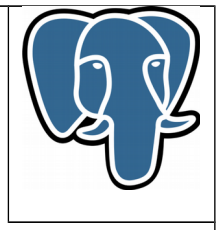


« E-Maj » PostgreSQL extension

-

Reference Guide

Version: 2.1.0

Last update: 02 August 2017

Summary

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 the 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.

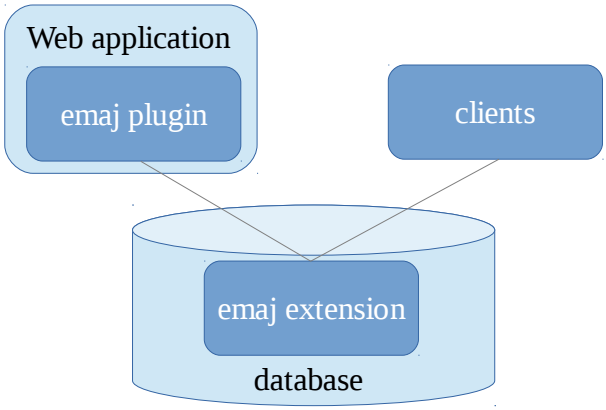
In a **test** environment, E-Maj also brings smoothness into operations. It is possible to very easily restore database subsets into predefined stable states, so that tests can be replayed as many times as needed.

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 GUI as a plugin for the *phpPgAdmin* tool or an independent 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 with this type of group, it is possible to record tables updates for auditing purposes, even with tables that do not have primary key known in PostgreSQL catalogue.

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*

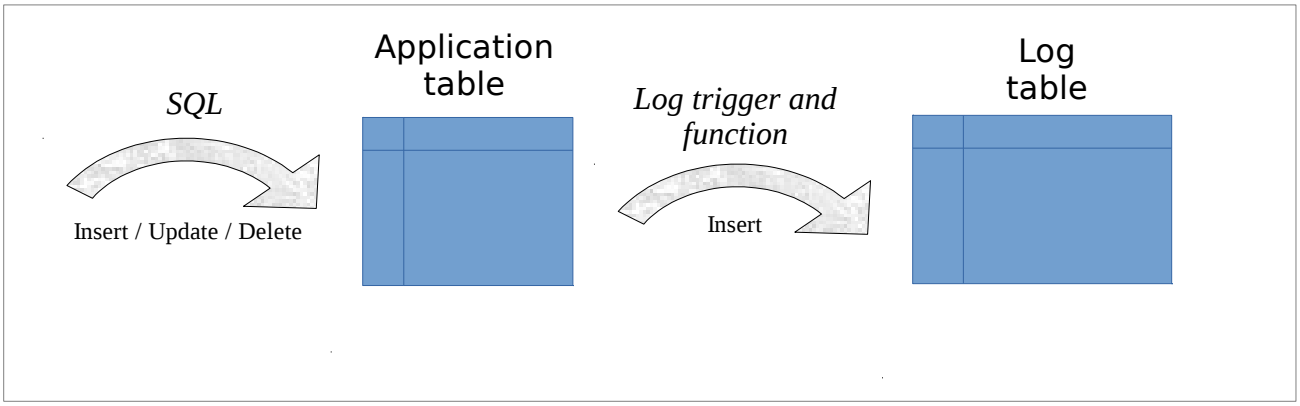
For statements that process several rows, each creation, update or deletion is individually recorded. For instance, if a "*DELETE FROM <table>*" is performed against a table having 1 million rows, 1 million row deletion events are recorded.

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:

- a unique identifier, as an integer associated to a global sequence,
- the precise date and time of the update,
- the type of the executed SQL operation: INS for *INSERT*, UPD for *UPDATE* and DEL for *DELETE*,
- an attribute taking either 'OLD' or 'NEW' value, allowing to distinguish old and new values of updated rows,
- the internal transaction identifier (PostgreSQL *txid*) that performed the update,
- the connection role who performed the update,
- the ip address of the user who performed the update,
- the ip port of the user who performed the update.

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

- 13 tables,
- 5 composite and 2 enum types,
- 1 view,
- more than 90 functions, about half of them being directly callable by users,
- 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
- 2 event triggers when the PostgreSQL version is 9.3 or 9.4 and 3 event triggers when the PostgreSQL version is at least 9.5.

Technical tables, whose structure is interesting to know, are described in the coming chapters (*emaj_group_def* is described in §4.2.2, *emaj_param* is described in §5.1 and *emaj_hist* is described in §5.3).

2.2.3 Norm for E-Maj objects naming

All objects associated to application tables have names built by default with the name of their related table and schema. More precisely, for an application table in a given schema:

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

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

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

It is also possible to define for each application table the prefix of the associated E-Maj objects name. This allows to manage tables with very long names.

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*".

2.2.4 Schemas

All technical objects created at E-Maj installation are located into the schema named *emaj*.

By default, all objects linked to a tables group are created in the main schema *emaj*. But it is possible to locate these objects in one or several secondary schemas. Secondary schemas' names start with "emaj", only their suffix being parametrized in tables groups definition. (Refer to §4.2.2)

Only one technical object is not located into the *emaj* schema: the event trigger « *emaj_protection_trg* » belongs to the *public* schema.

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 §4.2.2), 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 DOWNLOADING AND PREPARING THE EXTENSION

3.1.1 Download

E-Maj is available for download on the Internet sitePGXN, the PostgreSQL Extension Network (http://pgxn.org).

E-Maj and its add-ons are also available on the github.org Internet site:
- source components (https://github.com/beaud76/emaj)
- documentation (https://github.com/beaud76/emaj_doc)
- plug-in for phpPgAdmin (https://github.com/beaud76/emaj_ppa_plugin)
- Emaj_web GUI (https://github.com/beaud76/emaj_web)

3.1.2 Decompression

The extension is delivered as a single compressed file. To be usable, this file must be decompressed.

Under Windows, you can use your favourite decompression utility (Winzip, 7zip,...). Under Unix/Linux, a command like:

```
tar -xvzf emaj-<version>.tar.gz
```

can be used for *.tar.gz* file or

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

for a *.zip* file.

A new emaj-<version> directory is now available, containing the following files tree:

sql/emaj--2.1.0.sql	installation script of the extension (vers. 2.1.0)
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 E-Maj version

- sql/emaj-1.1.0-to-1.2.0.sql psql script that upgrades a 1.1.0 E-Maj version
- sql/emaj-1.0.2-to-1.1.0.sql psql script that upgrades a 1.0.2 E-Maj version
- sql/emaj-1.0.1-to-1.0.2.sql psql script that upgrades a 1.0.1 E-Maj version
- sql/emaj-1.0.0-to-1.0.1.sql psql script that upgrades a 1.0.0 E-Maj version
- sql/emaj-0.11.1-to-1.0.0.sql psql script that upgrades a 0.11.1 E-Maj version
- sql/emaj-0.11.0-to-0.11.1.sql psql script that upgrades a 0.11.0 E-Maj 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 release notes
- LICENSE information about E-Maj license
- AUTHORS 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.pdf English version of the E-Maj presentation
- doc/Emaj.<version>_pres.fr.pdf French version of the E-Maj presentation
- php/emajParallelRollback.php php tool for parallel rollback
- php/emajRollbackMonitor.php php tool for rollbacks monitoring

3.1.3 Preparing the emaj_control file

Starting from version 2.0.0, E-Maj is installed into PostgreSQL databases as an *extension*.

To let E-Maj be known by the integrated extension manager, an *emaj.control* file must be placed into the *SHAREDIR* directory of the PostgreSQL version.

To perform this task:

- identify the precise location of the *SHAREDIR* directory by using the "*pg_config --sharedir*" shell command,
- copy the emaj.contol file from the root directory of the decompressed structure into the *SHAREDIR* directory,
- adjust the *directory* parameter of the *emaj.control* file to reflect the actual location of the E-Maj components.

3.2 E-MAJ EXTENSION SETUP

If E-Maj is already installed in the database, please go on with §3.3.

Some preliminary operations are required.

3.2.1 Preliminary operations

For these operations, the user must log on the concerned database as a superuser.

3.2.1.1 DBLINK extension

Supplied with PostgreSQL, the dblink extension is used during E-Maj rollback operations to help in monitoring those operations. If the dblink extension is not installed, it must be installed before installing E-Maj.

To do it, just issue the following command:

```
CREATE EXTENSION dblink;
```

3.2.1.2 Tablespace

Optionally, if the E-Maj administrator wants to store E-Maj technical tables 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>;
```

E-Maj components installation

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

```
CREATE EXTENSION emaj;
```

To start with, the script verifies that the PostgreSQL version is at least 9.1, 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 §4.10.2).

3.2.2 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 § 4.10), it will be probably necessary to adjust the ***max_prepared_transaction*** parameter.

3.2.3 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 §4.8.2.1)

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.

3.3 UPDATE AN EXISTING E-MAJ VERSION

3.3.1 General approach

The process to update E-Maj version 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 §3.3.2. 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 §3.3.3.
The upgrade from version 1.3.1 to a higher version is described in §3.3.4.
The upgrade from a version greater or equal 2.0.0 is described in §3.3.5.

Starting from version 2.0.0, E-Maj no longer supports PostgreSQL versions prior 9.1. If an older PostgreSQL version is used, it must be updated before migrating E-Maj to a higher version.

3.3.2 Upgrade by deletion and re-installation

For this upgrade path, it is not necessary to use the full un-installation procedure described in §3.4. 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.

3.3.2.1 Stop tables groups

If some tables groups are in *LOGGING* state, they must be stopped, using the *emaj_stop_group()* function (see §4.2.8) (or the *emaj_force_stop_group()* function if *emaj_stop_group()* (see §4.4.4) returns an error).

3.3.2.2 Save user data

It may be useful to save the content of the *emaj_group_def* table in order to be able to easily reload it after the version update, by copying it outside the instance with a *\copy* command, or by duplicating the table outside the *emaj* schema with a SQL statement like:

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

The same way, if the E-Maj administrator had changed parameters value into the *emaj_param* table, it may also be useful to keep a trace of these changes, for instance with:

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

3.3.2.3 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*.

3.3.2.4 Restore user data

Data previously saved can now be restored into E-Maj both technical tables, for instance with *INSERT … SELECT* statements.

