

« E-Maj » PostgreSQL extension

-

User's Guide

Version: 1.0.1

Summary

1 - Introduction	5
1.1 - Document content	5
1.2 - License	5
1.3 - E-Maj's objectives	5
2 - How E-Maj works	7
2.1 - Concepts	7
2.1.1 - Tables Group	7
2.1.2 - Mark	7
2.1.3 - Rollback	7
2.2 - Architecture	8
2.2.1 - Logged SQL statements	8
2.2.2 - Created objects	8
2.2.3 - Norm for E-Maj objects naming	9
2.2.4 - Schemas	10
2.2.5 - Tablespaces	10
3 - How to install E-Maj	11
3.1 - extension download and decompression	11
3.1.1 - Download	11
3.1.2 - Decompression	11
3.2 - E-Maj extension setup	12
3.2.1 - Preliminary operations	12
3.2.2 - E-Maj components installation	13
3.2.3 - Changes in postgresql.conf configuration file	13
3.2.4 - Test and demonstration	14
3.3 - Update an existing E-Maj version	15
3.3.1 - General approach	15
3.3.2 - Un-registering an old E-Maj extension	15
3.3.3 - Migration by deletion and re-installation	16
3.3.4 - Migration from E-Maj 0.10.0 to 0.10.1	17
3.3.5 - Migration from E-Maj 0.10.1 to 0.11.0	17
3.3.6 - Migration from E-Maj 0.11.0 to 0.11.1	19
3.3.7 - Migration from E-Maj 0.11.1 to 1.0.0	19
3.3.8 - Migration from E-Maj 1.0.0 to 1.0.1	20
3.4 - E-Maj uninstall	21
4 - How to use E-Maj	22

4.1 - Set-up the E-Maj access policy	22
4.1.1 - E-Maj roles	22
4.1.2 - Giving E-Maj rights	22
4.1.3 - Giving rights on application tables and objects	22
4.1.4 - Synthesis	23
4.2 - Main functions	24
4.2.1 - Operations chain	24
4.2.2 - Define tables groups	25
4.2.3 - Create a tables group	27
4.2.4 - Start a tables group	28
4.2.5 - Set an intermediate mark	29
4.2.6 - Rollback a tables group	30
4.2.7 - Perform a logged rollback of a tables group	31
4.2.8 - Stop a tables group	32
4.2.9 - Drop a tables group	33
4.2.10 - Alter a tables group	34
4.3 - Multi-groups functions	36
4.3.1 - General information	36
4.3.2 - Functions list	36
4.3.3 - Syntax for groups array	36
4.3.4 - Other considerations	37
4.4 - Marks management functions	38
4.4.1 - Comments on marks	38
4.4.2 - Search a mark	38
4.4.3 - Rename a mark	39
4.4.4 - Delete a mark	39
4.4.5 - Delete oldest marks	40
4.5 - Statistics functions	41
4.5.1 - Global statistics about logs	41
4.5.2 - Detailed statistics about logs	42
4.5.3 - Estimate the rollback duration	43
4.6 - Data extraction functions	45
4.6.1 - Snap tables of a group	45
4.6.2 - Snap log tables of a group	46
4.6.3 - SQL script generation to replay logged updates	47
4.7 - Other functions	49
4.7.1 - Reset log tables of a group	49
4.7.2 - Comments on groups	49
4.7.3 - Check the consistency of the E-Maj environment	49
4.7.4 - Forced stop of a tables group	50
4.7.5 - Forced suppression of a tables group	51

4.8 - Parallel Rollback	52
4.8.1 - Sessions	52
4.8.2 - Prerequisites	52
4.8.3 - Syntax	52
4.8.4 - Examples	53
5 - Miscellaneous.....	55
5.1 - Parameters	55
5.2 - Internal checks	56
5.3 - Traces of operations	56
5.4 - Impacts on cluster and database administration	58
5.4.1 - Stopping and restarting the cluster	58
5.4.2 - Saving and restoring the database	59
5.4.3 - Data load	60
5.4.4 - Tables reorganisation	60
5.4.5 - Using E-Maj with replication	60
5.4.6 - PostgreSQL version upgrade	61
5.5 - Sensitivity to system time change	61
5.6 - Performances	63
5.6.1 - Updates recording overhead	63
5.6.2 - E-Maj rollback duration	63
5.6.3 - Optimizing E-Maj operations	63
5.7 - Usage limits	64
5.8 - User's responsibility	65
5.8.1 - Defining tables groups content	65
5.8.2 - Appropriate call of main functions	65
5.8.3 - Management of application triggers	65
5.8.4 - Internal E-Maj table or sequence change	66
6 - phpPgAdmin plugin.....	67
6.1 - General presentation	67
6.2 - Using phpPgAdmin plugin	67
6.2.1 - How to reach E-Maj from phpPgAdmin interface	67
6.2.2 - Tables groups list	68
6.2.3 - Tables groups composition	70
6.2.4 - Tables group details	71
6.2.5 - Statistics for rollback	73
6.2.6 - Statistics about log tables content	73
7 - Appendix.....	75
7.1 - E-Maj functions list	75

1 INTRODUCTION

1.1 DOCUMENT CONTENT

This document is a user's 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 in detail how to setup and use E-Maj.

Chapter 4 gives some additional information needed for a good understanding of how the extension works.

Then, chapter 5 presents the E-Maj extension of the administration tool phpPgAdmin.

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 ».

The main goal of E-Maj is to supply a way 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.

But E-Maj may also be used to trace updates performed by application programs on the table's content .

It provides a good solution to :

- define save points at precise time on a tables group,
- restore if needed this tables group into a stable state, without stopping the cluster,
- manage several save points, each of them being usable at any time as 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 databases into predefined stable states, so that tests can be replayed as many times as needed.

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 that are updated by one or several processings. Each tables group is defined by a name which must be unique inside its database. By extent, a tables group can also contain application sequences (in the RDBMS sense). Tables 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 later case, it is not possible to rollback the group. But with this type of group, it is possible to record tables updates for auditing purpose, 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 in resetting all tables and sequences of a group in the state they had when a mark has been set.

Actually, 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* », updates cancellations are recorded in log tables, so that they can be later cancelled: the rollback operation can be ... rolled back.

2.2 ARCHITECTURE

In order to be able to perform a rollback operation without having previously kept a physical image of the PostgreSQL cluster'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* (starting from PostgreSQL 8.4)

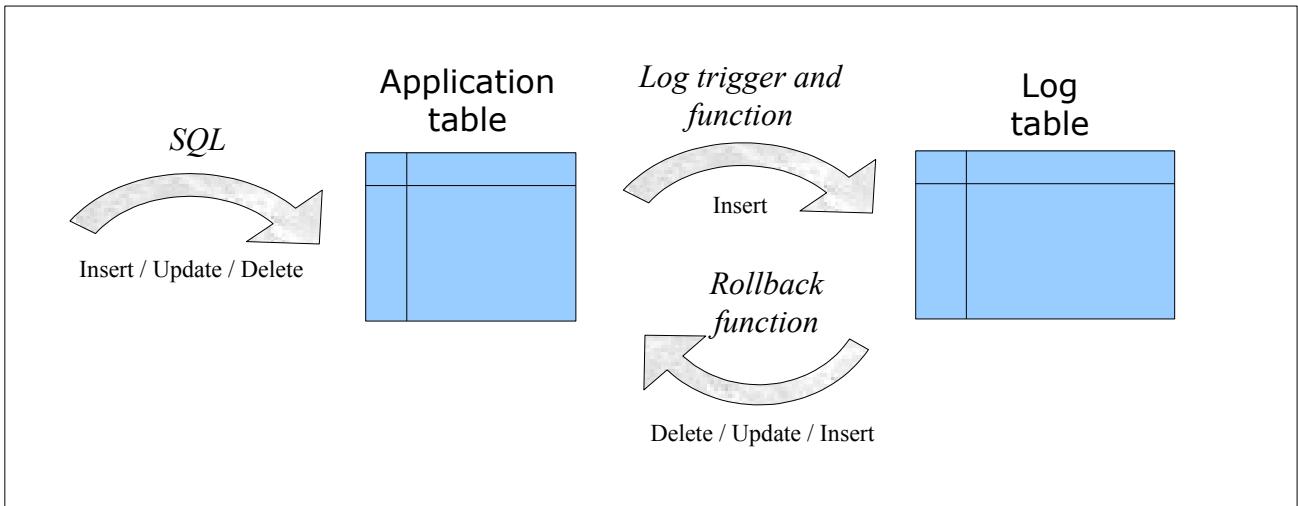
For statements that process several rows, each creation, update or deletion is individually recorded. For instance, if a “*DELETE FROM <table>*” is performed against a table having 1 million rows, 1 million row deletion events are recorded.

