns the number of tables and sequences contained in the group.

If the mark parameter is not specified or is empty or NULL, a mark name is generated: STOP_% where % represents the current time expressed as “hh.mn.ss.mmmm”.

Stopping a tables group simply deactivates log triggers of application tables of the group. The setting of *SHARE ROW EXCLUSIVE* locks can lead to *deadlock*. If the *deadlock* processing impacts the execution of the E-Maj function, the error is trapped and the lock operation is repeated, with a maximum of 5 attempts.

Additionally, the *emaj_stop_group()* function changes the status of all marks set for the group into a *DELETED* state. Then, it is not possible to execute a rollback command any more, even though no updates have been applied on tables between the execution of both *emaj_stop_group()* and *emaj_rollback_group()* functions.

But the content of log tables and E-Maj technical tables can be examined.

When a group is stopped, its state becomes “*IDLE”* again.

Executing the *emaj_stop_group()* function for a tables group already stopped does not generate an error. Only a warning message is returned.

Using the *emaj_stop_groups()* function, several groups can be stopped at once:

```
SELECT emaj.emaj_stop_groups('<group.names.array>'[,
'<mark.name>')];
```

The chapter §4.10.3 explains how to describe the group names array.

4.4 MODIFYING TABLES GROUPS

4.4.1 General information

Several event types may lead to alter a tables group:

- the tables group definition may change, some tables or sequences may have been added or suppressed,
- one of the E-Maj parameters linked to a table (priority, tablespaces,...) may have been modified,
- the structure of one or several application tables of the tables group may have changed, such as an added or dropped column or a column type change,
- a table or sequence may change its name or its schema.

When the modification concerns a tables group in LOGGING state, it may be necessary to temporarily remove the table or sequence from its tables group, with some impacts on potential future E-Maj rollback operations.

In most cases, depending whether the emaj_group_def configuration table is used of not (Cf §4.2.2), the changes can be performed:

- either dynamically, with dedicated functions,
- or by adjusting the *emaj_group_def* table and by executing an *emaj_alter_group()* function.

Here are the possible actions, depending on the choosen method.

Actions	***Dynamic Adjustment***	***emaj_group_def adjustment***
Add a table/sequence to a group	Dedicated functions	emaj_alter_group()
Remove a table/sequence from a group	Dedicated functions	emaj_alter_group()
Move a table/sequence to another group	Dedicated functions	emaj_alter_group()
Change the log data or index tablespace for a table	Dedicated functions	*emaj_alter_group()*
Change the E-Maj priority for a table	Dedicated functions	emaj_alter_group()
Repair a table	Remove from the group + add to the group	emaj_alter_group()
Rename a table	Remove from the group + ALTER TABLE + add	Remove from the group + ALTER TABLE + add
Rename a sequence	Remove from the group + ALTER SEQUENCE + add	Remove from the group + ALTER SEQUENCE + add
Change the schema of a table	Remove from the group	Remove from the group +

	+ ALTER TABLE + add	ALTER TABLE + add
Change the schema of a sequence	Remove from the group + ALTER SEQUENCE + add	Remove from the group + ALTER SEQUENCE + add
Rename a table's column	Remove from the group + ALTER TABLE + add	Remove from the group + ALTER TABLE + add
Change a table's structure	Remove from the group + ALTER TABLE + add	Remove from the group + ALTER TABLE + add
Other forms of ALTER TABLE	No E-Maj impact	
Other forms of ALTER SEQUENCE	No E-Maj impact	

Adjusting the structure of in *LOGGING* state groups may have consequences on E-Maj rollback or SQL script generation. This is explained into details in §4.4.4.

Even if the tables group is in *LOGGING* state, an E-Maj rollback operation targeting a mark set before a group's change do NOT automatically revert this group's change. However the E-Maj administrator can perform by himself the changes that would reset the group to its previous state.

4.4.2 The "Dynamic adjustment" method

Some functions allow to dynamically adjust the tables groups content without modifying *the* emaj_group_def table.

4.4.2.1 Add tables or sequences to a tables group

The functions that assign one or several tables or sequences into a tables group that are used at group's creation time (§4.2.2.1) are also usable during the whole group's life.

When executing these functions, the tables group can be either in *IDLE* or in *LOGGING* state.

When the group is in *LOGGING* state, an exclusive lock is set on all tables of the group.

When the tables group is in *LOGGING* state, a mark is set. Its name is defined by the last parameter of the function. This parameter is optional. If not supplied, the mark name is generated, with a "ASSIGN_" prefix.

4.4.2.2 Remove tables from their tables group

The 3 following functions allow to remove one or several tables from their tables group:

```
SELECT emaj.emaj_remove_table('<schema>','<table>'
      [,'<mark>'] );
```

or

```
SELECT emaj.emaj_remove_tables('<schema>','<tables.array>'
      [,'<mark>'] );
```

or

```
SELECT emaj.emaj_remove_tables('<schema>',
      '<tables.to.include.filter>','<tables.to.exclude.filter>'
      [,'<mark>'] );
```

They are very similar to the tables assignment functions.

When several tables are removed, they do not necessarily belongs to the same group.

When the tables group or groups are in *LOGGING* state and no mark is supplied in parameters, the mark is generated with a 'REMOVE_' prefix.

4.4.2.3 Remove sequences from their tables group

The 3 following functions allow to remove one or several sequences from their tables group:

```
SELECT emaj.emaj_remove_sequence('<schema>','<sequence>'
      [,'<mark>'] );
```

or

```
SELECT emaj.emaj_remove_sequences('<schema>'
      '<sequences.array>' [,'<mark>'] );
```

or

```
SELECT emaj.emaj_remove_sequences('<schema>'
      '<sequences.to.include.filter>','<sequences.to.exclude.filter>'
      [,'<mark>'] );
```

They are very similar to the sequences assignment functions.

When the tables group is in *LOGGING* state and no mark is supplied in parameters, the mark is generated with a 'REMOVE_' prefix,

4.4.2.4 Move tables to another tables group

3 functions allow to move one or several tables to another tables group:

```
SELECT emaj.emaj_move_table('<schema>','<table>',
     '<new.group>' [,'<mark>'] );
```

or

```
SELECT emaj.emaj_move_tables('<schema>','<tables.array>',
     '<new.group>' [,'<mark>'] );
```

or

```
SELECT emaj.emaj_move_tables('<schema>',
     '<tables.to.include.filter>','<tables.to.exclude.filter>',
     '<new.group>' [,'<mark>'] );
```

When serveral tables are moved to another tables group, they do not necessarily belong to the same source group.

When the tables group is in *LOGGING* state and no mark is supplied in parameters, the mark is generated with a 'MOVE_' prefix,

4.4.2.5 Move sequences to another tables group

3 functions allow to move one or several sequences to another tables group:

```
SELECT emaj.emaj_move_sequence('<schema>','<sequence>',
     '<new.group>' [,'<mark>'] );
```

or

```
SELECT emaj.emaj_move_sequences('<schema>'
     '<sequences.array>', '<new.group>' [,'<mark>'] );
```

or

```
SELECT emaj.emaj_move_sequences('<schema>'
     '<sequences.to.include.filter>','<sequences.to.exclude.filter>',
     '<new.group>' [,'<mark>'] );
```

When serveral sequences are moved to another tables group, they do not necessarily belong to the same source group.

When the tables group is in *LOGGING* state and no mark is supplied in parameters, the mark is generated with a 'MOVE_' prefix,

4.4.2.6 Modify tables properties

3 functions allow to modify the properties of one or several tables from a single schema:

```
SELECT emaj.emaj_modify_table('<schema>','<table>',
     '<modified.properties>' [,'<mark>']);
```

or

```
SELECT emaj.emaj_modify_tables('<schema>','<tables.array>',
     '<modified.properties>' [,'<mark>']);
```

or

```
SELECT emaj.emaj_modify_tables('<schema>',
     '<tables.to.include.filter>','<tables.to.exclude.filter>',
     '<modified.properties>' [,'<mark>']);
```

The <modified.properties> parameter is of type JSONB. Its elementary fields are the same as the <properties> parameter of the tables assignment functions (cf §4.2.2.1). But this <modified.properties> parameter only contains … the properties to modify. The not listed properties remain unchanged. It is possible to reset a property to its default value by setting a NULL value (the json null).

The functions return the number of tables that have effectively changed at least one property.

When the tables group is in *LOGGING* state and no mark is supplied in parameters, the mark is generated with a 'MODIFY_' prefix,

4.4.3 The « emaj_group_def update » method

To avoid to be obliged to drop and recreate the tables groups when the *emaj_group_def* table is modified, a function allows to only take into account the configuration changes:

```
SELECT emaj.emaj_alter_group('<group.name>' [,'<mark>']);
```

The function returns the number of tables and sequences that now belong to the tables group.

The *emaj_alter_group()* function also recreates E-Maj objects that may be missing (log tables, functions, …). It also creates and drops the log schemas when needed.

If the group is in *IDLE* state, its log tables become empty.

If the tables group is in *LOGGING* state, the function:

- sets a *ROW EXCLUSIVE* lock on all application tables of the tables group,
- sets a mark, whose name can be supplied as parameter.

If the parameter representing the mark is not specified, or is empty or *NULL*, a name is automatically generated: "ALTER_*%"*, where the '%' character represents the current time with a "hh.mn.ss.mmmm" pattern.

All actions that are chained by the *emaj_alter_group()* function are executed on behalf of a unique transaction. As a consequence, if an error occurs during the operation, the tables group remains in its previous state.

It is possible to update the *emaj_group_def* table, when the tables group is in *LOGGING* state. However it will have no effect until the group is altered (or dropped and re-created).

Using the *emaj_alter_groups()* function, several groups can be modified at once:

```
SELECT emaj.emaj_alter_groups('<group.names.array>' [,'<mark>']);
```

This function especially allows to move a table or a sequence from one tables group to another in a single operation.

The chapter §4.10.3 explains how to describe the group names array.

4.4.4 *Incidence of tables or sequences addition or removal in a group in LOGGING state*

Once a table or a sequence is removed from a tables group, any rollback operation will leave this object unchanged.

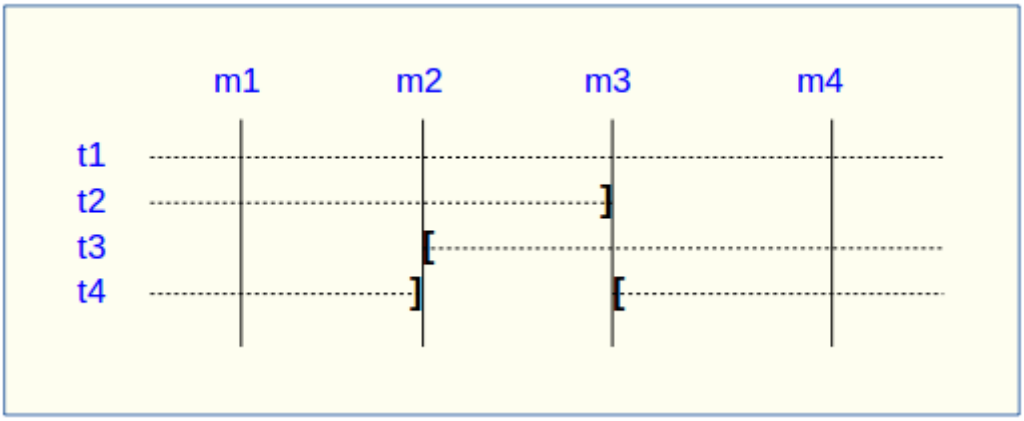
Once unlinked from its tables group, the application table or sequence can be altered or dropped. The historical data linked to the object (logs, marks traces,...) are kept as is so that they can be later examined. However, they remain linked to the tables group that owned the object. To avoid any confusion, log tables are renamed, adding a numeric suffix to its name. These logs and marks traces will only be deleted by a group's reset operation (Cf §4.5.1) or by the deletion of the oldest marks of the group (Cf §4.6.5).

When a table or a sequence is added into a tables group in *LOGGING* state, it is then processed by any further rollback operation. But if the rollback operation targets a mark set before the addition into the group, the table or the sequence is left in its state at the time of the addition into the group and a warning message is issued. Such a table or sequence will not be processed by a SQL

script generation function call if the requested start mark has been set before the addition of the table or sequence into the group

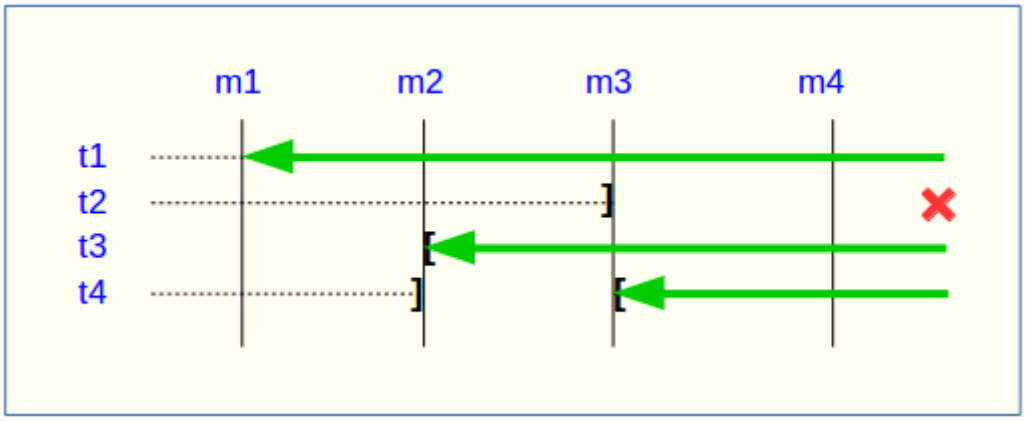
Some graphs help to more easily visualize the consequences of the addition or the removal of a table or a sequence into/from a tables group in *LOGGING* state.

Let's use a tables group containing 4 tables (t1 to t4) and 4 marks set over time (m1 to m4). At m2, t3 has been added to the group while t4 has been removed. At m3, t2 has been removed from the group while t4 has been re-added.



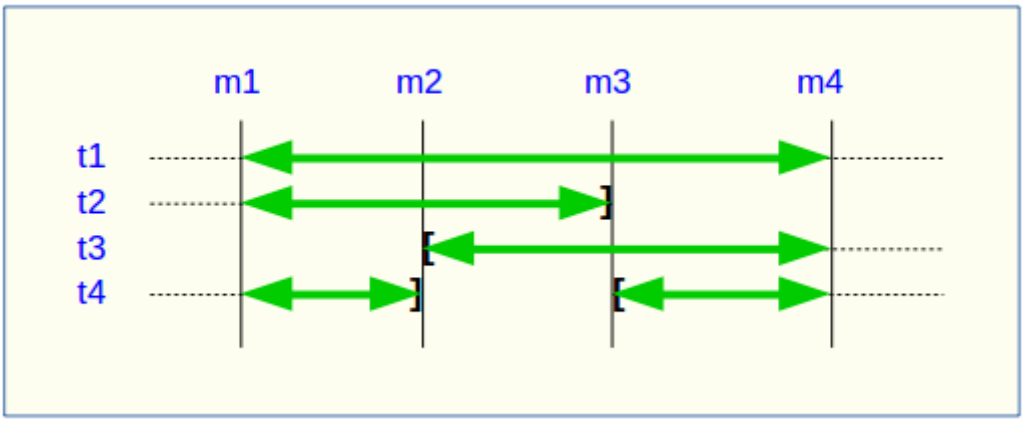
A rollback to the mark m1:

- would process the table t1,
- would **NOT** process the table t2, for lack of log after m3,
- would process the table t3, but only up to m2,
- would process the table t4, but only up to m3, for lack of log between m2 and m3.



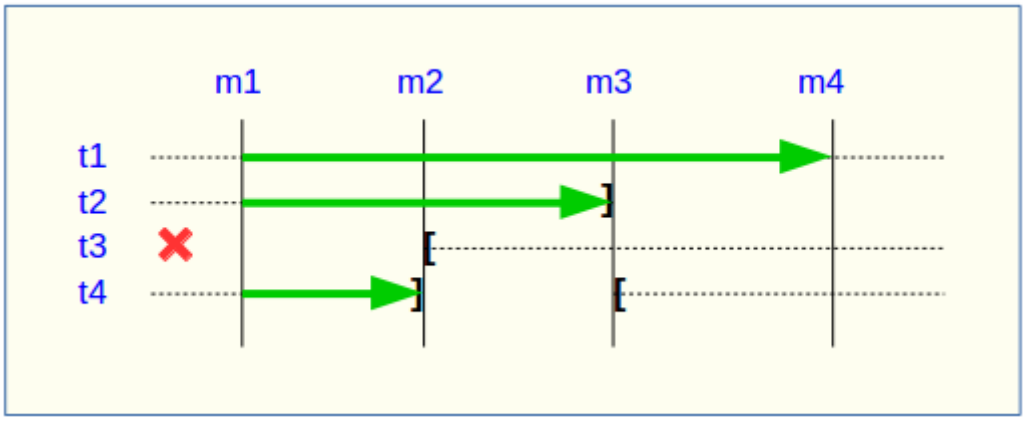
A log statistics report between the marks m1 and m4 would contain:

- 1 row for t1 (m1,m4),
- 1 row for t2 (m1,m3),
- 1 row for t3 (m2,m4),
- 2 rows for t4 (m1,m2) and (m3,m4).



The SQL script generation for the marks interval m1 to m4:

- would process the table t1,
- would process the table t2, but only up the mark m3,
- would **NOT** process the table t3, for lack of log before m2,
- would process the table t4, but only up to the mark m2, for lack of log between m2 and m3.



If the structure of an application table has been inadvertently changed while it belonged to a tables group in LOGGING state, the mark set and rollback operations will be blocked by the E-Maj internal checks. To avoid stopping, altering and then restarting the tables group, it is possible to only remove the concerned table from its group and then to re-add it.

When a table changes its affected group, the impact on the ability to generate a SQL script or to rollback the source and destination tables groups is similar to removing the table from its source group and then adding the table to the destination group.

4.4.5 *Reparing a tables group*

Eventhough the event triggers created with E-Maj limit the risk, some E-Maj components that support an application table (log table, function or trigger) may have been dropped. In such a case, the associated tables group cannot work correctly anymore.

In order to solve the issue without stopping the tables group if it is in *LOGGING* state (and thus loose the benefits of the recorded logs), it is possible to remove the table from its group and then re-add it, by chaining both commands:

```
SELECT emaj.emaj_remove_table('<schema>', '<table>'
      [,'<mark>']);

SELECT emaj.emaj_assign_table('<schema>', '<table>', '<group>'
      [,'properties' [,'<mark>']] );
```

Of course, once the table is removed from its group, the content of the associated logs cannot be used for a potential rollback or script generation anymore.

However, if the log sequence is missing (which should never be the case) and the tables group is in *LOGGING* state, it is necessary to force the group's stop (Cf §4.5.4) before removing and re-assigning the table.

It may also happen that an application table or sequence has been accidentaly dropped. In this case, the table of sequence can be simply a posteriori removed from its group:

- either by executing the *emaj_remove_table()* or *emaj_remove_sequence()* function,
- or by chaining the deletion of the row corresponding to the application table or sequence from the *emaj_group_def* table, and the call of the *emaj_alter_group()* function for the related tables group,.

4.4.6 Combining dynamic tables groups management and configuration with emaj_group_def

Normally, the tables group mangement method has to be choosen first: dynamic management or use of the *emaj_group_def* configuration table. However, it is possible to mix both methods.

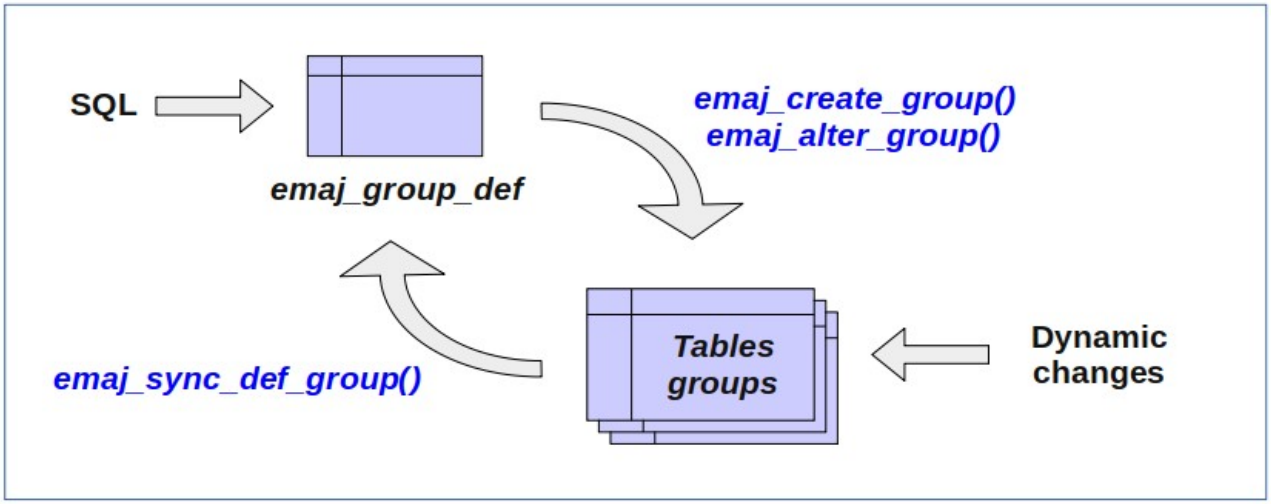
The difficulty comes when, once dynamic changes performed on tables groups content, the *emaj_group_def* table does not reflect the configuration of the tables groups anymore.

To avoid to manually report the configuration changes into the *emaj_group_def* table, with errors risk, the E-Maj administrator can synchronize the *emaj_group_def* table with the actual groups configuration, for a given tables group. To achive this, he can execute:

```
SELECT emaj.emaj_sync_def_group('<group>');
```

The function returns the number of tables and sequences contained in the tables group.

