

E-Maj

A PostgreSQL extension

Reference Guide

Version: 4.7.1

Summary

1 - Introduction	8
1.1 - Document content	8
1.2 - License	8
1.3 - E-Maj's objectives	8
1.4 - Main components	9
2 - How E-Maj works	10
2.1 - Concepts	10
2.1.1 - Tables Group	10
2.1.2 - Mark	10
2.1.3 - Rollback	10
2.2 - Architecture	11
2.2.1 - Logged SQL statements	11
2.2.2 - Created objects	11
2.2.3 - Schemas	12
2.2.4 - Norm for E-Maj objects naming	13
2.2.5 - Tablespaces	13
3 - How to install E-Maj	14
3.1 - Quick start	14
3.1.1 - Install software	14
3.1.2 - Create the extension	14
3.1.3 - Use the extension	15
3.2 - Install the E-Maj software	17
3.2.1 - Download sources	17
3.2.2 - Standart installation on Linux	17
3.2.3 - Manual installation under Linux	18
3.2.4 - Minimum installation on Linux	19
3.2.5 - Installation on Windows	19
3.3 - Create the emaj extension in a database	20
3.3.1 - Optional preliminary operation	20
3.3.2 - Standart creation of the emaj EXTENSION	20
3.3.3 - Creating the extension by script	21
3.3.4 - Changes in the PostgreSQL instance configuration	21
3.3.5 - E-Maj parameters	21
3.3.6 - Test and demonstration	22
3.4 - Upgrade an existing E-Maj version	23

3.4.1 - General approach	23
3.4.2 - Upgrade by deletion and re-installation	23
3.4.3 - Upgrade an E-Maj version installed as an EXTENSION	25
3.4.4 - Compatibility break	26
3.5 - Uninstall E-Maj	28
3.5.1 - Remove an E-Maj extension from a database	28
3.5.2 - Uninstall the E-Maj software	29
3.6 - Upgrade the PostgreSQL version	29
3.6.1 - Changing PostgreSQL minor versions	29
3.6.2 - Changing the major PostgreSQL version and the E-Maj version simultaneously	29
3.6.3 - Changing the PostgreSQL major version and keeping the existing E-Maj environment	29
4 - How to use E-Maj	31
4.1 - Set-up the E-Maj access policy	31
4.1.1 - E-Maj roles	31
4.1.2 - Giving E-Maj rights	31
4.1.3 - Giving rights on application tables and objects	31
4.1.4 - Synthesis	32
4.2 - Create and drop tables groups	33
4.2.1 - Tables groups configuration principles	33
4.2.2 - Create a tables group	34
4.2.3 - Assign tables and sequences into a tables group	35
4.2.4 - Drop a tables group	37
4.3 - Main functions	39
4.3.1 - Operations chain	39
4.3.2 - Start a tables group	40
4.3.3 - Set an intermediate mark	41
4.3.4 - Rollback a tables group	42
4.3.5 - Perform a logged rollback of a tables group	43
4.3.6 - Stop a tables group	45
4.4 - Modifying tables groups	47
4.4.1 - General information	47
4.4.2 - Add tables or sequences to a tables group	48
4.4.3 - Remove tables from their tables group	48
4.4.4 - Remove sequences from their tables group	49
4.4.5 - Move tables to another tables group	49
4.4.6 - Move sequences to another tables group	50
4.4.7 - Modify tables properties	50
4.4.8 - Incidence of tables or sequences addition or removal in a group in LOGGING state	51

4.4.9 - Repare a tables group	53
4.5 - Other tables groups management functions	55
4.5.1 - Reset log tables of a group	55
4.5.2 - Comment a group	55
4.5.3 - Protect a tables group against rollbacks	55
4.5.4 - Forced stop of a tables group	56
4.5.5 - Forced drop of a tables group	57
4.5.6 - Exporting and importing tables groups configurations	57
4.5.7 - Knowing the existence or state of a tables group or a mark	59
4.5.8 - Erase traces from a dropped tables group	60
4.6 - Marks management functions	61
4.6.1 - Comment a mark	61
4.6.2 - Search a mark	61
4.6.3 - Rename a mark	62
4.6.4 - Delete a mark	62
4.6.5 - Delete oldest marks	63
4.6.6 - Protect a mark against rollbacks	63
4.7 - Rollbacks Administration functions	65
4.7.1 - Estimate the rollback duration	65
4.7.2 - Monitor rollback operations	66
4.7.3 - Comment a rollback operation	67
4.7.4 - "Consolidate" a logged rollback	68
4.7.5 - List "consolidable rollbacks"	69
4.7.6 - Update rollback operations state	70
4.8 - Count data content changes	71
4.8.1 - Tables group level statistics	71
4.8.2 - Table or sequence level statistics	75
4.9 - Examine data content changes	79
4.9.1 - Introduction	79
4.9.2 - Output types	79
4.9.3 - Consolidation levels	79
4.9.4 - The emaj_dump_changes_group() function	81
4.9.5 - The emaj_gen_sql_dump_changes_group() function	83
4.9.6 - Impact of tables group structure changes	86
4.10 - Generate SQL scripts to replay logged changes	87
4.11 - Other functions	89
4.11.1 - Get the emaj extension current version	89
4.11.2 - Check the E-Maj environment consistency	89
4.11.3 - Exporting and importing parameters configurations	90
4.11.4 - Identify the current log table linked to an application table	92
4.11.5 - Purge history data	92

4.11.6 - Deactive/reactive event triggers	93
4.11.7 - Snap tables and sequences of a tables group	93
4.12 - Multi-groups functions	95
4.12.1 - General information	95
4.12.2 - Functions list	95
4.12.3 - Syntax for groups array	95
4.12.4 - Other considerations	96
4.13 - Parallel Rollback client	97
4.13.1 - Sessions	97
4.13.2 - Prerequisites	97
4.13.3 - Syntax	97
4.13.4 - Examples	98
4.14 - Rollback monitoring client	100
4.14.1 - Prerequisite	100
4.14.2 - Syntax	100
4.14.3 - Examples	101
4.15 - Changes recording monitoring client	102
4.15.1 - Prerequisite	102
4.15.2 - Syntax	102
4.15.3 - Example	103
4.15.4 - Display description	103
5 - Miscellaneous	105
5.1 - Parameters	105
5.2 - Log tables structure	106
5.2.1 - Standart structure	106
5.2.2 - Adding technical columns	107
5.3 - Reliability	107
5.3.1 - Internal checks	107
5.3.2 - Event triggers	108
5.4 - Management of generated columns	109
5.4.1 - Generated columns in log tables	109
5.4.2 - DDL changes on generated columns	109
5.5 - Traces of operations	110
5.5.1 - The emaj_hist table	110
5.5.2 - Other history tables	112
5.5.3 - Purge obsolete traces	112
5.6 - The E-Maj rollback under the Hood	113
5.6.1 - Planning and execution	113
5.6.2 - Rollbacking a table	114
5.6.3 - Foreign keys management	114
5.6.4 - Other integrity constraints	115

5.6.5 - Application triggers management	116
5.7 - Impacts on instance and database administration	117
5.7.1 - Stopping and restarting the instance	117
5.7.2 - Saving and restoring the database	117
5.7.3 - Data load	119
5.7.4 - Tables reorganisation	119
5.7.5 - Using E-Maj with replication	120
5.8 - Sensitivity to system time change	122
5.9 - Performance	123
5.9.1 - Updates recording overhead	123
5.9.2 - E-Maj rollback duration	123
5.9.3 - Optimizing E-Maj operations	123
5.10 - Usage limits	125
5.11 - User's responsibility	125
5.11.1 - Defining tables groups content	125
5.11.2 - Appropriate call of main functions	125
5.11.3 - Management of application triggers	125
5.11.4 - Internal E-Maj table or sequence change	126
6 - Emaj_web	127
6.1 - Overview	127
6.2 - Install the Emaj_web client	127
6.2.1 - Prerequisite	127
6.2.2 - Download the software	127
6.2.3 - Configure Emaj-web	128
6.3 - Use Emaj_web	128
6.3.1 - Access to Emaj_web and databases	128
6.3.2 - Tables groups lists	129
6.3.3 - Some details about the user interface	131
6.3.4 - Tables group details	132
6.3.5 - Statistics	133
6.3.6 - Tables group content	135
6.3.7 - Tables group history	136
6.3.8 - Schemas and tables groups configuration	136
6.3.9 - Table details	137
6.3.10 - Sequence details	139
6.3.11 - Triggers	141
6.3.12 - Monitoring rollback operations	142
6.3.13 - E-Maj activity	144
6.3.14 - E-Maj environment state	146
7 - Contribute to the E-Maj development	148

7.1 - Build the E-Maj environment	148
7.1.1 - Clone the E-Maj repository	148
7.1.2 - Description of the E-Maj tree	148
7.1.3 - Setting tools parameters	148
7.2 - Coding	149
7.2.1 - Versionning	149
7.2.2 - Coding rules	149
7.2.3 - Version upgrade script	150
7.3 - Testing	151
7.3.1 - Create PostgreSQL instances	151
7.3.2 - Install software dependancies	151
7.3.3 - Execute non regression tests	151
7.3.4 - Test coverage	154
7.3.5 - Evaluate the performances	154
7.4 - Documenting	156
7.5 - Submitting a patch	156
7.6 - Contributing to Emaj_web	156
8 - Appendix	158
8.1 - E-Maj functions list	158
8.1.1 - Tables or sequences level functions	158
8.1.2 - Groups level functions	160
8.1.3 - General purpose functions	163
8.2 - E-Maj distribution content	165
8.3 - PostgreSQL and E-Maj versions compatibility matrix	167

1 INTRODUCTION

1.1 DOCUMENT CONTENT

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

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

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

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

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

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

1.2 LICENSE

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

1.3 E-MAJ'S OBJECTIVES

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

It meets two main goals:

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

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

It provides a good solution to :

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

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

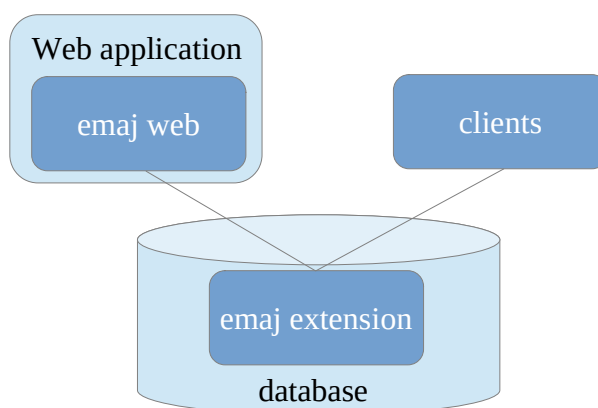
In a **test** environment, E-Maj also brings smoothness into operations. It is possible to very easily restore database subsets into predefined stable states, so that tests can be replayed as many times as needed.

1.4 MAIN COMPONENTS

E-Maj actually groups several components:

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

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



All these components are described in this documentation.

2 HOW E-MAJ WORKS

2.1 CONCEPTS

E-Maj is built on three main concepts.