```
INSERT INTO emaj.emaj_group_def SELECT * FROM public.sav_group_def;

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

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

3.3.3 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>

3.3.4 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.

3.3.5 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 automaticaly 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)

3.4 E-MAJ UNINSTALL

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 installed as an extension, it cannot be uninstalled with a simple *DROP EXTENSION* statement. An event trigger blocks such a statement (with PostgreSQL 9.3+).

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
```

3.5 POSTGRESQL VERSION UPGRADE

It may happen that a PostgreSQL version change has an impact on the E-Maj extension content. But the main principles apply:

- it is possible to upgrade PostgreSQL version, without reinstalling E-Maj,
- tables groups may even remain in *LOGGING* state at PostgreSQL upgrade,
- if the E-Maj extension content needs to be adapted, this must be performed using a script.

So a supplied *psql* script must be executed after each PostgreSQL version upgrade in order to process the potential impact. It 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.

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

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>;
```

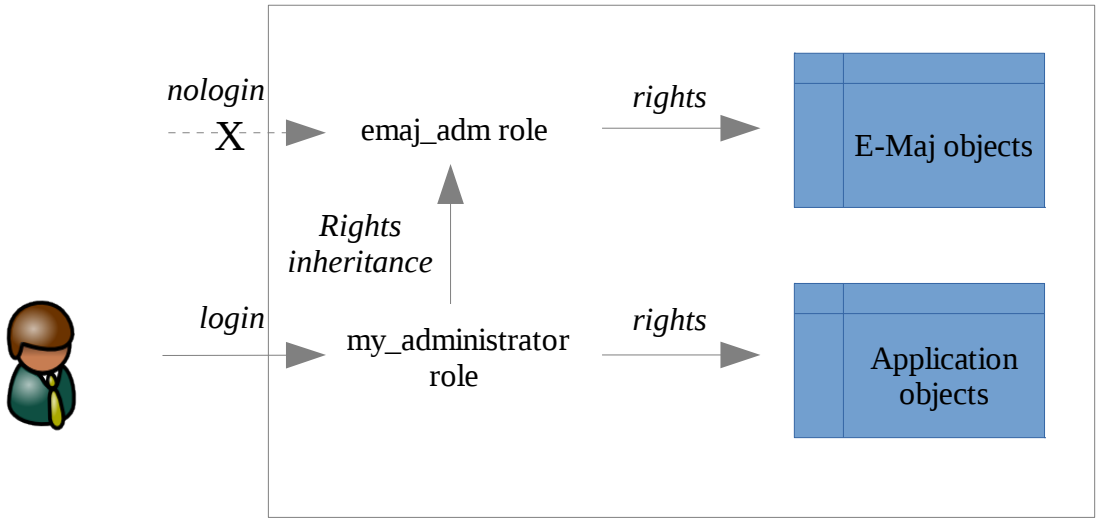
Of course, *emaj_adm* or *emaj_viewer* rights can be given to several roles.

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.

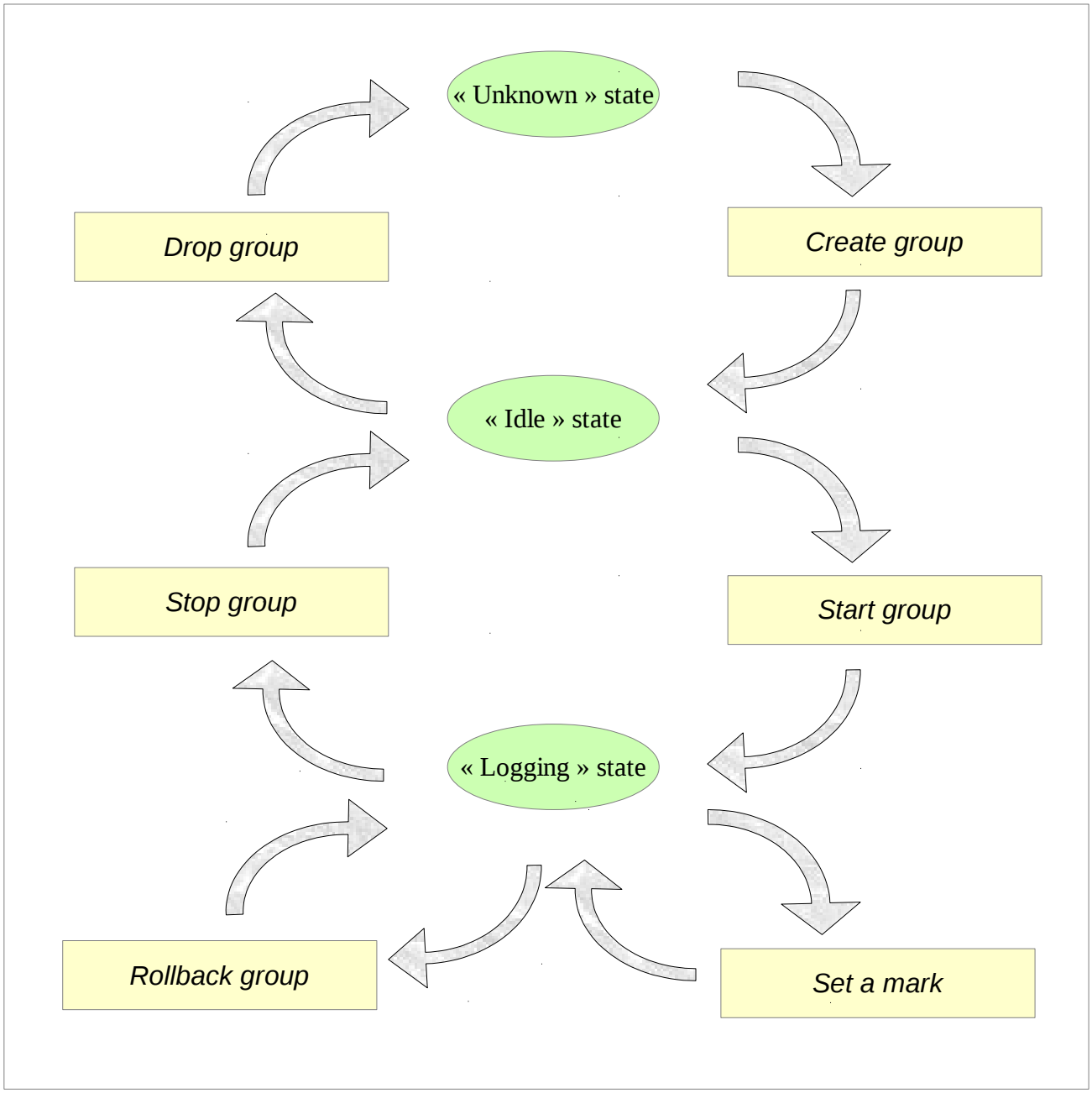
Except when explicitly noticed, the operations presented later can be indifferently executed by a superuser or by a role belonging to the *emaj_adm* group.

4.2 MAIN FUNCTIONS

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

4.2.1 Operations chain

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



4.2.2 Define tables groups

4.2.2.1 The *emaj_group_def* table

The content of tables groups E-Maj will manage 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 or sequence in E-Maj processing (optional)
grpdef_log_schema_suffix	TEXT	suffix used to build the name of the schema containing the E-Maj objects for the table (optional)
grpdef_emaj_names_prefix	TEXT	prefix of E-Maj objects names generated for the table (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.

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

4.2.2.2 Main columns

A tables group name (***grpdef_group*** column) 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.

A table or a sequence of a given schema (***grpdef_schema*** and ***grpdef_tblseq*** columns) cannot be assigned to more than one tables groups. 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.

All tables assigned to a group not created in "*audit_only*" mode 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, neither *TEMPORARY TABLE* nor *UNLOGGED TABLE* are supported by E-Maj. Tables must also be implicitely or explicitely defined as *WITHOUT OIDS*.

If a sequence is associated to an application table, it must be explicitly declared as member of the same group as its table, so that, in case of rollback, the sequence can be reset to its state at the set mark time.

On the contrary, log tables and their sequences should NOT be referenced in a tables group!

4.2.2.3 Optional columns

The type of the ***grpdef_priority*** column is *INTEGER* and may be NULL. It defines a priority order in E-Maj tables processing. This can be 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 *grpdef_priority* ascending order, NULL being processed last. For a same priority level, tables are processed in alphabetic order of schema name and table name.

For E-Maj installations having a large number of tables, it may be useful to spread all E-Maj objects on several schemas, instead of concentrating them in the unique *emaj* schema. The ***grpdef_log_schema_suffix*** column allows to specify the schema that will hold the log table, the log sequence, and the log and rollback functions for a particular application table.

It this *grpdef_log_schema_suffix* column contains a NULL or an empty chain, the *emaj* main schema will be used. Otherwise, a secondary schema will be used. Its name is then built as the concatenation of 'emaj' and the column's content.

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

For sequences, the *grpdef_log_schema_suffix* column must be NULL.

For tables having long names, the default prefix for E-Maj objects names may be too long to fit the PostgreSQL limits. But another prefix may be defined for each table, by setting the ***grpdef_emaj_names_prefix*** column.

If this *grpdef_emaj_names_prefix* column contains a NULL value, the default prefix <nom_schéma>_<nom_table> is used.

Two different tables cannot have the same prefix, explicitely or implicitely.

For sequences, the *grpdef_emaj_names_prefix* column must be NULL.

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. The ***grpdef_log_dat_tsp*** column specifies the name of the tablespace to use for the log table of an application table. Similarly, the ***grpdef_log_idx_tsp*** column specifies the name of the tablespace to use for the index of the log table.

If a column *grpdef_log_dat_tsp* or *grpdef_log_idx_tsp* is NULL (default value), the default tablespace of the current session at tables group creation is used.

For sequences, both *grpdef_log_dat_tsp* and *grpdef_log_idx_tsp* columns must be NULL.

4.2.3 Create a tables group

Once the content of a 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, 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 secondary E-Maj 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 verify that this trigger does not update any tables that would not belong to the group.

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, the group must not be referenced in the *emaj_group_def* table. Once created, an empty group can be then populated using the *emaj_alter_group*() function (see §4.3).

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 composition in the *emaj_relation* internal table, 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 §4.3) or suppressed by the *emaj_drop_group()* function (see §4.2.9).