The schema below shows the possible change flows.


SQL
emaj_group_def
emaj_create_group()
emaj_alter_group()
Tables
groups
Dynamic
changes
emaj_sync_def_group()

4.5 OTHER GROUPS MANAGEMENT FUNCTIONS

4.5.1 Reset log tables of a group

In standard use, all log tables of a tables group are purged at *emaj_start_group()* time. But, if needed, it is possible to reset log tables, using the following SQL statement:

```
SELECT emaj.emaj_reset_group('<group.name>');
```

The function returns the number of tables and sequences contained by the group.

Of course, in order to reset log tables, the tables group must be in *IDLE* state.

4.5.2 Comments on groups

In order to set a comment on any group, the following statement can be executed:

```
SELECT emaj.emaj_comment_group('<group.name>', '<comment>');
```

The function doesn't return any data.

To modify an existing comment, just call the function again for the same tables group, with the new comment.

To delete a comment, just call the function, supplying a NULL value as comment.

Comments are stored into the *group_comment* column from the *emaj_group* table, which describes … groups.

4.5.3 Protection of a tables group against rollbacks

It may be useful at certain time to protect tables groups against accidental rollbacks, in particular with production databases. Two functions fit this need.

The *emaj_protect_group()* function set a protection on a tables group.

```
SELECT emaj.emaj_protect_group('<group.name>');
```

The function returns the integer 1 if the tables group was not already protected, or 0 if it was already protected.

Once the group is protected, any logged or unlogged rollback attempt will be refused.

An "*audit_only*" or "*idle*" tables group cannot be protected.

When a tables group is started, it is not protected. When a tables group that is protected against rollbacks is stopped, it looses its protection.

The *emaj_unprotect_group()* function remove an existing protection on a tables group.

```
SELECT emaj.emaj_unprotect_group('<group.name>');
```

The function returns the integer 1 if the tables group was previously protected, or 0 if it was not already protected.

An "*audit_only*" tables group cannot be unprotected.

Once the protection of a tables group is removed, it becomes possible to execute any type of rollback operation on the group.

A protection mechanism at mark level complements this scheme (Cf §4.6.6).

4.5.4 Forced stop of a tables group

It may occur that a corrupted tables group cannot be stopped. This may be the case for instance if an application table belonging to a tables group has been inadvertently dropped while the group was in *LOGGING* state. If usual *emaj_stop_group()* or *emaj_stop_groups()* functions return an error, it is possible to force a group stop using the *emaj_force_stop_group()* function.

```
SELECT emaj.emaj_force_stop_group('<group.name>');
```

The function returns the number of tables and sequences contained by the group.

The *emaj_force_stop_group()* function performs the same actions as the *emaj_stop_group()* function, except that:

- it supports the lack of table or E-Maj trigger to deactivate, generating a "*warning*" message in such a case,
- it does NOT set a stop mark.

Once the function is completed, the tables group is in *IDLE* state. It may then be altered or dropped, using the *emaj_alter_group()* or *emaj_drop_group()* functions.

It is recommended to only use this function if it is really needed.

4.5.5 Forced suppression of a tables group

It may happen that a damaged tables group cannot be stopped. But not being stopped, it cannot be dropped. To be able to drop a tables group while it is still in logging state, a special function exists.

```
SELECT emaj.emaj_force_drop_group('<group.name>');
```

The function returns the number of tables and sequences contained by the group.

This *emaj_force_drop_group()* functions performs the same actions than the *emaj_drop_group()* function, but without checking the state of the group. So, it is recommended to only use this function if it is really needed.

Note: Since the *emaj_force_stop_group()* function has been created, this *emaj_force_drop_group()* function becomes useless. It may be removed in a future version.

4.5.6 Logged rollback consolidation

Following the execution of a "*logged rollback*", and once the rollback operation recording becomes useless, it is possible to "*consolidate*" this *rollback*, meaning to some extent to transform it into "*unlogged rollback*". A the end of the consolidation operation, marks and logs between the rollback target mark and the end rollback mark are deleted. The *emaj_consolidate_rollback_group()* function fits this need.

```
SELECT emaj.emaj_consolidate_rollback_group('<group.name>',
<end.rollback.mark>);
```

The concerned logged rollback operation is identified by the name of the mark generated at the end of the rollback. This mark must always exist, but may have been renamed.
The *'EMAJ_LAST_MARK'* keyword may be used as mark name to reference the last set mark.
The *emaj_get_consolidable_rollbacks()* function may help to identify the rollbacks that may be condolidated (See §4.5.7).

Like rollback functions, the emaj_consolidate_rollback_group() function returns the number of effectively processed tables and sequences.

The tables group may be in *LOGGING* or *IDLE* state.

The rollback target mark must always exist but may have been renamed. However, intermediate marks may have been deleted.

When the consolidation is complete, only the rollback target mark and the end rollback mark are kept.

The disk space of deleted rows will become reusable as soon as these log tables will be "vacuumed".

Of course, once consolidated, a "*logged rollback*" cannot be cancelled (or rolled back) any more, the start rollback mark and the logs covering this rollback being deleted.

The consolidation operation is not sensitive to the protections set on groups or marks, if any.

If a database has enough disk space, it may be interesting to replace a simple u*nlogged rollback* by a *logged rollback* followed by a *consolidation* so that the application tables remain readable during the rollback operation, thanks to the lower locking mode used for *logged rollbacks*.

4.5.7 List of "*consolidable rollbacks*"

The *emaj_get_consolidable_rollbacks()* function help to identify the rollbacks that may be consolidated.

```
SELECT * FROM emaj.emaj_get_consolidable_rollbacks();
```

The function returns a set of rows with the following columns:

- cons_group — rolled back tables group
- cons_target_rlbk_mark_name — rollback target mark name
- cons_target_rlbk_mark_time_id — temporal reference of the target mark (*)
- cons_end_rlbk_mark_name — rollback end mark name
- cons_end_rlbk_mark_time_id — temporal reference of the end mark (*)
- cons_rows — number of intermediate updates
- cons_marks — number of intermediate marks

(*) emaj_time_stamp table identifiers ; this table contains the time stamps of the most important events of the tables groups life.

Using this function, it is easy to consolidate at once all "consolidable" rollbacks for all tables groups in order to recover as much as possible disk space:

```
SELECT emaj.emaj_consolidate_rollback_group(cons_group,
cons_end_rlbk_mark__name) FROM
emaj.emaj_get_consolidable_rollbacks();
```

The *emaj_get_consolidable_rollbacks()* function may be used by *emaj_adm* and *emaj_viewer* roles.

4.5.8 Exporting and importing tables groups configurations

A set of functions allow to export and import tables groups configurations. They may be useful to deploy a standardized tables group configuration on several databases or to upgrade the E-Maj version by a complete extension un-install and re-install (Cf §3.4.2).

4.5.8.1 Export a tables groups configuration

Two versions of the *emaj_export_groups_configuration()* function export a description of one or several tables groups as a JSON structure.

A tables groups configuration can be written to a file with:

```
SELECT emaj_export_groups_configuration('<file.path>',
     <groups.names.array>);
```

The file path must be accessible in write mode by the PostgreSQL instance.

The second parameter is optional. It lists in an array the tables groups names to processed. If the parameter is not supplied or is set to NULL, the configuration of all tables groups is exported.

The function returns the number of exported tables groups.

If the file path is not supplied (i.e. is set to NULL), the function directly returns the JSON structure containing the configuration. This structure looks like this:

```
{
     "_comment": "Generated on database <db> with E-Maj version
<version> at <date_heure>",
     "tables_groups": [
          {
          "group": "...",
          "is_rollbackable": ...,
          "comment": "...",
          "tables": [
               {
               "schema": "...",
               "table": "...",
               "priority": ...,
               "log_data_tablespace": "...",
               "log_index_tablespace": "...",
```

```
                "ignored_triggers": [
                     {
                     "trigger": "...",
                     ...
                      },
                ]
                },
                {
                ...
                }
           ],
           "sequences": [
                {
                "schema": "myschema1",
                "sequence": "mytbl1",
                },
                {
                ...
                }
           ],
           },
           ...
      ]
}
```

4.5.8.2 Import a tables groups configuration

Two versions of the *emaj_import_groups_configuration()* function import a description of tables groups as a JSON structure.

A tables groups configuration can be read from a file with:

```
SELECT emaj_import_groups_configuration('<file.path>',
     <groups.names.array>, <alter_started_groups>);
```

The file must be accessible by the PostgreSQL instance.

The file must contain a JSON structure with an attribute named "*tables-groups*" of type array, and containing sub-structures describing each tables group, as described in the previous chapter about tables groups configuration exports.

The function can directly import a file generated by the *emaj_export_groups_configuration()* function.

The second parameter is of type array and is optional. It contains the list of the tables groups to import. By default, all tables groups described in the file are imported.

If a tables group to import does not exist, it is created and its tables and sequences are assigned into it.

If a tables group to import already exists, its configuration is adjusted to reflect the target configuration : some tables and sequences may be added or removed, and some attributes may be modified. When the tables group is in *LOGGING* state, its configuration adjustment is only possible if the third parameter is explicitly set to *TRUE*.

The function returns the number of imported tables groups.

Note that importing a tables group overrides the previous group's configuration, if any, in the *emaj_group_def* technical table.

In a variation of the function, the first input parameter directly contains the JSON description of the groups to load.

```
SELECT emaj_import_groups_configuration(
      '<JSON.structure>, <groups.names.array>,
      <alter_started_groups>);
```

4.6 MARKS MANAGEMENT FUNCTIONS

4.6.1 Comments on marks

In order to set a comment on any mark, the following statement can be executed:

```
SELECT emaj.emaj_comment_mark_group('<group.name>', '<mark>',
'<comment>');
```

The keyword *'EMAJ_LAST_MARK'* can be used as mark name. It then represents the last set mark.

The function doesn't return any data.

To modify an existing comment, just call the function again for the same tables group and the same mark, with the new comment.

To delete a comment, just call the function, supplying a NULL value as comment.

Comments are stored into the *mark_comment* column from the *emaj_mark* table, which describes … marks.

Comments are mostly interesting when using web clients (See §6). Indeed, they systematically display the comments in the groups marks list.

4.6.2 Search a mark

The *emaj_get_previous_mark_group*() function provides the name of the latest mark before either a given date and time or another mark for a tables group.

```
SELECT emaj.emaj_get_previous_mark_group('<group.name>',
'<date.time>');
```

or

```
SELECT emaj.emaj_get_previous_mark_group('<group.name>',
'<mark>');
```

In the first format, the date and time must be expressed as a *TIMESTAMPTZ* datum, for instance the literal '2011/06/30 12:00:00 +02'.

In the second format, the keyword *'EMAJ_LAST_MARK'* can be used as mark name. It then represents the last set mark.

If the supplied time strictly equals the time of an existing mark, the returned mark would be the preceding one.

4.6.3 Rename a mark

A mark that has been previously set by one of both *emaj_create_group()* or *emaj_set_mark_group()* functions can be renamed, using the SQL statement:

```
SELECT emaj.emaj_rename_mark_group('<group.name>',
'<mark.name>', '<new.mark.name>');
```

The keyword *'EMAJ_LAST_MARK'* can be used as mark name. It then represents the last set mark.

The function does not return any data.

A mark having the same name as the requested new name should not already exist for the tables group.

4.6.4 Delete a mark

A mark can also be deleted, using the SQL statement:

```
SELECT emaj.emaj_delete_mark_group('<group.name>',
'<mark.name>');
```

The keyword *'EMAJ_LAST_MARK'* can be used as mark name. It then represents the last set mark.

The function returns 1, corresponding to the number of effectively deleted marks.

As at least one mark must remain after the function has been performed, a mark deletion is only possible when there are at least two marks for the concerned tables group.

If the deleted mark is the first mark of the tables group, the useless rows of log tables are deleted.

If a table has been detached from a tables group (Cf. Erreur : source de la référence non trouvée), and the deleted mark corresponds to the last known mark for this table, the logs for the period between this mark and the preceeding one are deleted,

4.6.5 Delete oldest marks

To easily delete in a single operation all marks prior a given mark, the following statement can be executed:

```
SELECT emaj.emaj_delete_before_mark_group('<group.name>',
'<mark.name>');
```

The keyword *'EMAJ_LAST_MARK'* can be used as mark name. It then represents the last set mark.

The function deletes all marks prior the supplied mark, this mark becoming the new first available mark. It also suppresses from log tables all rows related to the deleted period of time.

The function returns the number of deleted marks.

The function also performs a purge of the oldest events in the *emaj_hist* technical table (see §5.4).

With this function, it is quite easy to use E-Maj for a long period of time, without stopping and restarting groups, while limiting the disk space needed for accumulated log records.

However, as the log rows deletion cannot use any *TRUNCATE* command (unlike with the *emaj_start_group()* or *emaj_reset_group()* functions), using *emaj_delete_before_mark_group()* function may take a longer time than simply stopping and restarting the group. In return, no lock is set on the tables of the group. Its execution may continue while other processes update the application tables. Nothing but other E-Maj operations on the same tables group, like setting a new mark, would wait until the end of an *emaj_delete_before_mark_group()* function execution.

When associated, the functions *emaj_delete_before_mark_group()* and *emaj_get_previous_mark_group()* allow to delete marks older than a retention delay. For example, to suppress all marks (and the associated log rows) set since more than 24 hours, the following statement can be executed:

```
SELECT emaj.emaj_delete_before_mark_group('<group>',
emaj.emaj_get_previous_mark_group('<group>', current_timestamp - '1
DAY'::INTERVAL));
```

4.6.6 Protection of a mark against rollbacks

To complement the mechanism of tables group protection against accidental rollbacks (see §4.5.3), it is possible to set protection at mark level. Two functions fit this need.

The *emaj_protect_mark_group()* function sets a protection on a mark for a tables group.

```
SELECT
emaj.emaj_protect_mark_group('<groupe.name>','<mark.name>');
```

The function returns the integer 1 if the mark was not previously protected, or 0 if it was already protected.

Once a mark is protected, any logged or unlogged rollback attempt is refused if it reset the tables group in a state prior this protected mark.

A mark of an “audit-only” or an “idle” tables group cannot be protected.

When a mark is set, it is not protected. Protected marks of a tables group automatically loose their protection when the group is stopped. Warning: deleting a protected mark also deletes its protection. This protection is not moved on an adjacent mark.

The *emaj_unprotect_mark_group()* function remove an existing protection on a tables group mark.

```
SELECT
emaj.emaj_unprotect_mark_group('<group.name>','<mark.name>');
```

The function returns the integer 1 if the mark was previously protected, or 0 if it was not yet protected.

A mark of an “audit-only” tables group cannot be unprotected.

Once a mark protection is removed, it becomes possible to execute any type of rollback on a previous mark.

4.7 STATISTICS FUNCTIONS

There are two functions that return statistics on log tables content:

- *emaj_log_stat_group()* quickly delivers, for each table of a group, the number of updates that have been recorded in the related log tables, either between 2 marks or since a particular mark,
- *emaj_detailed_log_stat_group()* provides more detailed information than *emaj_log_stat_group()*, the number of updates been reported per table, SQL type (INSERT/UPDATE/DELETE) and connection role.

Two other E-Maj functions, *emaj_estimate_rollback_group()* and *emaj_estimate_rollback_groups()* , provide an estimate of how long a rollback for one or several groups to a given mark may last.

These functions can be used by *emaj_adm* and *emaj_viewer* E-Maj roles.

4.7.1 Global statistics about logs

Full global statistics about logs content are available with this SQL statement:

```
SELECT * FROM emaj.emaj_log_stat_group('<group.name>',
'<start.mark>', '<end.mark>');
```

The function returns a set of rows, whose type is named *emaj.emaj_log_stat_type*, and contains the following columns:

- *stat_group* : tables group name (type TEXT),
- *stat_schema* : schema name (type TEXT),
- *stat_table* : table name (type TEXT),
- *stat_first_mark* : mark name of the period start (type TEXT),
- *stat_first_mark_datetime* : mark timestamp of the period start (type TIMESTAMPTZ),
- *stat_last_mark* : mark name of the period end (type TEXT),
- *stat_last_mark_datetime* : mark timestamp of the period end (type TIMESTAMPTZ),
- *stat_rows* : number of updates recorded into the related log table (type BIGINT)

A NULL value or an empty string ('') supplied as start mark represents the oldest mark.

A NULL value supplied as end mark represents the current situation.

The keyword *'EMAJ_LAST_MARK'* can be used as mark name. It then represents the last set mark.

The function returns one row per table, even if there is no logged update for this table. In this case, *stat_rows* columns value is 0.

Most of the time, the *stat_first_mark, stat_first_mark_datetime, stat_last_mark* and *stat_last_mark_datetime* columns reference the start and end marks of the requested period. But they can contain other values when a table has been added or removed from the tables group during the requested time interval.

It is possible to easily execute more precise requests on these statistics. For instance, it is possible to get the number of database updates by application schema, with a statement like:

```
postgres=# SELECT stat_schema, sum(stat_rows)
FROM emaj.emaj_log_stat_group('myAppl1', NULL, NULL)
GROUP BY stat_schema;
 stat_schema | sum
-------------+-----
 myschema    |  41
(1 row)
```

There is no need for log table scans to get these statistics. For this reason, they are delivered quickly.

But returned values may be approximative (in fact over-estimated). This occurs in particular when transactions executed between both requested marks have performed table updates before being cancelled.

Using the *emaj_log_stat_groups()* function, log statistics can be obtained for several groups at once:

```
SELECT emaj.emaj_log_stat_groups('<group.names.array>',
'<start.mark>', '<end.mark>');
```

The chapter §4.10.3 explains how to describe the group names array.

4.7.2 Detailed statistics about logs

Scanning log tables brings a more detailed information, at a higher response time cost. So can we get fully detailed statistics with the following SQL statement:

```
SELECT * FROM emaj.emaj_detailed_log_stat_group('<group.name>',
'<start.mark>', '<end.mark>');
```

The function returns a set of rows, whose type is named *emaj.emaj_detailed_log_stat_type*, and contains the following columns:

- *stat_group* : tables group name (type TEXT),
- *stat_schema* : schema name (type TEXT),
- *stat_table* : table name (type TEXT),
- *stat_first_mark* : mark name of the period start (type TEXT),
- *stat_first_mark_datetime* : mark timestamp of the period start (type TIMESTAMPTZ),
- *stat_last_mark* : mark name of the period end (type TEXT),
- *stat_last_mark_datetime* : mark timestamp of the period end (type TIMESTAMPTZ),
- *stat_role* : connection role (type VARCHAR(32)),
- *stat_verb* : type of the SQL verb that has performed the update (type VARCHAR(6), with values: *INSERT* / *UPDATE* / *DELETE*),
- *stat_rows* : number of updates recorded into the related log table (type BIGINT)

A NULL value or an empty string ('') supplied as start mark represents the oldest mark.

A NULL value supplied as end mark represents the current situation.

The keyword *'EMAJ_LAST_MARK'* can be used as mark name. It then represents the last set mark.

Unlike *emaj_log_stat_group()*, the *emaj_detailed_log_stat_group()* function doesn't return any rows for tables having no logged updates inside the requested marks range. So *stat_rows* column never contains 0.

Most of the time, the *stat_first_mark, stat_first_mark_datetime, stat_last_mark* and *stat_last_mark_datetime* columns reference the start and end marks of the requested period. But they can contain other values when a table has been added or removed from the tables group during the requested time interval.

Using the *emaj_detailed_log_stat_groups()* function, detailed log statistics can be obtained for several groups at once:

```
SELECT emaj.emaj_detailed_log_stat_groups('<group.names.array>',
'<start.mark>', '<end.mark>');
```

The chapter §4.10.3 explains how to describe the group names array.

4.7.3 Estimate the rollback duration

The *emaj_estimate_rollback_group()* function returns an idea of the time needed to rollback a tables group to a given mark. It can be called with a statement like:

```
SELECT emaj.emaj_estimate_rollback_group('<group.name>',
'<mark.name>', <is.logged>);
```

The keyword *'EMAJ_LAST_MARK'* can be used as mark name. It then represents the last set mark.

The third parameter indicates whether the E-Maj rollback to simulate is a logged rollback or not.

The function returns an *INTERVAL* value.

The tables group must be in *LOGGING* state and the supplied mark must be usable for a rollback, i.e. it cannot be logically deleted.

This duration estimate is approximative. It takes into account:

- the number of updates in log tables to process, as returned by the *emaj_log_stat_group()* function,
- recorded duration of already performed rollbacks for the same tables,
- 6 generic parameters (see §5.1) that are used as default values when no statistics have been already recorded for the tables to process.

The precision of the result cannot be high. The first reason is that, INSERT, UPDATE and DELETE having not the same cost, the part of each SQL type may vary. The second reason is that the load of the server at rollback time can be very different from one run to another. However, if there is a time constraint, the order of magnitude delivered by the function can be helpful to determine of the rollback operation can be performed in the available time interval.

If no statistics on previous rollbacks are available and if the results quality is poor, it is possible to adjust parameters listed in chapter 5.1. It is also possible to manually change the *emaj.emaj_rlbk_stat* table's content that keep a trace of the previous rollback durations, for instance by deleting rows corresponding to rollback operations performed in unusual load conditions.

Using the *emaj_estimate_rollback_groups()* function, it is possible to estimate the duration of a rollback operation on several groups:

```
SELECT emaj.emaj_estimate_rollback_groups('<group.names.array>',
'<mark.name>', <is.logged>);
```

The chapter §4.10.3 explains how to describe the group names array.

4.8 DATA EXTRACTION FUNCTIONS

Three functions extract data from E-Maj infrastructure and store them into external files.