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 only accepted 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,
- a rollback function, that can cancel all or some recorded updates for the table, (except for tables belonging to group created in “*audit_only*” mode),
- starting from PostgreSQL 8.4, another trigger that blocks any execution of a *TRUNCATE* SQL verb while log triggers are activated,
- 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* et 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.

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

- 10 tables,
- 3 types,
- 75 technical functions, 35 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.

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 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 log trigger is:
 <schema.name>_<table.name>_emaj_log_trg
- the name of the trigger that blocks *TRUNCATE* verb is:
 <schema.name>_<table.name>_emaj_trunc_trg
- the name of the rollback function is:
 <schema.name>_<table.name>_rlbk_fnct
- the name of the sequence associated to the log table is:
 <schema.name>_<table.name>_log_seq

Other E-Maj functions name is also normalised:

- functions names that begins with 'emaj_' are functions that are callable by users,
- functions names that begins with '_' are internal functions that should not be called directly.

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' name starts with "emaj", only their suffix being parametrized in tables groups definition. (Refer to §4.2.2)

2.2.5 Tablespaces

When the extension is installed and when a log tables are created, E-Maj can use the default tablespace.

But it is also possible to create a dedicated tablespace named *tspemaj*. If it exists when the extension is installed or when a tables group is created, it will be used to hold new tables.

Using tables group parameters, it is also possible to store log tables and/or their index into specific tablespaces. (Refer to §4.2.2)

3 HOW TO INSTALL E-MAJ

In this chapter, we will describe how to download and install the E-Maj extension. A last chapter deals with uninstall.

3.1 EXTENSION DOWNLOAD AND DECOMPRESSION

3.1.1 Download

E-Maj is available for download on two Internet sites:

- PGXN, the PostgreSQL Extension Network (<http://pgxn.org>),
- pgFoundry.org (<http://pgfoundry.org/projects/emaj/>).

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. 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.sql* psql script to install the E-Maj components
- *sql/emaj-1.0.0-to-1.0.1.sql* psql script to migrate E-Maj from version 1.0.0 to 1.0.1
- *sql/emaj-0.11.1-to-1.0.0.sql* psql script to migrate E-Maj from version 0.11.1 to 1.0.0
- *sql/emaj-0.11.0-to-0.11.1.sql* psql script to migrate E-Maj from version 0.11.0 to 0.11.1
- *sql/check-0.10.1-to-0.11.0-conditions.sql* psql script that verifies that conditions to migrate from 0.10.1 to 0.11.0 are met
- *sql/emaj-0.10.1-to-0.11.0.sql* psql script to migrate E-Maj from version 0.10.1 to 0.11.0

- sql/emaj-0.10.0-to-0.10.1.sql psql script to migrate E-Maj from version 0.10.0 to 0.10.1
- sql/emaj--0.10.1--unpackaged.sql script to “deconstruct” an installed 0.10.1 E-Maj extension
- sql/emaj--0.10.0--unpackaged.sql script to “deconstruct” an installed 0.10.0 E-Maj extension
- sql/demo.sql psql E-Maj demonstration script
- sql/prep-pr.sql psql test script for parallel rollbacks
- sql/uninstall.sql psql script to uninstall the E-Maj components
- README reduced extension's documentation
- CHANGES release notes
- LICENSE information about E-Maj license
- AUTHORS who are the authors
- META.json technical data for PGXN
- 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

3.2 E-MAJ EXTENSION SETUP

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

Note that PostgreSQL versions 9.1+ include an integrated extensions management feature to simplify the installation of additional components into the RDBMS. Unfortunately, E-Maj characteristics do not allow to reliably use this integrated feature.

Some preliminary operations are required.

3.2.1 Preliminary operations

For these operations, the user must log on the concerned database as a superuser, using for instance *psql*.

If the PL/PGSQL language is not activated (it is not activated by default with PostgreSQL versions prior 9.0), it must be activated by the following SQL command:

```
CREATE LANGUAGE plpgsql;
```

The second preliminary operation is optional. It consists in creating the tablespace named *tspemaj*. If it exists, and except if specific parameters are set at tables groups definition

level (see §4.2.2), tables and indexes created by E-Maj will be stored in it. Once created, this tablespace is shared among all databases of the PostgreSQL cluster.

To create *tspemaj* tablespace, the associated storage space (a directory for Unix/Linux, or a folder for Windows) must first be created, this storage space being left empty. Then the following SQL command must be executed:

```
CREATE TABLESPACE tspemaj LOCATION '<tablespace.directory/folder>';
```

For performance reasons, it is recommended to put the *tspemaj* tablespace and the application tables on separate disk spaces.

3.2.2 E-Maj components installation

The E-Maj components can now be installed into the database, by executing under `psql` the supplied *emaj.sql* script.

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

To start with, the script verifies that the PostgreSQL version is at least 8.2, and that the current user has the *superuser* attribute.

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



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 cluster configuration and may display a warning message regarding the *-max-prepared-statements* parameter (see §4.8.2).

At the end of its execution, the script displays the following message:

```
>>> E-Maj objects successfully created
```

3.2.3 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 cluster. 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 § 4.8), it will be probably necessary to adjust the *max_prepared_transaction* parameter.

3.2.4 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 *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!
```

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 and the installation method that has been used.

For E-Maj versions prior 0.10.0, there is no particular update procedure. A simple E-Maj deletion and then re-installation has to be done, as described in §3.3.3. 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 0.10.0 and later, it is possible to perform a migration without E-Maj deletion. Depending on cases, this can be achieved in one or several steps:

- the migration from 0.10.0 to 0.10.1 is described in §3.3.4
- the migration from 0.10.1 to 0.11.0 is described in §3.3.5
- the migration from 0.11.0 to 0.11.1 is described in §3.3.6
- the migration from 0.11.1 to 1.0.0 is described in §3.3.7
- the migration from 1.0.0 to 1.0.1 is described in §3.3.8

But if E-Maj has been created with the integrated extensions manager (with a *CREATE EXTENSION* statement), it will be necessary to first de-register the extension, as described in §3.3.2.

3.3.2 Un-registering an old E-Maj extension

If E-Maj has been installed using a *CREATE EXTENSION* statement, it is necessary to un-register E-Maj from the integrated extensions management system.

To do this, just chain both following commands:

```
\i <emaj_directory>/sql/emaj--<emaj_version>--unpackaged.sql  
DROP EXTENSION emaj;
```

where <emaj_version> takes either 0.10.0 or 0.10.1 value, depending on the installed E-Maj version.

After this operation, *emaj extension* doesn't exist any more but all components it contained (tables, functions, types) still exist.

3.3.3 Migration by deletion and re-installation

For this migration 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.

3.3.3.1 Stop tables groups

If some tables groups are in *ACTIVE* state, they must be stopped, using the *emaj_stop_group()* function (see §4.2.8), or the *emaj_force_stop_group()* function is *emaj_stop_group()* (see §4.7.4) returns an error.

3.3.3.2 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 cluster 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 *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';
```

3.3.3.3 E-Maj deletion and re-installation

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

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

3.3.3.4 Restore user data

Data previously saved can now be restored into E-Maj 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.

3.3.4 Migration from E-Maj 0.10.0 to 0.10.1

If a 0.10.0 E-Maj version is already installed, it is possible to perform a simple E-Maj update to migrate into 0.10.1.

This update can be done without dropping existing tables groups, and even without stopping them if they are in active state. 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.

This migration is very quick. It only consists in adding or modifying a few functions.

To update E-Maj from 0.10.0 to 0.10.1, the *emaj-0.10.0-to-0.10.1.sql* delivered psql script must be executed.

```
\i <emaj_directory>/sql/emaj-0.10.0-to-0.10.1.sql
```

The script reports the list of tables groups that will need to be recreated after the installation to take benefit of all enhancements brought by the 0.10.1 version. But the next migration towards 0.11.0 will implicitly perform the requested changes.

At the end of its execution, the script displays the following message :

```
>>> E-Maj successfully migrated to 0.10.1
```

3.3.5 Migration from E-Maj 0.10.1 to 0.11.0

If a 0.10.1 E-Maj version is already installed, it is possible, under some conditions (see §3.3.5.1), to perform an E-Maj update to migrate into 0.11.0.

In this case, this update can be done without dropping existing tables groups, and even without stopping them if they are in active state. 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.