4.2.4 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 transaction start time, with the pattern “hh.mn.ss.mmm“,

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 transaction start time with a “hh.mn.ss.mmm” 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 on each table of the group an *ACCESS EXCLUSIVE* lock if the PostgreSQL version is prior 9.5, or *SHARE ROW EXCLUSIVE* lock in other cases. 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 §5.3).

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 §4.9.3 explains how to describe the group names array.

4.2.5 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 transaction start time, with the pattern “hh.mn.ss.mmm”,

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 transaction start time with a “hh.mn.ss.mmm” 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 §4.9.3 explains how to describe the group names array.

4.2.6 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 §4.3). 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 the PostgreSQL version is prior 9.5, the lock mode is even *ACCESS EXCLUSIVE* for tables having updates to cancel and whose log trigger must consequently be disabled during the operation. 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, it may be necessary to de-activate them before the rollback and re-activate them after (see §5.5.3).

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 (§4.8.2.2) or the *emajRollbackMonitor.php* client (§4.11).

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 §4.9.3 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.

4.2.7 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 §4.3). 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, it may be necessary to de-activate them before the rollback and re-activate them after (see §5.5.3).

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 (§4.8.2.2) or the *emajRollbackMonitor.php* client (§4.11).

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 §4.4.6).

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 §4.9.3 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.

4.2.8 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 transaction start time expressed as “hh.mn.ss.mmm”.

Stopping a tables group simply deactivates log triggers of application tables of the group. The setting of *ACCESS EXCLUSIVE* locks for PostgreSQL versions prior 9.5, or *SHARE ROW EXCLUSIVE* locks in other cases, 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 §4.9.3 explains how to describe the group names array.

4.2.9 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 § 4.2.8).

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 *emaj_create_group()* function: log tables, log and rollback functions, log triggers.

The function also drops all secondary schemas that have become empty.
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 MODIFYING TABLES GROUPS

Several types of events 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, schema, 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 change in a column type.

4.3.1 Modifying a tables group in IDLE state

In all cases, the following steps can be performed:

- stop the group, if it is in *LOGGING* state, using the *emaj_stop_group()* function,
- update the *emaj_group_def* table and/or modify the application schema,
- drop and recreate the tables group, using the *emaj_drop_group()* and *emaj_create_group()* functions.

But this last step can be also performed by the *emaj_alter_group()* function, with a statement like:

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

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, …).

The function creates and drops the secondary schemas when needed.

Once altered, a tables group remains in IDLE state, but its log tables become empty.

The "*rollbackable*" or "*audit_only*" characteristic of the tables group cannot be changed using the *emaj_alter_group()* function. To change it, the tables group must be dropped and re-created using the *emaj_drop_group()* and *emaj_create_group()* functions.

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.

In most cases, executing the *emaj_alter_group()* function is much more efficient than chaining both *emaj_drop_group()* and *emaj_create_group()* functions.

It is possible to update the *emaj_group_def* table, when the tables group is in logging state. However it will not have an 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>');
```

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

The chapter §4.9.3 explains how to describe the group names array.

4.3.2 Modifying a tables group in LOGGING state

But the previous method has several drawbacks:
- logs recorded before the operation are lost,
- it is not possible to rollback a tables group to a previous state anymore.

However, some actions are possible while the tables groups are in *LOGGING* state. The following table lists these allowed actions.

Action	LOGGING group	Method
Change the groupe ownership	No	
Change the log schema suffix	Yes	*emaj_group_def* update
Change the E-Maj names prefix	Yes	*emaj_group_def* update
Change the log data tablespace	Yes	*emaj_group_def* update
Change the log index tablespace	Yes	*emaj_group_def* update
Change the E-Maj priority	Yes	*emaj_group_def* update
Remove a table from a group	No	
Remove a sequence from a group	No	
Add a table to a group	No	
Add a sequence to a group	No	
Repair a table or a sequence	No	
Rename a table	No	
Rename a sequence	No	
Change the schema of a table	No	
Change the schema of a sequence	No	
Rename a table's column	No	
Change a table's structure	No	
Other forms of ALTER TABLE	Yes	No E-Maj impact
Other forms of ALTER SEQUENCE	Yes	No E-Maj impact

4.3.2.1 The « emaj_group_def **update** » method

Most attributes of the *emaj_group_def* table describing the tables groups can be dynamicaly changed while groups have not been stopped.

To do this, the following steps can be performed:

- modify the *emaj_group_def* table,
- call one of the *emaj_alter_group()* or *emaj_alter_groups()* functions.

For tables groups in *LOGGING* state, these functions set a *ROW EXCLUSIVE* lock on each application table of these groups.

On these same tables groups, they also set a mark whose name can be suppled as parameter. The syntax of these calls becomes:

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

or

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

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 transaction start time with a "hh.mn.ss.mmm" pattern.

An E-Maj rollback operation targeting a mark set before such groups changes does NOT automaticaly cancel these changes.

However, the administrator can apply the same procedure to reset a tables group to a prior state.

4.4 OTHER GROUPS MANAGEMENT FUNCTIONS

4.4.1 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.

4.4.2 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.

4.4.3 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 §4.5.6).

4.4.4 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 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.

4.4.5 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.

4.4.6 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 §4.4.7).

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*.

4.4.7 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_id — internal identifier of the target mark
- cons_end_rlbk_mark_name — rollback end mark name
- cons_end_rlbk_mark_id — internal identifier of the end mark
- cons_rows — number of intermediate updates
- cons_marks — number of intermediate marks

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 MARKS MANAGEMENT FUNCTIONS

4.5.1 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.

4.5.2 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.

4.5.3 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.

4.5.4 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.

4.5.5 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 §5.3).

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));
```

4.5.6 Protection of a mark against rollbacks

To complement the mechanism of tables group protection against accidental rollbacks (see §4.4.3), 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 automaticaly 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.

4.6 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.

4.6.1 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_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.

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.

4.6.2 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_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.

It is possible to easily execute more precise requests on these statistics. For instance, it is possible to get the number of updates for a given table, here mytbl1, per SQL verb, using a statement like:

```
postgres=# SELECT stat_table, stat_verb, stat_rows
FROM emaj.emaj_detailed_log_stat_group('myAppl1', NULL, NULL)
WHERE stat_table='mytbl1';
 stat_table | stat_verb | stat_rows
------------+-----------+-----------
 mytbl1     | DELETE    |         1
 mytbl1     | INSERT    |         6
 mytbl1     | UPDATE    |         2
(3 rows)
```

4.6.3 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 §4.9.3 explains how to describe the group names array.

4.7 DATA EXTRACTION FUNCTIONS

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

4.7.1 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

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.

4.7.2 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 tables and sequences contained by the group.

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

<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>

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

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.

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 directly derived from the structure of the related application table. 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 that performed the update
- *emaj_user* connection role that performed the update
- *emaj_user_ip* ip address of the client that performed the update (if the client was connected with ip protocol)

4.7.3 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, and record them into a file. 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.

The output file name must be supplied as 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 last parameter 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* 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 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 §4.9.3 explains how to describe the group names array.

4.8 OTHER FUNCTIONS

4.8.1 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* schema and each secondary schema if any) 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 §5.2.1).

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 *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.

4.8.2 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 §4.11), fit this need.

4.8.2.1 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 (§3.2.1.1) 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).

4.8.2.2 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 (§4.6.3).

4.8.3 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 §4.8.2.1), 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.

4.8.4 Deactivating or reactivating event triggers

The E-Maj extension installation procedure activates event triggers to protect it (See §5.2.2). 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.

4.9 MULTI-GROUPS FUNCTIONS

4.9.1 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.

4.9.2 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()*	4.3
emaj.emaj_start_groups()	*emaj.emaj_start_group()*	4.2.4
emaj.emaj_stop_groups()	*emaj.emaj_stop_group()*	4.2.8
emaj.emaj_set_mark_groups()	*emaj.emaj_set_mark_group()*	4.2.5
emaj.emaj_rollback_groups()	*emaj.emaj_rollback_group()*	4.2.6
emaj.emaj_logged_rollback_groups()	*emaj.emaj_logged_rollback_group()*	4.2.7
emaj.emaj_estimate_rollback_groups()	*emaj.emaj_estimate_rollback_group()*	4.6.3
emaj.emaj_gen_sql_groups()	*emaj.emaj_gen_sql_group()*	4.7.3

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.

4.9.3 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.

4.9.4 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. In all these cases, and when the tables groups list is empty, a warning message is generated.

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.

4.10 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.

4.10.1 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.

4.10.2 Prerequisites

The command to run parallel rollbacks is written in php. As a consequence, *php* software and its PostgreSQL interface has to be installed on the server that executes the command (which is not necessarily the same as the one that hosts the PostgreSQL instance).

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.

4.10.3 Syntax

The command that performs a parallel rollback has the following syntax:

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

General options:
 -l specifies that the requested rollback is a "*logged rollback*" (see §4.2.7)
 -a specifies that the requested rollback is allowed to reach a mark set before an alter group operation (see §4.3)
 -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 the *emajParallelRollback.php* command, 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.

4.10.4 Examples

The command:

```
./php/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:

```
./php/emajParallelRollback.php -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.

4.11 ROLLBACK MONITORING CLIENT

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

4.11.1 Prerequisite

The command to monitor rollback operations is written in php. As a consequence, *php* software and its PostgreSQL interface has to be installed on the server that executes the command (which is not necessarily the same as the one that hosts the PostgreSQL instance).

4.11.2 Syntax

The command that monitors rollback operations has the following syntax:

emajRollbackMonitor.php [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).

4.11.3 Examples

The command:

```
./php/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:

```
./php/emajRollbackMonitor.php -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 1.1.0) - Monitoring rollbacks activity
------------------------------------------------------------
04/07/2013 - 12:07:17
** rollback 34 started at 2013-07-04 12:06:20.350962+02 for groups {myGroup1,myGroup2}
   status: COMMITTED ; ended at 2013-07-04 12:06:21.149111+02
** rollback 35 started at 2013-07-04 12:06:21.474217+02 for groups {myGroup1}
   status: COMMITTED ; ended at 2013-07-04 12:06:21.787615+02
-> rollback 36 started at 2013-07-04 12:04:31.769992+02 for groups {group1232}
   status: EXECUTING ; completion 89 % ; 00:00:20 remaining
-> rollback 37 started at 2013-07-04 12:04:21.894546+02 for groups {group1233}
   status: LOCKING ; completion 0 % ; 00:22:20 remaining
-> rollback 38 started at 2013-07-04 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_int	INT	parameter value, if its type is integer (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:

- *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 §4.6.3).
- *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 §4.6.3).
- *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 §4.6.3).
- *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 §4.6.3).*
- *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 §4.6.3).