4.8.1 Snap tables of a group

It may be useful to take images of all tables and sequences belonging to a group to be able to analyse their content or compare them. It is possible to dump to files all tables and sequences of a group with:

```
SELECT emaj.emaj_snap_group('<group.name>', '<storage.directory>',
'<COPY.options>');
```

The directory/folder name must be supplied as an absolute pathname and must have been previously created. This directory/folder must have the appropriate permission so that the PostgreSQL instance can write in it.

The third parameter defines the output files format. It is a character string that matches the precise syntax available for the COPY TO SQL statement.

The function returns the number of tables and sequences contained by the group.

This *emaj_snap_group()* function generates one file per table and sequence belonging to the supplied tables group. These files are stored in the directory or folder corresponding to the second parameter.

New files will overwrite existing files of the same name.

Created files are named with the following pattern:

<schema.name>_<table/sequence.name>.snap

Some unconvenient in file name characters, namely spaces, “/”, “\”, “$”, “>”, “<”, and “*” are replaced by “_”.

Each file corresponding to a sequence has only one row, containing all characteristics of the sequence.

Files corresponding to tables contain one record per row, in the format corresponding to the supplied parameter. These records are sorted in ascending order of the primary key.

At the end of the operation, a file named *_INFO* is created in this same directory/folder. It contains a message including the tables group name and the date and time of the snap operation.

It is not necessary that the tables group be in idle state to snap tables.

As this function may generate large or very large files (of course depending on tables sizes), it is user's responsibility to provide a sufficient disk space.

Thanks to this function, a simple test of the E-Maj behaviour could chain:

- emaj_create_group(),
- emaj_start_group(),
- emaj_snap_group(<directory_1>),
- updates of application tables,
- emaj_rollback_group(),
- emaj_snap_group(<directory_2>),
- comparison of both directories content, using a diff command for instance.

4.8.2 Snap log tables of a group

It is also possible to record a full or a partial image of all log tables related to a group. This provides a way to archive updates performed by one or more previous operations. It is possible to dump on files all tables and sequences of a group with:

```
SELECT emaj.emaj_snap_log_group('<group.name>', '<start.mark>',
'<end.mark>', '<storage.directory>', '<COPY.options>');
```

A *NULL* value or an empty string may be used as start mark, representing the first known mark.
A *NULL* value or an empty string may be used as end mark, representing the current situation.

The keyword *'EMAJ_LAST_MARK'* can be used as mark name, representing the last set mark.

The directory/folder name must be supplied as an absolute pathname and must have been previously created. This directory/folder must have the appropriate permission so that the PostgreSQL instance can write in it.

The fifth parameter defines the output files format. It is a character string that matches the precise syntax available for the COPY TO SQL statement.

The function returns the number of generated files.

This *emaj_snap_log_group()* function generates one file per log table, containing the part of this table that corresponds to the updates performed between both supplied marks. Created files name has the following pattern:

<log.table.name>.snap

So most of the time, they look like:

<schema.name>_<table/sequence.name>_log.snap

The function also generates two files, containing the application sequences state at the time of the respective supplied marks, and named:

<group.name>_sequences_at_<mark.name>

All these files are stored in the directory or folder corresponding to the fourth parameter. New files will overwrite existing files of the same name.

Some unconvenient in file name characters, namely spaces, “/”, “\”, “$”, “>”, “<”, and “*” are replaced by “_”.

At the end of the operation, a file named *_INFO* is created in this same directory/folder. It contains a message including the table's group name, the mark's name that defined the mark range and the date and time of the snap operation.

It is not necessary that the tables group be in idle state to snap log tables. If no end mark has been supplied, the log tables snap is bounded by a pseudo mark set at the function start. This ensures that, if the group is in logging state, output files will not contain updates recorded after the function start.

As this function may generate large or very large files (of course depending on tables sizes), it is user's responsibility to provide a sufficient disk space.

The structure of log tables is described in §5.2.1.

4.8.3 SQL script generation to replay logged updates

Log tables contain all needed information to replay updates. Therefore, it is possible to generate SQL statements corresponding to all updates that occurred between two marks or between a mark and the current situation. This is the purpose of the *emaj_gen_sql_group()* function.

So these updates can be replayed after the corresponding tables have been restored in their state at the initial mark, without being obliged to rerun application programs.

To generate this SQL script, just execute the following statement:

```
SELECT emaj.emaj_gen_sql_group('<group.name>', '<start.mark>',
       '<end.mark>', '<file>' [, <tables/sequences.array>);
```

A *NULL* value or an empty string may be used as start mark, representing the first known mark.
A *NULL* value or an empty string may be used as end mark, representing the current situation.

The keyword *'EMAJ_LAST_MARK'* can be used as mark name, representing the last set mark.

If supplied, the output file name must be an absolute pathname. It must have the appropriate permission so that the PostgreSQL instance can write to it. If the file already exists, its content is overwritten.

The output file name may be set to NULL. In this case, the SQL script is prepared in a temporary table that can then be accessed through a temporary view, *emaj_sql_script*. Using *psql*, the script can be exported with both commands:

```
SELECT emaj.emaj_gen_sql_group('<group.name>', '<start.mark>',
       '<end.mark>', NULL [, <tables/sequences.array>);
\copy (SELECT * FROM emaj_sql_script) TO 'file'
```

This method allows to generate a script in a file located outside the file systems accessible by the PostgreSQL instance.

The last parameter of the *emaj_gen_sql_group()* function is optional. It allows filtering of the tables and sequences to process. If the parameter is omitted or has a *NULL* value, all tables and sequences of the tables group are processed. If specified, the parameter must be expressed as a non empty array of text elements, each of them representing a schema qualified table or sequence name. Both syntaxes can be used:
ARRAY['sch1.tbl1','sch1.tbl2']
or
'{ "sch1.tbl1" , "sch1.tbl2" }'

The function returns the number of generated statements (not including comments and transaction management statements).

The tables group may be in *IDLE* or in *LOGGING* state while the function is called.

In order to generate the script, all tables must have an explicit *PRIMARY KEY*.

If a tables and sequences list is specified to limit the *emaj_gen_sql_group()* function's work, it is the user's responsibility to take into account the possible presence of foreign keys, in order to let the function produce a viable SQL script.

All statements, *INSERT*, *UPDATE*, *DELETE* and *TRUNCATE* (for *audit_only* tables groups), are generated in the order of their initial execution.

The statements are inserted into a single transaction. They are surrounded by a *BEGIN TRANSACTION;* statement and a *COMMIT;* statement. An initial comment specifies the characteristics of the script generation: generation date and time, related tables group and used marks.

TRUNCATE statements recorded for *audit_only* tables groups are also included into the script.

At the end of the script, sequences belonging to the tables group are set to their final state.

Then, the generated file may be executed as is by *psql* tool, using a connection role that has enough rights on accessed tables and sequences.

The used technology may result to doubled backslashes in the output file. These doubled characters must be suppressed before executing the script, for instance, in Unix/Linux environment, using a command like:
sed 's/\\\\/\\/g' <file.name> | psql ...

As the function can generate a large, or even very large, file (depending on the log volume), it is the user's responsibility to provide a sufficient disk space.

It is also the user's responsibility to deactivate application triggers, if any exist, before executing the generated script.

Using the *emaj_gen_sql_groups()* function, it is possible to generate a sql script related to several groups:

```
SELECT emaj.emaj_gen_sql_groups('<group.names.array>',
'<start.mark>', '<end.mark>', '<file>' [, <tables/sequences.array>);
```

The chapter §4.10.3 explains how to describe the group names array.

4.9 OTHER FUNCTIONS

4.9.1 Check the consistency of the E-Maj environment

A function is also available to check the consistency of the E-Maj environment. It consists in checking the integrity of all E-Maj schemas and all created tables groups. This function can be called with the following SQL statement:

```
SELECT * FROM emaj.emaj_verify_all();
```

For each E-Maj schema (*emaj* and each log schema) the function verifies that:

- all tables, functions, sequences and types contained in the schema are either objects of the extension, or linked to created tables groups,
- they don't contain any view, foreign table, domain, conversion, operator or operator class.

Then, for each created tables group, the function performs the same checks as those performed when a group is started, a mark is set, or a rollback is executed (see §5.3.1).

The function returns a set of rows describing the detected discrepancies. If no error is detected, the function returns a single row containing the following messages:

'No error detected'

The function also returns warnings when:

- a sequence linked to a column belongs to a tables group, but the associated table does not belong to the same tables group,
- a table of a tables group is linked to another table by a foreign key, but the associated table does not belong to the same tables group.

The *emaj_verify_all()* function can be executed by any role belonging to *emaj_adm* or *emaj_viewer* roles.

If errors are detected, for instance after an application table referenced in a tables group has been dropped, appropriate measures must be taken. Typically, the potential orphan log tables or functions must be manually dropped.

4.9.2 Exporting and importing parameters configurations

Two functions sets allow to respectively export and import parameters configurations. They can be useful to deploy a standardized parameters set on several databases, or during E-Maj version upgrades by a full extension uninstallation and reinstallation (Cf §3.4.2).

4.9.2.1 Exporting a parameters configuration

Two versions of the *emaj_export_parameters_configuration()* function export all the parameters registered in the *emaj_param* table in a JSON structure, except the parameter of key "emaj_version", which is linked to the emaj extension itself and is not really a configuration parameter.

The parameters data can be written to a file with:

```
SELECT emaj_export_parameters_configuration('<file.path>');
```

The file path must be accessible in write mode by the PostgreSQL instance.

The function returns the number of exported parameters.

If the file path is not supplied, the function directly returns the JSON structure containing the parameters value. This structure looks like this:

```
{
     "_comment": "E-Maj parameters, generated from the database
<db> with E-Maj version <version> at <date_heure>",
     "parameters": [
       {
          "key": "...",
          "value": "..."
       },
       {
          ...
       }
     ]
}
```

4.9.2.2 Importing a parameters configuration

Two versions of the *emaj_import_parameters_configuration()* function import parameters from a JSON structure into the *emaj_param* table.

A file containing parameters to load can be read with:

```
SELECT emaj_import_parameters_configuration('<file.path>',
     <delete.current.configuration>);
```

The file path must be accessible by the PostgreSQL instance.

The file must contain a JSON structure having an attribute named "parameters", of array type, and containing sub-structures with the attributes "key" and "value".

```
{"parameters": [
  {
     "key": "...",
     "value": "..."
  },
  {
     ...
  }
]}
```

If a paramater has no *"value"* attribute or if this attribute is set to *NULL*, the parameter is not inserted into the *emaj_param* table, and is deleted if it already exists in the table. So the parameter's default value will be used by the *emaj* extension.

The function can directly load a file generated by the *emaj_export_parameters_configuration()* function.

If present, the paramètre of key "*emaj_version*" is not processed.

The second parameter, boolean, is optional. It tells whether the current parameter configuration has to be deleted before the load. It is *FALSE* by default, meaning that the keys currenly stored into the *emaj_param* table, but not listed in the JSON structure are kept (differential mode load). If the value of this second parameter is set to *TRUE*, the function performs a full replacement of the parameters configuration (full mode load).

The function returns the number of imported parameters.

As an alternative, the first input parameter of the function directly contains the JSON structure of the parameters to load.

```
SELECT emaj_import_parameters_configuration(
     '<JSON.structure>', <delete.current.configuration>);
```

4.9.3 Getting the current log table linked to an application table

The *emaj_get_current_log_table()* function allows to get the schema and table names of the current log table linked to a given application table.

```
SELECT log_schema, log_table FROM
     emaj_get_current_log_table(<schema>, <table>);
```

The function always returns 1 row. If the application table does not currently belong to any tables group, the *log_schema* and *log_table* columns are set to NULL.

The *emaj_get_current_log_table()* function can be used by *emaj_adm* and *emaj_viewer* E-Maj roles.

It is possible to build a statement accessing a log table. For instance:

```
SELECT 'select count(*) from '
      || quote_ident(log_schema) || '.' || quote_ident(log_table)
      FROM emaj.emaj_get_current_log_table('myschema','mytable');
```

4.9.4 Not disabling application triggers at E-Maj rollback time

Application triggers are automatically disabled during E-Maj rollback operations. Under some circumstances, it may be desirable to keep them enabled (See §5.9.3 for more details). The *emaj_ignore_app_trigger()* function achieves this. It allows to add or remove triggers into/from a list of triggers that do not need to be disabled during rollback operations.

```
SELECT emaj.emaj_ignore_app_trigger(<action>, <schema.name>,
<table.name>, <trigger.name>);
```

The <action> parameter accepts 2 values: 'ADD' to add a trigger to the list or 'REMOVE' to delete a trigger from the list.

The trigger is identified by the 3 components: schema name, table name and trigger name.

The trigger name may contain '%' and '_' wildcard characters. These characters have the same meaning as in the *LIKE* clause of the SQL language. Thus several triggers of a single table can be processed by a unique function call.

The function returns the number of triggers effectively added or removed.

The function does not process E-Maj triggers (log triggers or triggers protecting against *TRUNCATE*).

The triggers referenced as "not to be automatically disabled during E-Maj rollbacks" are registered into the *emaj.emaj_ignored_app_trigger* table. This table contains 3 columns:

- trg_schema schema of the table holding the trigger
- trg_table table holding the trigger
- trg_name trigger name

In order to know the list of registered triggers, just display the table's content.

4.9.5 Monitoring rollback operations

When the volume of recorded updates to cancel leads to a long rollback, it may be interesting to monitor the operation to appreciate how it progresses. A function, named *emaj_rollback_activity()*, and a client, *emajRollbackMonitor.php* (see §4.12), fit this need.

4.9.5.1 Prerequisite

To allow E-Maj administrators to monitor the progress of a rollback operation, the activated functions update several technical tables as the process progresses. To ensure that these updates are visible while the transaction managing the rollback is in progress, they are performed through a *dblink* connection.

As a result, monitoring rollback operations requires the installation of the *dblink* extension (§3.3.2) as well as the insertion of a connection identifier usable by *dblink* into the *emaj_param* table.

Recording the connection identifier can be performed with a statement like:

```
INSERT INTO emaj.emaj_param (param_key, param_value_text)
VALUES ('dblink_user_password','user=<user> password=<password>');
```

The declared connection role must have been granted the *emaj_adm* rights (or be a superuser).

Lastly, the main transaction managing the rollback operation must be in a “read committed” concurrency mode (the default value).

4.9.5.2 Monitoring function

The *emaj_rollback_activity()* function allows to see the progress of rollback operations.

Invoke it with the following statement:

```
SELECT * FROM emaj.emaj_rollback_activity();
```

The function does not require any input parameter.

It returns a set of rows of type *emaj.emaj_rollback_activity_type*. Each row represents an in progress rollback operation, with the following columns:

- *rlbk_id* rollback identifier
- *rlbk_groups* tables groups array associated to the rollback
- *rlbk_mark* mark to rollback to
- *rlbk_mark_datetime* date and time when the mark to rollback to has been set
- *rlbk_is_logged* boolean taking the “true” value for logged rollbacks
- *rlbk_nb_session* number of parallel sessions
- *rlbk_nb_table* number of tables contained in the processed tables groups

- *rlbk_nb_sequence* number of sequences contained in the processed tables groups
- *rlbk_eff_nb_table* number of tables having updates to cancel
- *rlbk_status* rollback operation state
- *rlbk_start_datetime* rollback operation start timestamp
- *rlbk_elapse* elapse time spent since the rollback operation start
- *rlbk_remaining* estimated remaining duration
- *rlbk_completion_pct* estimated percentage of the completed work

An in progress rollback operation is in one of the following state:
- *PLANNING* the operation is in its initial planning phase,
- *LOCKING* the operation is setting locks,
- *EXECUTING* the operation is currently executing one of the planned steps.

If the functions executing rollback operations cannot use *dblink* connections (extension not installed, missing or incorrect connection parameters,...), the *emaj_rollback_activity()* does not return any rows.

The remaining duration estimate is approximate. Its precision is similar to the precision of the emaj_estimate_rollback_group() function (§4.7.3).

4.9.6 Updating rollback operations state

The *emaj_rlbk* technical table and its derived tables contain the history of E-Maj rollback operations.

When rollback functions cannot use *dblink* connections (see the conditions at §4.9.5.1), all updates of these technical tables are all performed inside a single transaction. Therefore:
- any rollback operation that has not been completed is invisible in these technical tables,
- any rollback operation that has been validated is visible in these technical tables with a "*COMMITTED*" state.

When rollback functions can use *dblink* connections, all updates of *emaj_rlbk* and its related tables are performed in autonomous transactions. In this working mode, rollback functions leave the operation in a "*COMPLETED*" state when finished. A dedicated internal function is in charge of transforming the "*COMPLETED*" operations either into a "*COMMITTED*" state or into an "*ABORTED*" state, depending on how the main rollback transaction has ended. This function is automatically called when a new mark is set and when the rollback monitoring function is used.

If the E-Maj administrator wishes to check the status of recently executed rollback operations, he can use the *emaj_cleanup_rollback_state()* function at any time:

```
SELECT emaj.emaj_cleanup_rollback_state();
```

The function returns the number of modified rollback operations.

4.9.7 Deactivating or reactivating event triggers

The E-Maj extension installation procedure activates event triggers to protect it (See §5.3.2). Normally, these triggers must remain in their state. But if the E-Maj administrator needs to deactivate and the reactivate them, he can use 2 dedicated functions.

To deactivate the existing event triggers:

```
SELECT emaj.emaj_disable_protection_by_event_triggers();
```

The function returns the number of deactivated event triggers (this value depends on the installed PostgreSQL version).

To reactivate existing event triggers:

```
SELECT emaj.emaj_enable_protection_by_event_triggers();
```

The function returns the number of reactivated event triggers.

4.10 MULTI-GROUPS FUNCTIONS

4.10.1 General information

To be able to synchronize current operations like group start or stop, set mark or rollback, usual functions dedicated to these tasks have twin-functions that process several tables groups in a single call.

The resulting advantages are:

- to process all tables group in a single transaction,
- to lock tables belonging to all groups at the beginning of the operation to minimize the risk of deadlock.

4.10.2 Functions list

The following table lists the multi-groups functions, with their relative mono-group functions, some of them being discussed later.

Multi-groups functions	Relative mono-group function	§
emaj.emaj_alter_groups()	*emaj.emaj_alter_group()*	4.4
emaj.emaj_start_groups()	*emaj.emaj_start_group()*	4.3.2
emaj.emaj_stop_groups()	*emaj.emaj_stop_group()*	4.3.6
emaj.emaj_set_mark_groups()	*emaj.emaj_set_mark_group()*	4.3.3
emaj.emaj_rollback_groups()	*emaj.emaj_rollback_group()*	4.3.4
emaj.emaj_logged_rollback_groups()	*emaj.emaj_logged_rollback_group()*	4.3.5
emaj.emaj_estimate_rollback_groups()	*emaj.emaj_estimate_rollback_group()*	4.7.3
emaj.emaj_log_stat_groups()	*emaj.emaj_log_stat_group()*	4.7.1
emaj.emaj_detailed_log_stat_groups()	*emaj.emaj_detailed_log_stat_group()*	4.7.2
emaj.emaj_gen_sql_groups()	*emaj.emaj_gen_sql_group()*	4.8.3

The parameters of multi-groups functions are the same as those of their related mono-group function, except the first one. The *TEXT* table group parameter is replaced by a *TEXT ARRAY* parameter representing a tables groups list.

4.10.3 Syntax for groups array

The SQL type of the <groups.array> parameter passed to the multi-groups functions is *TEXT[ ]*, i.e. an array of text data.

According to SQL standard, there are 2 possible syntaxes to specify a groups array, using either braces { }, or the *ARRAY* function.

When using { and }, the full list is written between single quotes, then braces frame the comma separated elements list, each element been placed between double quotes. For instance, in our case, we can write:

```
' { "group 1" , "group 2" , "group 3" } '
```

The SQL function *ARRAY* builds an array of data. The list of values is placed between brackets [ ], and values are separated by comma. For instance, in our case, we can write :

```
ARRAY [ 'group 1' , 'group 2' , 'group 3' ]
```

Both syntax are equivalent.

4.10.4 Other considerations

The order of the groups in the groups list is not meaningful. During the E-Maj operation, the processing order of tables only depends on the priority level defined for each table, and, for tables having the same priority level, from the alphabetic order of their schema and table names.

It is possible to call a multi-groups function to process a list of … one group, or even an empty list. This may allows a set oriented build of this list, using for instance the *array_agg()* function.

A tables groups list may contain duplicate values, NULL values or empty strings. These NULL values or empty strings are simply ignored. If a tables group name is listed several times, only one occurrence is kept.

Format and usage of these functions are strictly equivalent to those of their twin-functions.

However, an additional condition exists for rollback functions: the supplied mark must correspond to the same point in time for all groups. In other words, this mark must have been set by the same *emaj_set_mark_group()* function call.

4.11 PARALLEL ROLLBACK CLIENT

On servers having several processors or processor cores, it may be possible to reduce rollback elapse time by paralleling the operation on multiple threads of execution. For this purpose, E-Maj delivers a specific client to run as a command. It activates E-Maj rollback functions though several parallel connections to the database.

4.11.1 Sessions

To run a rollback in parallel, E-Maj spreads tables and sequences to process for one or several tables groups into “sessions”. Each session is then processed in its own thread.

However, in order to guarantee the integrity of the global operation, the rollback of all sessions is executed inside a single transaction.

To build the most balanced sessions as possible, E-Maj takes into account:

- the number of sessions specified by the user in its command,
- statistics about rows to rollback, as reported by the *emaj_log_stat_group()* function,
- foreign key constraints that link several tables between them, 2 updated tables linked by a foreign key constraint being affected into the same session.

4.11.2 Prerequisites

Two equivalent tools are actually provided, one coded in php and the other in perl. Both need that some software components be installed on the server that executes the command (which is not necessarily the same as the one that hosts the PostgreSQL instance):

- for the php client, the *php* software and its PostgreSQL interface,
- for the perl client, the perl software with the *DBI* and *DBD::Pg* modules.