But this operation can take a long time. Indeed, with this version, some functions are created or updated, the `emaj_mark` table changes, and overall the log tables structure evolves. So the migration script must recreate all log tables. The duration of this step obviously depends on the existing log volume. To limit this migration duration, if this is acceptable, it may be preferable to delete the oldest marks or even to stop tables groups and purge log tables (`emaj_stop_group()` and `emaj_reset_group()` functions), or even to drop tables groups before the migration and recreate them after.

To be sure that log tables are not updated by other processes during the migration, an exclusive lock is set on all tables of all tables groups in `LOGGING` state. This means that this migration can only be achieved when there is no activity on tables protected by E-Maj.

3.3.5.1 Validating migration conditions

The migration reassign a new sequence number to each log row. This number is now unique inside the database. To guarantee the integrity of data stored by E-Maj, it is essential that no server time change in the past prevents to reliably serialize all log rows for all log tables.

A script, named `check-0.10.1-to-0.11.0-conditions.sql`, is supplied with the version. It analyses the E-Maj environment state and indicates whether the migration can be simply done.

```
\i <emaj_directory>/sql/check-0.10.1-to-0.11.0-conditions.sql
```

“*warning*” or “*notice*” messages may be generated by this script.

“*notice*” messages simply report de-synchronisation of two consecutive log rows but that is not blocking for a migration. On the contrary, “*warning*” messages report cases that the migration process cannot safely handle.

The executed function returns a text message representing the analysis result. If the migration is possible, the following message is returned:

This E-Maj environment can be migrated into 0.11.0.

In the other case, one gets this message:

This E-Maj environment can NOT be migrated into 0.11.0.

In this latest case, there are two solutions:

- either delete old marks to suppress periods of time that generates the issue,
- or drop and then recreate tables groups.

The migration script contains the same checks.

3.3.5.2 Updating E-Maj components

If the test presented in the previous chapter reports that the migration from 0.10.1 to 0.11.0 is possible, the *emaj-0.11.1-to-0.11.0.sql* delivered psql script can be executed.

```
\i <emaj_directory>/sql/emaj-0.10.1-to-0.11.0.sql
```

At the end of its execution, the script displays the following message :

```
>>> E-Maj successfully migrated to 0.11.0
```

3.3.6 Migration from E-Maj 0.11.0 to 0.11.1

If a 0.11.0 E-Maj version is already installed, it is possible to perform a simple E-Maj update to migrate into 0.11.1.

This update can be done without dropping existing tables groups, and even without stopping them if they are in active state. 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.

This migration is very quick.

To update E-Maj from 0.11.0 to 0.11.1, the *emaj-0.11.0-to-0.11.1.sql* delivered psql script must be executed.

```
\i <emaj_directory>/sql/emaj-0.11.0-to-0.11.1.sql
```

At the end of its execution, the script displays the following message :

```
>>> E-Maj successfully migrated to 0.11.1
```

3.3.7 Migration from E-Maj 0.11.1 to 1.0.0

If a 0.11.1 E-Maj version is already installed, it is possible to perform a simple E-Maj update to migrate into 1.0.0.

This update can be done without dropping existing tables groups, and even without stopping them if they are in active state. 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.

This migration is very quick.

To update E-Maj from 0.11.1 to 1.0.0, the *emaj-0.11.1-to-1.0.0.sql* delivered psql script must be executed.

```
\i <emaj_directory>/sql/emaj-0.11.1-to-1.0.0.sql
```

At the end of its execution, the script displays the following message :

```
>>> E-Maj successfully migrated to 1.0.0
```

3.3.8 Migration from E-Maj 1.0.0 to 1.0.1

If a 1.0.0 E-Maj version is already installed, it is possible to perform a simple E-Maj update to migrate into 1.0.1.

This update can be done without dropping existing tables groups, and even without stopping them if they are in active state. 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.

This migration is very quick.

To update E-Maj from 1.0.0 to 1.0.1, the *emaj-1.0.0-to-1.0.1.sql* delivered psql script must be executed.

```
\i <emaj_directory>/sql/emaj-1.0.0-to-1.0.1.sql
```

At the end of its execution, the script displays the following message :

```
>>> E-Maj successfully migrated to 1.0.1
```

3.4 E-MAJ UNINSTALL

If some tables groups in logging state remain, they must be stopped with the *emaj_stop_group()* function (see § 4.2.8) or the *emaj_force_stop_group()* function (see §4.7.4) if the *emaj_stop_group()* function returns an error.

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.

Then, the *uninstall.sql* script delivered with the installed E-Maj version has to be executed.

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

This script drops all schemas of the extension (the *emaj* main schema and the secondary schemas that may exist), with all contained objects. It also drops objects created by the *demo.sql* script and that would not have been previously suppressed.

If *emaj_adm* and *emaj_viewer* roles are not associated to other roles or other databases in the cluster, and do not own rights on other tables, they are dropped.

However, the *tspemaj* tablespace, if it exists, or any other tablespaces containing log tables and indexes are NOT dropped by the script.

4 HOW TO USE E-MAJ

4.1 SET-UP THE E-MAJ ACCESS POLICY

A bad usage of E-Maj can break databases 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 reason, it is preferable to take advantage of both roles created by the installation script:

- *emaj_adm* is used as 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.

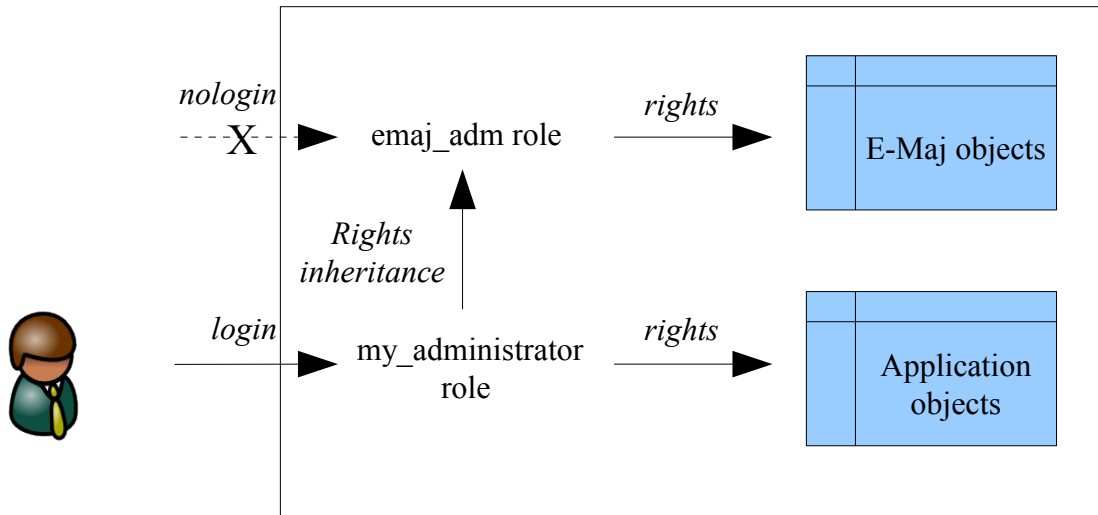
4.1.3 Giving rights on application tables and objects

To let an E-Maj administrator also access to application tables or to 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.