2.1.1 Tables Group

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

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

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

2.1.2 Mark

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

2.1.3 Rollback

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

There are two rollback types:

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

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

2.2 ARCHITECTURE

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

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

2.2.1 Logged SQL statements

The recorded update operations concerns the following SQL verbs:

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

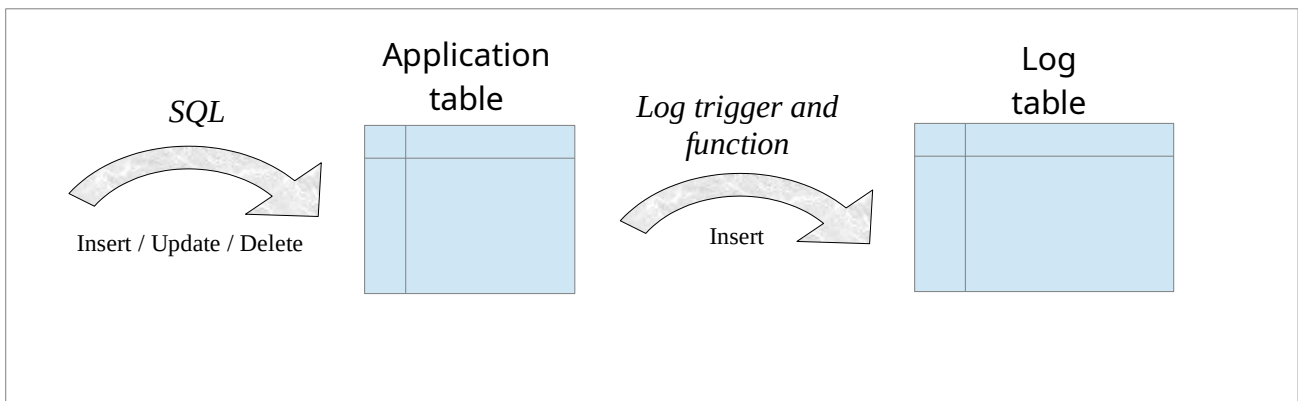
For statements that process several rows, each creation, update or deletion is individually recorded. For instance, if a “*DELETE FROM <table>*” is performed against a table having 1 million rows, 1 million row deletion events are recorded.

At *TRUNCATE* SQL execution time, the whole table content is recorded before its effective deletion.

2.2.2 Created objects

For each application table, the following objects are created:

- a dedicated log table, containing data corresponding to the updates applied on the application table,
- a trigger and a specific function, that, for each row creation (*INSERT*, *COPY*), change (*UPDATE*) or suppression (*DELETE*), record into the log table all data needed to potentially cancel later this elementary action,
- another trigger, that processes *TRUNCATE* SQL statements ,
- a sequence used to quickly count the number of updates recorded in log tables between 2 marks.



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

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

- 19 tables,
- 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,
- 8 composite and 3 enum types,
- 1 view,
- more than 180 functions, more than 80 of them being directly callable by users (cf §8.1),
- 1 trigger,
- 1 specific schema, named *emaj*, that contains all these relational objects,
- 2 roles acting as groups (NOLOGIN): *emaj_adm* to manage E-Maj components, and *emaj_viewer* to only look at E-Maj components
- 3 event triggers.

Technical tables, whose structure is interesting to know, are described in the coming chapters (*emaj_param* is described in §5.1 and *emaj_hist* is described in §5.5).

The *emaj_adm* role is the owner of all log schemas, tables, sequences and functions.

2.2.3 Schemas

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

All objects linked to application tables are stored into schemas named :
emaj_<schema>

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

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

2.2.4 Norm for E-Maj objects naming

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

- the name of the log table is:
`<table.name>_log`
- the name of the log function is:
`<table.name>_log_fnct`
- the name of the sequence associated to the log table is:
`<table.name>_log_seq`

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

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

Other E-Maj function names are also normalised:

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

Triggers created on application tables have the same name:

- `emaj_log_trg` for the log triggers,
- `emaj_trunc_trg` for the triggers that manage TRUNCATE verbs.

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

2.2.5 Tablespaces

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

The same rule applies for log tables and index. But using tables group parameters, it is also possible to store log tables and/or their index into specific tablespaces.

3 HOW TO INSTALL E-MAJ

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

3.1 QUICK START

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

3.1.1 Install software

If the *pgxn* client is installed, a single command is required:

```
pgxn install E-Maj --sudo
```

Otherwise:

```
wget https://api.pgxn.org/dist/e-maj/<version>/e-maj-<version>.zip  
unzip e-maj-<version>.zip  
cd e-maj-<version>/  
sudo make install
```

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

3.1.2 Create the extension

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

```
CREATE EXTENSION emaj CASCADE;  
GRANT emaj_adm TO <role>;
```

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

3.1.3 Use the extension

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

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

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

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

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

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

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

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

Then the typical commands sequence:

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

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

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

3.2 INSTALL THE E-MAJ SOFTWARE

3.2.1 Download sources

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

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

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



Caution: installing the extension from the github.org repository creates the extension in its development version (“devel”), even when downloading a ‘tagged’ version. In this case, no future extension update is possible. For a stable E-Maj use, it is highly recommended to use the packets available from PGXN.

3.2.2 Standart installation on Linux

3.2.2.1 With the pgxn client

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

```
pgxn install E-Maj --sudo
```

3.2.2.2 Without the pgxn client

Download the latest E-Maj version by any convenient way, for instance using the *wget* command:

```
wget https://api.pgxn.org/dist/e-maj/<version>/e-maj-<version>.zip
```

Then decompress the downloaded archive file and install the components with the commands:

```
unzip e-maj-<version>.zip  
cd e-maj-<version>/  
sudo make install
```

3.2.2.3 Components localization

In both cases, the components are installed into the usual PostgreSQL directories. In particular:

- SQL scripts are into *<SHAREDIR_directory>/emaj/*
- CLI clients are into *<BINDIR_directory>*, with PostgreSQL clients
- the documentation is into *<DOCDIR_directory>/emaj*

The physical localization of *SHAREDIR*, *BINDIR* and *DOCDIR* directories can be found with the *pg_config* command.

3.2.3 Manual installation under Linux

Download the latest E-Maj version by any convenient way, for instance using the *wget* command:

```
wget https://api.pgxn.org/dist/e-maj/<version>/e-maj-<version>.zip
```

Then decompress the downloaded archive file with the commands:

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

Edit the *emaj.control* file to set the *directory* parameter to the actual location of the E-Maj install SQL scripts (the absolute path of the *e-maj-<version>/sql* directory).

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

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

Then copy the modified *emaj.control* file into the extension directory of the PostgreSQL version you want to use. As a super-user or pre-pended with *sudo*, type:

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

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

3.2.4 Minimum installation on Linux

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

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

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

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

3.2.5 Installation on Windows

To install E-Maj on Windows:

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

3.3 CREATE THE emaj EXTENSION IN A DATABASE

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

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

In environments for which this is not possible (cases of minimum installation – see §3.2.4), a *psql* script can be executed.

3.3.1 Optional preliminary operation

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

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

3.3.2 Standart creation of the emaj EXTENSION

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

```
CREATE EXTENSION emaj CASCADE;
```

After having verified that the PostgreSQL version is compatible with this E-Maj version, the script creates the *emaj* schema and populate it with technical tables, functions and some other objects.



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

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

Finally, the installation script looks at the instance configuration and may display a warning message about the *max_prepared_transactions* parameter (see §4.13.2).

3.3.3 Creating the extension by script

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

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

where <emaj_directory> is the directory generated by the E-Maj installation (see §3.2.4) and <version> the current E-Maj version.



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

In this installation mode, all optimizations regarding E-Maj rollbacks are not available, leading to a decreased performance level of these operations.

3.3.4 Changes in the PostgreSQL instance configuration

Two configuration parameters may need to be changed in the *postgresql.conf* file:

- ***max_locks_per_transaction*** to manage tables groups with a high number of tables. Main E-Maj functions set a lock on each table of a processed tables group. If an E-Maj operation fails with a message indicating that all entries of the “*shared lock table*” have been used, this parameter must be increased. Its default value equals 64.
- ***max_prepared_transactions***, to use the E-Maj parallel rollback client (see § 4.13). Its default value is 0, blocking the use of this tool. Its value must be greater or equal the maximum number of concurrent parallel rollbacks operations.

In both cases, the PostgreSQL instance must be restarted.

3.3.5 E-Maj parameters

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

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

However, if the E-Maj administrator wishes to take benefit from the rollback operations monitoring capabilities, it is necessary to set the “*dblink_user_password*” parameter into the *emaj_param* table and give the E-Maj administrator role the right to execute the *dblink_connect_u* function (cf §4.7.2.1).

3.3.6 Test and demonstration

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

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

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

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

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

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

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

3.4 UPGRADE AN EXISTING E-MAJ VERSION

3.4.1 General approach

The first step consists in installing the new version of the E-Maj software, using the procedure described in §3.2.

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

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

Any E-Maj environment installed into a database can be upgraded by a simple uninstall and re-install. This procedure is described au §3.4.2.

For E-Maj versions installed as an *EXTENSION*, and with version id greater or equal 2.3.1, it is possible to upgrade the version without being obliged to uninstall the version . This approach, described in chapter §3.4.3 keeps all logs, allowing to examine prior changes and even process E-Maj rollbacks targeting a mark set before the version upgrade.

Note that for E-Maj versions that have been installed by *psql* script (so not as an *EXTENSION*), there is no particular update procedure. On such environments, a simple E-Maj deletion and then re-installation is required.



It is important to verify the PostgreSQL and E-Maj versions compatibility matrix to be sure that the E-Maj version upgrade is possible. If the PostgreSQL version is too old, it must be upgraded **before** migrating E-Maj to a higher version.

3.4.2 Upgrade by deletion and re-installation

For this upgrade path, it is not necessary to use the full un-installation procedure described in §3.5.1. In particular, the tablespace and both roles can remain as is. However, it may be judicious to save some useful pieces of information. Here is a suggested procedure.

3.4.2.1 Stop tables groups

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

3.4.2.2 Save user data

The procedure depends on the installed E-Maj version.

Installed version >= 3.3

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

```
SELECT emaj.emaj_export_groups_configuration('<file.path.1>');  
  
SELECT emaj.emaj_export_parameters_configuration  
('<file.path.2>');
```

Installed version < 3.3

If the installed E-Maj version is prior 3.3.0, these export functions are not available.

As starting from E-Maj 4.0 the tables groups configuration doesn't use the *emaj_group_def* table anymore, rebuilding the tables groups after the E-Maj version upgrade will need either to edit a JSON configuration file to import or to execute a set of tables/sequences assignment functions.

If the *emaj_param* tables contains specific parameters, it can be saved on file with a *copy* command, or duplicated outside the *emaj* schema.

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

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

3.4.2.3 E-Maj deletion and re-installation

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

```
\i <old_emaj_directory>/sql/emaj_uninstall.sql  
  
CREATE EXTENSION emaj CASCADE;
```

3.4.2.4 Restore user data

Previous version >= 3.3

The exported tables groups and parameters configurations can be reloaded with:

```
SELECT emaj.emaj_import_parameters_configuration  
    ('<file.path.2>', TRUE);  
  
SELECT emaj.emaj_import_groups_configuration('<file.path.1>');
```

Previous version < 3.3

The saved parameters and application triggers configurations can be reloaded for instance with INSERT SELECT statements:

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

The tables groups need to be rebuilt using the standard methods of the new version (See §4.2).

Then, temporary tables or files can be deleted.