Rolling back each session on behalf of a unique transaction implies the use of two phase commit. As a consequence, the *max_prepared_transaction* parameter of the *postgresql.conf* file must be adjusted. As the default value of this parameter equals 0, it must be modified by specifying a value at least equal to the maximum number of sessions that will be used.

4.11.3 Syntax

Both php and perl commands share the same syntax:

emajParallelRollback.php -g <group(s).name> -m <mark> -s <number.of.sessions> [OPTIONS]...
and
emajParallelRollback.pl -g <group(s).name> -m <mark> -s <number.of.sessions> [OPTIONS]...

General options:
-l specifies that the requested rollback is a "*logged rollback*" (see §4.3.5)
-a specifies that the requested rollback is allowed to reach a mark set before an alter group operation (see §4.4)
-v displays more information about the execution of the processing
--help only displays a command help
--version only displays the software version

Connection options:
-d database to connect to
-h host to connect to
-p ip-port to connect to
-U connection role to use
-W password associated to the role, if needed

To replace some or all these parameters, the usual *PGDATABASE*, *PGPORT*, *PGHOST* and/or *PGUSER* environment variables can be used.

To specify a list of tables groups in the -g parameter, separate the name of each group by a comma.

The supplied connection role must be either a superuser or a role having *emaj_adm* rights.

For safety reasons, it is not recommended to use the -W option to supply a password. It is rather advisable to use the *.pgpass* file (see PostgreSQL documentation).

To allow the rollback operation to work, the tables group or groups must be in logging state. The supplied mark must also correspond to the same point in time for all groups. In other words, this mark must have been set by the same *emaj_set_mark_group()* function call.

The *'EMAJ_LAST_MARK'* keyword can be used as mark name, meaning the last set mark.

It is possible to monitor the multi-session rollback operations with the same tools as for mono-session rollbacks.

In order to test both *emajParallelRollback* commands, the E-Maj extension supplies a test script, emaj_prepare_parallel_rollback_test*.sql*. It prepares an environment with two

tables groups containing some tables and sequences, on which some updates have been performed, with intermediate marks. Once this script has been executed under *psql*, the command displayed at the end of the script can be simply run.

4.11.4 Examples

The command:

```
./client/emajParallelRollback.php -d mydb -g myGroup1 -m Mark1 -s 3
```

logs on database mydb and executes a rollback of group myGroup1 to mark Mark1, using 3 parallel sessions.

The command:

```
./client/emajParallelRollback.pl -d mydb -g "myGroup1,myGroup2" -m Mark1 -s 3 -l
```

logs on database mydb and executes a *logged rollback* of both groups myGroup1 and myGroup2 to mark Mark1, using 3 parallel sessions.

4.12 ROLLBACK MONITORING CLIENT

E-Maj delivers an external client to run as a command that monitors the progress of rollback operations in execution.

4.12.1 Prerequisite

Two equivalent tools are actually provided, one coded in php and the other in perl. Both need that some software components be installed on the server that executes the command (which is not necessarily the same as the one that hosts the PostgreSQL instance):

- for the php client, the *php* software and its PostgreSQL interface,
- for the perl client, the perl software with the *DBI* and *DBD::Pg* modules.

4.12.2 Syntax

Both php and perl commands share the same syntax:

emajRollbackMonitor.php [OPTIONS]...
and
emajRollbackMonitor.pl [OPTIONS]...

General options:

-i	time interval between 2 displays (in seconds, default = 5s)
-n	number of displays (default = 1)
-a	maximum time interval for rollback operations to display (in hours, default = 24h)
-l	maximum number of completed rollback operations to display (default = 3)
--help	only displays a command help
--version	only displays the software version

Connection options:

-d	database to connect to
-h	host to connect to
-p	ip-port to connect to
-U	connection role to use
-W	password associated to the role, if needed

To replace some or all these parameters, the usual *PGDATABASE*, *PGPORT*, *PGHOST* and/or *PGUSER* environment variables can be used.

The supplied connection role must either be a super-user or have *emaj_adm* or *emaj_viewer* rights.

For security reasons, it is not recommended to use the -W option to supply a password. Rather, it is advisable to use the *.pgpass* file (see PostgreSQL documentation).

Examples

The command:

```
./client/emajRollbackMonitor.php -i 3 -n 10
```

displays 10 times and every 3 seconds, the list of in progress rollback operations and the list of the at most 3 latest rollback operations completed in the latest 24 hours.

The command:

```
./client/emajRollbackMonitor.pl -a 12 -l 10
```

displays only once the list of in progress rollback operations and the list of at most 10 operations completed in the latest 12 hours.

Example of display:

```
 E-Maj (version 3.3.0) - Monitoring rollbacks activity
----------------------------------------------------------------
04/02/2018 - 12:07:17
** rollback 34 started at 2020 -02-04 12:06:20.350962+02 for groups
{myGroup1,myGroup2}
   status: COMMITTED ; ended at 2020 -02-04 12:06:21.149111+02
** rollback 35 started at 2020 -02-04 12:06:21.474217+02 for groups {myGroup1}
   status: COMMITTED ; ended at 2020 -02-04 12:06:21.787615+02
-> rollback 36 started at 2020 -02-04 12:04:31.769992+02 for groups {group1232}
   status: EXECUTING ; completion 89 % ; 00:00:20 remaining
-> rollback 37 started at 2020 -02-04 12:04:21.894546+02 for groups {group1233}
   status: LOCKING ; completion 0 % ; 00:22:20 remaining
-> rollback 38 started at 2020 -02-04 12:05:21.900311+02 for groups {group1234}
   status: PLANNING ; completion 0 %
```

5 MISCELLANEOUS

5.1 PARAMETERS

The E-Maj extension works with some parameters. Those are stored into the *emaj_param* internal table.

The emaj_param table structure is the following:

Column	**Type**	**Description**
param_key	TEXT	keyword identifying the parameter
param_value_text	TEXT	parameter value, if its type is text (otherwise NULL)
param_value_numeric	NUMERIC	parameter value, if its type is numeric (otherwise NULL)
param_value_boolean	BOOLEAN	parameter value, if its type is boolean (otherwise NULL)
param_value_interval	INTERVAL	parameter value, if its type is time interval (otherwise NULL)

The E-Maj extension installation procedure inserts a single row into the *emaj_param* table. This row, that should not be modified, describes parameter:

- *version* (text) current E-Maj version.

But the E-Maj administrator may insert other rows into the *emaj_param* table to change the default value of some parameters.

Presented in alphabetic order, the existing key values are:

- *alter_log_table* (text) *ALTER TABLE* directive executed at the log table creation ; no *ALTER TABLE* exectuted by default (see §5.2.2).
- *avg_fkey_check_duration* (interval) default value = 20 µs ; defines the average duration of a *foreign key* value check ; can be modified to better represent the performance of the server that hosts the database (see §4.7.3).
- *avg_row_delete_log_duration* (interval) default value = 10 µs ; defines the average duration of a log row deletion ; can be modified to better represent the performance of the server that hosts the database (see §4.7.3).
- *avg_row_rollback_duration* (interval) default value = 100 µs ; defines the average duration of a row rollback ; can be modified to better represent the performance of the server that hosts the database (see §4.7.3).
- *fixed_dblink_rollback_duration* (interval) default value = 4 ms ; defines an additional cost for each rollback step when a *dblink* connection is used ; can be modified to better represent the performance of the server that hosts the database (see §4.7.3).

- *fixed_table_rollback_duration* (interval) default value = 1 ms ; defines a fixed rollback cost for any table belonging to a group ; can be modified to better represent the performance of the server that hosts the database (see §4.7.3).
- *fixed_step_rollback_duration* (interval) default value = 2,5 ms ; defines a fixed cost for each rollback step ; can be modified to better represent the performance of the server that hosts the database (see §4.7.3).
- *history_retention* (interval) default value = 1 year ; it can be adjusted to change the retention delay of rows in the *emaj_hist* history table (see § 5.4),

Below is an example of a SQL statement that defines a retention delay of history table's rows equal to 3 months:

```
INSERT INTO emaj.emaj_param (param_key, param_value_interval)
VALUES ('history_retention','3 months'::interval);
```

Any change in the *emaj-param* table's content is logged into the *emaj_hist* table (cf §5.4).

Only super-user and roles having *emaj_adm* rights can access the *emaj_param* table.

Roles having *emaj_viewer* rights can only access a part of the *emaj_param* table, through the *emaj.emaj_visible_param* view. This view just masks the real value of the *param_value_text* column for the '*dblink_user_password*' key.

The *emaj_export_parameters_configuration()* and *emaj_import_parameters_configuration()* functions allow to save the parameters values and restore them (Cf §4.9.2).

5.2 LOG TABLES STRUCTURE

5.2.1 Standart structure

The structure of log tables is directly derived from the structure of the related application tables. The log tables contain the same columns with the same type. But they also have some additional technical columns:

- *emaj_verb* type of the SQL verb that generated the update (INS, UPD, DEL)
- *emaj_tuple* row version (OLD for DEL and UPD, NEW for INS and UPD)
- *emaj_gid* log row identifier
- *emaj_changed* log row insertion timestamp
- *emaj_txid* transaction id (the PostgreSQL *txid*) that performed the update
- *emaj_user* connection role that performed the update

5.2.2 Adding technical columns

It is possible to add one or several technical columns to enrich the traces. These columns value must be set as a default value (a *DEFAULT* clause) associated to a function (so that the log triggers are not impacted).

To add one or several technical columns, a parameter of key '*alter_log_table'* must be inserted into the *emaj_param* table (Cf §5.1). The associated text value must contain an ALTER TABLE clause. At the log table creation time, if the parameter exists, an ALTER TABLE statement with this parameter is executed.

For instance, one can add to log tables a column to record the value of the '*application_name'* connection field with:

```
INSERT INTO emaj.emaj_param (param_key, param_value_text) VALUES
  ('alter_log_table', 'ADD COLUMN extra_col_appname TEXT
    DEFAULT current_setting("application_name")');
```

Several ADD COLUMN directives may be concatenated, separated by a comma. For instance, to create columns recording the ip adress and port of the connected client:

```
INSERT INTO emaj.emaj_param (param_key, param_value_text) VALUES
  ('alter_log_table', 'ADD COLUMN emaj_user_ip INET DEFAULT inet_client_addr(),
ADD COLUMN emaj_user_port INT DEFAULT inet_client_port()');
```

To change the structure of existing log tables once the *alter_log_table* parameter has been set, the tables groups must be dropped and then recreated.

5.3 RELIABILITY

Two additional elements help in ensuring the E-Maj reliability: internal checks are performed at some key moments of tables groups life and event trigers can block some risky operations.

5.3.1 Internal checks

When a function is executed to start a tables group, to set a mark or to rollback a tables group, E-Maj performs some checks in order to verify the integrity of the tables groups to process.

These tables group integrity checks verify that:

- the PostgreSQL version at tables group creation time is compatible with the current version,

- each application sequence or table of the group always exists,
- each table of the group has its log table, its log function and its triggers,
- the log tables structure always reflects the related application tables structure, and contains all required technical columns,
- for *ROLLBACKABLE* tables groups, no table has been altered as *UNLOGGED* or *WITH OIDS*,
- for *ROLLBACKABLE* tables groups, application tables have their primary key and their structure has not changed.

By using the *emaj_verify_all()* function (§4.9.1), the administrator can perform the same checks on demand on all tables groups.

5.3.2 Event triggers

Installing E-Maj adds 2 event triggers of type "*sql_drop*":

- *emaj_sql_drop_trg* blocks the drop attempts of:
  - ✔ any E-Maj object (log schema, log table, log sequence, log function and log trigger),
  - ✔ any application table or sequence belonging to a table group in "*LOGGING*" state,
  - ✔ any schema containing at least one table or sequence belonging to a table group in "*LOGGING*" state.

- *emaj_protection_trg* blocks the drop attempts of the *emaj* extension itself and the main *emaj* schema.

Installing E-Maj also adds 1 event trigger of type "*table_rewrite*":

- *emaj_table_rewrite_trg* blocks any structure change of application or log table.

It is possible to deactivate and reactivate these event triggers thanks to 2 functions: *emaj_disable_protection_by_event_triggers()* and *emaj_enable_protection_by_event_triggers()* (see §4.9.7).

However, the protections do not cover all risks. In particular, they do not prevent any tables or sequences renaming or any schema change. And some other DDL statements altering tables structure will not fire any trigger.

5.4 TRACES OF OPERATIONS

All operations performed by E-Maj, and that impact in any way a tables group, are traced into a table named *emaj_hist*.

The emaj_hist table structure is the following:

Column	Type	Description
hist_id	BIGSERIAL	serial number identifying a row in this history table
hist_datetime	TIMESTAMPTZ	recording date and time of the row
hist_function	TEXT	function associated to the traced event
hist_event	TEXT	kind of event
hist_object	TEXT	object related to the event (group, table or sequence)
hist_wording	TEXT	additional comments
hist_user	TEXT	role whose action has generated the event
hist_txid	BIGINT	identifier of the transaction that has generated the event

The *hist_function* column can take the following values:

- *ADJUST_GROUP_PROPERTIES* ajust the *group_has_waiting_changes* column content of the *emaj_group* table
- *ALTER_GROUP* tables group change
- *ALTER_GROUPS* tables groups change
- *ASSIGN_SEQUENCE* sequence assigned to a tables group
- *ASSIGN_SEQUENCES* sequences assigned to a tables group
- *ASSIGN_TABLE* table assigned to a tables group
- *ASSIGN_TABLES* tables assigned to a tables group
- *CLEANUP_RLBK_STATE* cleanup the state of recently completed rollback operations
- *COMMENT_GROUP* comment set on a group
- *COMMENT_MARK_GROUP* comment set on a mark for a tables group
- *CONSOLIDATE_RLBK_GROUP* consolidate a logged rollback operation
- *CREATE_GROUP* tables group creation
- *DBLINK_OPEN_CNX* open a *dblink* connection for a rollback operation
- *DBLINK_CLOSE_CNX* close a *dblink* connection for a rollback operation
- *DELETE_MARK_GROUP* mark deletion for a tables group
- *DISABLE_EVENT_TRIGGERS* desactivate event triggers
- *DROP_GROUP* tables group suppression
- *EMAJ_INSTALL* E-Maj installation or version update
- *ENABLE_EVENT_TRIGGERS* activate event triggers
- *EXPORT_GROUPS* export a tables groups configuration
- *EXPORT_PARAMETERS* export an E-maj parameters configuration
- *FORCE_DROP_GROUP* tables group forced suppression
- *FORCE_STOP_GROUP* tables group forced stop
- *GEN_SQL_GROUP* generation of a psql script to replay updates for a tables group
- *GEN_SQL_GROUPS* generation of a psql script to replay updates for several tables groups
- *IMPORT_GROUPS* import a tables groups configuration

- *IMPORT_PARAMETERS* import an E-maj parameters configuration
- *LOCK_GROUP* lock set on tables of a group
- *LOCK_GROUPS* lock set on tables of several groups
- *LOCK_SESSION* lock set on tables for a rollback session
- *MODIFY_TABLE* table properties change
- *MODIFY_TABLES* tables properties change
- *MOVE_SEQUENCE* sequence moved to another tables group
- *MOVE_SEQUENCES* sequences moved to another tables group
- *MOVE_TABLE* table moved to another tables group
- *MOVE_TABLES* tables moved to another tables group
- *PROTECT_GROUP* set a protection against rollbacks on a group
- *PROTECT_MARK_GROUP* set a protection against rollbacks on a mark for a group
- *PURGE_HISTORY* delete from the *emaj_hist* table the events prior the retention delay
- *REMOVE_SEQUENCE* sequence removed from its tables group
- *REMOVE_SEQUENCES* sequences removed from their tables group
- *REMOVE_TABLE* table removed from its tables group
- *REMOVE_TABLES* tables removed from their tables group
- *RENAME_MARK_GROUP* mark rename for a tables group
- *RESET_GROUP* log tables content reset for a group
- *ROLLBACK_GROUP* rollback updates for a tables group
- *ROLLBACK_GROUPS* rollback updates for several tables groups
- *ROLLBACK_TABLE* rollback updates for one table
- *ROLLBACK_SEQUENCE* rollback one sequence
- *SET_MARK_GROUP* mark set on a tables group
- *SET_MARK_GROUPS* mark set on several tables groups
- *SNAP_GROUP* snap all tables and sequences for a group
- *SNAP_LOG_GROUP* snap all log tables for a group
- *START_GROUP* tables group start
- *START_GROUPS* tables groups start
- *STOP_GROUP* tables group stop
- *STOP_GROUPS* tables groups stop
- *SYNC_DEF_GROUP* *emaj_group_def* table synchronization
- *UNPROTECT_GROUP* remove a protection against rollbacks on a group
- *UNPROTECT_MARK_GROUP* remove a protection against rollbacks on a mark for a group

The *hist_event* column can take the following values:

- *BEGIN*
- *DELETED PARAMETER* parameter deleted from *emaj_param*
- *EMAJ_GROUP_DEF SYNCHRONIZED* tables group synchronized in the *emaj_group_def* table
- *END*
- *EVENT TRIGGERS DISABLED*
- *EVENT TRIGGERS ENABLED*
- *GROUP_CREATED* new tables group created
- *INSERTED PARAMETER* parameter inserted into *emaj_param*
- *LOG DATA TABLESPACE CHANGED* tablespace for the log table modified
- *LOG INDEX TABLESPACE CHANGED* tablespace for the log index modified

- *LOG_SCHEMA CREATED*
- *LOG_SCHEMA DROPPED*
- *MARK DELETED*
- *NAMES PREFIX CHANGED* — E-Maj names prefix modified
- *NOTICE* — notice message issued by a rollback
- *PRIORITY CHANGED* — priority level modified
- *SEQUENCE ADDED* — sequence added to a logging tables group
- *SEQUENCE MOVED* — sequence moved from one group to another
- *SEQUENCE REMOVED* — sequence removed from a logging tables group
- *TABLE ADDED* — table added to a logging tables group
- *TABLE MOVED* — table moved from one group to another
- *TABLE REMOVED* — table removed from a logging tables group
- *UPDATED PARAMETER* — parameter updated in *emaj_param*
- *WARNING* — warning message issued by a rollback

The *emaj_hist* content can be viewed by anyone who has the proper access rights on this table (superuser, *emaj_adm* or *emaj_viewer* roles).

Two other internal tables keep traces of groups alter or rollback operations:

- *emaj_alter_plan* lists the elementary steps performed during the execution of *emaj_alter_group()* and related functions (Cf §4.4),
- *emaj_rlbk_plan* lists the elementary steps performed during the execution of *emaj_rollback_group()* and related functions (Cf §4.3.4 and 4.3.5).

When a tables group is started, using the *emaj_start_group()* function, or when old marks are deleted, using the *emaj_delete_before_mark_group()* function, the oldest events are deleted from *emaj_hist* tables. The events kept are those not older than a parametrised retention delay and not older than the oldest active mark and not older than the oldest uncompleted rollback operation. By default, the retention delay for events equals 1 year. But this value can be modified at any time by inserting the *history_retention* parameter into the *emaj_param* table with a SQL statement (see §5.1). The same retention applies to the tables that log elementary steps of tables groups alter or rollback operations.

5.5 IMPACTS ON INSTANCE AND DATABASE ADMINISTRATION

5.5.1 Stopping and restarting the instance

Using E-Maj doesn't bring any particular constraint regarding stopping and restarting a PostgreSQL instance.

5.5.1.1 General rule

At instance restart, all E-Maj objects are in the same state as at instance stop: log triggers of tables groups in *LOGGING* state remain enabled and log tables contain cancel-able updates already recorded.

If a transaction with table updates were not committed at instance stop, it would be rolled back during the recovery phase of the instance start, the application tables updates and the log tables updates being cancelled at the same time.

This rule also applies of course to transactions that execute E-Maj functions, like a tables group start or stop, a rollback, a mark deletion,...

5.5.1.2 Sequences rollback

Due to a PostgreSQL constraint, the rollback of application sequences assigned to a tables group is the only operation that is not protected by transactions. That is the reason why application sequences are processed at the very end of the rollback operations (See §4.3.4). (For the same reason, at set mark time, application sequences are processed at the beginning of the operation.)

In case of an instance stop during an E-Maj rollback execution, it is recommended to rerun this rollback just after the instance restart, to ensure that application sequences and tables remain properly synchronised.

5.5.2 Saving and restoring the database

Using E-Maj allows a reduction in the database saves frequency. But E-Maj cannot be considered as a substitute to regular database saves that remain indispensable to keep a full image of databases on an external support.

5.5.2.1 File level saves and restores

When saving or restoring instances at file level, it is essential to save or restore ALL instance files, including those stored on dedicated tablespaces.

After a file level restore, tables groups are in the very same state as at the save time, and the database activity can be restarted without any particular E-Maj operation.

5.5.2.2 Logical saves and restores of entire database

To properly save and restore a database with E-Maj, using *pg_dump*, and *psql* or *pg_restore*, it is essential that both source and restored databases use the **same E-Maj version**. If this is not the case, the content of some technical tables may be not synchronised with their structure. Reading the row of key '*emaj_version*' in the

emaj.emaj_param table may help in knowning the version of an E-Maj extension created in a database.

Regarding stopped tables groups (in *IDLE* state), as log triggers are disabled and the content of related log tables is meaningless, there is no action required to find them in the same state as at save time.

Concerning tables groups in *LOGGING* state at save time, it is important to be sure that log triggers will only be activated after the application tables rebuild. Otherwise, during the tables rebuild, tables updates would also be recorded in log tables!

When using *pg_dump* command for saves and *psql* or *pg_restore* commands for restores, and processing full databases (schema + data), these tools recreate triggers, E-Maj log triggers among them, after tables have been rebuilt. So there is no specific precaution to take.