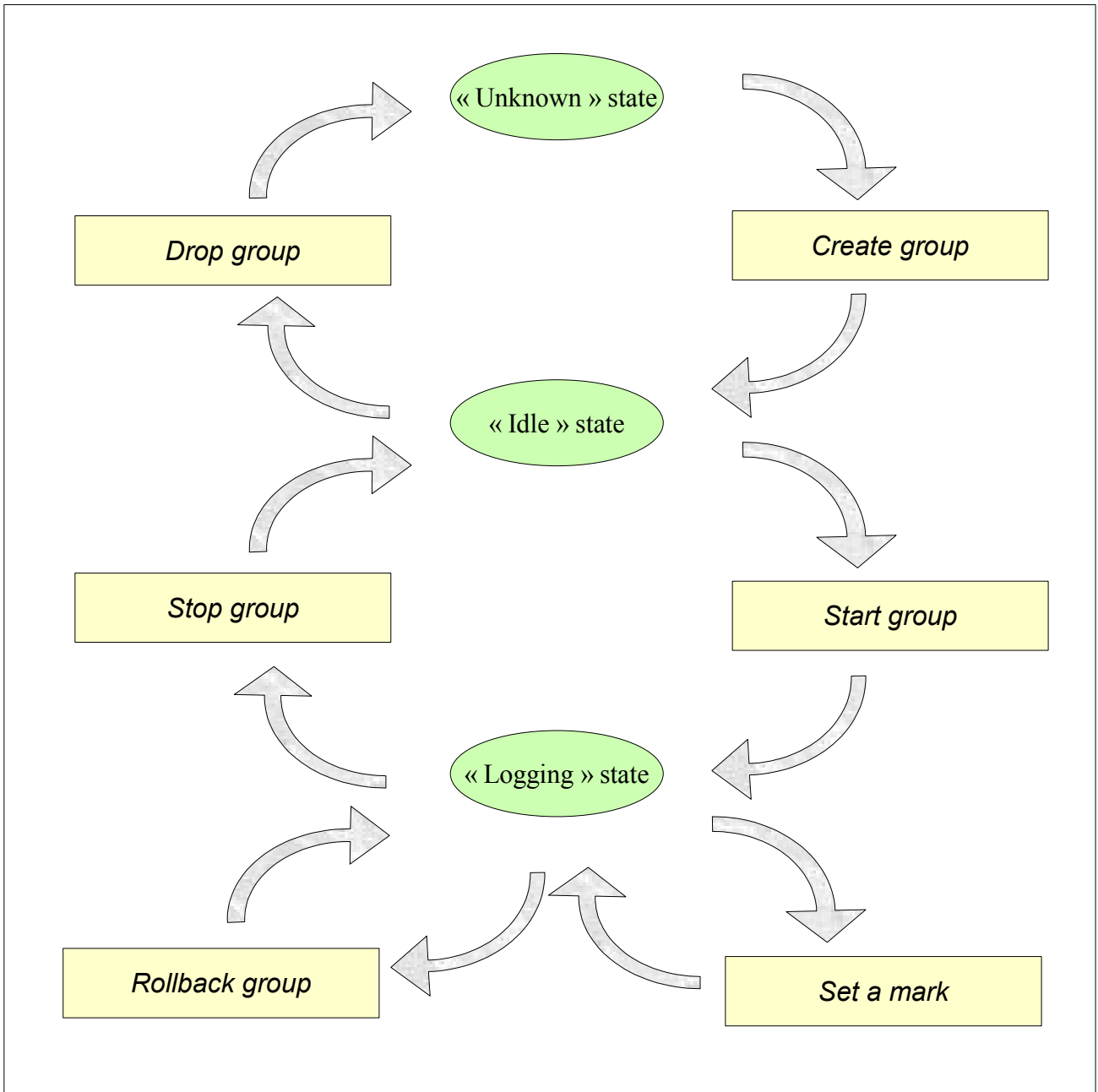
Except when explicitly noticed, the operations presented later can be indifferently executed by a superuser or by a role belonging to the *emaj_adm* group.

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
<i>grpdef_group</i>	TEXT	tables group name
<i>grpdef_schema</i>	TEXT	name of the schema containing the application table or sequence
<i>grpdef_tblseq</i>	TEXT	application table or sequence name
<i>grpdef_priority</i>	INT	priority level for the table or sequence in E-Maj processing (optional)
<i>grpdef_log_schema_suffix</i>	TEXT	suffix used to build the name of the schema containing the E-Maj objects for the table (optional)
<i>grpdef_log_dat_tsp</i>	TEXT	name of the tablespace containing the log table (optional)
<i>grpdef_log_idx_tsp</i>	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 table's 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 affected to several 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 affected to a group not created in "audit_only" mode must have an explicit primary key (***PRIMARY KEY*** clause in ***CREATE TABLE*** or ***ALTER TABLE***).

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 sequence 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 processings. 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 installation 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 allow to specify the schema that will hold the log table, the log sequence and the log and rollback functions for a particular application table.

If this ***grpdef_log_schema_suffix*** column contains a NULL or an empty chain, the ***emaj*** main schema will be used. Otherwise, the 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.

To optimize performances of E-Maj installation 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 tablespace that is used at tables group creation is either *tspemaj* if it exists or the default tablespace of the current session.

If a column *grpdef_log_dat_tsp* or *grpdef_log_idx_tsp* is not NULL, its value is used to define the tablespace to use at tables group creation.

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 abbreviate 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 (starting PostgreSQL 8.4). If the group is “*rollbackable*”, the rollback function is also created.

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 to a *SERIAL* or *BIGSERIAL* column and the table that owns this column does not belong to the same tables group, the function also issues a *WARNING* message.

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 § 4.2.10) 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>',  
[<are.old.logs.to.be.deleted?>]);
```

or in an abbreviate form:

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

The group must be first in *IDLE* state.

A mark name must be specified. It will be the first mark on which a rollback will be later possible.

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 empty or *NULL*, a name is automatically generated: « *MARK_%* », where the '%' character represents the current transaction start time.

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 set an *ACCESS EXCLUSIVE* lock on each table of the group. If transactions accessing these tables are running, this can lead to *deadlock*. If the *deadlock* processing impacts the execution of the E-Maj function, the error is trapped and the lock operation is repeated, with a maximum of 5 attempts.

The function also performs a purge of the oldest events in the *emaj_hist* technical table (see §5.3).

When a group is started, its state becomes « *LOGGING* ».

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 empty or *NULL*, a name is automatically generated: « *MARK_%* », where the '%' character represents the current transaction start time.

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

Note that it is possible to set two consecutive marks without any update on any table between these marks.

The `emaj_set_mark_group()` function set *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.

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

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 function returns the number of tables and sequences that have been **effectively** modified by the rollback operation.

To be sure that no concurrent transaction updates any table of the group during the rollback operation, the `emaj_rollback_group()` function explicitly set 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 deactivate them before the rollback and re-activate them after (see §5.8.3).

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.

When the rollback operation is completed, are deleted:

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

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



By nature, the reset of sequence 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.

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

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 function returns the number of tables and sequences that have been **effectively** modified by the rollback operation.

To be sure that no concurrent transaction updates any table of the group during the rollback operation, the *emaj_rollback_group()* function explicitly set 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 deactivate them before the rollback and re-activate them after (see §5.8.3).

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 post 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”.

Then, it is possible to continue updating processings, 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 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)
```

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>');
```

or, in its abbreviated form:

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

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

The function automatically sets a mark corresponding to the end of the recording. If a mark name is supplied in parameters, this name is used. Otherwise, the mark is named:

`STOP_<stop_time>`

where `<stop_time>` is expressed as “hours.minutes.seconds.milliseconds”.

If the supplied mark name is NULL or equals an empty string the mark is named:

`MARK_<stop_time>`

Stopping a tables group simply deactivate log triggers of application tables of the group. The locks set 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 twice the `emaj_stop_group()` function for a tables group already stopped does not generate an error. Only a warning message is returned.

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 (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 by 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 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.

4.2.10 Alter a tables group

Two types of events may lead to alter a tables group:

- the tables group definition may change, some tables or sequences been added or suppressed, or one of the parameters linked to a table (priority, schema, or tablespaces) been modified,
- the structure of one or several application tables of the tables group may change, a added or dropped column or a change in a column type having an impact in the log table structure.

In both cases, the following steps must 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 *emaj_drop_group()* and *emaj_create_group()* functions.

But this last step can be also performed with 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 *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()*.

It is possible to anticipate the update of the *emaj_group_def* table, even if the tables group is in logging state.

In case of discrepancy between the structure of both application and related log tables, E-Maj generates an error at start group time, or set mark time or rollback time.

4.3 MULTI-GROUPS FUNCTIONS

4.3.1 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.

4.3.2 Functions list

The following functions process several groups:

- `emaj.emaj_start_groups(<groups.array>,<start.mark>[,<reset.log>])` starts several groups,
- `emaj.emaj_stop_groups(<groups.array>[,<stop.mark>])` stops several groups,
- `emaj.emaj_set_mark_groups(<groups.array>,<mark>)`, sets a mark on several groups,
- `emaj.emaj_rollback_groups(<groups.array>,<mark>)` process a simple rollback for several groups,
- `emaj.emaj_logged_rollback_groups(<groups.array>,<mark>)` process a “logged” rollback for several groups.

4.3.3 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 simple 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.

4.3.4 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. So a set oriented build of this list is possible (using for instance the *array_agg()* function that is available from PostgreSQL version 8.4).

It is also permitted to have a list with duplicate values, NULL values or empty strings. In all these cases, the value is ignored and a warning message is generated, like in case of empty list.

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.

4.4 MARKS MANAGEMENT FUNCTIONS

4.4.1 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 the E-Maj phpPgAdmin plug-in (See §6). Indeed, the plug-in systematically displays the comments in the groups marks list.

4.4.2 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.

This function is particularly useful when used with the *emaj_delete_before_mark_group()*. 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));
```

4.4.3 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.