- *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 §4.6.3).
- *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 § 5.3),

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);
```

It is also possible to manage parameter values using any graphic tool such as PgAdmin or phpPgAdmin.

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.

5.2 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.

5.2.1 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,
- no table has been altered as *UNLOGGED* or *WITH OIDS* table,
- for *ROLLBACKABLE* tables groups, application tables have their primary key.

By using the *emaj_verify_all()* function (§4.8.1), the administrator can perform the same checks on demand on all tables groups.

5.2.2 Event triggers

Installing E-Maj on instances using PostgreSQL version 9.3 or higher adds 2 event triggers of type "*sql_drop*":

- *emaj_sql_drop_trg* blocks the drop attempts of:
  - ✔ any E-Maj object (log table, log sequence, log function, log trigger and secondary schema),
  - ✔ 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 on instances using PostgreSQL version 9.5 or higher 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 §4.8.4).

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.

5.3 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:

- *ALTER_GROUP* tables group change
- *ALTER_GROUPS* tables groups change
- *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
- *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
- *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
- *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
- *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
- *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*
- *END*
- *EVENT TRIGGERS DISABLED*
- *EVENT TRIGGERS ENABLED*
- *MARK DELETED*
- *SCHEMA CREATED* secondary schema created
- *SCHEMA DROPPED* secondary schema dropped
- *LOG SCHEMA CHANGED*
- *NAMES PREFIX CHANGED* E-Maj names prefix modified
- *LOG DATA TABLESPACE CHANGED* Tablespace for the log table modified
- *LOG INDEX TABLESPACE CHANGED* Tablespace for the log index modified

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 §4.3),
- *emaj_rlbk_plan* lists the elementary steps performed during the execution of *emaj_rollback_group()* and related functions (Cf §4.2.6 and 4.2.7).

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.

5.1 IMPACTS ON INSTANCE AND DATABASE ADMINISTRATION

5.1.1 Stopping and restarting the instance

Using E-Maj doesn't bring any particular constraint regarding stopping and restarting a PostgreSQL instance.

5.1.1.1 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,...

5.1.1.2 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 §4.2.6). (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.

5.1.2 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.

5.1.2.1 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.

5.1.2.2 Logical saves and restores of entire 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.

5.1.2.3 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.

5.1.3 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.

5.1.4 Tables reorganisation

5.1.4.1 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.

5.1.4.2 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.

5.1.5 Using E-Maj with replication

5.1.5.1 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.

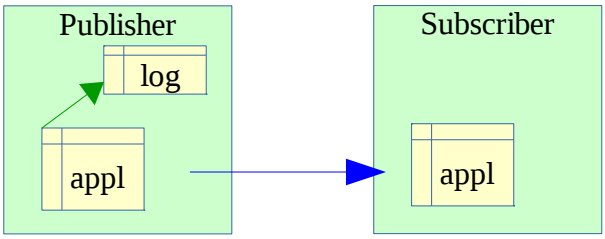
However, because of the way PostgreSQL manages sequences, the sequences' current values may be a little forward on slave instances than on the master instance. For E-Maj, this may lightly overestimate the number of log rows in general statistics. But there is no consequence on the data integrity.

5.1.5.2 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.

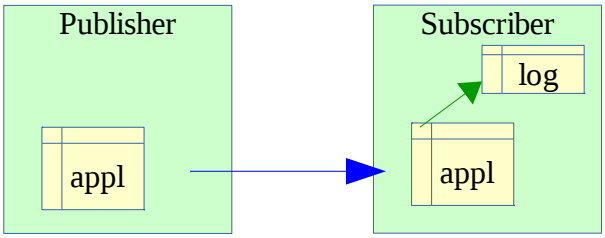
Several cases have to be examined.

Replication of application tables managed by E-Maj



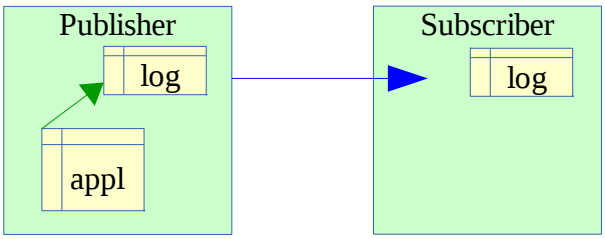
An application table that belongs to a tables group can be replicated, without any particular condition. The effect of any rollback operation that may occur would be simply replicated on subscriber side.

Replication of application tables with E-Maj activated on subscriber side



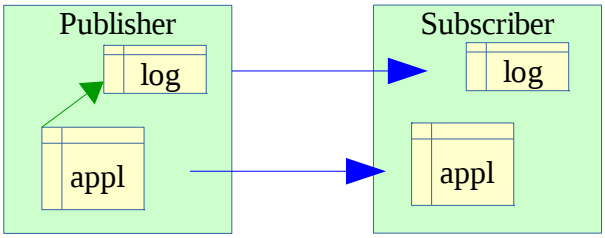
It is possible to include an application table into a tables group, with updates coming from a logical replication flow. But all E-Maj operations (starting/stopping the group, setting marks,…) must of course be executed on the subscriber side. An E-Maj rollback operation can be launched once the replication 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.

5.1.5.3 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.

5.2 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.

5.3 PERFORMANCE

5.3.1 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.

5.3.2 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 §4.6.3).

5.3.3 Optimizing E-Maj operations

Here are some advice to optimize E-Maj operations:

5.3.3.1 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 §3.2.1.2)

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 §4.2.2.3)

5.3.3.2 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.

5.4 USAGE LIMITS

The E-Maj extension usage has some limits.

- The minimum required PostgreSQL version is 9.1.
- All tables belonging to a “rollbackable” tables group must have an explicit *PRIMARY KEY*.
- *TEMPORARY*, *UNLOGGED* or *WITH OIDS* 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.

5.5 USER'S RESPONSIBILITY

5.5.1 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.

5.5.2 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.

5.5.3 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 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 value of columns. So the rollback would reset the old values without any problem. But may be it would be necessary to deactivate such a trigger during a rollback operation.

If the trigger updates another table, two cases must be considered:

- if the updated table belong to the same tables group, it would be necessary to deactivate the trigger during a rollback operation, so that E-Maj and only E-Maj performs the updates required by the rollback operation,
- 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. In such a case, merely deactivating the trigger may not be sufficient.

5.5.4 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 WEB CLIENTS

To make E-Maj use easier, two web applications are also available:
- a plug-in for the phpPgAdmin administration tool in its versions 5.1 and higher,
- an independent web application, Emaj_web.

6.1 OVERVIEW

Both web clients provide the same functionalities with E-Maj, and have a very similar user interface.

Emaj_web borrows 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 PHPPGADMIN PLUGIN INSTALLATION

6.2.1 Prerequisite

A version 5.1 or higher of *phpPgAdmin* must be installed and operational in a web server.

6.2.2 Plug-in download

The E-Maj plug-in for phpPgAdmin can be downloaded from the following git repository:
https://github.com/beaud76/emaj_ppa_plugin

The downloaded *Emaj* directory must be copied into the *plugin* directory of the installed *phpPgAdmin* root directory.

6.2.3 Plug-in activation

To activate the plug-in, just open the *conf/config.inc.php* file from the phpPgAdmin root directory, and add the character string 'Emaj' to the variable *$conf['plugins']*.

For instance, one may have:

```
$conf['plugins'] = array('Emaj');
```

or, if another plug-in is already activated:

```
$conf['plugins'] = array('Report','Emaj');
```

6.2.4 Plug-in parametrization

In order to submit batch rollback (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 contained in the *Emaj/conf/config.inc.php* file:

- `$plugin_conf['psql_path']` defines the access path of the *psql* executable file,
- `$plugin_conf['temp_dir']` defines a temporary directory that rollback functions can use.

  The distributed *config.inc.php-dist* file can be used as a configuration template.

6.3 EMAJ_WEB CLIENT INSTALLATION

6.3.1 Prerequisite

Emaj_web requires a web server with a php interpreter.

6.3.2 Download

The *Emaj_web* application can be downloaded from the following git repository:
https://github.com/beaud76/emaj_web

6.3.3 Application configuration

Two configuration files have to be set up.

6.3.3.1 General parameters

The file *emaj_web/conf/config.inc.php* contains the general parameters of the applications. It includes in particular the description of the PostgreSQL instances connections.

The *emaj_web/conf/config.inc.php-dist* file may be used as a configuration template.

6.3.3.2 Plugin parameters

As *Emaj_web* reuses the plugin for *phpPgAdmin*, the parameters that are specific to the plugin are defined into a separate file: *emaj_web/plugins/Emaj/conf/config.inc.php.*

In order to submit batch rollback (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 contained in the *Emaj/conf/config.inc.php* file:

- `$plugin_conf['psql_path']` defines the access path of the *psql* executable file,
- `$plugin_conf['temp_dir']` defines a temporary directory that rollback functions can use.

The distributed *config.inc.php-dist* file can be used as a configuration template.

6.4 USING THE WEB CLIENTS

6.4.1 Accessing E-Maj from the phpPgAdmin interface

Once connected to a database where the E-Maj extension has been installed, and using a role having sufficient privileges (super-user, *emaj_adm* or *emaj_viewer*), a new red icon appears on the right in the horizontal database icons tab. Obviously, the *emaj* schema appears in schemas list.

In the browser tree on the left, a new E-Maj object also appears. By opening it, the list of created tables groups becomes directly accessible.

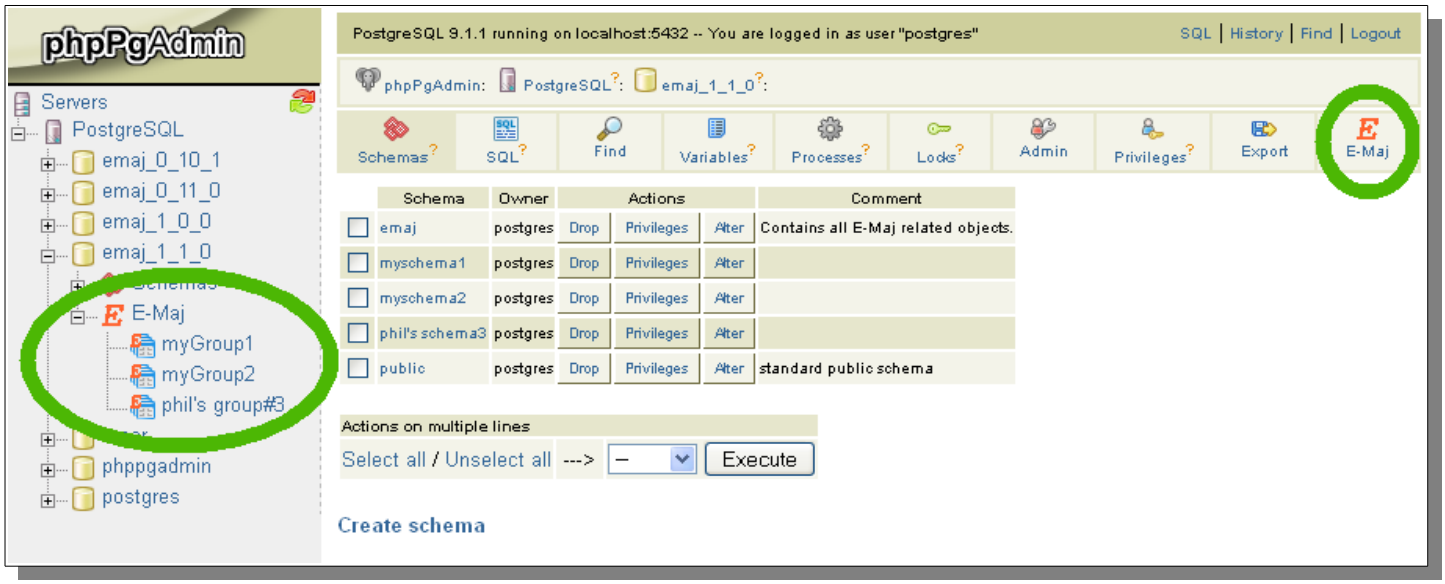


Figure 1a – phpPgAdmin: connection to a database where E-Maj is installed.

6.4.2 Access to Emaj_web

The connection to a database is similar to *phpPgAdmin*.

Once connected to a database where the E-Maj extension has been installed, and using a role having sufficient privileges (super-user, *emaj_adm* or *emaj_viewer*), a red icon appears on the right in the horizontal database icons tab.

In the browser tree on the left, the E-Maj object also appears. By opening it, the list of created tables groups becomes directly accessible.

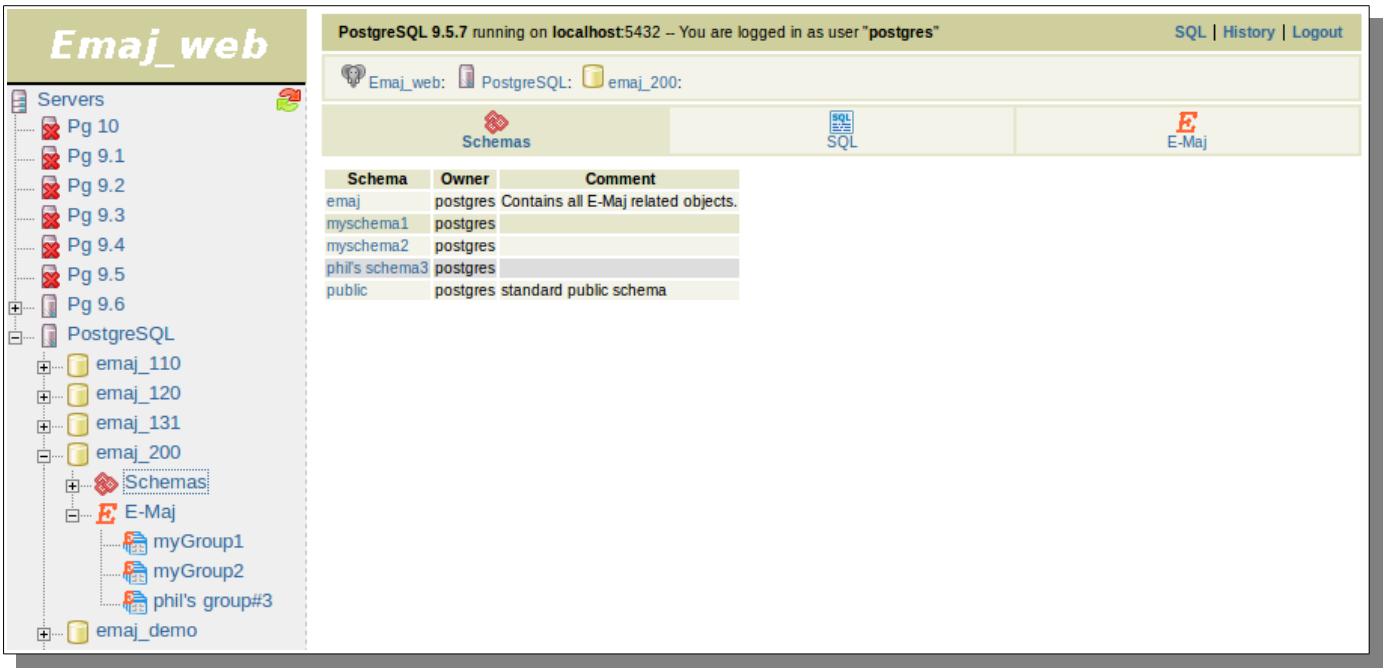


Figure 1b – Emaj_web: connection to a database where E-Maj is installed.

6.4.3 Tables groups list

By clicking on one of the E-Maj icons, the user reaches a page that lists all tables groups created in this database.