On the other hand, in case of data only save or restore (i.e. without schema, using *-a* or *--data-only* options), the *--disable-triggers* must be supplied:

- with *pg_dump* (or *pg_dumpall*) with save in *plain* format (and *psql* is used to restore),
- with *pg_restore* command with save in *tar* or *custom* format.

Restoring the database structure generates 2 error messages reporting that the *_emaj_protection_event_trigger_fnct()* function and the *emaj_protection_trg* event trigger already exist:

```
...
ERROR:  function "_emaj_protection_event_trigger_fnct" already exists with same
argument types
...
ERROR:  event trigger "emaj_protection_trg" already exists
...
```

This message display is normal and does not indicate a defective restore. Indeed, both objects are created with the extension and are then detached from it, so that the trigger can block any attempt of the extension drop. As a result, the *pg_dump* tool saves them as independent objects. And when restoring, these objects are created twice, first with the emaj extension creation, and then as independent objects, this second attempt generating both error messages.

5.5.2.3 Logical save and restore of partial database

With *pg_dump* and *pg_restore* tools, database administrators can perform on a subset of database schemas or tables.

Restoring a subset of application tables and/or log tables generates a heavy risk of data corruption in case of later E-Maj rollback of concerned tables. Indeed, it is impossible to

guarantee in this case that application tables, log tables and internal E-Maj tables that contain essential data for rollback, remain coherent.

If it is necessary to perform partial application tables restores, a drop and recreation of all tables groups concerned by the operation must be performed just after.

The same way, it is strongly recommended to NOT restore a partial *emaj* schema content.

The only case of safe partial restore concerns a full restore of the *emaj* schema content as well as all tables belonging to all groups that are created in the database.

5.5.3 Data load

Beside using *pg_restore* or *psql* with files produced by *pg_dump*, it is possible to efficiently load large amounts of data with the *COPY* SQL verb or the *\copy psql* meta-command. In both cases, this data loading fires *INSERT* triggers, among them the E-Maj log trigger. Therefore, there is no constraint to use *COPY* or *\copy* in E-Maj environment.

With other loading tools, it is important to check that triggers are effectively fired for each row insertion.

5.5.4 Tables reorganisation

5.5.4.1 Reorganisation of application tables

Application tables protected by E-Maj can be reorganised using the SQL *CLUSTER* command. Whether or not log triggers are enabled, the organisation process has no impact on log tables content.

5.5.4.2 Reorganisation of E-Maj tables

The index corresponding to the primary key of each table from E-Maj schemas (neither log tables nor technical tables) is declared “cluster”.

So using E-Maj may have an operational impact regarding the execution of *CLUSTER* SQL commands at database level.

When E-Maj is used in continuous mode (with deletion of oldest marks instead of regular tables groups stop and restart), it is recommended to regularly reorganize E-Maj log tables. This reclaims unused disk space following mark deletions.

5.5.5 Using E-Maj with replication

5.5.5.1 Integrated physical replication

E-Maj is totally compatible with the use of the different PostgreSQL integrated physical replication modes (WAL archiving and PITR, asynchronous and synchronous Streaming Replication). Indeed, all E-Maj objects hosted in the instance are replicated like all other objects of the instance.

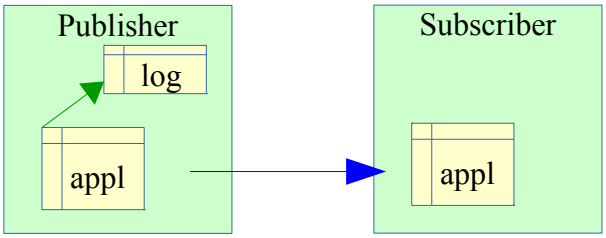
However, because of the way PostgreSQL manages sequences, the sequences' current values may be a little forward on slave instances than on the master instance. For E-Maj, this may lightly overestimate the number of log rows in general statistics. But there is no consequence on the data integrity.

5.5.5.2 Integrated logical replication

Starting with version 10, PostgreSQL includes logical replication mechanisms. The replication granularity is the table. The *publication* object used in logical replication is quite close to the E-Maj tables group concept, except that a *publication* cannot contain sequences.

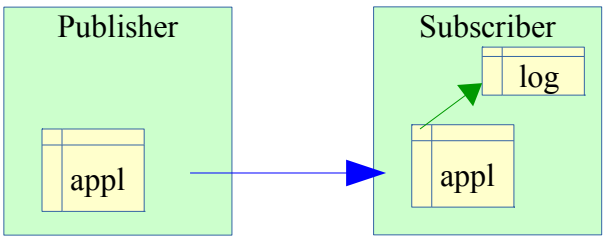
Several cases have to be examined.

Replication of application tables managed by E-Maj



An application table that belongs to a tables group can be replicated. The effect of any rollback operation that may occur would be simply replicated on subscriber side, as long as no filter has been applied on replicated SQL verbs types.

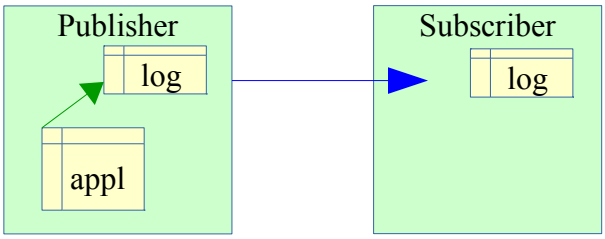
Replication of application tables with E-Maj activated on subscriber side



It is possible to include an application table into a tables group, with updates coming from a logical replication flow. But all E-Maj operations (starting/stopping the group, setting

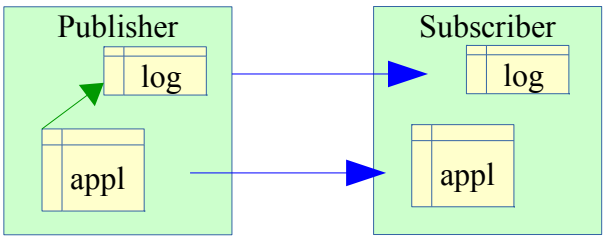
marks,…) must of course be executed on the subscriber side. An E-Maj rollback operation can be launched once the replication flow has been stopped (to avoid updates conflicts). But then, tables on both publisher and subscriber sides are not coherent anymore.

Replication of E-Maj log tables



It is technicaly possible to replicate an E-Maj log table (once found a way to get the DDL that creates the log table – using *pg_dump* for instance). This allows to duplicate or concentrate logs content on another server. But the replicated log table can only be used for log **auditing**. As log sequences or *TRUNCATE* verbs are not replicated, these logs cannot be used for other purposes.

Replication of application tables and E-Maj log tables



Application tables and log tables can be simultaneously replicated. But as seen previously, these replicated logs can only be used for **auditing** purpose. E-Maj rollback operations can only be executed on publisher side.

5.5.5.3 Other replication solutions

Using E-Maj with external replication solutions based on triggers like Slony or Londiste, requires some attention... It is probably advisable to avoid replicating log tables and E-Maj technical tables.

5.6 SENSITIVITY TO SYSTEM TIME CHANGE

To ensure the integrity of tables managed by E-Maj, it is important that the rollback mechanism be insensitive to potential date or time change of the server that hosts the PostgreSQL instance.

The date and time of each update or each mark is recorded. But nothing other than sequence values recorded when marks are set, are used to frame operation in time. So rollbacks and mark deletions are insensitive to potential system date or time change.
However, two minor actions may be influenced by a system date or time change:

- the deletion of oldest events in the *emaj_hist* table (the retention delay is a time interval),
- finding the name of the mark immediately preceding a given date and time as delivered by the *emaj_get_previous_mark_group()* function.

5.7 PERFORMANCE

5.7.1 Updates recording overhead

Recording updates in E-Maj log tables has necessarily an impact on the duration of these updates. The global impact of this log on a given processing depends on numerous factors. Among them:

- the part that the update activity represents on the global processing,
- the intrinsic performance characteristics of the storage subsystem that supports log tables.

However, the E-Maj updates recording overhead is generally limited to a few per-cents. But this overhead must be compared to the duration of potential intermediate saves avoided with E-Maj.

5.7.2 E-Maj rollback duration

The duration of an E-Maj rollback depends on several factors, like:

- the number of updates to cancel,
- the intrinsic characteristics of the server and its storage material and the load generated by other activities hosted on the server,
- triggers or foreign keys on tables processed by the rollback operation,
- contentions on tables at lock set time.

To get an order of magnitude of an E-Maj rollback duration, it is possible to use the *emaj_estimate_rollback_group()* and *emaj_estimate_rollback_groups() functions* (See §4.7.3).

5.7.3 Optimizing E-Maj operations

Here are some advice to optimize E-Maj operations:

5.7.3.1 Use tablespaces

Creating tables into tablespaces located in dedicated disks or file systems is a way to more efficiently spread the access to these tables. To minimize the disturbance of application tables access by log tables access, the E-Maj administrator has two ways to use tablespaces for log tables and indexes location.

By setting a specific default tablespace for the session before the tables groups creation, log tables and indexes are created by default into this tablespace, without any additional action (see §4.2.2.2).

But through parameters set into the *emaj_group_def* table, it is also possible to specify a tablespace to use for any log table or log index (see §4.2.2.2).

5.7.3.2 Declare foreign keys as DEFERRABLE

Foreign keys can be explicitly declared as *DEFERRABLE* at creation time. If a foreign key is declared *DEFERRABLE* and no *ON DELETE* or *ON UPDATE* clause is used, this foreign key is not dropped at the beginning and recreated at the end of an E-Maj rollback operation. The foreign key checks of updated rows are just deferred to the end of the rollback function execution, once all log tables are processed. This generally greatly speeds up the rollback operation.

5.7.3.3 Modify memory parameters

Increasing the value of the *work_mem* parameter when performing an E-Maj rollback may bring some performance gains.

If foreign keys have to be recreated by an E-Maj rollback operation, increasing the value of the *maintenance_work_mem* parameter may also help.

If the E-Maj rollback functions are directly called in SQL, these parameters can be previously set at session level, with statements like:

```
SET work_mem = <value>;
SET maintenance_work_mem = <value>;
```

If the E-Maj rollback operations are executed by a web client, it is also possible to set these parameters at function level, as *superuser*:

```
ALTER FUNCTION emaj._rlbk_tbl(emaj.emaj_relation, BIGINT, BIGINT, INT,
BOOLEAN) SET work_mem = <value>;
ALTER FUNCTION emaj._rlbk_session_exec(INT, INT) SET maintenance_work_mem
= <value>;
```

5.8 USAGE LIMITS

The E-Maj extension usage has some limits.

- The minimum required PostgreSQL version is 9.5..
- All tables belonging to a "*rollbackable*" tables group must have an explicit *PRIMARY KEY*.
- *UNLOGGED* and *WITH OIDS* tables can only be members of "*audit_only*" tables groups.
- *TEMPORARY* tables are not supported by E-Maj.
- If a *TRUNCATE* SQL verb is executed on an application table belonging to a group, E-Maj is not able to reset this table in a previous state. Indeed, when a *TRUNCATE* is executed, no trigger is executed at each row deletion. A trigger, created by E-Maj, blocks any *TRUNCATE* statement on any table belonging to a tables group in logging state.
- Using a global sequence for a database leads to a limit in the number of updates that E-Maj can manage throughout its life. This limit equals 2^63, about 10^19 (but only 10^10 on oldest platforms), which still allow to record 10 million updates per second (100 times the best performance benchmarks results in 2012) during … 30,000 years (or at worst 100 updates per second during 5 years). Would it be necessary to reset the global sequence, the E-Maj extension would just have to be un-installed and re-installed.
- If a DDL operation is executed on an application table belonging to a tables group, E-Maj is not able to reset the table in its previous state.

To understand this last point, it may be interesting to understand the consequences of a DDL statement execution on the way E-Maj works, depending on the kind of executed operation.

- If a new table were created, it would be unable to enter into a group's definition until this group is stopped, dropped and then recreated.
- If a table belonging to a group in logging state were dropped, there would be no way for E-Maj to recover it's structure and its content.
- For a table belonging to a tables group in logging state, adding or deleting a column would generate an error at the next *INSERT/UPDATE/DELETE* SQL verb execution.
- For a table belonging to a tables group in logging state, renaming a column would not necessarily generate any error during further log recording. But the checks that E-Maj performs would block any attempt to set a new mark or rollback the related group.
- For a table belonging to a tables group in logging state, changing the type of a column would lead to an inconsistency between the application table and the log table. But, depending on the change of data type applied, updates logging could either work or not. Furthermore, data could be corrupted, for instance in case of increased data length not propagated in log tables. Anyway, due to the checks performed by E-Maj, any attempt to set a new mark or rollback the related group would then fail.
- However, it is possible to create, modify or drop indexes, rights or constraints for a table belonging to a tables group in logging state. But of course, cancelling these changes could not be done by E-Maj.

5.9 USER'S RESPONSIBILITY

5.9.1 Defining tables groups content

Defining the content of tables group is essential to guarantee the database integrity. It is the E-Maj administrator's responsibility to ensure that all tables updated by a given operation are really included in a single tables group.

5.9.2 Appropriate call of main functions

emaj_start_group(), *emaj_set_mark_group()*, *emaj_rollback_group()* and *emaj_logged_rollback_group()* functions (and their related multi-groups functions) set explicit locks on tables of the group to be sure that no transactions updating these tables are running at the same time. But it is the user's responsibility to execute these operations “at the right time”, i.e. at moments that really correspond to a stable point in the life of these tables. He must also take care of warning messages that may be reported by E-Maj rollback functions.

5.9.3 Management of application triggers

Triggers may have been created on application tables. It is not rare that these triggers perform one or more updates on other tables. In such a case, it is the E-Maj administrator's responsibility to understand the impact of E-Maj rollback operations on tables concerned by triggers, and if needed, to take the appropriate measures.

If the trigger simply adjusts the content of the row to insert or update, the logged data will contain the final columns value. So in case of rollback, the log table contains the right data to process. And as the trigger is by default automatically disabled at rollback time, the trigger cannot disturb the rollback processing

If the trigger updates another table, two cases must be considered:

- if the updated table belongs to the same tables group, the automatic trigger disabling and the rollback of both tables will let them in the expected state,
- if the updated table does not belong to the same tables group, it is essential to analyse the consequences of a rollback operation, in order to avoid a de-synchronisation between both tables. If needed, the *emaj_ignore_app_trigger()* function can be used to not disable the trigger at rollback time (see §4.9.4). But merely deactivating the trigger may not be sufficient and some other actions may be required.

For more complex triggers, it is essential to perfectly understand their impacts on E-Maj rollbacks and take any appropriate mesure at rollback time.

5.9.4 Internal E-Maj table or sequence change

With the rights they have been granted, *emaj_adm* roles and super-users can update any E-Maj internal table.

But in practice, only two tables may be updated by these users: *emaj_group_def* and *emaj_param*. Any other internal table or sequence update my lead to data corruption during rollback operations.

6 EMAJ_WEB

A web application, Emaj_web, makes E-Maj use much easier.

For the records, a plugin for *phpPgAdmin* also existed. But it is not maintained any more since E-Maj 3.0.

6.1 OVERVIEW

Emaj_web has borrowed to phpPgAdmin its infrastructure (browser, icon trails, database connection, management,…) and some useful functions like browsing the tables content or editing SQL queries.

For databases into which the E-Maj extension has been installed, and if the user is connected with a role that owns the required rights, all E-Maj objects are accessible.

It is then possible to:

- define or modify groups content,
- see the list of tables groups and perform any possible action, depending on groups state (create, drop, start, stop, set or remove a mark, rollback, add or modify a comment),
- see the list of the marks that have been set for a group, and perform any possible action on them (delete, rename, rollback, add or modify a comment),
- get statistics about log tables content and see their content,
- monitor in progress rollback operations.

6.2 EMAJ_WEB CLIENT INSTALLATION

6.2.1 Prerequisite

Emaj_web requires a web server with a php interpreter.

6.2.2 Download

The *Emaj_web* application can be downloaded from the following git repository:
https://github.com/dalibo/emaj_web

6.2.3 Application configuration

The configuration is centralized into a single file: *emaj_web/conf/config.inc.php.* It contains the general parameters of the applications, and the description of the PostgreSQL instances connections.

When the number of instances is large, it is possible to split them into “instances groups”. A group can contain instances or other instance groups.

In order to submit batch rollbacks (i.e. without blocking the use of the browser while the rollback operation is in progress), it is necessary to specify a value for two configuration parameters:

- `$conf['psql_path']` defines the access path of the *psql* executable file,
- `$conf['temp_dir']` defines a temporary directory that rollback functions can use.

The distributed *emaj_web/conf/config.inc.php-dist* file can be used as a configuration template.

6.3 USING EMAJ_WEB

6.3.1 Access to Emaj_web and databases

Accessing Emaj_web in a browser displays the welcome page.

To sign in to a database, select the target instance in the left browser or in the ‘*servers*’ tab, and fill the connection identifier and password. Several connections can remain opened simultaneously.

Once connected to a database where the emaj extension has been installed, the user interacts with the extension, depending on the role it owns (super-user, *emaj_adm* or *emaj_viewer*).

On the left, the browser tree shows all the configured instances, that may be split into instances groups, and all the databases they contain. By unfolding a database object, the user reaches the E-Maj tables groups and the existing schemas.

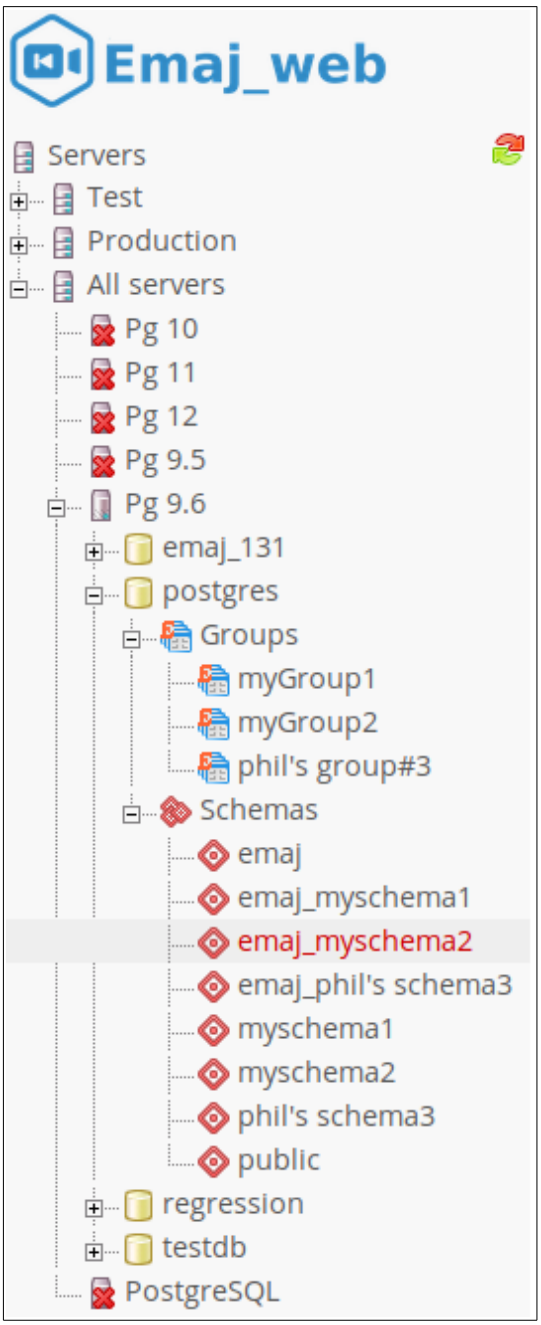


Figure 1 – The browser tree.

6.3.2 Tables groups list

By selecting a database, the user reaches a page that lists all tables groups created in this database.

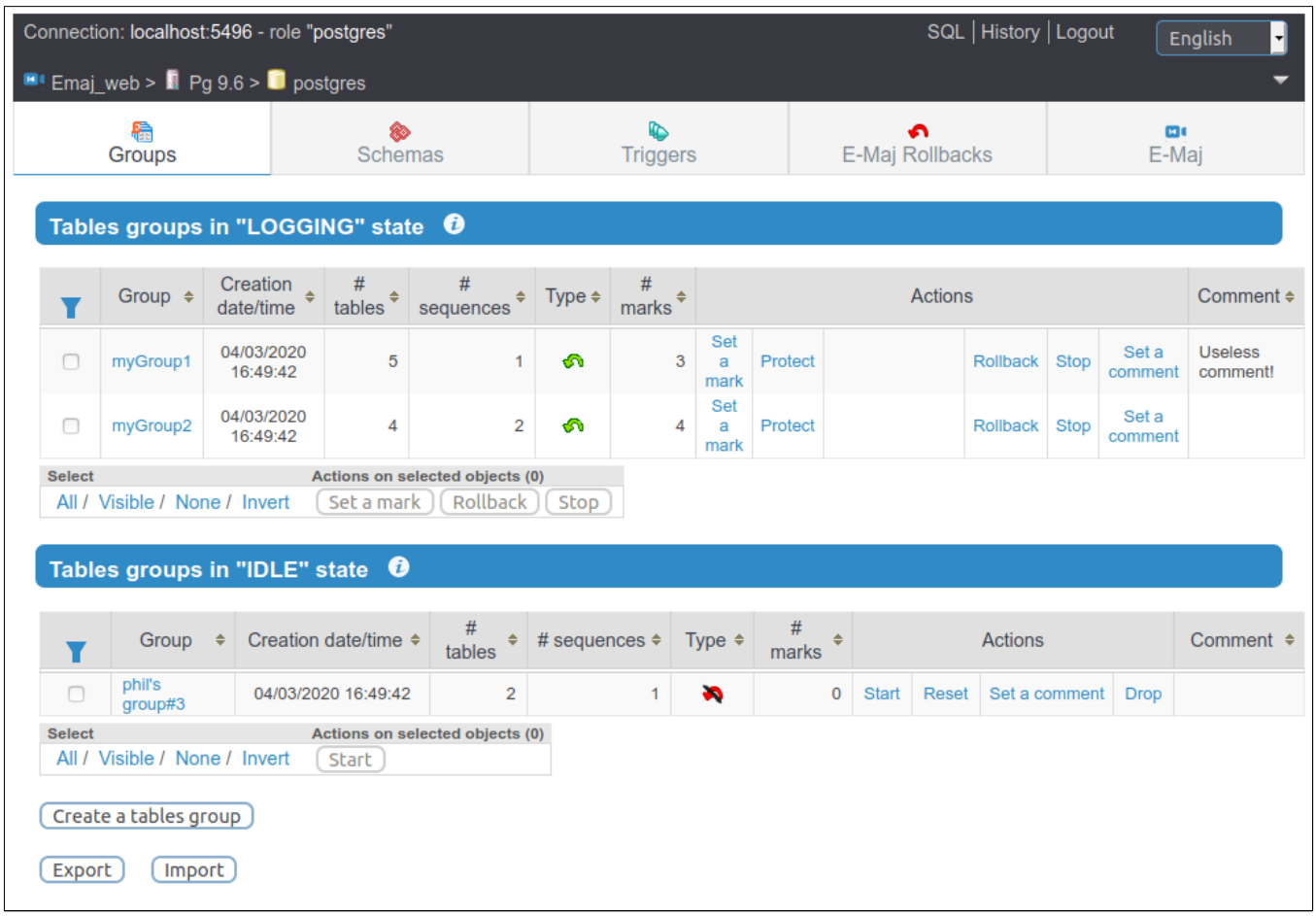


Figure 2 – Tables groups list.