4.4.4 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.

4.4.5 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 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 during 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 (as with the *emaj_start_group()* or *emaj_reset_group()* functions), using *emaj_delete_before_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 updates 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.

4.5 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()* brings a more detailed information than *emaj_log_stat_group()*, the number of updates been reported per table, SQL type (INSERT/UPDATE/DELETE) and connection role.

Another E-Maj function, *emaj_estimate_rollback_duration()*, provides an estimate of how long a rollback for a group to a given mark may last.

Finally, a function named *emaj_get_previous_mark_group()*, returns for a group the name of the latest mark preceding a given date and time.

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

4.5.1 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, once the *test-emaj-2.sql* test script has been executed, 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 tables updates before been cancelled.

4.5.2 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*, *emaj_detailed_log_stat_group* function doesn't return any row 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, once the *test-emaj-2.sql* test script has been executed, 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('myApp11', NULL, NULL)
WHERE stat_table='mytbl1';
 stat_table | stat_verb | stat_rows
-----+-----+-----
 mytbl1    | DELETE   |         1
 mytbl1    | INSERT   |         6
 mytbl1    | UPDATE   |         2
(3 rows)
```

4.5.3 Estimate the rollback duration

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

```
SELECT emaj.emaj_estimate_rollback_duration('<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 an *INTERVAL* data.

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,
- 5 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.

4.6 DATA EXTRACTION FUNCTIONS

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

4.6.1 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 on 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 cluster can write in it.

The third parameter defines the output files format. It is a characters 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 with the same name as existing files will overwrite them.

Created files name has the following pattern:

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

Each file corresponding to a sequence has only one row, that contains 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.

4.6.2 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 provide a way to archive updates performed by one or several processings. 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. They represents then the first known mark.

A `NULL` value or an empty string may be used as end mark. They represents then the current situation.

The keyword '`EMAJ_LAST_MARK`' can be used as mark name. It then represents 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 cluster can write in it.

The fifth parameter defines the output files format. It is a characters 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 with the same name as existing files will overwrite them.

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, the marks 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 their related application table. They 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)

4.6.3 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_generate_sql()` 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_generate_sql('<group.name>', '<start.mark>',  
'<end.mark>', '<file>');
```

A `NULL` value or an empty string may be used as start mark. They represents then the first known mark.

A `NULL` value or an empty string may be used as end mark. They represents then the current situation.

The keyword '*EMAJ_LAST_MARK*' can be used as mark name. It then represents 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 cluster can write into it. If the file already exists, its content is overwritten.

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.

The function is only available with PostgreSQL version 8.3 and higher.

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

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

They are inserted into a single transaction. They are surrounded by a *BEGIN TRANSACTION;* statement and a *COMMIT;* statement. An initial comment reminds 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 leads to have doubled backslashes in 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 exists before executing the generated script.

4.7 OTHER FUNCTIONS

4.7.1 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.

4.7.2 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.

4.7.3 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 than those performed when a group is started, a mark is set or a rollback is executed (see §5.2).

The function returns a set of rows describing the detected discrepancies. If no error is detected, the function returns a single rows 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.

4.7.4 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 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.

4.7.5 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.

NB: Since the *emaj_force_stop_group()* has been created, this *emaj_force_drop_group()* function should likely be useless. So it may disappear in a future version.

4.8 PARALLEL ROLLBACK

On servers having several processors or processor cores, it may be interesting to reduce rollback elapse time by paralleling the operation on several corridors. 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.

4.8.1 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 each own corridor.

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

To built 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.

4.8.2 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 cluster).

Rollbacking 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 *postgres.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.

4.8.3 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)
- 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 let the rollback operation 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.

In order to test the *emajParallelRollback.php* command, the E-Maj extension supplies a test script, *prep-pr.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.

4.8.4 Examples

The command:

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

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

5 MISCELLANEOUS

5.1 PARAMETERS

The E-Maj extension works with some parameters. Those are stored into the *emaj_param* internal table.

emaj_param table structure is the following:

Column	Type	Description
<i>param_key</i>	TEXT	keyword identifying the parameter
<i>param_value_text</i>	TEXT	parameter value, if its type is text (otherwise NULL)
<i>param_value_int</i>	INT	parameter value, if its type is integer (otherwise NULL)
<i>param_value_boolean</i>	BOOLEAN	parameter value, if its type is boolean (otherwise NULL)
<i>param_value_interval</i>	INTERVAL	parameter value, if its type is time interval (otherwise NULL)

The E-Maj extension installation procedure inserts a single row into *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 *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 = 5 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 § 4.5.3).
- *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 § 4.5.3).
- *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 § 4.5.3).
- *fixed_table_rollback_duration* (interval) default value = 5 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 § 4.5.3).
- *fixed_table_with_rollback_duration* (interval) default value = 2,5 ms ; defines an additional fixed rollback cost for any table that effectively has updates to rollback ;

can be modified to better represent the performance of the server that hosts the database (see § 4.5.3).

- *history_retention* (interval) default value = 1 mois ; 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 , different from the default value (1 month):