3.4.3 Upgrade an E-Maj version installed as an EXTENSION

An existing version that has been installed as an extension can be upgraded using the SQL statement:

```
ALTER EXTENSION emaj UPDATE;
```

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

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

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

Version specific details:

- The procedure that upgrades a version **2.3.1** into **3.0.0** changes the structure of log tables: both *emaj_client_ip* and *emaj_client_port* columns are not created anymore. Existing log tables are not modified. Only the new log tables are impacted. But the administrator can add these columns, by using the 'alter_log_tables' parameter (Cf §5.2.2).
- The procedure that upgrades a version **3.0.0** into **3.1.0** renames existing log objects. This leads to locking the application tables, which may generate conflicts with the parallel use of these tables. This procedure also issues a warning message indicating that the changes in E-Maj rollback functions regarding the application triggers processing may require changes in user's procedures.
- The procedure that upgrades a version **3.4.0** into **4.0.0** updates the log tables content for *TRUNCATE* recorded statements. The upgrade duration depends on the global log tables size.
- The procedure that upgrades a version **4.1.0** into **4.2.0** checks that all event triggers exist. If some are missing, the E-Maj environment must be recreated (alternatively, the *sql/emaj_upgrade_after_postgres_upgrade.sql* script provided by the 4.1.0 E-maj version could be searched and executed to create the missing event triggers).
- The procedure that upgrades a version **4.3.1** into **4.4.0** reads the *emaj_hist* table content in order to build 3 histories populating the 3 new internal tables. Although rather short, the upgrade duration depends on the *emaj_hist* table size.

3.4.4 Compatibility break

As a general rule, upgrading the E-Maj version does not change the way to use the extension. There is an ascending compatibility between versions. The exceptions to this rule are presented below.

3.4.4.1 Upgrading towards version 4.0.0

The compatibility breaks of the 4.0.0 E-Maj version mainly deal with the way to manage tables groups configurations. The 3.2.0 version brought the ability to dynamically manage the assignment of tables and sequences into tables groups. The 3.3.0 version allowed to describe tables groups configuration with JSON structures. Since, these technics have existed beside the historical way to handle tables group using the *emaj_group_def* table. Starting with the 4.0.0 version, this historical way to manage tables groups configurations has disappeared.

More precisely:

- The table *emaj_group_def* does not exist anymore.
- The *emaj_create_group()* function only creates empty tables groups, that must be then populated with functions of the *emaj_assign_table()* / *emaj_assign_sequence()* family, or the *emaj_import_groups_configuration()* function. The third and last parameter of the *emaj_create_group()* function has disappeared. It allowed to create empty tables groups.
- The now useless *emaj_alter_group()*, *emaj_alter_groups()* and *emaj_sync_def_group()* functions also disappear.

Furthermore:

- The *emaj_ignore_app_trigger()* function is deleted. The triggers to ignore at E-Maj rollback time can be registered with the functions of the *emaj_assign_table()* family.
- In JSON structures managed by the *emaj_export_groups_configuration()* and *emaj_import_groups_configuration()* functions, the format of the "ignored_triggers" property that lists the triggers to ignore at E-Maj rollback time has been simplified. It is now a simple text array.
- The old family of E-Maj rollback functions that just returned an integer has been deleted. Only the functions returning a set of messages remain.
- The name of function parameters have changed: "v_" prefixes have been transformed into "p_". This only impacts function calls formatted with named parameters. But this practice is unusual.

3.4.4.2 Upgrading towards version 4.3.0

Before E-Maj 4.3.0, the *emaj_log_stat_group()*, *emaj_gen_sql_group()* and *emaj_snap_log_group()* functions families accepted a NULL value or an empty string as the first mark name of the requested time range, this value representing the oldest known mark for the tables group or groups. The concept being ambiguous, especially with multi-groups functions, this feature has been removed in version 4.3.0.

The *emaj_snap_log_group()* function has been replaced by both *emaj_dump_changes_group()* and *emaj_gen_sql_dump_changes_group()* functions, providing much larger features (see §4.9). In order to create a files set of log tables extracts, the statement:

```
SELECT emaj.emaj_snap_log_group(<group>, <start.mark>, <end.mark>,  
    <directory>, <copy.options>);
```

can be easily changed into:

```
SELECT emaj.emaj_dump_changes_group(<group>, <start.mark>,  
    <end.mark>, 'COPY_OPTIONS=(<copy.options>)', NULL, <directory>);
```

Note that none of the start and end marks can now be NULL. Furthermore, data format about sequences has changed: while 2 files grouped the initial and final sequences states respectively, there is now one file per sequence with the same elementary information.

3.5 UNINSTALL E-MAJ

3.5.1 Remove an E-Maj extension from a database

To remove 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 removal operation.

Although the *emaj* extension is usually installed with a *CREATE EXTENSION* statement, it cannot be removed with a simple *DROP EXTENSION* statement. An event trigger blocks such a statement.

Whatever the manner the *emaj* extension has been installed (using the standart *CREATE EXTENSION* statement or a *psql* script when adding an extension is forbidden), its removal just needs to execute the *emaj_drop_extension()* function.

```
SELECT emaj.emaj_drop_extension();
```

This function performs the following steps:

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

In E-Maj versions 4.4.0 and previous, the *emaj* extension removal was done by the execution of a *sql/emaj_uninstall.sql* script. Although deprecated, the removal can always be done the same manner.

3.5.2 Uninstall the E-Maj software

The way to uninstall the E-Maj software depends on the way it has been installed.

For a standart install with the *pgxn* client, a single command is required:

```
pgxn uninstall E-Maj --sudo
```

For a standart install without the *pgxn* client, reach the initial directory of the E-Maj distribution and type:

```
sudo make uninstall
```

For a manual install, the installed components must be removed by reverting the initial installation steps.

3.6 UPGRADE THE POSTGRESQL VERSION

3.6.1 Changing PostgreSQL minor versions

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

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

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

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

Nevertheless, it is possible to keep the existing E-Maj components (tables groups, logs, marks,...) while changing the PostgreSQL major version. And the tables groups may event stay in logging state during the operation. But one condition must be met: the old and new instances must share the **same E-Maj version**.

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

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

4 HOW TO USE E-MAJ

4.1 SET-UP THE E-MAJ ACCESS POLICY

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

4.1.1 E-Maj roles

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

- *emaj_adm* is used as the administration role ; it can execute all functions and access to all E-Maj tables, with reading and writing rights ; *emaj_adm* is the owner of all log objects (schemas, tables, sequences, functions),
- *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>;
```

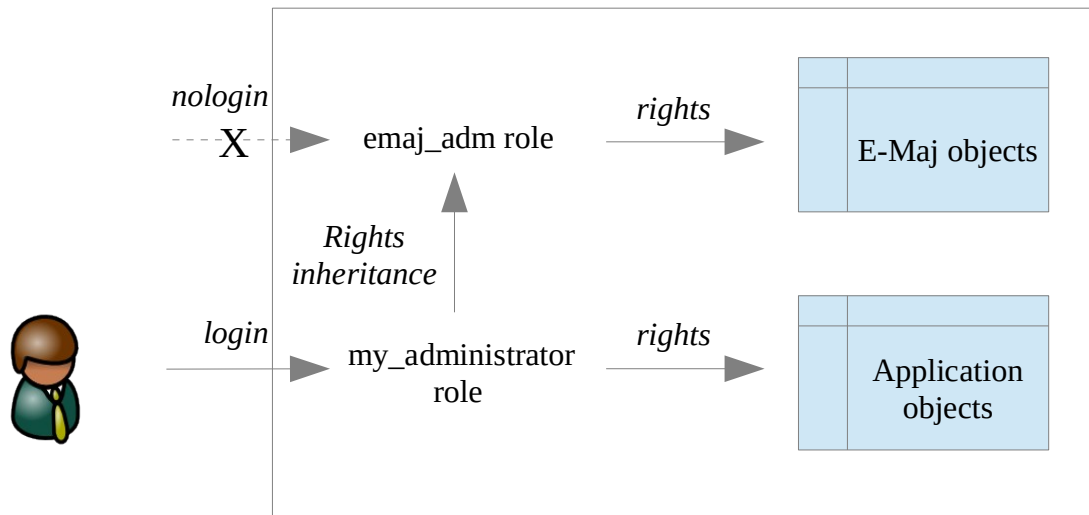
Of course, *emaj_adm* or *emaj_viewer* rights can be given to several roles.

4.1.3 Giving rights on application tables and objects

It is not necessary to grant any privilege on application tables and sequences to *emaj_adm* and *emaj_viewer*. The functions that need to access these objects are executed with the extension installation role, i.e. a superuser role.

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 CREATE AND DROP TABLES GROUPS

4.2.1 Tables groups configuration principles

Configuring a tables group consists in:

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

4.2.1.1 The tables group

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

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

4.2.1.2 The tables and sequences to assign

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

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

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



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

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

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

By their nature, *TEMPORARY TABLE* are not supported by E-Maj. *UNLOGGED* tables can only be members of “*audit_only*” tables groups.

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

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

4.2.1.3 Specific tables properties

Four properties are associated to tables assigned to tables group:

- the priority level,
- the tablespace for log data,
- the tablespace for log index,
- the list of triggers whose state (ENABLED/DISABLED) must be left unchanged during E-Maj rollback operations.

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

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

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

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

When an E-Maj rollback is performed on a tables group, enabled triggers of concerned tables are neutralized, so that table's content changes generated by the operation do not fire them. But this by default behaviour can be changed if needed. Note that this does not concern E-Maj or system triggers.

4.2.2 Create a tables group

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

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

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

The third parameter is an optional comment to describe the group. If it is not provided or if it is set to *NULL*, no comment is registered for the group. The comment can be modified or deleted later using the *emaj_comment_group()* function (See §4.5.2).

The function returns the number of created groups, i.e. 1.

To insert a tables group creation into an idempotent script, it is possible to condition the creation to its non-existence, by using the *emaj_does_exist_group()* function in a *WHERE* clause (See §4.5.7).

4.2.3 Assign tables and sequences into a tables group

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

To add one or several tables into a tables group:

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

or

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

or

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

To add one or several sequences into a tables group:

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

or

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

or

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

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

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

A *TEXT* array is typically expressed with a syntax like:

```
ARRAY['element_1','element_2', ...]
```

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

- selects all tables or sequences of the schema *my_schema*

```
'my_schema', '.*', ''
```
- select all tables of this schema and whose name start with 'tbl'

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

```
'my_schema', '^tbl.*', '_sav$'
```

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

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