This page displays two lists:

- the tables groups in *LOGGING* state
- the tables groups in *IDLE* state.

For each created tables group, the following attributes are displayed:

- its creation date and time,
- the number of application tables and sequences it contains,
- its type ("*ROLLBACKABLE*" or "*AUDIT_ONLY*", protected against rollback or not),
- the number of marks it owns,
- its associated comment, if any.

For each tables group, several buttons are available so that the user can perform any possible action, depending on the group state.

At the bottom of the page, three buttons allow to create a new tables group, to export or import a tables groups configuration to or from a local file.

6.3.3 Some details about the user interface

The page headers contain:

- some information regarding the current connection,
- 3 links to reach the SQL statements editor, the history of submitted statements and to logout the current connection,
- a combo box to select the language used by the user interface,
- a breadcrumb trail,
- and a button allowing to directly go to the page bottom.

The user can navigate in Emaj_web functions using two icon bars: one for the general purpose functions and the other for the functions concerning a single tables group.

Figure 3 – Main icons bar.

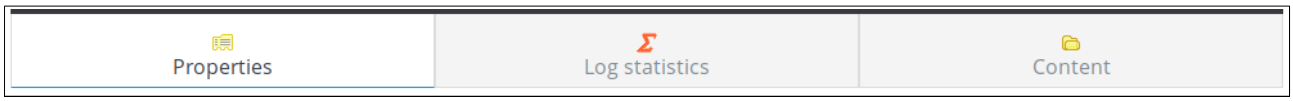


Figure 4 – Tables groups icons bar.

For *emaj_viewer* roles, some icons are not visible.

On some tables, it is possible to dynamically sort displayed rows, using small vertical arrows on the right of column titles.

On some tables too, an icon located at the left of the header row, let show or hide input fields that can be used to dynamically filter displayed rows.

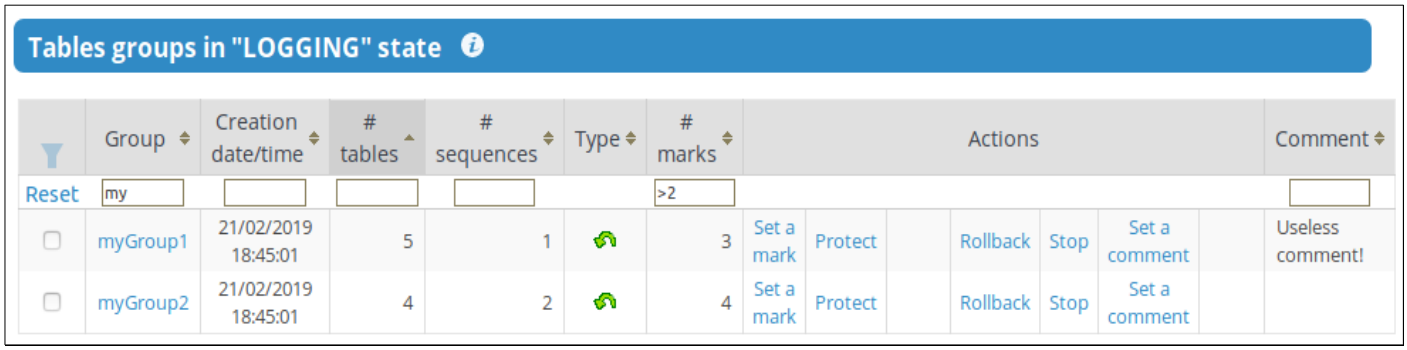


Figure 5 – Filtering the tables groups in logging state.
Here, only tables groups whose name contains "my" and having more than 2 marks are displayed, sorted in descending order by number of tables.

Some tables allow to perform actions on several objects at once. In this case, the user selects the objects with the checkboxes on the first column of the table and choose the action to perform among the available buttons under the table.

6.3.4 Tables group details

From the tables groups list page, it is possible to get more information about a particular tables group by clicking on its name. This page is also accessible with the "*Properties*" icon of the groups bar and through the left browsing tree.

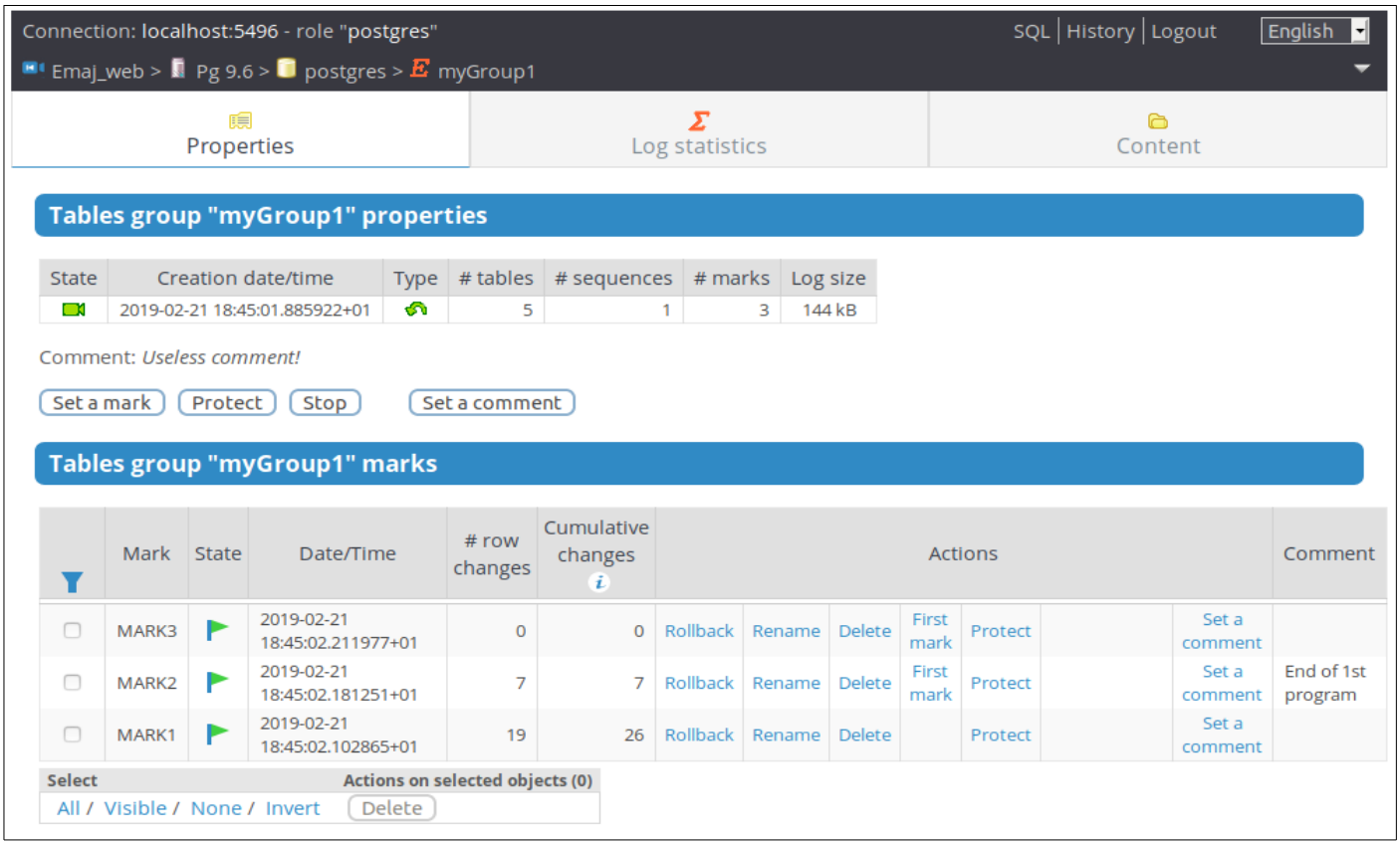


Figure *6 – Details of a tables group*

A first line repeats information already displayed on the groups list (number of tables and sequences, type, state and number of marks). It also shows the disk space used by its log tables.

This line is followed by the group's comment, if any has been recorded for this group.

Next is a set of buttons to execute actions depending on the group's state.

Then, the user can see the list of all marks that have been set on the group. For each of them, the following is displayed:

- its name,
- the date and time it has been set,
- its state (active or not, protected against rollback or not),
- the number of recorded log rows between this mark and the next one (or the current situation if this is the last set mark),
- the total number of recorded log rows from when the mark was set,
- the comment associated to the mark, if it exists.

For each mark, several buttons are available to perform the actions permitted by the mark's state.

6.3.5 Statistics

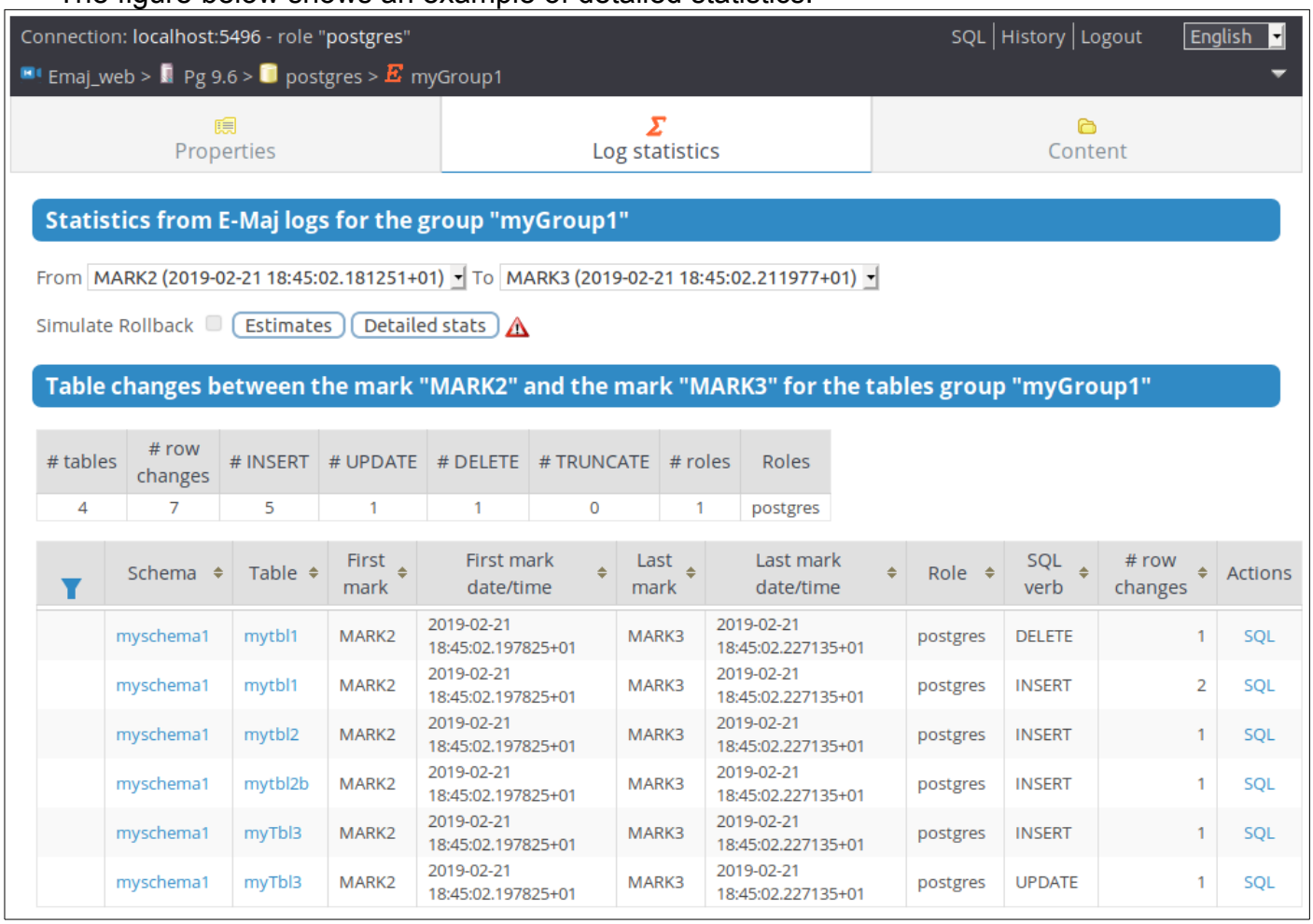
Using the "*Log statistics*" tab of the group's bar, one gets statistics about updates recorded into the log tables for the selected tables group.

Two types of statistics can be produced:

- some estimates about the number of updates per table, recorded between two marks or between one mark and the current situation,
- a precise numbering of updates per tables, per statement type (*INSERT*/*UPDATE*/*DELETE*/*TRUNCATE*) and role.

If the end of the range corresponds to the current situation, a check box allows one to request a rollback simulation to the selected mark in order to quickly get an approximate duration of this operation.

The figure below shows an example of detailed statistics.



# tables	# row changes	# INSERT	# UPDATE	# DELETE	# TRUNCATE	# roles	Roles
4	7	5	1	1	0	1	postgres

Schema	Table	First mark	First mark date/time	Last mark	Last mark date/time	Role	SQL verb	# row changes	Actions
myschema1	mytbl1	MARK2	2019-02-21 18:45:02.197825+01	MARK3	2019-02-21 18:45:02.227135+01	postgres	DELETE	1	SQL
myschema1	mytbl1	MARK2	2019-02-21 18:45:02.197825+01	MARK3	2019-02-21 18:45:02.227135+01	postgres	INSERT	2	SQL
myschema1	mytbl2	MARK2	2019-02-21 18:45:02.197825+01	MARK3	2019-02-21 18:45:02.227135+01	postgres	INSERT	1	SQL
myschema1	mytbl2b	MARK2	2019-02-21 18:45:02.197825+01	MARK3	2019-02-21 18:45:02.227135+01	postgres	INSERT	1	SQL
myschema1	myTbl3	MARK2	2019-02-21 18:45:02.197825+01	MARK3	2019-02-21 18:45:02.227135+01	postgres	INSERT	1	SQL
myschema1	myTbl3	MARK2	2019-02-21 18:45:02.197825+01	MARK3	2019-02-21 18:45:02.227135+01	postgres	UPDATE	1	SQL

Figure 7 – Detailed statistics about updates recorded between two marks

The displayed page contains a first line returning global counters.

On each line of the statistics table, the user can click on a "*SQL*" button to easily look at the log tables content. A click on this button opens the SQL editor window and proposes the statement displaying the content of the log table that corresponds to the selection (table, time frame, role, statement type). The user can modify this suggested statement before executing it to better fit his needs.

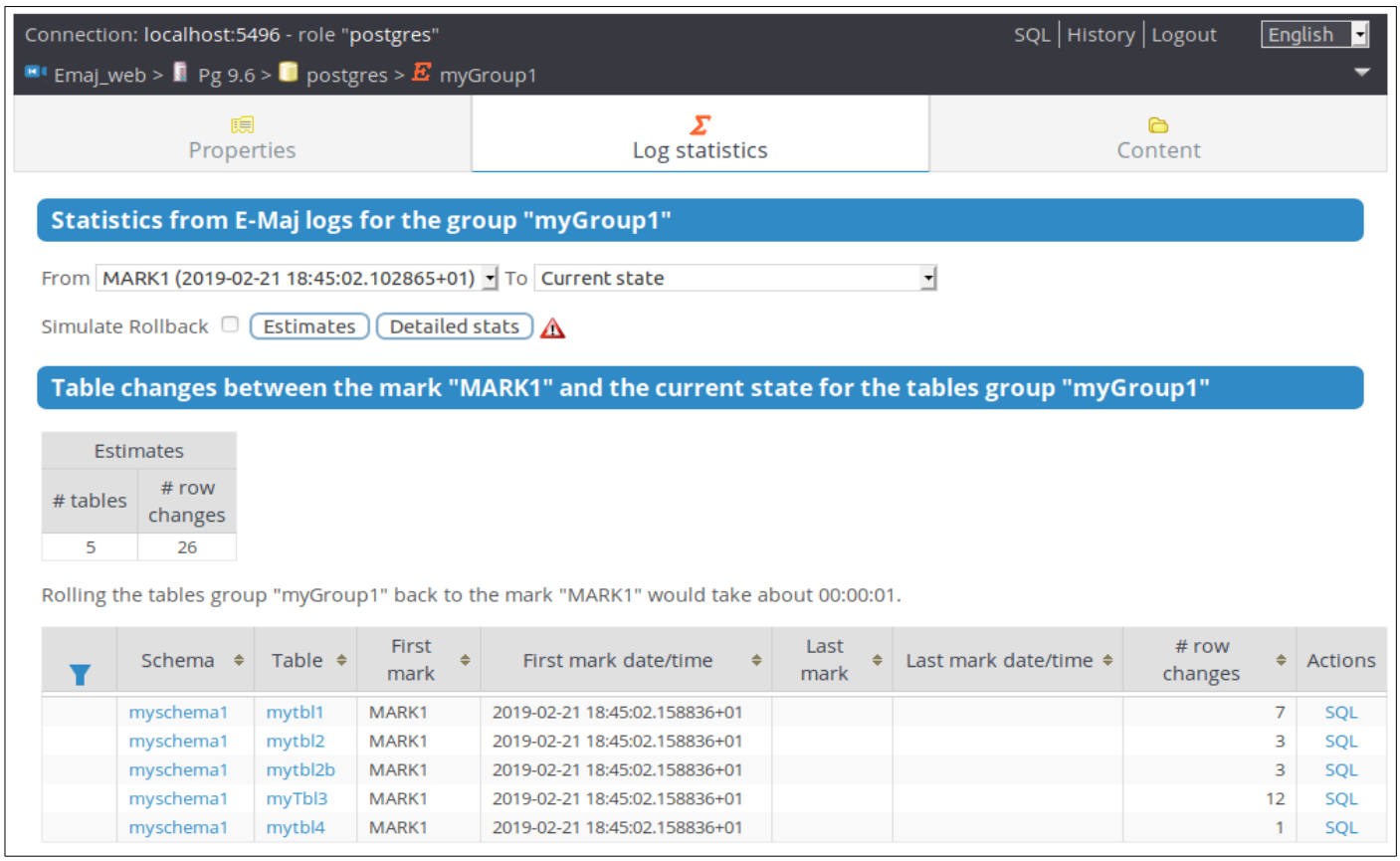


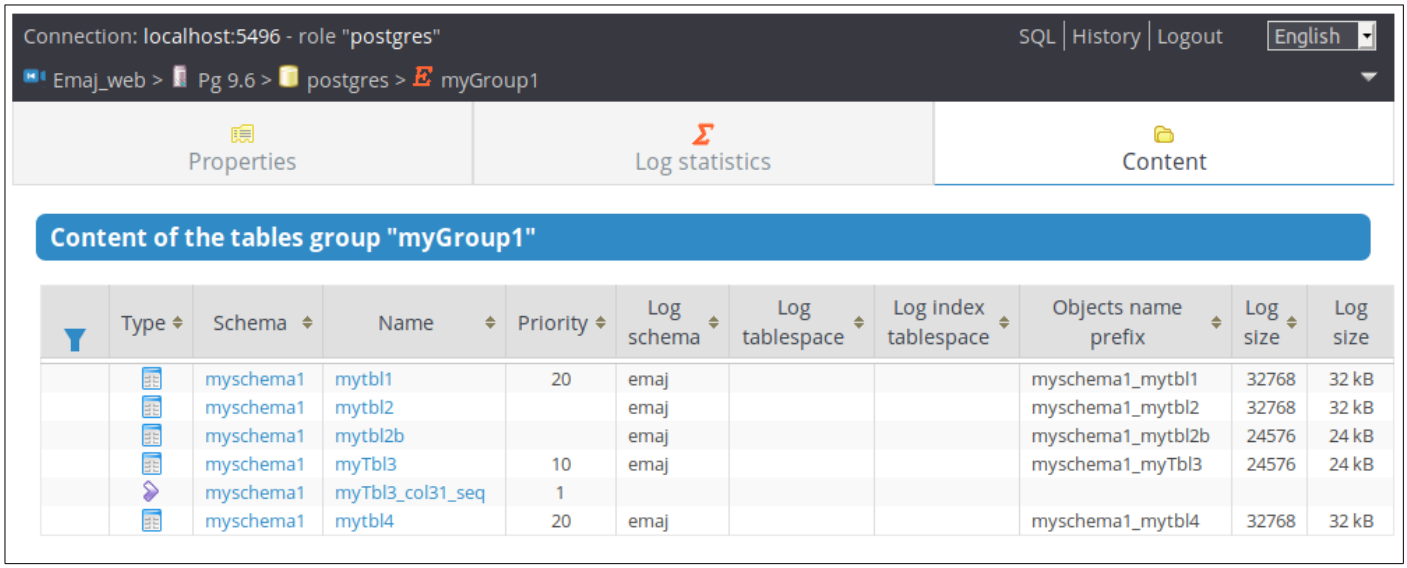
Figure 8 – Result of the rollback simulation, with the estimated number of updates for each table.