```
INSERT INTO emaj.emaj_param (param_key, param_value_interval) VALUES ('history_retention','15 days'::interval);
```

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

5.2 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 verifications include:

- a check that the PostgreSQL version at tables group creation time is compatible with the current version,
- a check that each application sequence or table of the group always exists,
- a check that each table of the group has its log table, its log and rollback functions and its triggers,
- a check that the log tables structure always reflects the related application tables structure.

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*.

emaj_hist table structure is the following:

Column	Type	Description
<i>hist_id</i>	BIGSERIAL	serial number identifying a row in this history table

<i>hist_datetime</i>	TIMESTAMPTZ	recording date and time of the row
<i>hist_function</i>	TEXT	function associated to the traced event
<i>hist_event</i>	TEXT	kind of event
<i>hist_object</i>	TEXT	object related to the event (group, table or sequence)
<i>hist_wording</i>	TEXT	additional comments
<i>hist_user</i>	TEXT	role whose action has generated the event
<i>hist_txid</i>	BIGINT	identifier of the transaction that has generated the event

The *hist_function* column can take the following values:

- *EMAJ_INSTALL* E-Maj installation or version update
- *CREATE_GROUP* tables group creation
- *COMMENT_GROUP* comment set on a group
- *DROP_GROUP* tables group suppression
- *ALTER_GROUP* tables group change
- *FORCE_DROP_GROUP* tables group forced suppression
- *START_GROUP* tables group start
- *START_GROUPS* tables groups start
- *STOP_GROUP* tables group stop
- *STOP_GROUPS* tables groups stop
- *FORCE_STOP_GROUP* tables group forced stop
- *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
- *SET_MARK_GROUP* mark set on a tables group
- *SET_MARK_GROUPS* mark set on several tables groups
- *COMMENT_MARK_GROUP* comment set on a mark for a tables group
- *DELETE_MARK_GROUP* mark deletion for a tables group
- *RENAME_MARK_GROUP* mark rename for a tables group
- *ROLLBACK_GROUP* rollback updates for a tables group
- *ROLLBACK_GROUPS* rollback updates for several tables groups
- *RESET_GROUP* log tables content reset for a group
- *ROLLBACK_TABLE* rollback updates for one table
- *ROLLBACK_SEQUENCE* rollback one sequence
- *SNAP_GROUP* snap all tables and sequences for a group
- *SNAP_LOG_GROUP* snap all log tables for a group
- *GENERATE_SQL* generation of a psql script to replay updates

the *hist_event* column can take the following values:

- *BEGIN*
- *END*
- *MARK DELETED*
- *SCHEMA CREATED* secondary schema created

➤ *SCHEMA DROPPED* secondary schema dropped

emaj_hist content can be viewed by anyone who has the proper access rights on this table (superuser, *emaj_adm* or *emaj_viewer* roles).

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. By default, the retention delay for events equals 1 month. But this value can be modified at any time by inserting the *history_retention* parameter into *emaj_param* table with a SQL statement (see § 5.1).

5.4 IMPACTS ON CLUSTER AND DATABASE ADMINISTRATION

5.4.1 Stopping and restarting the cluster

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

5.4.1.1 General rule

At cluster restart, all E-Maj objects are in the same state as at cluster stop: log triggers of active tables groups remains enabled and log tables contain cancel-able updates already recorded.

If a transaction with table updates were not committed at cluster stop, it would be rolled back during the recovery phase of the cluster 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,...

5.4.1.2 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 for application sequences are processed at the very end end 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 a cluster stop during an E-Maj rollback execution, it is recommended to rerun this rollback just after the cluster restart, to ensure that application sequences and tables remain properly synchronised.

5.4.2 Saving and restoring the database



Using E-Maj may lead to reduce 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.

5.4.2.1 File level saves and restores

When saving or restoring clusters at file level, it is essential to save or restore ALL cluster files. This includes of course all files from the *tspemaj* tablespace, if it exists.

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.

5.4.2.2 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 nothing to take care 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.

5.4.2.3 Logical save and restore of partial database

With *pg_dump* and *pg_restore* tools, database administrators can perform 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.

5.4.3 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.

5.4.4 Tables reorganisation

5.4.4.1 Reorganisation of application table

Application tables protected by E-Maj can be reorganised using the SQL *CLUSTER* command. Log triggers being enabled or not, the organisation process has no impact on log tables content.

5.4.4.2 Reorganisation of E-Maj tables

No table from E-Maj schemas (neither log tables nor technical tables) has “cluster” index. So using E-Maj has no operational impact regarding the execution of *CLUSTER* SQL commands at database level.

5.4.5 Using E-Maj with replication

5.4.5.1 Integrated replication

E-Maj est totally compatible with the use of the different PostgreSQL integrated replication modes (WAL archiving and PITR, asynchronous and synchronous Streaming Replication). Indeed, all E-Maj objects hosted in the cluster are replicated like all other objects of the cluster.

However, because of the way PostgreSQL manages sequences, the sequences current values may be a little forward on slave clusters than on the master cluster. For E-Maj, this

may lightly overestimate the number of log rows in general statistics. But there is no consequence on the data integrity.

5.4.5.2 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 to replicate log tables and E-Maj technical tables.

5.4.6 PostgreSQL version upgrade

5.4.6.1 PostgreSQL versions 8.2 and 8.3

Tables groups that are created in PostgreSQL version 8.2 or 8.3 can only be managed in their creation version. Indeed, in these PostgreSQL versions, the E-Maj functions behave not totally similarly.

For this reason, when upgrading from 8.2 or 8.3 to a higher version, it is necessary to uninstall and then reinstall E-Maj (see §3.4 and §3.2). As a consequence, it is not possible to keep tables groups in logging state while migrating the PostgreSQL version.

5.4.6.2 PostgreSQL versions 8.4 and later

For all PostgreSQL version greater or equal to 8.4, E-Maj objects and functions are identical.

So it is possible to upgrade the PostgreSQL version without E-Maj re-installation. Tables groups may remain in logging state at PostgreSQL upgrade.

However, it is recommended to stop tables groups before the PostgreSQL upgrade, tables being normally in a stable state at that time. Moreover, if the PostgreSQL version upgrade is performed using a database dump and restore, the execution of an *emaj_reset_group()* function may reduce the volume of data to manipulate, thus reducing the time needed for the operation.

5.5 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 PostgreSQL cluster.

Date and time of each update or each mark is recorded. But nothing but 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.

5.6 PERFORMANCES

5.6.1 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.

5.6.2 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 locks set time.

To get an order of magnitude of an E-Maj rollback duration, it is possible to use the *emaj_estimate_rollback_duration()* (Cf §4.5.3).

5.6.3 Optimizing E-Maj operations

Here are some advice to optimize E-Maj operations:

5.6.3.1 Use tablespaces

Creating tables into tablespaces located in dedicated disks or file systems is a way to more efficiently spread the accesses to these tables. To minimize the disturbances of application tables accesses by log tables accesses, the E-Maj administrator has two ways to use tablespaces for log tables and indexes location.