```
'{ "priority" : <p> ,  
  "log_data_tablespace" : "<ldt>" ,  
  "log_index_tablespace" : "<lit>" ,  
  "ignored_triggers" : ["<tg1>" , "<tg2>" , ...] ,  
  "ignored_triggers_profiles" : ["<regexp1>" , "<regexp2>" , ...] }'
```

where:

- *<p>* is the priority level for the table or tables
- *<ldt>* is the name of the tablespace to handle log tables
- *<lit>* is the name of the tablespace to handle log indexes
- *<tg1>* and *<tg2>* are trigger names
- *<regexp1>* and *<regexp2>* are regular expressions that select triggers names among those that exist for the table or the tables to assign into the group

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

If specific tablespaces are referenced for any log table or log index, these tablespaces must exist before the function's execution and the user must have been granted the *CREATE* privilege on them.

Both "ignored_triggers" and "ignored_triggers_profiles" properties define the triggers whose state must remain unchanged during E-Maj rollback operations. Both properties are of type array. "ignored_triggers" can be a simple string if it only contains one trigger.

Triggers listed in the "ignored_triggers" property must exist for the table or the tables referenced by the function call. The triggers created by E-Maj (emaj_log_trg and emj_trunc_trg) cannot appear in this list.

If several regular expressions are listed in the "ignored_triggers_profiles" property, they each act as a filter selecting triggers.

Both "ignored_triggers" and "ignored_triggers_profiles" properties can be used jointly. In this case, the selected triggers set is the union of those listed by the "ignored_triggers" property and those selected by each regular expression of the "ignored_triggers_profiles" property.

For more details about the management of application triggers, refer to §5.11.3.

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

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

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

4.2.4 Drop a tables group

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

Then, just execute the SQL command:

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

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

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

The function also drops all log schemas that are now useless.

The locks set by this operation can lead to *deadlock*. If the *deadlock* processing impacts the execution of the E-Maj function, the error is trapped and the lock operation is repeated, with a maximum of 5 attempts.

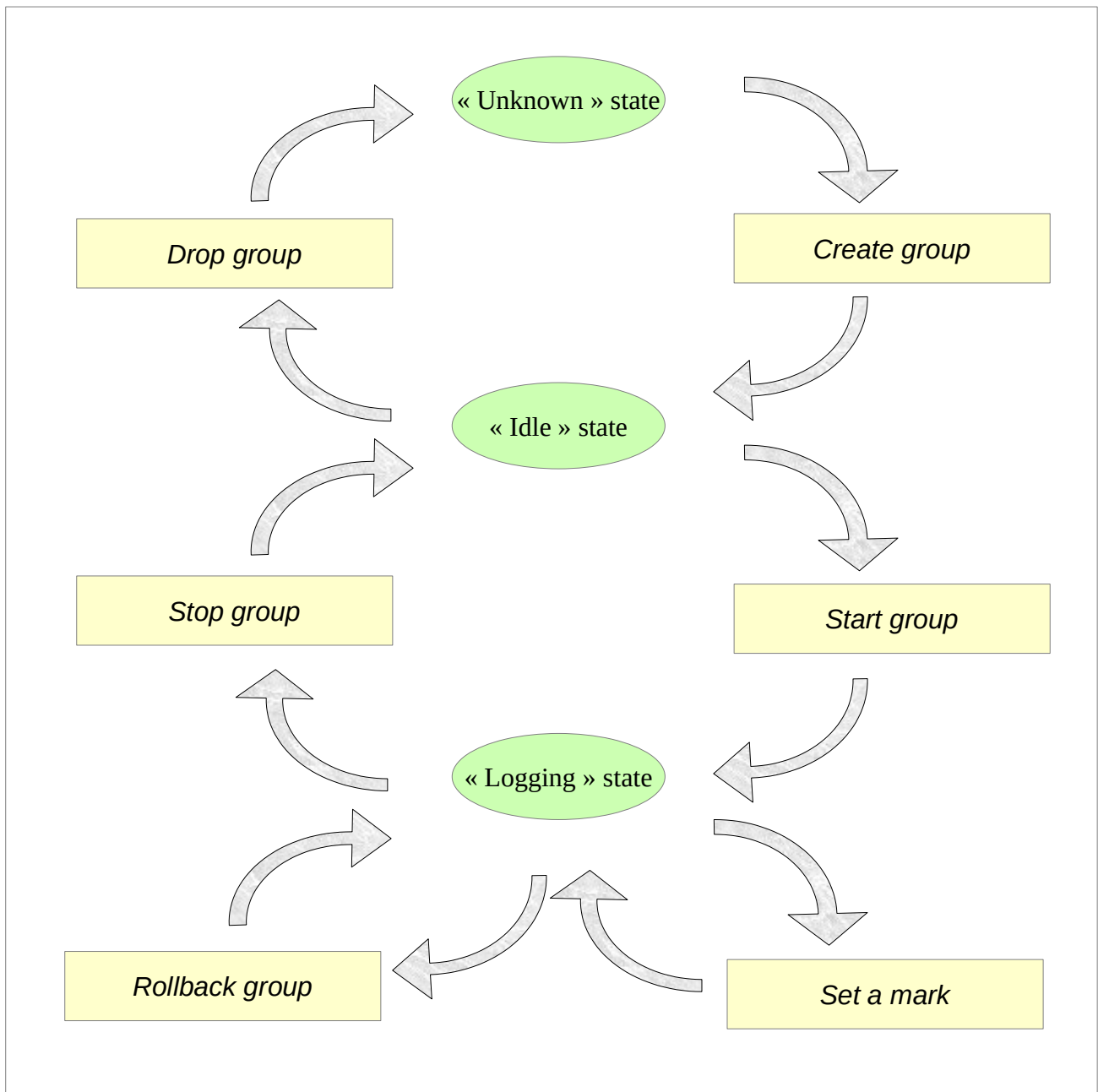
To insert a tables group drop into an idempotent script, it is possible to condition the operation to the group existence, by using the *emaj_does_exist_group()* function in a *WHERE* clause (See §4.5.7).

4.3 MAIN FUNCTIONS

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

4.3.1 Operations chain

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



4.3.2 Start a tables group

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

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

The group must be first in *IDLE* state.

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

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

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

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

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

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

The function also performs a purge of the oldest events in the `emaj_hist` technical table (see §5.5).

When a group is started, its state becomes "LOGGING".

To insert a tables group start into an idempotent script, it is possible to condition the operation to the group state, by using the `emaj_is_logging_group()` function in a *WHERE* clause (See §4.5.7).

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

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


The chapter §4.12.3 explains how to describe the group names array.

4.3.3 Set an intermediate mark

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

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

The tables group must be in *LOGGING* state.

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

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

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

The third parameter is an optional comment to describe the mark. If it is not provided or if it is set to *NULL*, no comment is registered for the mark. The comment can be modified or deleted later using the *emaj_comment_mark_group()* function (See §4.6.1).

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

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

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

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

To set a mark into an idempotent script, it is possible to condition the operation to the mark non-existence, by using the *emaj_does_exist_mark_group()* function in a *WHERE* clause (See §4.5.7).

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

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

The chapter §4.12.3 explains how to describe the group names array.

4.3.4 Rollback a tables group

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

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

The tables group must be in *LOGGING* state and not protected (Cf §4.5.3). The target mark cannot be prior a protected mark (Cf §4.6.6).

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

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

A comment associated to the rollback can be supplied as 4th parameter. It allows the administrator to annotate the operation, indicating for instance the reason for it has been launched or the reverted processing. The comment can also be added by the *emaj_comment_rollback()* function (cf §4.7.3), this function allowing also its update or deletion.

The function returns a set of rows with a severity level set to either “*Notice*” or “*Warning*” values, and a textual message. The function returns 3 “*Notice*” rows reporting the generated rollback identifier, the number of tables and and the number of sequences that have been **effectively** modified by the rollback operation. Other messages of type “*Warning*” may also be reported when the rollback operation has processed tables group changes.

To be sure that no concurrent transaction updates any table of the group during the rollback operation, the *emaj_rollback_group()* function explicitly sets an *EXCLUSIVE* lock on each table of the group. If transactions updating these tables are running, this can lead to

deadlock. If the *deadlock* processing impacts the execution of the E-Maj function, the error is trapped and the lock operation is repeated, with a maximum of 5 attempts. But tables of the group remain accessible for read only transactions during the operation.

The E-Maj rollback takes into account the existing triggers and foreign keys on the concerned tables. More details in the chapter §5.6)

When the volume of updates to cancel is high and the rollback operation is therefore long, it is possible to monitor the operation using the *emaj_rollback_activity()* function (§4.7.2) or the *emajRollbackMonitor* client (§4.14).

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

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

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

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

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

The chapter §4.12.3 explains how to describe the group names array.

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

4.3.5 Perform a logged rollback of a tables group

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

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

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

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

The tables group must be in *LOGGING* state and not protected (Cf §4.5.3). The target mark cannot be prior a protected mark (Cf §4.6.6).

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

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

A comment associated to the rollback can be supplied as 4th parameter. It allows the administrator to annotate the operation, indicating for instance the reason for it has been launched or the reverted processing. The comment can also be added by the *emaj_comment_rollback()* function (cf §4.7.3), this function allowing also its update or deletion.

The function returns a set of rows with a severity level set to either “*Notice*” or “*Warning*” values, and a textual message. The function returns 3 “*Notice*” rows reporting the generated rollback identifier, the number of tables and the number of sequences that have been **effectively** modified by the rollback operation. Other messages of type “*Warning*” may also be reported when the rollback operation has processed tables group changes.

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

The E-Maj rollback takes into account the existing triggers and foreign keys on the concerned tables. More details in the chapter §5.6)

Unlike with *emaj_rollback_group()* function, at the end of the operation, the log tables content as well as the marks following the rollback mark remain.

At the beginning and at the end of the operation, the function automatically sets on the group two marks named:

- 'RLBK_<rollback.identifier>_START'
- 'RLBK_<rollback.identifier>_DONE'

with a comment for each including the target mark name.

When the volume of updates to cancel is high and the rollback operation is therefore long, it is possible to monitor the operation using the *emaj_rollback_activity()* function (§4.7.2) or the *emajRollbackMonitor* client (§4.14).

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

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

Set Mark M1

```

...
Set Mark M2
...
Logged Rollback to M1,
generating RLBK_<rlbk.1.id>_STRT,
and RLBK_<rlbk.1.id>_DONE
...
Rollback to RLBK_<rlbk.1.id>_DONE
(to cancel the updates performed after the first rollback)
...
Rollback to RLBK_<rlbk.1.id>_STRT
(to finally cancel the first rollback)

```

A “*consolidation*” function for “*logged rollback*” allows to transform a logged rollback into a simple unlogged rollback (see §4.7.4).

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

```

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

```

The chapter §4.12.3 explains how to describe the group names array.

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

4.3.6 Stop a tables group

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