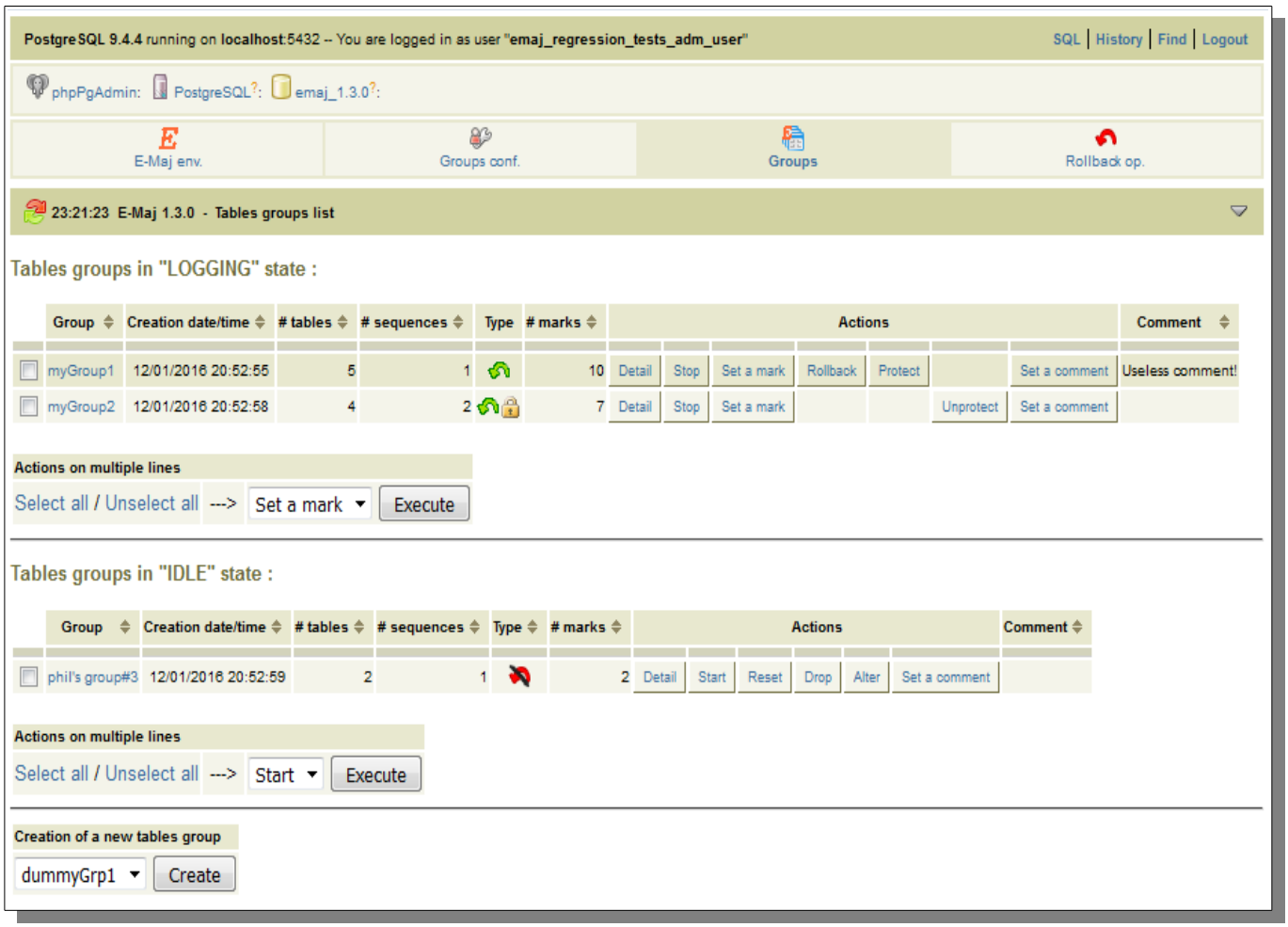


Figure 2 – List of the tables groups created in the database.

In fact, this page displays two lists: one for groups in *LOGGING* state and the other for groups in *IDLE* state.

For each 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.

Several buttons are available so that the user can perform any possible action, depending on the group state.

Under both lists, a combo box and a button are dedicated to multi-group actions.

At the bottom of the page, a list box presents the table groups that may be created (those known in the *emaj_group_def* table but not yet created).

6.4.4 Some details about the user interface

The user can navigate in E-Maj 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.

All pages displayed by the E-Maj plug-in have a header that contains:
- a button to refresh the current page,
- the time of current page display,
- the E-Maj version installed on the database,
- the page title,
- a bottom link, located at the extreme right of the header, to reach the bottom of the page.

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, hovering the mouse over the grey bar located just below the header row displays input fields that can be used to filter rows to display.

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.

6.4.5 E-Maj environment state

By clicking on the "*E-Maj env.*" icon of the main bar, the user reaches an overview of the E-Maj environment state.

Items displayed first:
- the installed E-Maj version,

- 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.

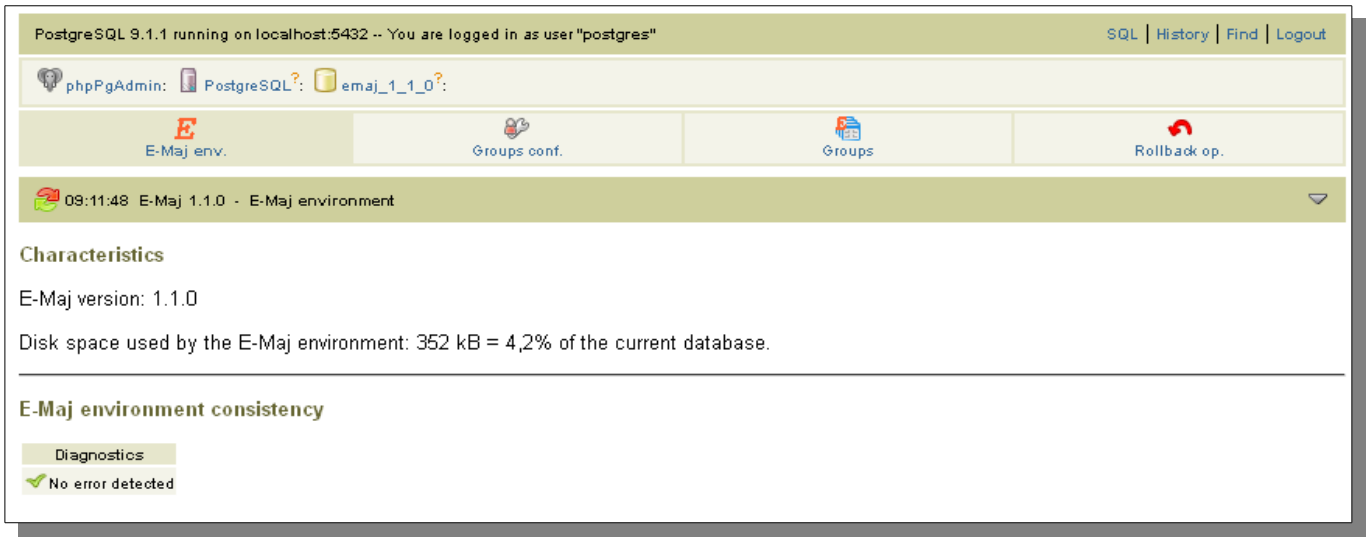


Figure 6 – E-Maj environment state.

6.4.6 Tables groups content

With a click on the "*Groups conf.*" icon of the main bar, the user reaches the function that manages the tables groups content.

The upper part of the page lists the existing schemas (except schemas dedicated to E-Maj). By selecting a schema, the list of its tables and sequences appears.

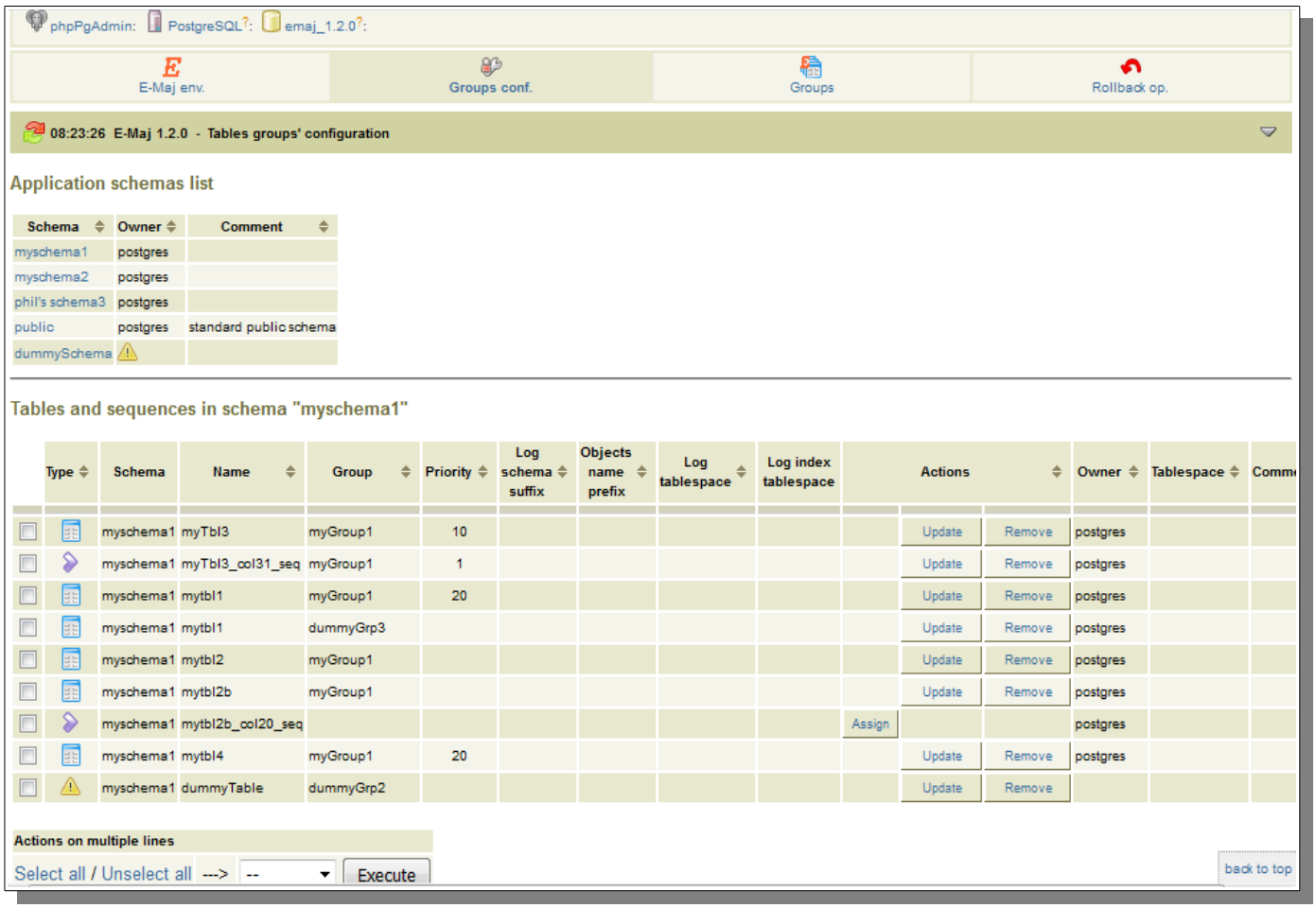


Figure 7 – Tables groups content.

The user can then view or modify the content of the *emaj_group_def* table used for the tables groups creation (*emaj_create_group()* function).

The following are listed for each table or sequence:

- its type
- the tables group it belongs to, if any,
- the following attributes of the table or sequence in the *emaj_group_def* table, if assigned: (see §4.2.2):
  - the priority level in the group,
  - the suffix that defines log schema,
  - the prefix used to build the E-Maj object names for this table,
  - the optional tablespace name for the log table,
  - the optional tablespace name for the log table's index,
- its owner,
- the tablespace it belongs to, if any,
- the associated comment in the database.

The schemas list and the tables and sequences list also display the objects that are known in the *emaj_group_def* table but don't exist in the database. These objects are identified with a “!” icon in the first column of each list.