By creating a tablespace named *tspemaj* before the tables groups creation, log tables are created by default into this tablespace, without any additional action.

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)

5.6.3.2 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 at the end of the rollback function execution, once all log tables are processed. This generally speeds up a lot the rollback operation.

5.7 USAGE LIMITS

The E-Maj extension usage has some limits.

- The minimum required PostgreSQL version is 8.2.
- All tables belonging to a “rollbackable” tables group must have an explicit *PRIMARY KEY*.
- For a table declared in a group, the sum of the schema name length and the table name length can not exceed 52 characters.
- The schema named “*emaj*” is created at E-Maj initialisation. If its name should be changed, the *emaj.sql* scripts, as well as test scripts and the *emajParallelRollback.php* command should be adapted consequently.
- 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. Starting from 8.4 PostgreSQL version, a trigger, created by E-Maj, blocks any *TRUNCATE* statement on any table belonging to a tables group in logging state. For older PostgreSQL versions, this detection is not possible.
- 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 able to enter into a group's definition until this group be 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 its 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 at 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.

5.8 USER'S RESPONSABILITY

5.8.1 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 processing are really included in a single tables group.

5.8.2 Appropriate call of main functions

emaj_start_group(), *emaj_set_mark_group()*, *emaj_rollback_group()* and *emaj_rollback_and_stop_group()* 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 stable point in the life of these tables.

5.8.3 Management of application triggers

Triggers may have been created on application tables. It is not rare that these triggers performs one or several 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 adjust 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, only deactivating the trigger may not be sufficient.

5.8.4 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 may lead to data corruption during rollback operations.

6 PHPPGADMIN PLUGIN

To make E-Maj use easier, a plugin for the phpPgAdmin 5 administration tool is also available.

6.1 GENERAL PRESENTATION

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, E-Maj objects are accessible. It is then possible to:

- see the list of tables groups and perform any possible action, depending on groups state (start, stop, set or remove a mark, rollback, add or modify a comment),
- define or modify groups composition,
- 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.

6.2 USING PHPPGADMIN PLUGIN

6.2.1 How to reach E-Maj from phpPgAdmin interface

On figure 1 below, once being connected with a proper role to a database where the E-Maj extension has been installed, a new icon is easily visible 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.

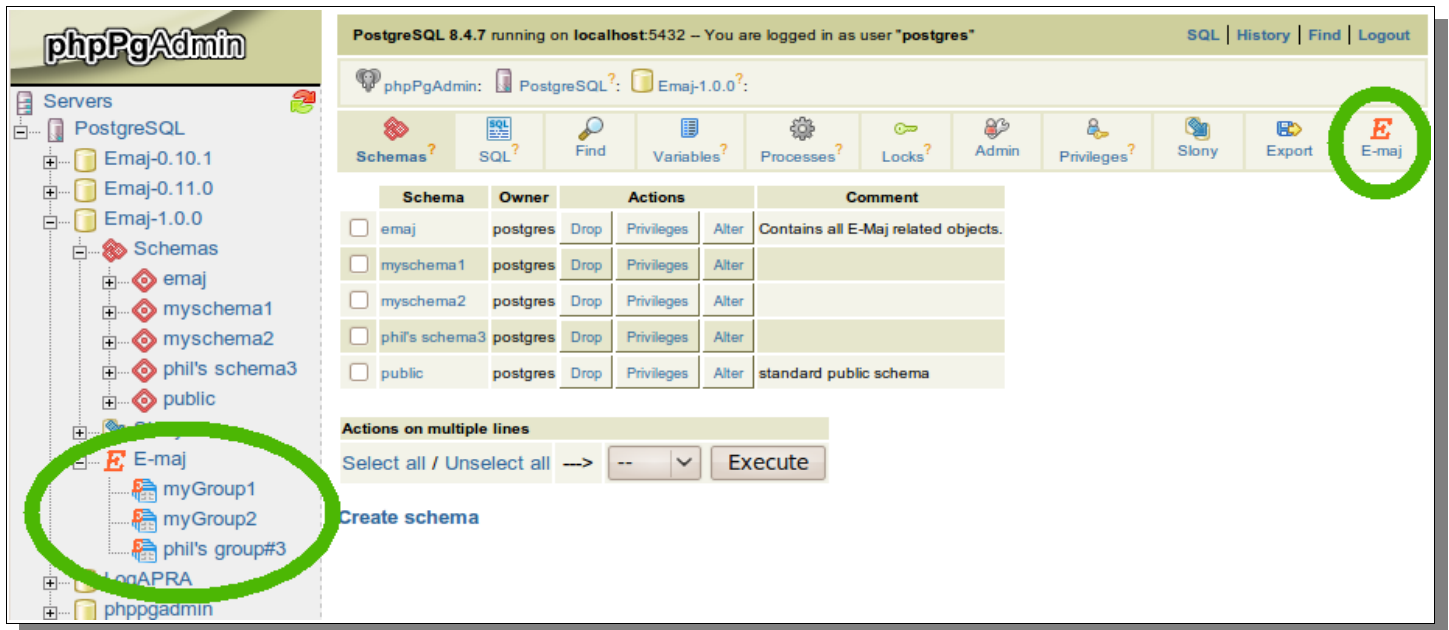


Figure 1 – Connection to a database where E-Maj is installed.

6.2.2 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 (see figure 2 above).

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

Below these both groups lists, a list box presents tables groups that may be created (those known in the *emaj_group_def* table but not yet created).

At the bottom of the page, a link allows the user to define or modify the tables groups content.

All pages displayed by the E-Maj plugin have a header that contains:

- a button to refresh the current page,
- the time at current page display,
- the E-Maj version installed on the database, a click on this field verifying the state of the E-Maj environment,
- the disk space used by the E-Maj tablespace and the part of the whole database it represents,
- the page title.

phpPgAdmin: PostgreSQL? Emaj-1.0.0?

Schemas? SQL? Find Variables? Processes? Locks? Admin Privileges? Slony Export E-maj

21:29:08 E-Maj 1.0.0 [392 kB = 4,8%] - Tables groups list

Tables groups in "LOGGING" state :

	Group	Creation date/time	# tables	# sequences	Type	# marks	Actions						Comment
<input type="checkbox"/>	myGroup1	12/11/2012 21:24:32	5	1	ROLLBACK-ABLE	3	Detail	Stop	Set a mark	Rollback	Content	Set a comment	Useless comment!
<input type="checkbox"/>	myGroup2	12/11/2012 21:24:33	4	2	ROLLBACK-ABLE	2	Detail	Stop	Set a mark	Rollback	Content	Set a comment	

Actions on multiple lines

Select all / Unselect all --> Set a mark v Execute

Tables groups in "IDLE" state :

	Group	Creation date/time	# tables	# sequences	Type	# marks	Actions						Comment	
<input type="checkbox"/>	phil's group#3	12/11/2012 21:24:33	2	1	AUDIT-ONLY	0	Detail	Start	Reset	Drop	Alter	Content	Set a comment	

Actions on multiple lines

Select all / Unselect all --> Start v Execute

Creation of a new tables group

dummyGrp1 v Create

Define groups' content

Figure 2 – List of the tables groups created in the database, with two groups in logging state and one group in idle state.

For each tables group, are displayed the following attributes:

- its creation date and time,
- the number of application tables and sequences it contains,
- its type (« ROLLBACKABLE » or « AUDIT_ONLY »),
- 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 each list, a combo box and a button are dedicated to multi-groups actions.

6.2.3 Tables groups composition

With a click on the link located at the bottom of tables groups list page, the user reach 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.

The screenshot shows the phpPgAdmin interface for PostgreSQL 8.4.7. The top navigation bar includes links for SQL, History, Find, and Logout. Below the navigation bar, there are icons for various database management functions: Schemas, SQL, Find, Variables, Processes, Locks, Admin, Privileges, Slony, Export, and E-maj. The main content area displays the 'Tables groups' setup for the 'myschema1' schema. It includes a 'Back to the tables groups list' link and a table of application schemas. Below this, a table lists the tables and sequences in the 'myschema1' schema, including their types, names, groups, priorities, and actions.

Application schemas list

Schema	Owner	Comment
myschema1	postgres	
myschema2	postgres	
phil's schema3	postgres	
public	postgres	standard public schema
dummySchema		

Tables and sequences in schema « myschema1 »

	Type	Schema	Name	Group	Priority	Log schema suffix	Log tablespace	Log index tablespace	Actions	Owner	Tablespace	Comment
<input type="checkbox"/>	Table	myschema1	myTbl3	myGroup1	10				Remove	postgres		
<input type="checkbox"/>	Sequence	myschema1	myTbl3_col31_seq	myGroup1	1				Remove	postgres		
<input type="checkbox"/>	Table	myschema1	mytbl1	myGroup1	20				Remove	postgres		
<input type="checkbox"/>	Table	myschema1	mytbl1	dummyGrp3					Remove	postgres		
<input type="checkbox"/>	Table	myschema1	mytbl2	myGroup1					Remove	postgres		
<input type="checkbox"/>	Table	myschema1	mytbl2b	myGroup1					Remove	postgres		
<input type="checkbox"/>	Sequence	myschema1	mytbl2b_col20_seq						Assign	postgres		
<input type="checkbox"/>	Table	myschema1	mytbl4	myGroup1	20				Remove	postgres		
<input type="checkbox"/>	!	myschema1	dummyTable	dummyGrp2					Remove			

Figure 3 – 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).

Are listed for each table or sequence:

- the tables group it belongs to, if any,

- the attributes of the table or sequence in the *emaj_group_def* table, if it is already affected to it: (see §4.2.2):
 - the priority level in the group,
 - the suffix that defines log schema
 - the optional tablespace name for the log table,
 - the optional tablespace name for the log table's index,
- the identity of its owner,
- the tablespace it belongs to, if any
- the associated comment in the database.

Both schemas list and the tables and sequences list also contains 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 "owner" column.

With a button, it is possible to either assign the table or the sequence to a new or already known tables group, or detach it from the tables group it is already assigned.

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