```

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

```

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

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

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

The *emaj_stop_group()* function closes the current log session. Then, it is not possible to execute an E-Maj rollback targeting an existing mark anymore, even though no changes have been applied since the tables group stop.

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

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

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

To insert a tables group stop into an idempotent script, it is possible to condition the operation to the group state, by using the *emaj_is_logging_group()* function in a *WHERE* clause (See §4.5.7).

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

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

The chapter §4.12.3 explains how to describe the group names array.

4.4 MODIFYING TABLES GROUPS

4.4.1 General information

Several event types may lead to alter a tables group:

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

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

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

<i>Actions</i>	<i>Method</i>
Add a table/sequence to a group	Tables/sequences assignment functions
Remove a table/sequence from a group	Tables/sequences removal functions
Move a table/sequence to another group	Tables/sequences move functions
Change the log data or index tablespace for a table	Tables properties modification functions
Change the E-Maj priority for a table	Tables properties modification functions
Repair a table	Remove from the group + add to the group
Rename a table	Remove from the group + ALTER TABLE + add
Rename a sequence	Remove from the group + ALTER SEQUENCE + add
Change the schema of a table	Remove from the group + ALTER TABLE + add
Change the schema of a sequence	Remove from the group + ALTER SEQUENCE + add
Rename a table's column	Remove from the group + ALTER TABLE + add
Change a table's structure	Remove from the group + ALTER TABLE + add
Other forms of ALTER TABLE	No E-Maj impact
Other forms of ALTER SEQUENCE	No E-Maj impact

Adjusting the structure of in *LOGGING* state groups may have consequences on E-Maj rollback or SQL script generation. This is explained into details in §4.4.8.

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

4.4.2 Add tables or sequences to a tables group

The functions that assign one or several tables or sequences into a tables group that are used at group's creation time (§4.2.3) are also usable during the whole group's life.

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

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

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

4.4.3 Remove tables from their tables group

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

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

or

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

or

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

They are very similar to the tables assignment functions.

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

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

4.4.4 Remove sequences from their tables group

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

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

or

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

or

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

They are very similar to the sequences assignment functions.

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

4.4.5 Move tables to another tables group

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

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

or

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

or

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

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

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

4.4.6 Move sequences to another tables group

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

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

or

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

or

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

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

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

4.4.7 Modify tables properties

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

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

or

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

or

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

The <modified.properties> parameter is of type JSONB. Its elementary fields are the same as the <properties> parameter of the tables assignment functions (cf §4.2.3). But this <modified.properties> parameter only contains ... the properties to modify. The not listed properties remain unchanged. It is possible to reset a property to its default value by setting a NULL value (the json null).

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

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

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



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

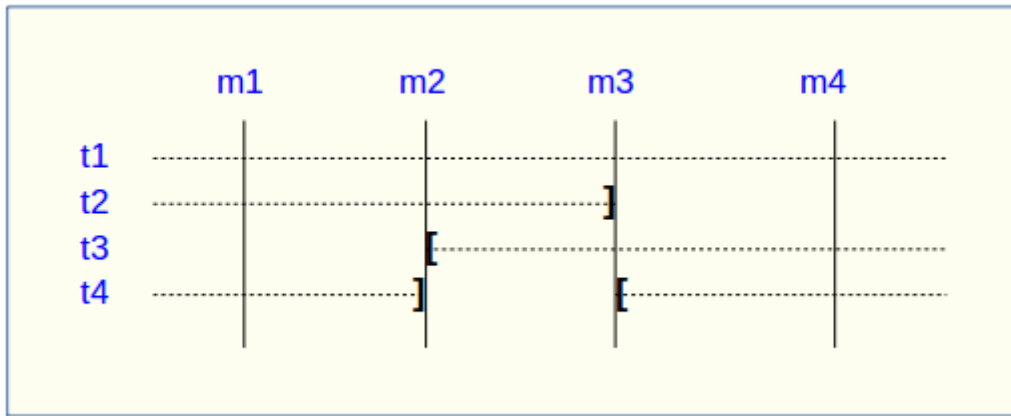
Once unlinked from its tables group, the application table or sequence can be altered or dropped. The historical data linked to the object (logs, marks traces,...) are kept as is so that they can be later examined. However, they remain linked to the tables group that owned the object. To avoid any confusion,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



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

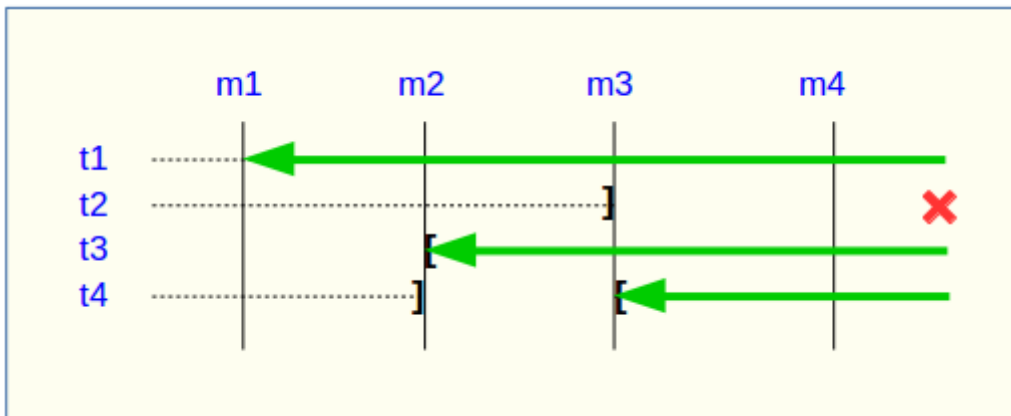
Some graphs help to more easily visualize the consequences of the addition or the removal of a table or a sequence into/from a tables group in *LOGGING* state.

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



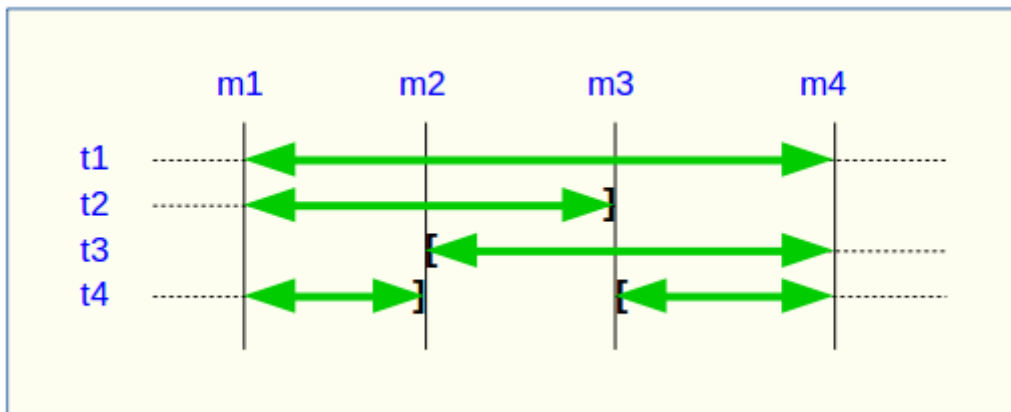
A rollback to the mark m1:

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



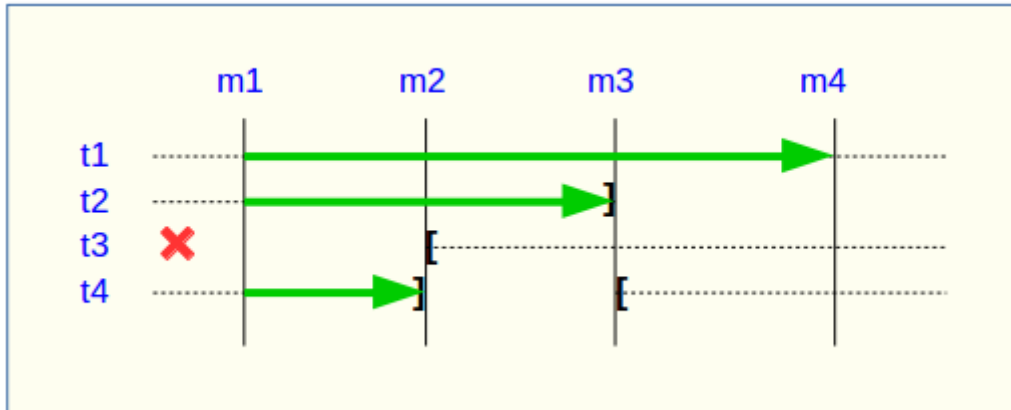
A log statistics report between the marks m1 and m4 would contain:

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



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

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



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

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

4.4.9 Repair a tables group

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

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

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

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

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

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

It may also happen that an application table or sequence has been accidentally dropped. In this case, the table or sequence can be simply a posteriori removed from its group by executing the *emaj_remove_table()* or *emaj_remove_sequence()* function.

4.5 OTHER TABLES GROUPS MANAGEMENT FUNCTIONS

4.5.1 Reset log tables of a group

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

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

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

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

4.5.2 Comment a group

It is possible to set a comment on a group when it is created (Cf §4.2.2). But this can be also done later with:

```
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 mostly interesting when using *Emaj_web*, that systematically displays them in the groups lists (See §6.3.2). But they can also be found in the *group_comment* column from the *emaj.emaj_group* table.

4.5.3 Protect a tables group against rollbacks

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

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

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

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

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

An “*audit_only*” or “*idle*” tables group cannot be protected.

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

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

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

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

An “*audit_only*” tables group cannot be unprotected.

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

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

4.5.4 Forced stop of a tables group

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

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

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

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

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

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

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

4.5.5 Forced drop of a tables group

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

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

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

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

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

4.5.6 Exporting and importing tables groups configurations

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

4.5.6.1 Export a tables groups configuration

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

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

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

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

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

The function returns the number of exported tables groups.

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

```
{
  "_comment": "Generated on database <db> with E-Maj version
<version> at <date_heure>",
  "tables_groups": [
    {
      "group": "ggg",
      "is_rollbackable": true|false,
      "comment": "ccc",
      "tables": [
        {
          "schema": "sss",
          "table": "ttt",
          "priority": ppp,
          "log_data_tablespace": "lll",
          "log_index_tablespace": "lll",
          "ignored_triggers": [ "tg1", "tg2", ... ]
        },
        {
          ...
        }
      ],
      "sequences": [
        {
          "schema": "sss",
          "sequence": "sss",
        },
        {
          ...
        }
      ],
    },
    ...
  ]
}
```

4.5.6.2 Import a tables groups configuration

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

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

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

The file must be accessible by the PostgreSQL instance.

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

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

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

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

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

The fourth parameter defines the mark to set on tables groups in *LOGGING* state. By default, the generated mark is *"IMPORT_%"*, where the % character represents the current time, formatted as *"hh.min.ss.mmmm"*.

The function returns the number of imported tables groups.

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

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

4.5.7 Knowing the existence or state of a tables group or a mark

The E-Maj administrator who wishes to write idempotent SQL scripts to manage its tables groups can take benefit from a few useful functions: [*emaj_does_exist_group\(\)*](#), [*emaj_is_logging_group\(\)*](#) and [*emaj_does_exist_mark_group\(\)*](#).

```
SELECT emaj.emaj_does_exist_group('<nom.du.groupe>');  
  
SELECT emaj.emaj_is_logging_group('<nom.du.groupe>');  
  
SELECT emaj.emaj_does_exist_mark_group('<nom.du.groupe>',  
'<nom.de.marque>');
```

They all return a boolean set to TRUE when respectively :

- a given tables group exists,
- a given tables group is in LOGGING state,
- a given mark exists.

By using these functions in a WHERE clause, it is possible for instance to only create a tables group if it does not exist yet.

```
SELECT emaj.emaj_create_group('<nom.du.groupe>')  
WHERE NOT emaj.emaj_does_exist_group('<nom.du.groupe>');
```

4.5.8 Erase traces from a dropped tables group

When a tables group is dropped, data about its previous life (creations, drops, starts, stops) are retained into two historical tables, with the same retention as for other historical data (Cf §5.5). But when dropping a tables group that had been mistakenly created, it may be useful to erase this traces immediately to avoid a pollution of these histories. A dedicated function is available for this purpose:

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

The tables group must not exist anymore.

The function returns the number of deleted traces.

4.6 MARKS MANAGEMENT FUNCTIONS

4.6.1 Comment a mark

It is possible to set a comment on a mark when it is set (Cf §4.3.3). But this can be also done later with:

```
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 mostly interesting when using *Emaj_web*, that systematically displays them in the group marks list (See §6.3.4). But they can also be found in the *mark_comment* column from the *emaj.emaj_mark* table.

4.6.2 Search a mark

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

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

or

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

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

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

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

4.6.3 Rename a mark

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

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

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

The function does not return any data.

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

4.6.4 Delete a mark

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

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

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

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

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

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

If a table has been detached from a tables group (Cf. 4.4.3), and the deleted mark corresponds to the last known mark for this table, the logs for the period between this mark and the preceding one are deleted,

4.6.5 Delete oldest marks

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

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

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

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

The function returns the number of deleted marks.

The function also performs a purge of the oldest events in the *emaj_hist* technical table (see §5.5).

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

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

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

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

4.6.6 Protect a mark against rollbacks

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

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

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

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

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

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

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

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

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

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

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

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

4.7 ROLLBACKS ADMINISTRATION FUNCTIONS

4.7.1 Estimate the rollback duration

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

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

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

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

The function returns an *INTERVAL* value.

The tables group must be in *LOGGING* state and the supplied mark must be usable for an E-Maj rollback.

This duration estimate is approximative. It takes into account:

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

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

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

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

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

The chapter §4.12.3 explains how to describe the group names array.

4.7.2 Monitor rollback operations

When the volume of recorded updates to cancel leads to a long rollback, it may be interesting to monitor the operation to appreciate how it progresses. A function, named *emaj_rollback_activity()*, and a client, *emajRollbackMonitor.php* (see §4.14), fit this need.

4.7.2.1 Prerequisite

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

If not already present, the *dblink* extension is automatically installed at *emaj* extension creation. But monitoring rollback operations also requires:

- to give the E-Maj administrator, and only him, the right to execute the *dblink_connect_u(text,text)* function, this right being not granted by default for security reasons;
- to insert a connection identifier usable by *dblink* into the *emaj_param* table (Cf §5.1).