With buttons, it is possible to:

- assign a table or a sequence to a new or an already known tables group,
- modify the properties of a table or a sequence inside its tables group,
- remove a table or a sequence from its tables group.

Note that any change applied in the *emaj_group_def* table's content will only be effective when the concerned tables groups are altered or dropped and re-created.

6.4.7 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 or on its “Detail” button. This page is also accessible with the “*Properties*” icon of the groups bar and through the left browsing tree.

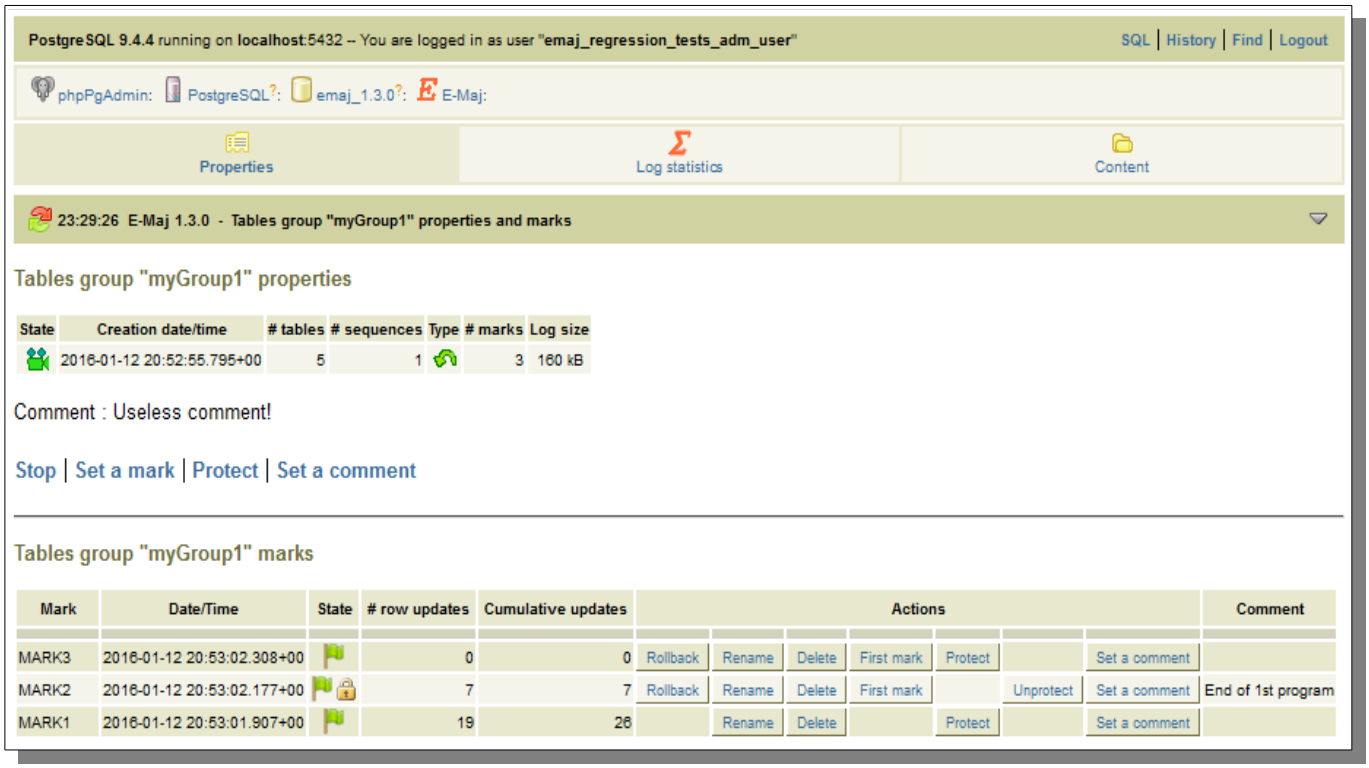


Figure *8 – Details of a tables group*

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 list of links enabling execution of actions applicable to 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.

Several buttons are available to perform the actions permitted by the mark's state.

6.4.8 Statistics

Using the "*Log statistics*" icon of the groups 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.

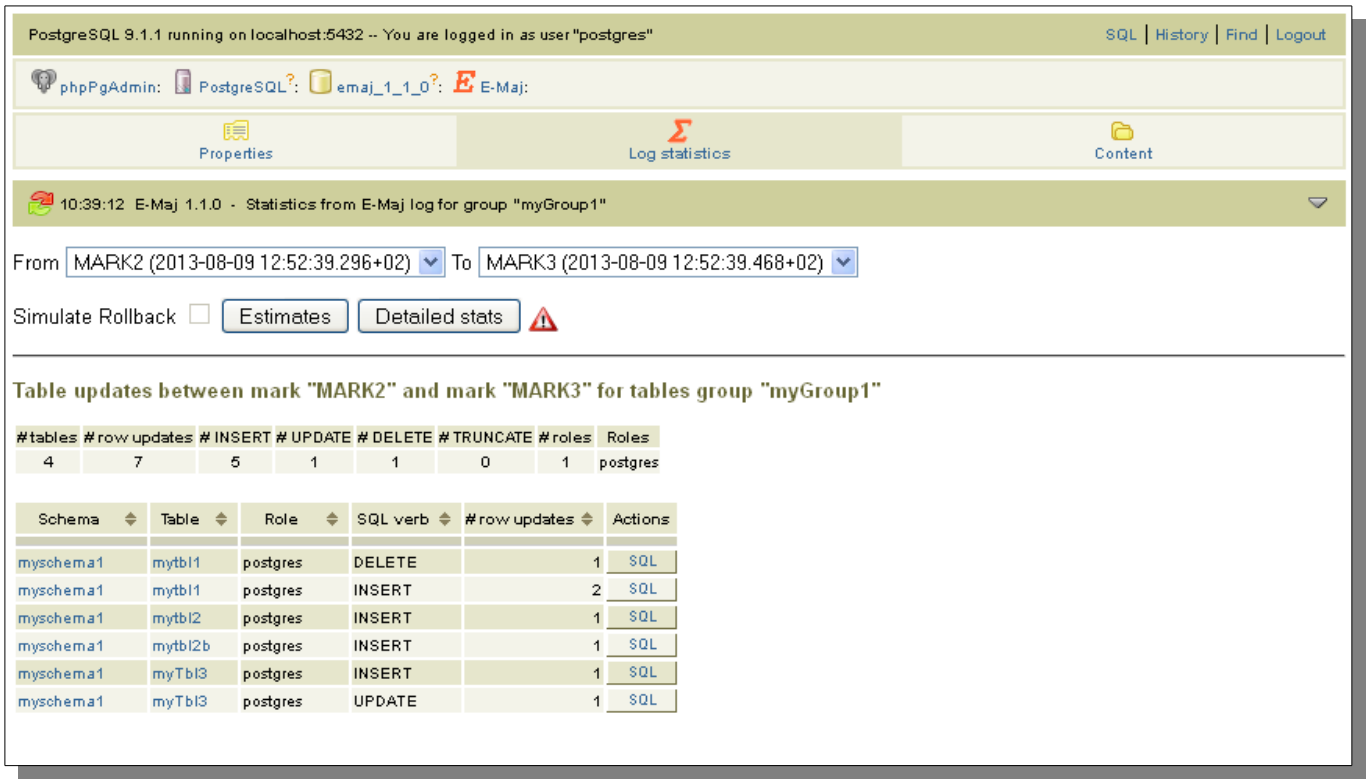


Figure 9 – Detailed statistics about updates recorded between two marks

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.

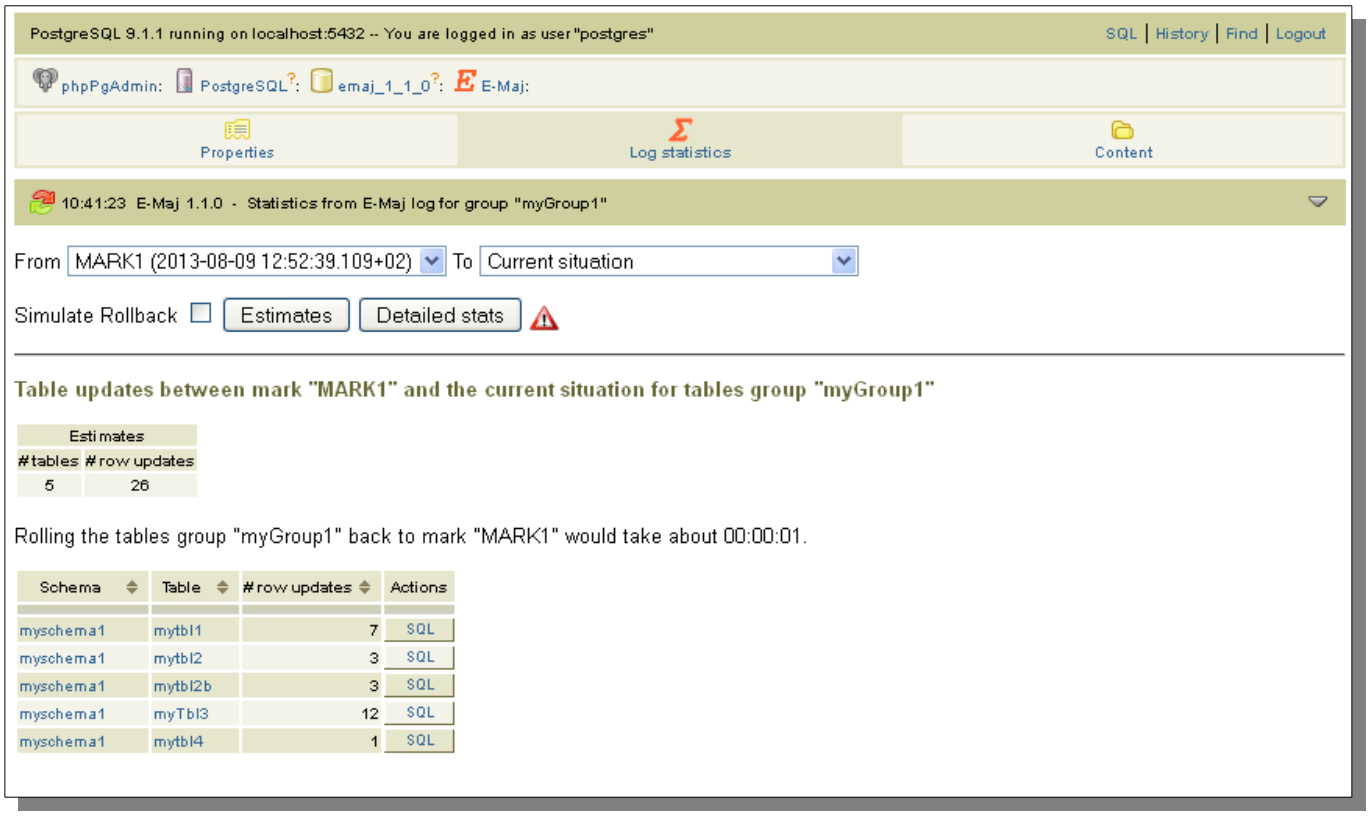


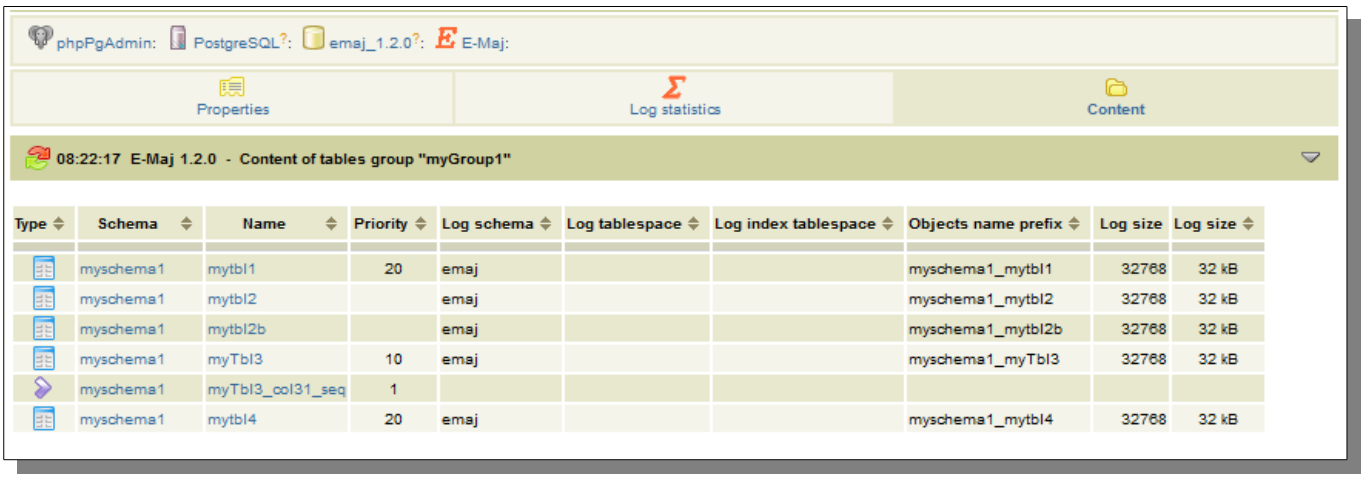
Figure 10 – Result of the rollback simulation, with the estimated number of updates for each table.

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.

6.4.9 Tables group content

Using the "*Content*" icon of the groups icon 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	32768	32 kB
	myschema1	myTbl3	10	emaj			myschema1_myTbl3	32768	32 kB
	myschema1	myTbl3_col31_seq	1						
	myschema1	mytbl4	20	emaj			myschema1_mytbl4	32768	32 kB

Figure 11 – Content of a tables group.

6.4.10 Monitoring rollback operations

Using the "*Rollback op.*" icon 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.

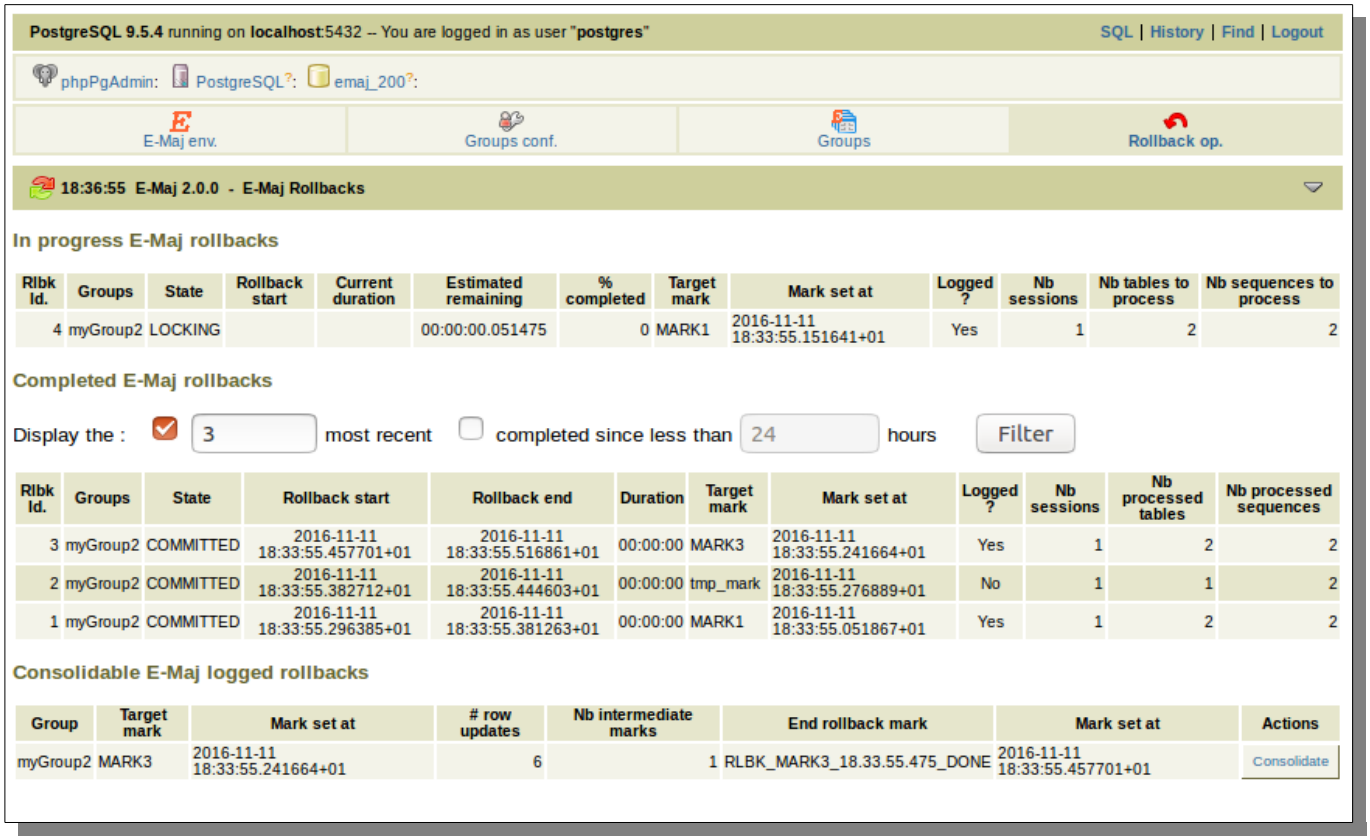


Figure 12 – Rollback operation monitoring.

7 APPENDIX

7.1 E-MAJ FUNCTIONS LIST

E-Maj functions that are available to users are listed in alphabetic order below. They are all callable by roles having *emaj_adm* privileges. The chart also specifies those callable by *emaj_viewer* roles.

Functions	Input parameters	Output data	Callable by emaj_viewer	Ref.
emaj_alter_group	group TEXT	# tables.and.seq INT		§ 4.3
emaj_alter_groups	groups.array TEXT	# tables.and.seq INT		§ 4.3
emaj_cleanup_rollback_state	-	# rollback INT		§ 14.8.3
emaj_comment_group	group TEXT comment TEXT	-		§ 4.4.2
emaj_comment_mark_group	group TEXT mark TEXT comment TEXT	-		§ 4.5.1
emaj_consolidate_rollback_group	group TEXT end.rollback.mark TEXT	# tables.and.seq INT		§4.4.6
emaj_create_group	group TEXT [is.rollbackable BOOLEAN] [is.empty BOOLEAN]	#.tables.and.seq INT		§ 4.2.3
emaj_delete_before_mark_group	group TEXT mark TEXT	#.deleted.marks INT		§ 4.5.5
emaj_delete_mark_group	group TEXT mark TEXT	1 INT		§ 4.5.4