6.2.4 Tables group details

From the tables groups list page, it is possible to get more information about a particular tables group by clicking on its name or on its « Detail » button.

PostgreSQL 8.4.7 running on localhost:5432 – You are logged in as user "postgres" SQL | History | Find | Logout

phpPgAdmin: PostgreSQL? E-maj-1.0.0?

Schemas? SQL? Find Variables? Processes? Locks? Admin Privileges? Slony Export E-maj

21:52:35 E-Maj 1.0.0 [392 kB = 4,8%] - Tables group's detail

[Back to the tables groups list](#) | [Go to bottom](#)

Tables group « myGroup1 » characteristics:

State	Creation date/time	# tables	# sequences	Type	# marks	Log size	Actions
Logging	2012-11-12 21:24:32.617445+01	5	1	ROLLBACK-ABLE	3	160 kB	Stop Set a mark Content Set a comment

Comment : Useless comment!

Tables group « myGroup1 » marks:

Mark	Date/Time	State	# row updates	Cumulative updates	Actions						Comment
MARK3	2012-11-12 21:24:35.185207+01	Active	0	0	Rollback Stats	Rollback	Rename	Delete	First mark	Set a comment	
MARK2	2012-11-12 21:24:35.079443+01	Active	7	7	Rollback Stats	Rollback	Rename	Delete	First mark	Set a comment	End of 1st program
MARK1	2012-11-12 21:24:34.936043+01	Active	19	26	Rollback Stats	Rollback	Rename	Delete	First mark	Set a comment	

Statistics:

Range start: Range end: [Global stats](#) [Detailed stats](#)

[Back to the tables groups list](#) | [back to top](#)

Figure 4 – 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 and a set of button to perform the possible actions its state allows.

This line is followed by the group's comment if any has been stored for this group.

Then, the user can see a list of all marks that have been set on the group. For each of them, are displayed:

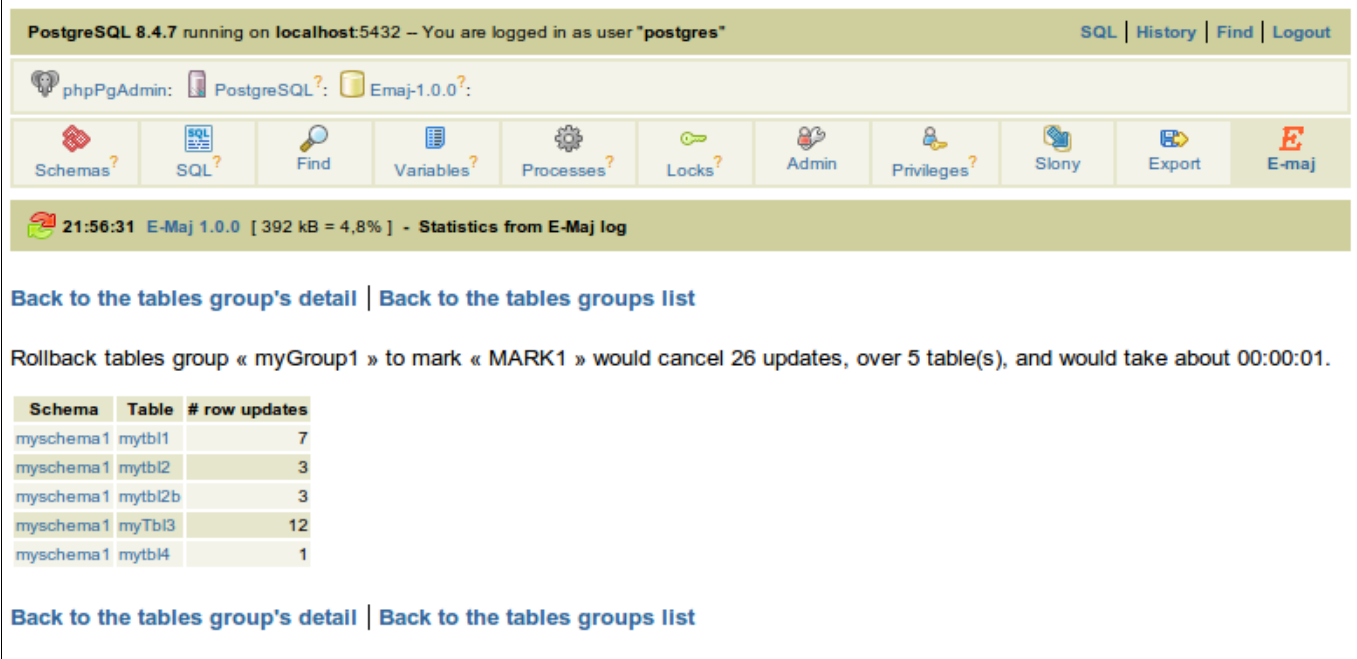
- its name,
- the date and time it has been set,
- its state,
- the number of recorded log rows between this mark and the next one (or the current situation if this is the last set mark),
- the total number of recorded log rows from when the mark has been set,
- the comment associated to the mark if it exists.

Several buttons are available to perform the actions permitted by its state.

A last line brings a way to get either global or detailed statistics between 2 marks or between a mark and the current situation.

6.2.5 Statistics for rollback

The figure below shows a page obtained by clicking on the « Rollback Stat » button of a mark.



The screenshot shows the phpPgAdmin interface for PostgreSQL 8.4.7. The top navigation bar includes links for SQL, History, Find, and Logout. Below the navigation bar, there are several menu items: Schemas, SQL, Find, Variables, Processes, Locks, Admin, Privileges, Slony, Export, and E-maj. The main content area displays the title "21:56:31 E-Maj 1.0.0 [392 kB = 4,8%] - Statistics from E-Maj log". Below the title, there are two links: "Back to the tables group's detail" and "Back to the tables groups list". The main text states: "Rollback tables group « myGroup1 » to mark « MARK1 » would cancel 26 updates, over 5 table(s), and would take about 00:00:01." Below this text is a table with the following data:

Schema	Table	# row updates
myschema1	mytb11	7
myschema1	mytb12	3
myschema1	mytb12b	3
myschema1	myTb13	12
myschema1	mytb14	1

At the bottom of the main content area, there are two more links: "Back to the tables group's detail" and "Back to the tables groups list".

Figure 5 – Before rollback statistics

The displayed page contains a first line returning the number of log rows that would be concerned by a potential rollback to this mark and an estimate of the rollback duration.

Then, the number of log rows to processed are presented on a per table basis.

6.2.6 Statistics about log tables content

Global or detailed statistics about log tables content are obtained from the detail page of a tables group.

PostgreSQL 8.4.7 running on localhost:5432 – You are logged in as user "postgres" SQL | History | Find | Logout

phpPgAdmin: PostgreSQL? Emaj-1.0.0?

Schemas? SQL? Find Variables? Processes? Locks? Admin Privileges? Slony Export E-maj

21:57:26 E-Maj 1.0.0 [392 kB = 4,8%] - Statistics from E-Maj log

[Back to the tables group's detail](#) | [Back to the tables groups list](#)

Table updates between mark MARK1 and mark MARK3 for tables group « myGroup1 »:

Schema	Table	# row updates	Log examination SQL
myschema1	mytbl1	7	select * from emaj.myschema1_mytbl1_log where emaj_changed >= '2012-11-12 21:24:34.936043+01' and emaj_changed <= '2012-11-12 21:24:35.185207+01' order by emaj_gid
myschema1	mytbl2	3	select * from emaj.myschema1_mytbl2_log where emaj_changed >= '2012-11-12 21:24:34.936043+01' and emaj_changed <= '2012-11-12 21:24:35.185207+01' order by emaj_gid
myschema1	mytbl2b	3	select * from emaj.myschema1_mytbl2b_log where emaj_changed >= '2012-11-12 21:24:34.936043+01' and emaj_changed <= '2012-11-12 21:24:35.185207+01' order by emaj_gid
myschema1	myTbl3	12	select * from emaj."myschema1_myTbl3_log" where emaj_changed >= '2012-11-12 21:24:34.936043+01' and emaj_changed <= '2012-11-12 21:24:35.185207+01' order by emaj_gid
myschema1	mytbl4	1	select * from emaj.myschema1_mytbl4_log where emaj_changed >= '2012-11-12 21:24:34.936043+01' and emaj_changed <= '2012-11-12 21:24:35.185207+01' order by emaj_gid

[Back to the tables group's detail](#) | [Back to the tables groups list](#)

Figure 6 – Global statistics with updates only concerning one application table.

The displayed statistics contain:

- table identity (schema and table names),
- the number of updates recorded into the log tables between both supplied marks,
- a SQL statement that can be copied/pasted to examine the log table's content corresponding to this time frame.

For detailed statistics, there are two additional columns:

- the connection role who performed the updates,
- the SQL verb type (*insert/update/delete*).

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 sheet also specifies those callable by *emaj_viewer* roles.

Functions	Parameters	Return type	Callable by emaj_viewer	Ref.
emaj_alter_group	group TEXT	# tables.and.seq INT		§ 4.2.10
emaj_comment_group	group TEXT comment TEXT	-		§ 4.7.2
emaj_comment_mark_group	group TEXT mark TEXT comment TEXT	-		§ 4.4.1
emaj_create_group	group TEXT [is.rollbackable BOOLEAN]	#.tables.and.seq INT		§ 4.2.3
emaj_delete_before_mark_group	group TEXT mark TEXT	#.deleted.marks INT		§ 4.4.5
emaj_delete_mark_group	group TEXT mark TEXT	1 INT		§ 4.4.4
emaj_detailed_log_stat_group	group TEXT start.mark TEXT end.mark TEXT	SETOF emaj_detailed_log_stat _type	Yes	§ 4.5.2
emaj_drop_group	group TEXT	#.tables.and.seq INT		§ 4.2.9
emaj_estimate_rollback_duration	group TEXT mark TEXT	duration INTERVAL	Yes	§ 4.5.3
emaj_force_drop_group	group TEXT	#.tables.and.seq INT		§ 4.7.5
emaj_force_stop_group	group TEXT	#.tables.and.seq INT		§ 4.7.4
emaj_generate_sql	group TEXT start.mark TEXT end.mark TEXT output.file.path TEXT	#.gen.statements INT		§ 4.6.3
emaj_get_previous_mark_group	group TEXT date.time TIMESTAMPTZ	mark TEXT	Yes	§ 4.4.2
emaj_get_previous_mark_group	group TEXT mark TEXT	mark TEXT	Yes	§ 4.4.2
emaj_log_stat_group	group TEXT start.mark TEXT end.mark TEXT	SETOF emaj_log_stat_type	Yes	§ 4.5.1

Functions	Parameters	Return type	Callable by emaj_viewer	Ref.
emaj_logged_rollback_group	group TEXT mark TEXT	#.proc.tables.and.seq INT		§ 4.2.7
emaj_logged_rollback_groups	groups.array TEXT[] mark TEXT	#.proc.tables.and.seq INT		§ 4.3.2
emaj_rename_mark_group	group TEXT mark TEXT new.name TEXT	-		§ 4.4.3
emaj_reset_group	group TEXT	#.tables.and.seq INT		§ 4.7.1
emaj_rollback_group	group TEXT mark TEXT	#.proc.tables.and.seq INT		§ 4.2.6
emaj_rollback_groups	groups.array TEXT[] mark TEXT	#.proc.tables.and.seq INT		§ 4.3.2
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		§ 4.3.2
emaj_snap_group	group TEXT directory TEXT copy.options TEXT	#.tables.and.seq INT		§ 4.6.1
emaj_snap_log_group	group TEXT start.mark TEXT end.mark TEXT directory TEXT copy.options TEXT	#.tables.and.seq INT		§ 4.6.2
emaj_start_group	group TEXT mark TEXT [reset.log BOOLEAN]	#.tables.and.seq INT		§ 4.2.4
emaj_start_groups	groups.array TEXT[] mark TEXT [reset.log BOOLEAN]	#.tables.and.seq INT		§ 4.3.2
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		§ 4.3.2
emaj.emaj_verify_all	-	Setof TEXT	Yes	§ 4.7.3