The displayed page contains a first part indicating the number of tables and sequences concerned by the rollback operation to this mark, and an estimate of the operation duration.

6.3.6 Tables group content

Using the "*Content*" tab of the group's bar, it is possible to get a summary of a tables group content.

For each table and sequence belonging to the group, the displayed table shows the characteristics configured into the *emaj_group_def* table, as well as the disk space used by the log table and its index.



Type	Schema	Name	Priority	Log schema	Log tablespace	Log index tablespace	Objects name prefix	Log size	Log size
	myschema1	mytbl1	20	emaj			myschema1_mytbl1	32768	32 kB
	myschema1	mytbl2		emaj			myschema1_mytbl2	32768	32 kB
	myschema1	mytbl2b		emaj			myschema1_mytbl2b	24576	24 kB
	myschema1	myTbl3	10	emaj			myschema1_myTbl3	24576	24 kB
	myschema1	myTbl3_col31_seq	1						
	myschema1	mytbl4	20	emaj			myschema1_mytbl4	32768	32 kB

Figure 9 – A tables group's content.

6.3.7 Schemas and tables groups configuration

The "*schemas*" tab displays the list of schemas contained in the database.

By selecting one of them, two additional lists are displayed: the tables and the sequences contained by this schema.

For both lists, the E-Maj properties and some general properties of each object become visible. Some action buttons allow to reach their structure or content and manage their assignment to tables groups.

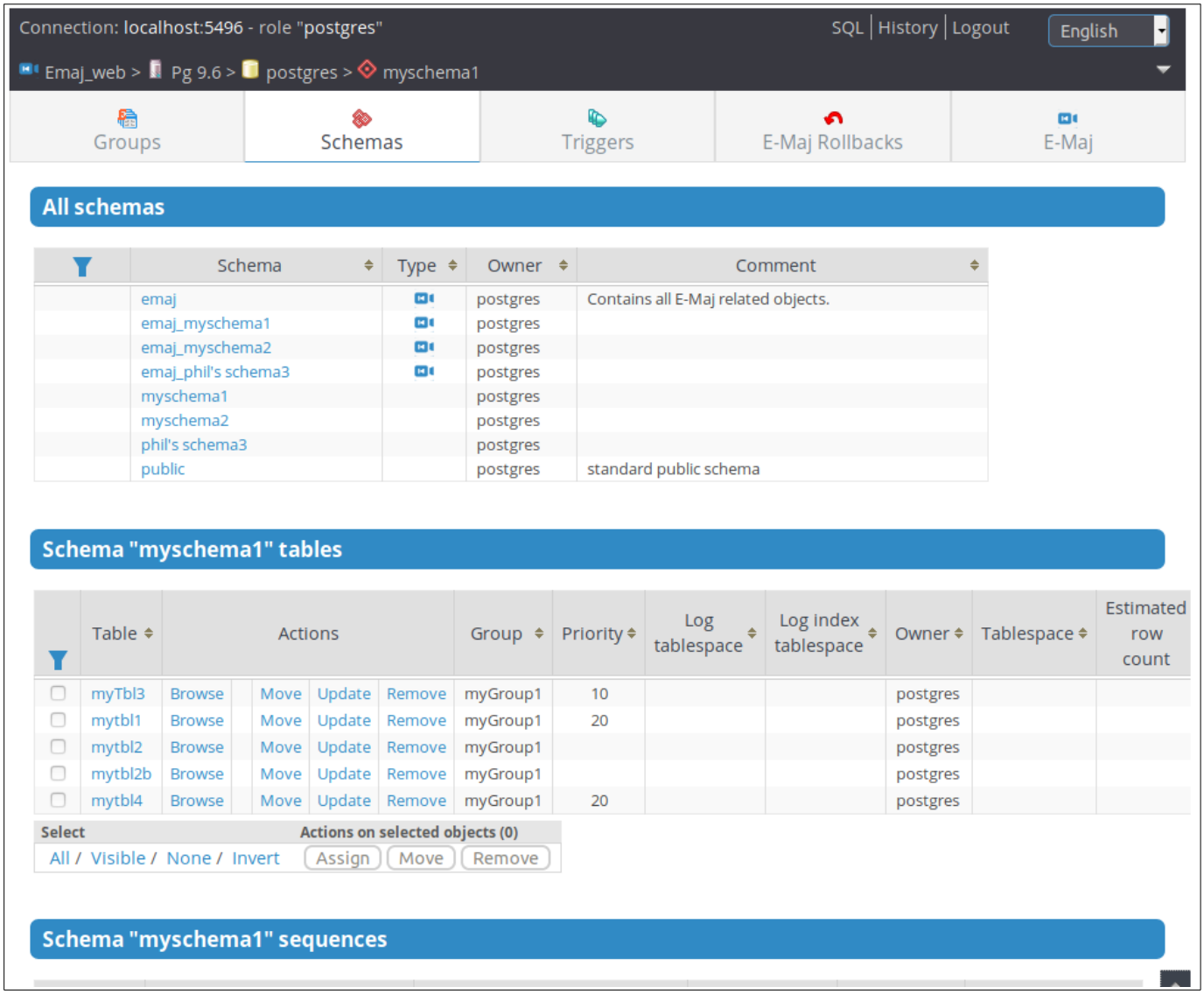


Figure 10 – Schema content and tables groups configuration.

6.3.7.1 Triggers

The "*Triggers*" tab lists the application triggers (those not linked to E-Maj), with their main characteristics.

A button allows to switch their de-activation mode at E-Maj rollback time.

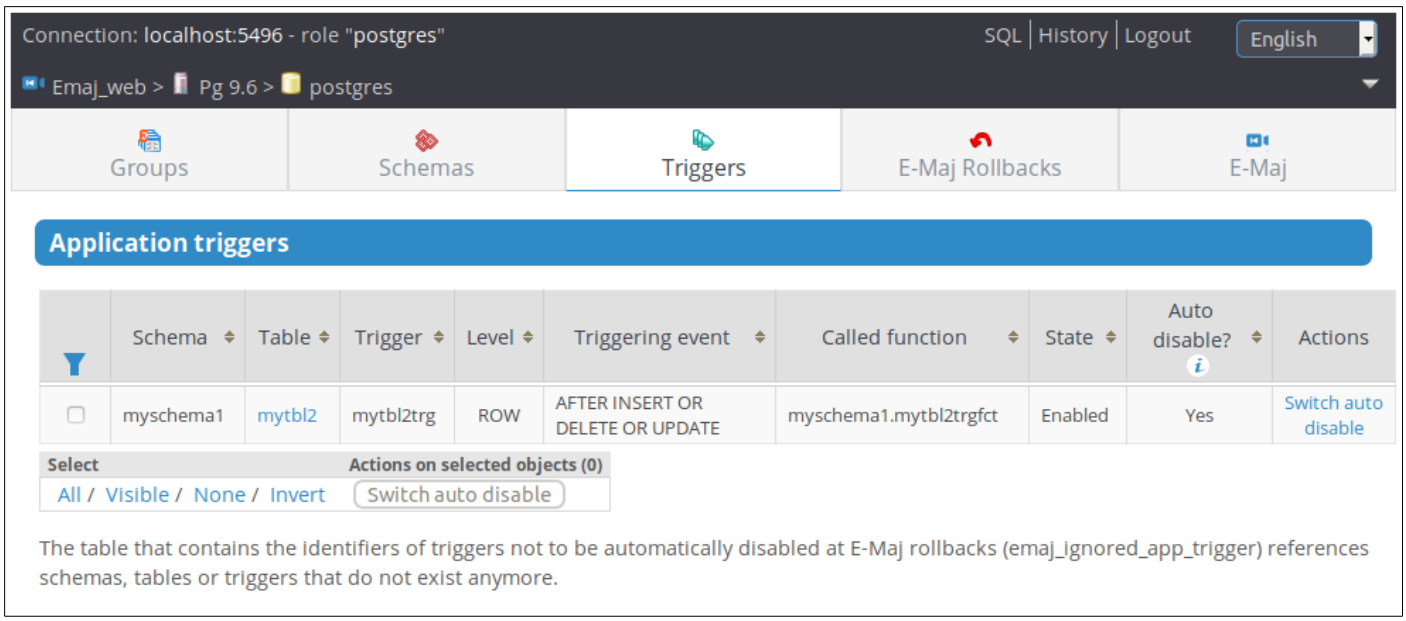


Figure 11 – Application triggers list.

6.3.8 Monitoring rollback operations

Using the "*Rollback operations*" tab of the main bar, users can monitor the rollback operations. Three different lists are displayed:

- in progress rollback operations, with the characteristics of the rollback operations and estimates of the percentage of the operation already done and of the remaining duration,
- the latest completed operations,
- logged rollback operations that are consolidable.

For completed operations, the user can filter on a more or less deep history.

For each consolidable rollback, a button allows to effectively consolidate the operation.

Connection: localhost:5496 - role "postgres" SQL | History | Logout English

Emaj_web > Pg 9.6 > postgres

Groups | Groups configuration | Rollback operations | Schemas | E-Maj

In progress E-Maj rollbacks

Rlbk Id.	Groups	State	Rollback start	Current duration	Estimated remaining	% completed	Target mark	Mark set at	Logged ?	Nb sessions	Nb tables to process	se to
4	myGroup2	LOCKING			00:00:00.044023	0	MARK1	2019-02-21 18:45:02.158836+01	Yes	1	2	

Completed E-Maj rollbacks

Display the : 3 most recent completed since less than 24 hours Filter

Rlbk Id.	Groups	State	Rollback start	Rollback end	Duration	Target mark	Mark set at	Logged ?	Nb sessions	Nb process table
3	myGroup2	COMMITTED	2019-02-21 18:45:02.378441+01	2019-02-21 18:45:02.414558+01	00:00:00	MARK3	2019-02-21 18:45:02.211977+01	Yes	1	
2	myGroup2	COMMITTED	2019-02-21 18:45:02.325197+01	2019-02-21 18:45:02.371898+01	00:00:00	tmp_mark	2019-02-21 18:45:02.23706+01	No	1	
1	myGroup2	COMMITTED	2019-02-21 18:45:02.257901+01	2019-02-21 18:45:02.322943+01	00:00:00	MARK1	2019-02-21 18:45:02.102865+01	Yes	1	

Consolidable E-Maj logged rollbacks

Group	Target mark	Mark set at	# row changes	Nb intermediate marks	End rollback mark	Mark set at	Actions
myGroup2	MARK3	2019-02-21 18:45:02.211977+01	6	1	RLBK_MARK3_18.45.02.395_DONE	2019-02-21 18:45:02.378441+01	Consolidate

Figure 12 – Rollback operations monitoring.

6.3.9 E-Maj environment state

By selecting the "*E-Maj*" tab of the main bar, the user reaches an overview of the E-Maj environment state.

First, some items are displayed:

- the installed PostgreSQL and E-Maj versions,
- the disk space used by E-Maj (log tables, technical tables and their indexes), and the part of the global database space it represents.

Then, the environment integrity is checked; the result of the *emaj_verify_all()* function execution is displayed.

The page ends with a list of the extension parameters value, be they present in the *emaj_param* table or set to their default value.

Two buttons allow to export and import parameters configurations to or from a local file.

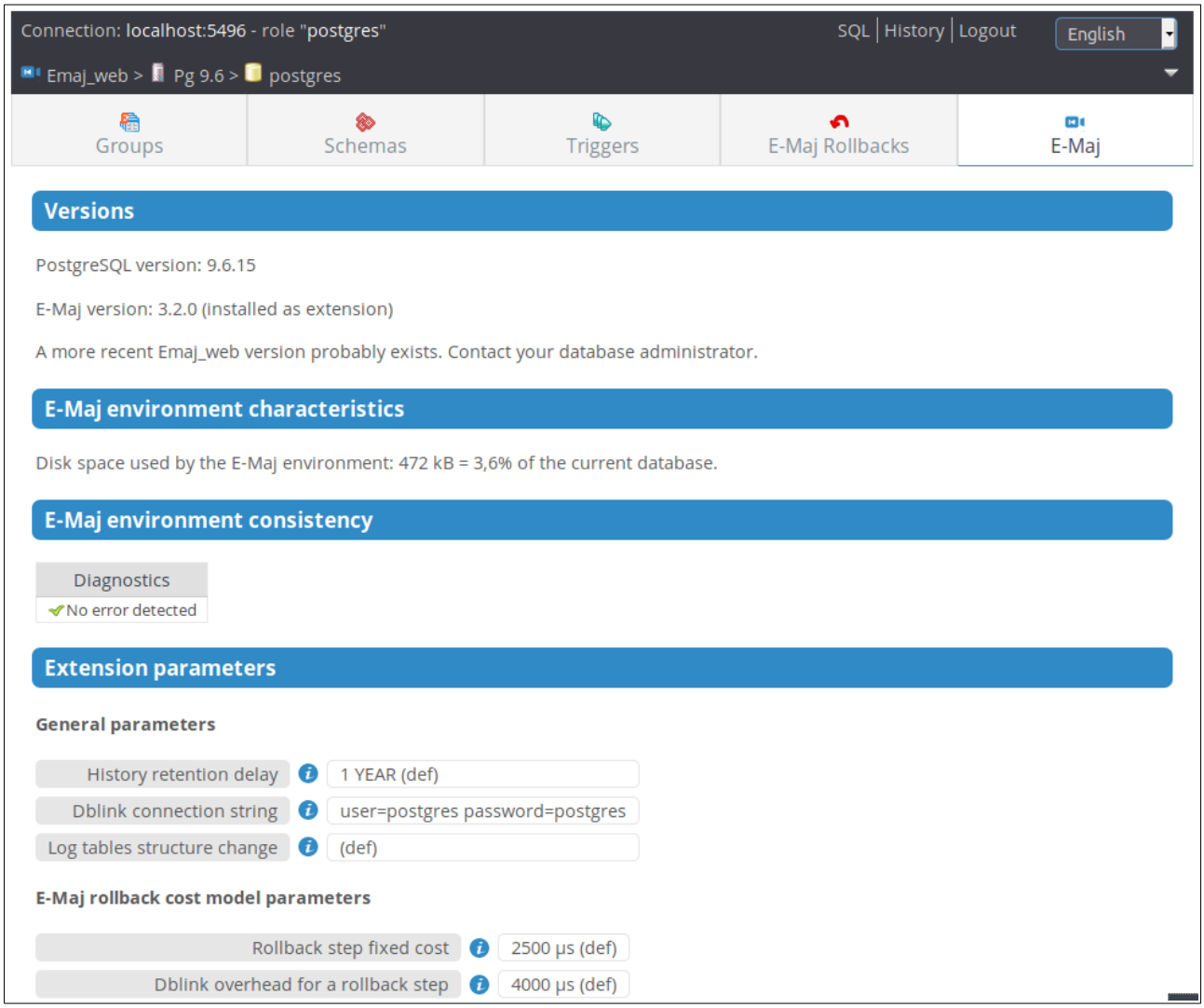


Figure 13 – The E-Maj environment state.

7 CONTRIBUTE TO THE E-MAJ DEVELOPMENT

Any contribution to the development and the improvement of the E-Maj extension is welcome. This chapter gives some information to make these contributions easier.

7.1 BUILD THE E-MAJ ENVIRONMENT

The E-Maj extension repository is hosted on the *github* site: *https://github.com/dalibo/emaj*

7.1.1 Clone the E-Maj repository

So the first acction to perform is to locally clone this repository on his/her own computer. This can be done by using the functionnalities of the *github* web interface or by typing the shell command:

```
git clone https://github.com/dalibo/emaj.git
```

7.1.2 Description of the E-Maj tree

So one has a full directory tree (except the web clients). It contains all directories and files described in the appendix 8.2, except the *doc* directory content that is separately maintained (see below).

The main directory also contains the following components:

- the *tar.index* file that is used to build the tarball of the E-Maj version distributed on *pgxn.org*
- the *docs* directory with all sources of the online documentation (see §7.4)
- in the *sql* directory:
  - the file *emaj--devel.sql*, source of the extension in its current version
  - the source of the previous version *emaj--<previous_version>.sql*
  - a *emaj_prepare_emaj_web_test.sql* script that prepares an E-Maj environment to test the *Emaj_web* client
- a *test* directory containing all components used to test the extension (see §7.3)
- a *tools* directory containing some … tools.

7.1.3 Setting tools parameters

The tools stored in the *tools* directory need some parameters to be set, depending on his/her own environment. A parameter system covers some tools. For the others, the *tools/README* file details the changes to apply.

7.1.3.1 Créating the emaj_tools.env file

The parameters that may be modified are grouped into the *tools/emaj_tools.env* file, which is called by *tools/emaj_tools.profile*.

The repository contains a file *tools/emaj_tools.env-dist* that may be used as a template to create the *emaj_tools.env* file.

The *emaj_tools.env* file must contain:

- the list of PostgreSQL versions that are supported by the current E-Maj version and for which a PostgreSQL instance exists for tests (EMAJ_USER_PGVER variable),
- for each PostgreSQL version used for the tests, 6 variables describing the location of binaries, the main directory of the related instance, the role and the ip-port to be used for the connection to the instance.

7.2 CODING

7.2.1 Versionning

The version currently under development is named *devel*.

Regularly and when it is justified, a new version is created. Its name has a X.Y.Z pattern.

The *tools/create_version.sh* shell script assists in creating this version. It is only used by the E-Maj maintainers. So its use is not described here.

7.2.2 Coding rules

Coding the *emaj--devel.sql* script must follow these rules:

- script structure: after some checks about the execution conditions that must be met, the objects are created in the following order: roles, enumerated types, sequences, tables (with their indexes and contraints), composite types, E-Maj parameters, low level functions, elementary functions that manage tables and sequences, functions that manage tables groups, general purpose functions, event triggers, grants, additional actions for the extensions. The script ends with some final operations.
- all objects are created in the *emaj* schema, except the *_emaj_protection_event_trigger_fnct()* function, created in the *public* schema,
- tables and sequences names are prefixed by '*emaj_*'
- functions names are prefixed by '*emaj_*' when they are usable by end users, or by '_' for internal functions,
- the internal tables and the functions callable by end users must have a comment,
- the language keywords are in upper case, objects names are in lower case,
- the code is indented with 2 space characters,

- lines must not contain tab characters, must not be longer than 140 characters long and must not end with spaces,
- in the functions structure, the code delimiters must contain the function name surrounded with a $ character (or *do* for code blocks),
- variables names are prefixed with '*v_*' for simple variables or '*r_*' for *RECORD* type variables,
- the code must be compatible with all PostgreSQL versions supported by the current E-Maj version. When this is striclty necessary, the code may be differenciated depending on the PostgreSQL version.

A perl script, *tools/check_code.pl* performs some checks on the code format of the script that creates the extension. It also detects unused variables. This script is directly called in non-regression tests scenarios (Cf §7.3).

7.2.3 Version upgrade script

E-Maj is installed into a database as an extension. The E-Maj administrator must be able to easily upgrade the extension version (cf §3.4). So an upgrade script is provided for each version, that upgrades from the previous version to the next version. It is named *emaj--<previous_version>--devel.sql*.

The development of this script follows these rules:

- Develop/maintain the upgrade script at the same time as the main *emaj--devel.sql* script, so that the tests of a change include upgrade version cases,
- Apply the same coding rules as for the main script,
- As far as possible, ensure that the upgrade operation is able to process tables groups in logging state, without loosing the capability to perform E-Maj rollbacks on marks set prior the version upgrade.

At the beginning of a version, the upgrade script is built using a template (the file *tools/emaj_upgrade.template)*.

As the development goes on, a perl script helps to synchronize the creation/deletion/replacement of functions. It compares the *emaj--devel.sql* script and the script that creates the previous version and updates the *emaj--<previous_version>--devel.sql* script. To let it work properly, it is essential to keep both tags that frame the part of the script that describes functions.

After having adapted the parameters (see the *TOOLS/README* file), just submit:

```
perl tools/sync_fct_in_upgrade_script.pl
```

The other parts of the script must be coded manually. If the structure of an internal table is changed, the table content must be migrated (scripts for prior version upgrade can be used as examples).

7.3 TESTING

Through the rollback functions, the E-Maj extension updates database content. So the reliability is a key characteristics. For this reason, it is essential to pay a great attention to the tests.

7.3.1 Create PostgreSQL instances

The ideal is to be able to test E-Maj with all PostgreSQL versions that are supported by the extension (currently from version 9.5 to version 12).

The *tools/create_cluster.sh* script helps in creating a test instance. Its content may show the characteristics of the instance to create. It can also be executed (after parameters setting as indicated in *tools/README*):

```
tools/create_cluster.sh <PostgreSQL_major_version>
```

7.3.2 Install software dependancies

Testing the clients may require to install some additional software components:

- the *php* software, with its PostgreSQL interface,
- the *perl* software, with the *DBI* and *DBD::Pg* modules.

7.3.3 Execute non regression tests

A solid test environment is supplied in the repository. It contains:

- a test tool,
- test scenarios,
- expected results.