emaj_detailed_log_stat_group	group TEXT start.mark TEXT end.markTEXT	SETOF emaj_detailed_log_stat_type	Yes	§ 4.6.2
emaj_disable_protection_by_event_triggers	-	# triggers INT		§ 4.8.4
emaj_drop_group	group TEXT	#.tables.and.seq INT		§ 4.2.9
emaj_enable_protection_by_event_triggers	-	# triggers INT		§ 4.8.4
emaj_estimate_rollback_group	group TEXT mark TEXT	duration INTERVAL	Yes	§ 4.6.3
emaj_estimate_rollback_groups	groups.array TEXT[] mark TEXT	duration INTERVAL	Yes	§ 4.6.3
emaj_force_drop_group	group TEXT	#.tables.and.seq INT		§4.4.5
emaj_force_stop_group	group TEXT	#.tables.and.seq INT		§ 4.4.4
emaj_gen_sql_group	group TEXT start.mark TEXT end.mark TEXT	#.gen.statements BIGINT		§ 4.7.3

Functions	Input parameters	Output data	Callable by emaj_viewer	Ref.
	output.file.path TEXT [tables.seq.array TEXT[]]			
emaj_gen_sql_groups	groups.array TEXT[] start.mark TEXT end.mark TEXT output.file.path TEXT [tables.seq.array TEXT[]]	#.gen.statements BIGINT		§ 4.7.3
emaj_get_consolidable_rollbacks	-	SETOF emaj_consolidable_rollback_type	Yes	§ 4.4.7
emaj_get_previous_mark_group	group TEXT date.time TIMESTAMPTZ	mark TEXT	Yes	§ 4.5.2
emaj_get_previous_mark_group	group TEXT mark TEXT	mark TEXT	Yes	§ 4.5.2
emaj_log_stat_group	group TEXT start.mark TEXT end.mark TEXT	SETOF emaj_log_stat_type	Yes	§ 4.6.1
emaj_logged_rollback_group	group TEXT mark TEXT is_alter_group_allowed BOOLEAN	SETOF (severity TEXT, message TEXT)		§ 4.2.7
emaj_logged_rollback_groups	groups.array TEXT[] mark TEXT is_alter_group_allowed BOOLEAN	SETOF (severity TEXT, message TEXT)		§ 4.2.7
emaj_protect_group	group TEXT	0/1 INT		§ 4.4.3
emaj_protect_mark_group	group TEXT mark TEXT	0/1 INT		§ 4.5.6
emaj_rename_mark_group	group TEXT mark TEXT new.name TEXT	-		§ 4.5.3
emaj_reset_group	group TEXT	#.tables.and.seq INT		§ 4.4.1
emaj_rollback_activity	-	SETOF emaj_rollback_activity_type	Yes	§ 4.8.2
emaj_rollback_group	group TEXT mark TEXT is_alter_group_allowed BOOLEAN	SETOF (severity TEXT, message TEXT)		§ 4.2.6
emaj_rollback_groups	groups.array TEXT[] mark TEXT is_alter_group_allowed BOOLEAN	SETOF (severity TEXT, message TEXT)		§ 4.2.6
emaj_set_mark_group	group TEXT [mark TEXT]	#.tables.and.seq INT		§ 4.2.5
emaj_set_mark_groups	groups.array TEXT[] [mark TEXT]	#.tables.and.seq INT		§ 4.2.5

Functions	Input parameters	Output data	Callable by emaj_viewer	Ref.
emaj_snap_group	group TEXT directory TEXT copy.options TEXT	#.tables.and.seq INT		§ 4.7.1
emaj_snap_log_group	group TEXT start.mark TEXT end.mark TEXT directory TEXT copy.options TEXT	#.tables.and.seq INT		§ 4.7.2
emaj_start_group	group TEXT [mark TEXT] [reset.loge BOOLEAN]	#.tables.and.seq INT		§ 4.2.4
emaj_start_groups	groups.array TEXT[] [mark TEXT] [reset.loge BOOLEAN]	#.tables.and.seq INT		§ 4.2.4
emaj_stop_group	group TEXT [mark TEXT]	#.tables.and.seq INT		§ 4.2.8
emaj_stop_groups	groups.array TEXT[] [mark TEXT]	#.tables.and.seq INT		§ 4.2.8
emaj_unprotect_group	group TEXT	0/1 INT		§ 4.4.3
emaj_unprotect_mark_group	group TEXT mark TEXT	0/1 INT		§ 4.5.6
emaj_verify_all	-	Setof TEXT	Yes	§ 4.8.1