```
GRANT EXECUTE ON FUNCTION dblink_connect_u(text,text) TO <admin_role>;  
INSERT INTO emaj.emaj_param (param_key, param_value_text)  
VALUES ('dblink_user_password','user=<user> password=<password>');
```

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

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

4.7.2.2 Monitoring function

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

Invoke it with the following statement:

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

The function does not require any input parameter.

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

- *rlbk_id* rollback identifier
- *rlbk_groups* tables groups array associated to the rollback
- *rlbk_mark* mark to rollback to
- *rlbk_mark_datetime* date and time when the mark to rollback to has been set
- *rlbk_is_logged* boolean taking the “true” value for logged rollbacks
- *rlbk_is_alter_group_allowed* *boolean indicating whether the rollback can target a mark set before a tables groups structure change*
- *rlbk_comment* comment
- *rlbk_nb_session* number of parallel sessions
- *rlbk_nb_table groups* number of tables contained in the processed tables groups
- *rlbk_nb_sequence groups* number of sequences contained in the processed tables groups
- *rlbk_eff_nb_table* number of tables having updates to cancel
- *rlbk_eff_nb_sequence* number of sequences having attributes to change
- *rlbk_status* rollback operation state
- *rlbk_start_datetime* rollback operation start timestamp
- *rlbk_planning_duration* planning phase duration
- *rlbk_locking_duration* tables locking phase duration
- *rlbk_elapse* elapse time spent since the rollback operation start
- *rlbk_remaining* estimated remaining duration
- *rlbk_completion_pct* estimated percentage of the completed work

An in progress rollback operation is in one of the following state:

- *PLANNING* the operation is in its initial planning phase,
- *LOCKING* the operation is setting locks,
- *EXECUTING* the operation is currently executing one of the planned steps.

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

The remaining duration estimate is approximate. Its precision is similar to the precision of the *emaj_estimate_rollback_group()* function (§4.7.1).

4.7.3 Comment a rollback operation

When calling *emaj_rollback_group()*, *emaj_logged_rollback_group()*, *emaj_rollback_groups()* or *emaj_logged_rollback_groups()* functions, one of the supplied parameters allows to record a comment associated to the rollback operation. Using the *emaj_comment_rollback()* function,

this comment can be updated or deleted. The same function allows to set a comment when it has not been done at rollback submission time.

```
SELECT emaj.emaj_comment_rollback('<rollback.id>', <comment>);
```

The rollback identifier is an integer. It is available in the execution report delivered at the rollback operation completion. It is also visible in the *emaj_rollback_activity()* function report (cf §4.7.2.2).

If the comment parameter is set to NULL, the existing comment, if any, is deleted.

The function does not return any data.

The comment can be added, modified or deleted when the operation is completed, but also when it is in progress if it is visible, i.e. if the *dblink_user_password* parameter is set into the *emaj_param* table (cf §5.1).

4.7.4 “Consolidate” a logged rollback

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

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

The concerned logged rollback operation is identified by the name of the mark generated at the end of the rollback. This mark must always exist, but may have been renamed. The 'EMAJ_LAST_MARK' keyword may be used as mark name to reference the last set mark. The *emaj_get_consolidable_rollbacks()* function may help to identify the rollbacks that may be consolidated (See §4.7.1).

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

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

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

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

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

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

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

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

4.7.5 List “consolidable rollbacks”

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

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

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

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

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

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

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

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

4.7.6 Update rollback operations state

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

When rollback functions cannot use *dblink* connections, all updates of these technical tables are all performed inside a single transaction. Therefore:

- any rollback operation that has not been completed is invisible in these technical tables,
- any rollback operation that has been validated is visible in these technical tables with a “*COMMITTED*” state.

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

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

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

The function returns the number of modified rollback operations.

4.8 COUNT DATA CONTENT CHANGES

Data stored into E-Maj technical tables or log tables allow to build statistics about recorded changes.

For this purpose, two functions sets are available for users. They produce statistics either at tables group level, or at individual table or sequence level.

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

4.8.1 Tables group level statistics

Six functions return statistics about recorded data content changes for **all tables or sequences** belonging to one or several **tables groups** on a **single marks interval** or since a mark:

- *emaj_log_stat_group()* and *emaj_log_stat_groups()* quickly deliver, for each table from one or several tables groups, the number of changes that have been recorded in their related log tables,
- *emaj_detailed_log_stat_group()* and *emaj_detailed_log_stat_groups()* provide more detailed information than *emaj_log_stat_group()*, the number of updates been reported per table, SQL verb type and connection role,
- *emaj_sequence_stat_group()* and *emaj_sequence_stat_groups()* return statistics about how sequences from one or several tables groups change.

4.8.1.1 Global statistics about log tables content for one or several tables groups

Full global statistics about logs content for a tables group 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 that contains the following columns:

- | | |
|-------------------------------------|--|
| ➤ <i>stat_group</i> : | tables group name (type TEXT), |
| ➤ <i>stat_schema</i> : | schema name (type TEXT), |
| ➤ <i>stat_table</i> : | table name (type TEXT), |
| ➤ <i>stat_first_mark</i> : | mark name of the period start (type TEXT), |
| ➤ <i>stat_first_mark_datetime</i> : | mark timestamp of the period start (type |
| | TIMESTAMPTZ), |
| ➤ <i>stat_first_time_id</i> : | internal time id of the period start (type BIGINT) |
| ➤ <i>stat_last_mark</i> : | mark name of the period end (type TEXT), |
| ➤ <i>stat_last_mark_datetime</i> : | mark timestamp of the period end (type |
| | TIMESTAMPTZ), |

- *stat_last_time_id*: internal time id of the period end (type BIGINT)
- *stat_rows*: number of recorded row changes

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

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

If the marks range is not contained by a single “log session”, i.e. if group stops/restarts occurred between these marks, a warning message is raised, indicating that data changes may have been not recorded.

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

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

If a table is removed from its group and later re-assigned to it during the requested time frame, several rows are returned in the statistics. In this case, *stat_first_time_id* and *stat_last_time_id* columns can be used to reliably sort these multiple time slices (internal server clock fluctuations may produce consecutive *stat_first_datetime* or *stat_last_datetime* not always in ascending order).

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

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

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

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

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


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

The chapter §4.12.3 explains how to describe the group names array.

4.8.1.2 Detailed statistics about logs for one or several tables groups

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 that contains the following columns:

- *stat_group*: tables group name (type TEXT),
- *stat_schema*: schema name (type TEXT),
- *stat_table*: table name (type TEXT),
- *stat_first_mark*: mark name of the period start (type TEXT),
- *stat_first_mark_datetime*: mark timestamp of the period start (type TIMESTAMPTZ),
- *stat_first_time_id*: internal time id of the period start (type BIGINT)
- *stat_last_mark*: mark name of the period end (type TEXT),
- *stat_last_mark_datetime*: mark timestamp of the period end (type TIMESTAMPTZ),
- *stat_last_time_id*: internal time id of the period end (type BIGINT)
- *stat_role*: connection role (type TEXT),
- *stat_verb*: type of the SQL verb that has performed the update (type TEXT, with values: *INSERT* / *UPDATE* / *DELETE* / *TRUNCATE*),
- *stat_rows*: number of recorded row changes (type BIGINT)

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

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

If the marks range is not contained by a single “log session”, i.e. if group stops/restarts occurred between these marks, a warning message is raised, indicating that data changes may have been not recorded.

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

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

If a table is removed from its group and later re-assigned to it during the requested time frame, several rows are returned in the statistics. In this case, *stat_first_time_id* and *stat_last_time_id* columns can be used to reliably sort these multiple time slices (internal server clock fluctuations may produce consecutive *stat_first_datetime* or *stat_last_datetime* not always in ascending order).

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

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

The chapter §4.12.3 explains how to describe the group names array.

4.8.1.3 Statistics about sequence changes for one or several tables groups

Global statistics about how sequences of a tables group change are available with this SQL statement:

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

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

- *stat_group*: tables group name (type TEXT),
- *stat_schema*: schema name (type TEXT),
- *stat_sequence*: sequence name (type TEXT),
- *stat_first_mark*: mark name of the period start (type TEXT),
- *stat_first_mark_datetime*: mark timestamp of the period start (type TIMESTAMPTZ),
- *stat_first_time_id*: internal time id of the period start (type BIGINT)
- *stat_last_mark*: mark name of the period end (type TEXT),
- *stat_last_mark_datetime*: mark timestamp of the period end (type TIMESTAMPTZ),
- *stat_last_time_id*: internal time id of the period end (type BIGINT)
- *stat_increments*: number of increments separating both sequence value at the period beginning and end (type BIGINT),
- *stat_has_structure_changed*: flag indicating whether any property of this sequence has changed during the period (type BOOLEAN).

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

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

The function returns one row per sequence, even if no change has been detected during the period.

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

If a sequence is removed from its group and later re-assigned to it during the requested time frame, several rows are returned in the statistics. In this case, *stat_first_time_id* and *stat_last_time_id* columns can be used to reliably sort these multiple time slices (internal server clock fluctuations may produce consecutive *stat_first_datetime* or *stat_last_datetime* not always in ascending order).

Sequence statistics are delivered quickly. Needed data are only stored into the small internal table that records the sequences states when marks are set.

But returned values may be approximative. Indeed, there is no way to detect temporary property changes during the period. Similarly, regarding the number of increments, there is no way to detect:

- *setval()* function calls (used by E-Maj rollbacks for instance),
- a return to the sequence minimum value (*MINVALUE*) if the sequence is cyclic (*CYCLE*) and the maximum value (*MAXVALUE*) has been reached,
- an increment change during the period.

For a given sequence, the number of increments is computed as the difference between the *LAST_VALUE* at the period end and the *LAST_VALUE* at the period beginning, divided by the *INCREMENT* value at the period beginning. As a consequence, it is possible to get negative numbers of increments.

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

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

The chapter §4.12.3 explains how to describe the group names array.

4.8.2 Table or sequence level statistics

Two functions return statistics about recorded data changes for a **single table or sequence** on **each elementary marks interval** of a given time frame:

- *emaj_log_stat_table()* quickly returns the number of changes that have been recorded for **a table** on each elementary marks interval,
- *emaj_log_stat_sequence()* returns the number of increments for **a sequence** on each elementary marks interval.

4.8.2.1 Statistics about changes recorded for a table

Statistics about changes recorded for a single table on a given time frame are available through one of these statements:

```
SELECT * FROM emaj.emaj_log_stat_table('<schema.name>', '<table.name>' [,
'<start.date-time>' [, '<end.date-time>']] );

or

SELECT * FROM emaj.emaj_log_stat_table('<schema.name>', '<table.name>',
'<start.tables.group>', '<start.mark>' [, '<end.tables.group>', '<end.mark>'] );
```

Both functions return a set of rows of type *emaj.emaj_log_stat_table_type* and containing the following columns:

- *stat_group*: tables group name (type TEXT),
- *stat_first_mark*: mark of the time slice lower bound (type TEXT),
- *stat_first_mark_datetime*: timestamp of the time slice lower bound (type TIMESTAMPTZ),
- *stat_first_time_id*: internal time id of the time slice lower bound (type BIGINT)
- *stat_is_log_start*: indicator of log start for the table (type BOOLEAN),
- *stat_last_mark*: mark of the time slice upper bound (type TEXT),
- *stat_last_mark_datetime*: timestamp of the time slice upper bound (type TIMESTAMPTZ),
- *stat_last_time_id*: internal time id of the time slice upper bound (type BIGINT)
- *stat_is_log_stop*: indicator of log stop for the table (type BOOLEAN),
- *stat_changes*: number of recorded row changes (type BIGINT)
- *stat_rollbacks*: number of E-Maj rollbacks executed on the time slice (type INT)

In the first function variant, the observation is framed by two start date-time and end date-time parameters of type *TIMESTAMPTZ*. The first returned interval surrounds the start date-time. The last returned interval surrounds the end date-time.

In the second function variant, the observation is framed by two marks defined by their tables group and mark names. These marks are just points in time: they not necessarily belong to the tables group owning the examined table. If the lower bound mark doesn't match a known state of the table (i.e. if the start tables group didn't owned the table at this start mark time), the first returned interval surrounds this first mark. Similarly, If the upper

bound mark doesn't match a known state of the table (i.e. if the end tables group didn't owned the table at this end mark time), the last returned interval surrounds this end mark.

If parameters defining the observation start are not set or are set to *NULL*, the observation starts at the oldest available data for the table.

If parameters defining the observation end are not set or are set to *NULL*, the observation ends at the table current state.

These functions don't return any rows for marks intervals when data changes were not recorded for the table. The *stat_is_log_start* and *stat_is_log_stop* columns help to detect gaps in the changes recording.

These statistics are quickly delivered because they do not need to scan log tables.

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

4.8.2.2 Statistics about changes recorded for a sequence

Statistics about changes recorded for a single sequence on a given time frame are available through one of these statements:

```
SELECT * FROM emaj.emaj_log_stat_sequence('<schema.name>',
'<sequence.name>' [, '<start.date-time>' [, '<end.date-time>']] );

or

SELECT * FROM emaj.emaj_log_stat_sequence('<schema.name>',
'<sequence.name>', '<start.tables.group>', '<start.mark>' [,
'<end.tables.group>', '<end.mark>'] );
```

Both functions return a set of rows of type *emaj.emaj_log_stat_sequence_type* and containing the following columns:

- *stat_group*: tables group name (type TEXT),
- *stat_first_mark*: mark of the time slice lower bound (type TEXT),
- *stat_first_mark_datetime*: timestamp of the time slice lower bound (type TIMESTAMPTZ),
- *stat_first_time_id*: internal time id of the time slice lower bound (type BIGINT)
- *stat_is_log_start*: indicator of log start for the sequence (type BOOLEAN),
- *stat_last_mark*: mark of the time slice upper bound (type TEXT),
- *stat_last_mark_datetime*: timestamp of the time slice upper bound (type TIMESTAMPTZ),
- *stat_last_time_id*: internal time id of the time slice upper bound (type BIGINT)
- *stat_is_log_stop*: indicator of log stop for the sequence (type BOOLEAN),

- *stat_increments*: number of sequence increments (type BIGINT)
- *stat_has_structure_changed*: TRUE if any property other than *last_value* has changed (type BOOLEAN),
- *stat_rollbacks*: number of E-Maj rollbacks executed on the time slice (type INT)

In the first function variant, the observation is framed by two start date-time and end date-time parameters of type *TIMESTAMPTZ*. The first returned interval surrounds the start date-time. The last returned interval surrounds the end date-time.

In the second function variant, the observation is framed by two marks defined by their tables group and mark names. These marks are just points in time: they not necessarily belong to the tables group owning the examined sequence. If the lower bound mark doesn't match a known state of the sequence (i.e. if the start tables group didn't owned the sequence at this start mark time), the first returned interval surrounds this first mark. Similarly, If the upper bound mark doesn't match a known state of the sequence (i.e. if the end tables group didn't owned the sequence at this end mark time), the last returned interval surrounds this end mark.

If parameters defining the observation start are not set or are set to *NULL*, the observation starts at the oldest available data for the sequence.

If parameters defining the observation end are not set or are set to *NULL*, the observation ends at the sequence current state.

These functions don't return any rows for marks intervals when data changes were not recorded for the sequence. The *stat_is_log_start* and *stat_is_log_stop* columns help to detect recording gaps.

4.9 EXAMINE DATA CONTENT CHANGES

4.9.1 Introduction

Log tables and the internal sequences states table are a real gold mine for the analysis of changes recorded between two marks. Aside already described cancellation (rollback) and statistics functions, it is possible to view the changes in different forms.

First of all, any user having *emaj_adm* or *emaj_viewer* privileges can directly query log tables. Their structure is described in §5.2.

But two functions, *emaj_dump_changes_group()* and *emaj_gen_sql_dump_changes_group()*, may help this examination. They allow to visualize data content changes for each table and sequence belonging to a tables group, for a period of time framed by two marks.

4.9.2 Output types

In order to cover many use cases, the data changes visualization may take different forms:

- a set of flat files created by *COPY TO* statements (thus stored into the PostgreSQL instance disk space);
- a *psql* script producing flat files using *\copy to* meta-commands (thus in the client disk space);
- a temporary table containing SQL statements allowing any client to directly visualize and analyze data changes.

4.9.3 Consolidation levels

Different levels of changes visualization are available through the concept of “consolidation”.

Without consolidation, each elementary change recorded into the log tables is returned. So one gets simple log table extracts for the targeted time period.

The consolidation process aims to only return the initial state (at the begin mark time) and/or the end state (at the end mark time) of each primary key, for which changes have been recorded. For each primary key, one gets a row of type ‘OLD’, representing the initial state if it already exists, and/or a row of type ‘NEW’ representing the final state, if it still exists. So the consolidation process needs that all examined tables own a *PRIMARY KEY*.

Two consolidation levels exists. The “partial consolidation” doesn’t take into account the data content. On the contrary, the “full consolidation” considers that changes producing strictly identical data between both marks are not to be returned.

Let’s take some examples, using a table described as (col1 INT PRIMARY KEY, col2 TEXT).

SQL beetwen both marks	Log table (*)	Partial consolidation	Full consolidation
INSERT (1,'A')	1,'A',NEW,1	1,'A',NEW	1,'A',NEW
UPDATE (1,'A') => (1,'B')	1,'A',OLD,1 1,'B',NEW,1	1,'A',OLD 1,'B',NEW	1,'A',OLD 1,'B',NEW
DELETE (1,'A')	1,'A',OLD,1	1,'A',OLD	1,'A',OLD
INSERT (1,'A') UPDATE (1,'A') => (1,'B')	1,'A',NEW,1 1,'A',OLD,2 1,'B',NEW,2	1,'B',NEW	1,'B',NEW
UPDATE (1,'A') => (1,'B') DELETE (1,'B')	1,'A',OLD,1 1,'B',NEW,1 1,'B',OLD,2	1,'A',OLD	1,'A',OLD
UPDATE (1,'A') => (1,'B') UPDATE (1,'B') => (1,'C') UPDATE (1,'C') => (1,'D')	1,'A',OLD,1 1,'B',NEW,1 1,'B',OLD,2 1,'C',NEW,2 1,'C',OLD,3 1,'D',NEW,3	1,'A',OLD 1,'D',NEW	1,'A',OLD 1,'D',NEW
INSERT (1,'A') DELETE (1,'A')	1,'A',NEW,1 1,'A',OLD,2	-	-
DELETE (1,'A') INSERT (1,'B')	1,'A',OLD,1 1,'B',NEW,2	1,'A',OLD 1,'B',NEW	1,'A',OLD 1,'B',NEW
UPDATE (1,'A') => (1,'B') UPDATE (1,'B') => (1,'A')	1,'A',OLD,1 1,'B',NEW,1 1,'B',OLD,2 1,'A',NEW,2	1,'A',OLD 1,'A',NEW	-
DELETE (1,'A') INSERT (1,'A')	1,'A',OLD,1 1,'A',NEW,2	1,'A',OLD 1,'A',NEW	-
UPDATE (1,'A') => (2,'A') UPDATE (2,'A') => (2,'B') UPDATE (2,'B') => (3,'B')	1,'A',OLD,1 2,'A',NEW,1 2,'A',OLD,2 2,'B',NEW,2 2,'B',OLD,3 3,'B',NEW,3	1,'A',OLD 3,'B',NEW	1,'A',OLD 3,'B',NEW

(*) the log table extract corresponds to columns (col1, col2, emaj_tuple, emaj_gid), other E-Maj technical columns not being mentioned.

NB: Some rare data types like JSON or XML have no equality operator. In this case, the full consolidation casts these columns into TEXT to compare initial and final values.

For each sequence, two rows are returned, corresponding to its initial and final state. In a full consolidation vision, no row is returned when both states are strictly identical.

4.9.4 The `emaj_dump_changes_group()` function

The `emaj_dump_changes_group()` function extracts changes from log tables and from the sequences states table and create files into the PostgreSQL instance disk space, using *COPY TO* statements.

```
SELECT emaj.emaj_dump_changes_group('<group.name>', '<start.mark>',  
'<end.mark>', '<options.list>', <tables/sequences.array>,  
'<output.directory>');
```

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

If the marks range is not contained by a single “log session”, i.e. if group stops/restarts occurred between these marks, a warning message is raised, indicating that data changes may have been not recorded.

The 4th parameter is a list of options, separated by commas. Options can be any of the following keywords (in alphabetic order):

- `COLS_ORDER = LOG_TABLE | PK`: defines the columns order in output files (`LOG_TABLE` = the same order than in log tables, `PK` = the primary key columns first);
- `CONSOLIDATION = NONE | PARTIAL | FULL`: defines the consolidation level; the default value is `NONE`;
- `COPY_OPTIONS = (options)`: defines the options to be used by the *COPY TO* statements; the list must be set between parenthesis; refer to the PostgreSQL documentation for the available options details (<https://www.postgresql.org/docs/current/sql-copy.html>);
- `EMAJ_COLUMNS = ALL | MIN | (columns list)`: restricts the returned E-Maj technical columns: `ALL` = all existing columns, `MIN` = a minimum number of columns, or an explicit columns list, set between parenthesis;
- `NO_EMPTY_FILES`: remove sfiles that do not contain any data;
- `ORDER_BY = PK | TIME`: defines the rows sort order in files; `PK` = the primary key order, `TIME` = the entry into the table log order;
- `SEQUENCES_ONLY`: only process sequences of the tables group; by default, tables are processed;

- **TABLES_ONLY**: only process tables of the tables groups; by default, sequences are processed.

The default value of the three **COLS_ORDER**, **EMAJ_COLUMNS** and **ORDER_BY** options depends on the consolidation level:

- when **CONSOLIDATION** = **NONE**, **COLS_ORDER** = **LOG_TABLE**, **EMAJ_COLUMNS** = **ALL** and **ORDER_BY** = **TIME**;
- when **CONSOLIDATION** = **PARTIAL** or **FULL**, **COLS_ORDER** = **PK**, **EMAJ_COLUMNS** = **MIN** and **ORDER_BY** = **PK**.

The 5th parameter allows to filter the tables and sequences to process. If the parameter is set to **NULL**, all tables and sequences of the tables group are processed. If specified, the parameter must be expressed as a non empty array of text elements, each of them representing a schema qualified table or sequence name. Both syntaxes can be used:

`ARRAY['sch1.tbl1','sch1.tbl2']`

or

`{ "sch1.tbl1" , "sch1.tbl2" }`

The effects of this tables/sequences selection and the **TABLES_ONLY** and **SEQUENCES_ONLY** options are cumulative. For instance, a sequence listed in the array will not be processed if the **TABLES_ONLY** option is set.

The output directory/folder set as 6th parameter must be an absolute pathname. It must have been created prior the function call and it must have the appropriate permission so that the PostgreSQL instance can write into it.

The function returns a textual message containing the number of generated files and their location.

When the tables group structure is stable between both targeted marks, the *emaj_dump_changes_group()* function generates one file per application table and sequence. Its name profile looks like:

`<schema.name>_<table/sequence.name>.changes`

The impact of tables group structure changes is presented below.

In order to manipulate generated files more easily, any inconvenient in file name characters, namely spaces, “/”, “\”, “\$”, “>”, “<”, “|”, single or double quotes and “*” are replaced by “_”. Beware, these file names adjustment may lead to duplicates, the last generated file overwriting then the previous ones.

All files are stored into the directory/folder set as 6th parameter. Already existing files are overwritten.

At the end of the operation, a file named **_INFO** is created in this same directory/folder. It contains:

- the operation characteristics, including the tables group, both selected marks, the options and the operation date and time;
- one line per created file, indicating the table or sequence name and the associated marks range.

During the extraction, the tables group may be in any idle or logging state.

As this function may generate large or very large files, it is user's responsibility to provide a sufficient disk space.

The log tables structure is described in §5.2.1.

4.9.5 The `emaj_gen_sql_dump_changes_group()` function

The `emaj_gen_sql_dump_changes_group()` function generates SQL statements that extract changes from log tables and from the sequences states table. Two versions exist, depending whether the 6th parameter is present.