7.3.3.1 The test scenarios

The test system contains 4 scenarios:

- a full standart scenario,
- the same scenario but installing the extension from the previous version with an immediate upgrade into the current version,
- the same scenario but installing the extension with the *emaj-devel.sql* script provided for cases when a “CREATE EXTENSION emaj” statement is not possible,
- a shorter scenario but with an upgrade into the current version while tables groups are in logging state.

These scenarios call *psql* scripts, all located into the *test/sql* directory. The scripts chain E-Maj function calls in different contexts, and SQL statements to prepare or check the results.

At the end of scripts, internal sequences are often reset, so that a single function call insertion does not produce impacts in the next scripts results.

The *psql* test scripts must be maintained in the same time as the extension source.

7.3.3.2 The expected results

For each *psql* script, the test tool produces a result file. These files are distinguished from a PostgreSQL version to another. They are located in the *test/<PostgreSQL_version>/results* directory.

At the end of a run, the test tool compares these files with a reference located into the *test/<PostgreSQL_version>/expected* directory.

Unlike for files in the *test/<PostgreSQL_version>/results* directory, files in the *test/<PostgreSQL_version>/expected* directory belong to the git repository. They must always remain consistent with the source of the extension and the *psql* test scripts.

7.3.3.3 The test tool

The test tool, *regress.sh*, combines all test functions.

Before using it, it is necessary to:

- have the PostgrSQL instances to be used already created and the *tools/emaj_tools.env* file already setup
- manually create the *test/<PostgreSQL_version>/results* directories.

The test tool can be launched with the command:

```
tools/regress.sh
```

As it starts with a copy of the *emaj.control* file into the *SHAREDIR/extension* directory of each configured PostgreSQL version, it may ask for the password of the Linux account to be able to execute *sudo* commands. It also automatically generates the *emaj-devel.sql* script used to create the extension with psql.

It then displays the list of test functions in a menu. Just enter the letter corresponding to the choosen test.

The test functions are:

- standart tests for each configured PostgreSQL version,

- the tests with the installation of the previous version followed by an upgrade,
- the tests with the installation of the version with the *emaj-devel.sql* script,
- the tests with an E-Maj version upgrade while tables groups are in logging state,
- tests chaining a database save with *pg_dump* and a restore, with different PostgreSQL versions,
- a PostgreSQL upgrade version test using *pg_upgrade* with a database containing the E-Maj extension.

It is important to execute the four first sets of tests for each E-Maj change.

7.3.3.4 Validate results

After having executed a *psql* script, *regress.sh* compares the outputs of the run with the expected outputs and reports the comparison result with the words '*ok*' or 'FAILED'.

Here is an example of the display issued by the test tool (in this case with the scenario chaining the installation and a version upgrade, and with a detected difference):

```
Run regression test
============== dropping database "regression"         ==============
DROP DATABASE
============== creating database "regression"         ==============
CREATE DATABASE
ALTER DATABASE
============== running regression test queries        ==============
test install_upgrade          ... ok
test setup                    ... ok
test create_drop              ... ok
test start_stop               ... ok
test mark                     ... ok
test rollback                 ... ok
test misc                     ... ok
test alter                    ... ok
test alter_logging            ... ok
test viewer                   ... ok
test adm1                     ... ok
test adm2                     ... ok
test adm3                     ... ok
test client                   ... ok
test check                    ... FAILED
test cleanup                  ... ok

=======================
 1 of 15 tests failed.
=======================

The differences that caused some tests to fail can be viewed in the
file "/home/postgres/proj/emaj/test/11/regression.diffs".  A copy of the test summary
that you see
above is saved in the file "/home/postgres/proj/emaj/test/11/regression.out".
```

When at least one script fails, it is important to closely analyze the differences, by reviewing the *test/<PostgreSQL_version>/regression.diffs* file content, and check that

the differences are directly linked to changes applied in the extension source code or in the test scripts.

Once the reported differences are considered as valid, the content of the *test/<PostgreSQL_version>/result* directories must be copied into the *test/<PostgreSQL_version>/expected* directories. A shell script processes all PostgreSQL versions in a single command:

```
sh tools/copy2Expected.sh
```

It may happen that some test outputs do not match the expected outputs, due to differences in the PostgreSQL behaviour from one run to another. Repeating the test allows to check these cases.

7.3.4 Test coverage

7.3.4.1 Functions test coverage

The PostgreSQL test instances are configured to count the functions executions. The *check.sql* test script displays the functions execution counters. It also displays E-Maj functions that have not been executed.

7.3.4.2 Error messages test coverage

A perl script extracts error and warning messages coded in the *sql/emaj--devel.sql* file. It then extracts the messages from the files of the *test/10/expected* directory. It finally displays error or warning messages that are not covered by tests.

The script can be run with the command:

```
perl tools/check_error_messages.pl
```

Some messages are known to not be covered by tests (for instance internal errors that are hard to reproduce). These messages, coded in the perl script, are excluded from the final report.

7.3.5 Evaluate the performances

The *tools/performance* directory contains some shell scripts helping in measuring performances. As the measurement results totally depend on the platform and the environment used, no reference results are supplied.

The scripts cover the following domains:

- *log_overhead/pgbench.sh* evaluates the log mechanism overhead, using pgbench,
- *large_group/large_group.sh* evaluates the behaviour of groups containing a large number of tables,
- *rollback/rollback_perf.sh* evaluates the E-Maj rollback performances with different tables profiles.

For all these files, some variables have to be configured at the begining of the scripts.

7.4 DOCUMENTING

A *LibreOffice* format documentation is managed by the maintainers. It has its own *github* reporistory: *emaj_doc*. Thus the *doc* directory of the main repository remains empty.

The online documentation is managed by *sphinx*. It is located in the *docs* directory.

To install *sphinx*, refer to the *docs/README.rst* file.

The documentation exists in two languages, English and French. Depending on the languages, document sources are located in */docs/en* and */docs/fr*. These documents are in *ReStructured Text* format.

To compile the documentation for a language, set the current directory to *docs/<language>* and execute the command:

```
make html
```

When there is no compilation error anymore, the documentation becomes available locally on a brower, by opening the *docs/<language>/_build/html/index.html* file.

The documentation on the *readthedocs.org* site is automatically updated as soon as the main *github* repository is updated.

7.5 SUBMITTING A PATCH

Patches can be proposed to the E-Maj maintainers through *Pull Requests* on the *github* site.

Before submitting a patch, it may be useful to create an *issue* on *github*, in order to start a discussion with the maintainers and help in working on the patch.

8 APPENDIX

8.1 E-MAJ FUNCTIONS LIST

The E-Maj functions that are available to users can be grouped into 3 categories. They are listed below, in alphabetic order. They are all callable by roles having *emaj_adm* privileges. The charts also specify those callable by *emaj_viewer* roles.

8.1.1 Tables or sequences level functions

Functions	Input parameters	Output data	Callable by emaj_viewer	Ref.
emaj_assign_sequence	schema TEXT sequence TEXT group TEXT [properties JSONB] [mark TEXT]	1 INT		§ 4.4.2
emaj_assign_sequences	schema TEXT sequences.array TEXT[] group TEXT [properties JSONB] [mark TEXT]	#.séquences INT		§ 4.4.2
emaj_assign_sequences	schema TEXT sequences.to.include.filter TEXT sequences.to.exclude.filter TEXT group TEXT [properties JSONB] [mark TEXT]	#.séquences INT		§ 4.4.2
emaj_assign_table	schema TEXT table TEXT groupe TEXT [properties JSONB] [mark TEXT]	1 INT		§ 4.4.2
emaj_assign_tables	schema TEXT tables.array TEXT[] group TEXT [properties JSONB] [mark TEXT]	#.tables INT		§ 4.4.2
emaj_assign_tables	schema TEXT tables.to.include.filter TEXT tables.to.exclude.filter TEXT group TEXT [properties JSONB] [mark TEXT]	#.tables INT		§ 4.4.2
emaj_get_current_log_table	schema TEXT	(log.schema TEXT,	Yes	§ 4.9.3

Functions	Input parameters	Output data	Callable by emaj_viewer	Ref.
	table TEXT	log.table TEXT)		
emaj_ignore_app_trigger	action TEXT schema TEXT table TEXT trigger TEXT	#.triggers INT		§ 4.9.4
emaj_modify_table	schema TEXT table TEXT properties JSONB [mark TEXT]	#.tables INT		§ 4.4.2
emaj_modify_tables	schema TEXT tables.array TEXT[] properties JSONB [mark TEXT]	#.tables INT		§ 4.4.2
emaj_modify_tables	schema TEXT tables.to.include.filter TEXT tables.to.exclude.filter TEXT properties JSONB [mark TEXT]	#.tables INT		§ 4.4.2
emaj_move_sequence	schema TEXT sequence TEXT new.group TEXT [mark TEXT]	1 INT		§ 4.4.2
emaj_move_sequences	schema TEXT sequences.array TEXT[] new.group TEXT [mark TEXT]	#.sequences INT		§ 4.4.2
emaj_move_sequences	schema TEXT sequences.to.include.filter TEXT sequences.to.exclude.filter TEXT new.group TEXT [mark TEXT]	#.sequences INT		§ 4.4.2
emaj_move_table	schema TEXT table TEXT new.group TEXT [mark TEXT]	1 INT		§ 4.4.2
emaj_move_tables	schema TEXT tables.array TEXT[] new.group TEXT [mark TEXT]	#.tables INT		§ 4.4.2
emaj_move_tables	schema TEXT tables.to.include.filter TEXT tables.to.exclude.filter TEXT new.group TEXT [mark TEXT]	#.tables INT		§ 4.4.2
emaj_remove_sequence	schema TEXT	1 INT		§ 4.4.2

Functions	Input parameters	Output data	Callable by emaj_viewer	Ref.
	sequence TEXT [mark TEXT]			
emaj_remove_sequences	schema TEXT sequences.array TEXT[] [mark TEXT]	#.sequences INT		§ 4.4.2
emaj_remove_sequences	schema TEXT sequences.to.include.filter TEXT sequences.to.exclude.filter TEXT [mark TEXT]	#.sequences INT		§ 4.4.2
emaj_remove_table	schema TEXT table TEXT [mark TEXT]	1 INT		§ 4.4.2
emaj_remove_tables	schema TEXT tables.array TEXT[] [mark TEXT]	#.tables INT		§ 4.4.2
emaj_remove_tables	schema TEXT tables.to.include.filter TEXT tables.to.exclude.filter TEXT [mark TEXT]	#.tables INT		§ 4.4.2

8.1.2 Groups level functions

Functions	Input parameters	Output data	Callable by emaj_viewer	Ref.
emaj_alter_group	group TEXT [mark TEXT]	# tables.and.seq INT		§ 4.4
emaj_alter_groups	groups.array TEXT [mark TEXT]	# tables.and.seq INT		§ 4.4
emaj_comment_group	group TEXT comment TEXT	-		§ 4.5.2
emaj_comment_mark_group	group TEXT mark TEXT comment TEXT	-		§ 4.6.1
emaj_consolidate_rollback_group	group TEXT end.rollback.mark TEXT	# tables.and.seq INT		§4.5.6
emaj_create_group	group TEXT [is.rollbackable BOOLEAN] [is.empty BOOLEAN]	#.tables.and.seq INT		§ 4.2.2
emaj_delete_before_mark_group	group TEXT mark TEXT	#.deleted.marks INT		§ 4.6.5
emaj_delete_mark_group	group TEXT mark TEXT	1 INT		§ 4.6.4

Functions	Input parameters	Output data	Callable by emaj_viewer	Ref.
emaj_detailed_log_stat_group	group TEXT start.mark TEXT end.markTEXT	SETOF emaj_detailed_log_stat_type	Yes	§ 4.7.2
emaj_detailed_log_stat_groups	groups.array TEXT[] start.mark TEXT end.mark TEXT	SETOF emaj_log_stat_type	Yes	§ 4.7.2
emaj_drop_group	group TEXT	#.tables.and.seq INT		§ 4.2.3
emaj_estimate_rollback_group	group TEXT mark TEXT	duration INTERVAL	Yes	§ 4.7.3
emaj_estimate_rollback_groups	groups.array TEXT[] mark TEXT	duration INTERVAL	Yes	§ 4.7.3
emaj_force_drop_group	group TEXT	#.tables.and.seq INT		§4.5.5
emaj_force_stop_group	group TEXT	#.tables.and.seq INT		§ 4.5.4
emaj_gen_sql_group	group TEXT start.mark TEXT end.mark TEXT output.file.path TEXT [tables.seq.array TEXT[]]	#.gen.statements BIGINT		§ 4.8.3
emaj_gen_sql_groups	groups.array TEXT[] start.mark TEXT end.mark TEXT output.file.path TEXT [tables.seq.array TEXT[]]	#.gen.statements BIGINT		§ 4.8.3
emaj_get_previous_mark_group	group TEXT date.time TIMESTAMPTZ	mark TEXT	Yes	§ 4.6.2
emaj_get_previous_mark_group	group TEXT mark TEXT	mark TEXT	Yes	§ 4.6.2
emaj_log_stat_group	group TEXT start.mark TEXT end.mark TEXT	SETOF emaj_log_stat_type	Yes	§ 4.7.1
emaj_log_stat_groups	groups.array TEXT[] start.mark TEXT end.mark TEXT	SETOF emaj_log_stat_type	Yes	§ 4.7.1
emaj_logged_rollback_group	group TEXT mark TEXT is.alter.group.allowed BOOLEAN	SETOF (severity TEXT, message TEXT)		§ 4.3.5
emaj_logged_rollback_groups	groups.array TEXT[] mark TEXT is.alter.group.allowed BOOLEAN	SETOF (severity TEXT, message TEXT)		§ 4.3.5
emaj_protect_group	group TEXT	0/1 INT		§ 4.5.3
emaj_protect_mark_group	group TEXT	0/1 INT		§ 4.6.6

Functions	Input parameters	Output data	Callable by emaj_viewer	Ref.
	mark TEXT			
emaj_rename_mark_group	group TEXT mark TEXT new.name TEXT	-		§ 4.6.3
emaj_reset_group	group TEXT	#.tables.and.seq INT		§ 4.5.1
emaj_rollback_group	group TEXT mark TEXT is_alter_group_allowed BOOLEAN	SETOF (severity TEXT, message TEXT)		§ 4.3.4
emaj_rollback_groups	groups.array TEXT[] mark TEXT is_alter_group_allowed BOOLEAN	SETOF (severity TEXT, message TEXT)		§ 4.3.4
emaj_set_mark_group	group TEXT [mark TEXT]	#.tables.and.seq INT		§ 4.3.3
emaj_set_mark_groups	groups.array TEXT[] [mark TEXT]	#.tables.and.seq INT		§ 4.3.3
emaj_snap_group	group TEXT directory TEXT copy.options TEXT	#.tables.and.seq INT		§ 4.8.1
emaj_snap_log_group	group TEXT start.mark TEXT end.mark TEXT directory TEXT copy.options TEXT	#.tables.and.seq INT		§ 4.8.2
emaj_start_group	group TEXT [mark TEXT] [reset.loge BOOLEAN]	#.tables.and.seq INT		§ 4.3.2
emaj_start_groups	groups.array TEXT[] [mark TEXT] [reset.loge BOOLEAN]	#.tables.and.seq INT		§ 4.3.2
emaj_stop_group	group TEXT [mark TEXT]	#.tables.and.seq INT		§ 4.3.6
emaj_stop_groups	groups.array TEXT[] [mark TEXT]	#.tables.and.seq INT		§ 4.3.6
emaj_sync_def_group	group TEXT	#.tables.and.seq INT		§ 4.4.6
emaj_unprotect_group	group TEXT	0/1 INT		§ 4.5.3
emaj_unprotect_mark_group	group TEXT mark TEXT	0/1 INT		§ 4.6.6

8.1.3 General purpose functions

Functions	Input parameters	Output data	Callable by emaj_viewer	Ref.
emaj_cleanup_rollback_state	-	# rollback INT		§ 14.9.6
emaj_disable_protection_by_event_triggers	-	# triggers INT		§ 4.9.7
emaj_enable_protection_by_event_triggers	-	# triggers INT		§ 4.9.7
emaj_export_groups_configuration	NULL [, groups.array TEXT[]]	configuration JSON		§ 4.5.8.1
emaj_export_groups_configuration	file TEXT [, groups.array TEXT[]]	#.groups INT		§ 4.5.8.1
emaj_export_parameters_configuration	-	parameters JSON		§4.9.2.1
emaj_export_parameters_configuration	file.path TEXT	# parameters INT		§ 4.9.2.1
emaj_get_consolidable_rollbacks	-	SETOF emaj_consolidable_rollback_type	Yes	§ 4.5.7
emaj_import_groups_configuration	groups JSON [, groups.array TEXT[]] [, alter.logging.groups BOOLEAN]	#.groups INT		§ 4.5.8.1
emaj_import_groups_configuration	file TEXT [, groups.array TEXT[]] [, alter.logging.groups BOOLEAN]	#.groups INT		§ 4.5.8.1
emaj_import_parameters_configuration	parameters JSON, [delete.current.conf BOOLEAN)]	# parameters INT		§ 4.9.2.2
emaj_import_parameters_configuration	file.path TEXT, [delete.current.conf BOOLEAN)]	# parameters INT		§ 4.9.2.2
emaj_rollback_activity	-	SETOF emaj_rollback_activity_type	Yes	§ 4.9.5
emaj_verify_all	-	SETOF TEXT	Yes	§ 4.9.1

8.2 E-MAJ DISTRIBUTION CONTENT

Once installed (see §3), an E-Maj version contents the following files.

sql/emaj--<version>.sql	installation script of the extension
sql/emaj-<version>.sql	alternate psql installation script
sql/emaj--3.2.0--3.3.0.sql	extension upgrade script from 3.2.0 to 3.3.0
sql/emaj--3.1.0--3.2.0.sql	extension upgrade script from 3.1.0 to 3.2.0
sql/emaj--3.0.1--3.1.0.sql	extension upgrade script from 3.0.0 to 3.1.0
sql/emaj--2.3.1--3.0.0.sql	extension upgrade script from 2.3.1 to 3.0.0
sql/emaj--2.3.0--2.3.1.sql	extension upgrade script from 2.3.0 to 2.3.1
sql/emaj--2.2.3--2.3.0.sql	extension upgrade script from 2.2.3 to 2.3.0
sql/emaj--2.2.2--2.2.3.sql	extension upgrade script from 2.2.2 to 2.2.3
sql/emaj--2.2.1--2.2.2.sql	extension upgrade script from 2.2.1 to 2.2.2
sql/emaj--2.2.0--2.2.1.sql	extension upgrade script from 2.2.0 to 2.2.1
sql/emaj--2.1.0--2.2.0.sql	extension upgrade script from 2.1.0 to 2.2.0
sql/emaj--2.0.1--2.1.0.sql	extension upgrade script from 2.0.1 to 2.1.0
sql/emaj--2.0.0--2.0.1.sql	extension upgrade script from 2.0.0 to 2.0.1
sql/emaj--1.3.1--2.0.0.sql	extension upgrade script from 1.3.1 to 2.0.0
sql/emaj--unpackaged--1.3.1.sql	script that transforms an existing 1.3.1 version into extension
sql/emaj-1.3.0-to-1.3.1.sql	psql script that upgrades a 1.3.0 version
sql/emaj-1.2.0-to-1.3.0.sql	psql script that upgrades a 1.2.0 version
sql/emaj-1.1.0-to-1.2.0.sql	psql script that upgrades a 1.1.0 version
sql/emaj-1.0.2-to-1.1.0.sql	psql script that upgrades a 1.0.2 version
sql/emaj-1.0.1-to-1.0.2.sql	psql script that upgrades a 1.0.1 version
sql/emaj-1.0.0-to-1.0.1.sql	psql script that upgrades a 1.0.0 version
sql/emaj-0.11.1-to-1.0.0.sql	psql script that upgrades a 0.11.1 version
sql/emaj-0.11.0-to-0.11.1.sql	psql script that upgrades a 0.11.0 version
sql/emaj_demo.sql	psql E-Maj demonstration script
sql/emaj_prepare_parallel_rollback_test.sql	psql test script for parallel rollbacks
sql/emaj_uninstall.sql	psql script to uninstall the E-Maj components
README.md	reduced extension's documentation
CHANGES.md	change log
LICENSE	information about E-Maj license

AUTHORS.md	who are the authors
META.json	technical data for PGXN
emaj.control	extension control flle used by the integrated extensions management
doc/Emaj.<version>_doc_en.pdf	English version of the full E-Maj documentation
doc/Emaj.<version>_doc_fr.pdf	French version of the full E-Maj documentation
doc/Emaj.<version>_pres_en.odp	English version of the E-Maj presentation
doc/Emaj.<version>_pres_fr.odp	French version of the E-Maj presentation
doc/Emaj.<version>_pres_en.pdf	English version of the E-Maj presentation (pdf version)
doc/Emaj.<version>_pres_fr.pdf	French version of the E-Maj presentation (pdf version)
client/emajParallelRollback.php	php tool for parallel rollback
client/emajParallelRollback.pl	perl tool for parallel rollback
client/emajRollbackMonitor.php	php tool for rollbacks monitoring
client/emajRollbackMonitor.pl	perl tool for rollbacks monitoring

8.3 POSTGRESQL AND E-MAJ VERSIONS COMPATIBILITY MATRIX

PostgreSQL versions		E-Maj versions	
Min	**Max**	**Min**	**Date**
8.2	8.4	0.4.0	01/2010
8.2	9.0	0.8.0	10/2010
8.2	9.1	0.10.0	11/2011
8.2	9.2	0.11.1	07/2012
8.3	9.3	1.1.0	10/2013
8.3	9.5	1.2.0	01/2016
8.3	9.6	1.3.1	09/2016
9.1	9.6	2.0.0	11/2016
9.1	10	2.1.0	08/2017
9.2	10	2.3.0	07/2018
9.2	11	2.3.1	09/2018
9.5	11	3.0.0	03/2019
9.5	12	3.1.0	06/2019