```
SELECT emaj.emaj_gen_sql_dump_changes_group('<group.name>',  
      '<start.mark>', '<end.mark>', '<options.list>',  
      '<tables/sequences.array>');
```

or

```
SELECT emaj.emaj_gen_sql_dump_changes_group('<group.name>',  
      '<start.mark>', '<end.mark>', '<options.list>',  
      '<tables/sequences.array>', '<script.location>');
```

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

If the marks range is not contained by a single “log session”, i.e. if group stops/restarts occurred between these marks, a warning message is raised, indicating that data changes may have been not recorded.

The 4th parameter is a list of options, separated by commas. Options can be any of the following keywords (in alphabetic order):

- `COLS_ORDER = LOG_TABLE | PK`: defines the columns order in output results (`LOG_TABLE` = the same order than in log tables, `PK` = the primary key columns first);
- `CONSOLIDATION = NONE | PARTIAL | FULL`: defines the consolidation level; the default value is `NONE`;
- `EMAJ_COLUMNS = ALL | MIN | (columns list)`: restricts the returned E-Maj technical columns: `ALL` = all existing columns, `MIN` = a minimum number of columns, or an explicit columns list, set between parenthesis;
- `ORDER_BY = PK | TIME`: defines the rows sort order in output results; `PK` = the primary key order, `TIME` = the entry into the table log order;
- `PSQL_COPY_DIR = (directory)`: generates a `psql \copy` meta-command for each statement, using the directory name provided by the option; the directory name must be surrounded by parenthesis;

- `PSQL_COPY_OPTIONS` = (options): when `PSQL_COPY_DIR` is set, defines the options to be used by the generated `lcopy` to statements; the list must be set between parenthesis; refer to the PostgreSQL documentation for the available options details (<https://www.postgresql.org/docs/current/sql-copy.html>);
- `SEQUENCES_ONLY`: only process sequences of the tables group; by default, tables are processed;
- `SQL_FORMAT` = `RAW` | `PRETTY`: defines how generated statements will be formatted: `RAW` = on a single line, `PRETTY` = on several lines and indented, for a better readability;
- `TABLES_ONLY`: only process tables of the tables groups; by default, sequences are processed.

The default value of the three `COLS_ORDER`, `EMAJ_COLUMNS` and `ORDER_BY` options depends on the consolidation level:

- when `CONSOLIDATION` = `NONE`, `COLS_ORDER` = `LOG_TABLE`, `EMAJ_COLUMNS` = `ALL` and `ORDER_BY` = `TIME`;
- when `CONSOLIDATION` = `PARTIAL` or `FULL`, `COLS_ORDER` = `PK`, `EMAJ_COLUMNS` = `MIN` and `ORDER_BY` = `PK`.

The 5th parameter allows to filter the tables and sequences to process. If the parameter is set to `NULL`, all tables and sequences of the tables group are processed. If specified, the parameter must be expressed as a non empty array of text elements, each of them representing a schema qualified table or sequence name. Both syntaxes can be used:

```
ARRAY['sch1.tbl1','sch1.tbl2']
```

or

```
{ "sch1.tbl1" , "sch1.tbl2" }
```

The effects of this tables/sequences selection and the `TABLES_ONLY` and `SEQUENCES_ONLY` options are cumulative. For instance, a sequence listed in the array will not be processed if the `TABLES_ONLY` option is set.

The script file name parameter supplied as 6th parameter is optional. If it is not present, generated statements are left at the caller's disposal into an *emaj_temp_sql* temporary table. Otherwise, they are written into the file defined by this parameter. It must be an absolute pathname. The directory must have been created prior the function call and it must have the appropriate permission so that the PostgreSQL instance can write into it.

If any schema, table or column name contains a “\” (antislash) character, the *COPY* command executed to build the output script file duplicates this character. If a *sed* command is available on the server hosting the PostgreSQL instance, the *emaj_gen_sql_dump_changes_group()* function automatically removes such duplicated characters. Otherwise, manual script changes are required.

The function returns a textual message containing the number of generated statements and their location.

The *emaj_temp_sql* temporary table left at the caller's disposal when the 6th parameter is not present has the following structure:

- `sql_stmt_number` (INT): statement number

- `sql_line_number` (INT): line number for the statement (0 for the comments , 1 for a full statement when `SQL_FORMAT = RAW`, 1 to n when `SQL_FORMAT = PRETTY`)
- `sql_rel_kind` (TEXT): kind of relation ("table" ou "sequence")
- `sql_schema` (TEXT): schema containing the application table or sequence
- `sql_tblseq` (TEXT): table or sequence name
- `sql_first_mark` (TEXT): the first mark for this table or sequence
- `sql_last_mark` (TEXT): the last mark for this table or sequence
- `sql_group` (TEXT): tables group owning the table or sequence
- `sql_nb_changes` (BIGINT): estimated number of changes to process (NULL for sequences)
- `sql_file_name_suffix` (TEXT): file name suffix when the `PSQL_COPY_DIR` option has been set
- `sql_text` (TEXT): a line of text of the generated statement
- `sql_result` (BIGINT): column dedicated to the caller for its own purpose when using the temporary table.

The table contents:

- a first statement which is a general comment, reporting the main SQL generation characteristics: tables group, marks, options, etc (*sql_stmt_number* = 0);
- in case of full consolidation, a statement that changes the *enable_nestloop* configuration variable ; this statement is needed to optimize the log tables analysis (*sql_stmt_number* = 1);
- then, for each table and sequence:
 - a comment related to this table or sequence (*sql_line_number* = 0);
 - the analysis statement, on one or several lines, depending on the `SQL_FORMAT` option;
- in case of full consolidation, a last statement resetting the *enable_nestloop* variable to its previous value.

An index is created on columns `sql_stmt_number` and `sql_line_number`.

Once the *emaj_gen_sql_dump_changes_group()* function has been executed the caller can use the temporary table as he wants. With `ALTER TABLE` statements, he can even add columns, rename the table, transform it into a permanent table; He can also add an additional index, if needed. The estimated number of changes can be used to efficiently parallelize the statements execution.

For instance, the caller can generate a SQL script and store it locally with:

```
\copy (SELECT sql_text FROM emaj_temp_sql) to <fichier>
```

He can get the SQL statement for a given table with:

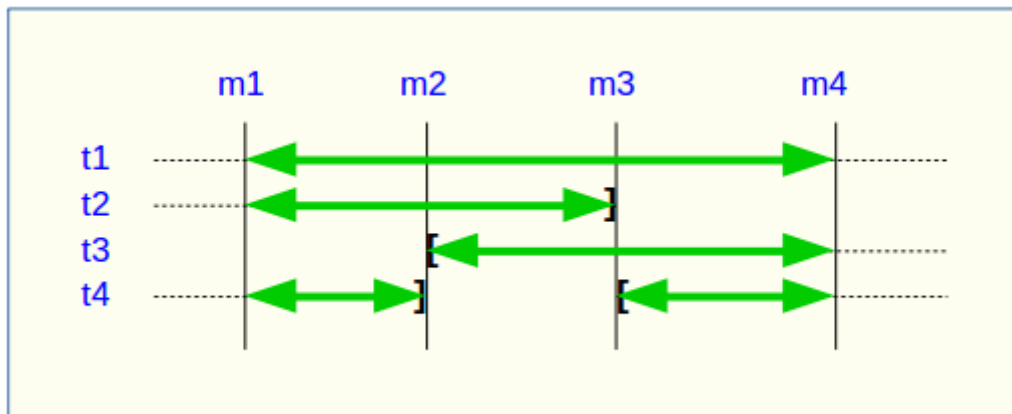
```
SELECT sql_text FROM emaj_temp_sql
WHERE sql_line_number >= 1
AND sql_schema = '<schema>' AND sql_tblseq = '<table>';
```

During the SQL generation, the tables group may be in any idle or logging state.

The *emaj_gen_sql_dump_changes_group()* function can be called by any role who has been granted *emaj_viewer* but not *emaj_adm* if no file is directly written by the function (i.e. if the 6th parameter is not present).

4.9.6 Impact of tables group structure changes

It may happen that the tables group structure changes during the examined marks frame.



A table or a sequence may have been removed from the group or assigned to the group between the selected start mark and end mark. In this case, as for table t2 and t3 in the example above, the extraction frames the real period of time the table or sequence belonged to the tables group. For this reason, the *_INFO* file and the *emaj_temp_sql* table contain information about the real marks frame used for each table or sequence.

A table or a sequence may even be removed from its group and reassigned to it later. In this case, as for table t4 above, there are several distinct extractions; the *emaj_dump_changes_group()* function generates several statements into the *emaj_temp_sql* table and the *emaj_gen_sql_dump_changes_group()* function writes several files for the same table or sequence. Then, the output file name suffix becomes *_1.changes*, *_2.changes*, etc.

4.10 GENERATE SQL SCRIPTS TO REPLAY LOGGED CHANGES

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

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

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

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

A *NULL* value or an empty string may be used as end mark, representing the current state.

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

If the marks range is not contained by a single “log session”, i.e. if group stops/restarts occurred between these marks, a warning message is raised, indicating that data changes may have been not recorded.

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

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

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

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

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

```
ARRAY['sch1.tbl1','sch1.tbl2']
```

or
'{ "sch1.tbl1" , "sch1.tbl2" }'

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

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

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



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

Statements are generated in the order of their initial execution.

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

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

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

The used technology may result to doubled backslashes in the output file. These doubled characters must be suppressed before executing the script, for instance, in Unix/Linux environment, using a command like:

```
sed 's/\\\\\\\\/g' <file.name> | psql ...
```

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

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

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

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

The chapter §4.12.3 explains how to describe the group names array.

4.11 OTHER FUNCTIONS

4.11.1 Get the emaj extension current version

The *emaj_get_version()* function returns the current version identifier of the *emaj* extension.

```
SELECT emaj.emaj_get_version();
```

4.11.2 Check the E-Maj environment consistency

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

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

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

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

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

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

'No error detected'

The function also returns warnings when:

- a sequence linked to a column belongs to a tables group, but the associated table does not belong to the same tables group,
- a table of a tables group is linked to another table by a foreign key, but the associated table does not belong to the same tables group,
- a foreign key is inherited from a partitionned table but either is not *DEFERRABLE* or holds an *ON DELETE* or *ON UPDATE* clause, blocking its potential drop / recreation during an E-Maj rollback in both cases,
- parameters in the *emaj_param* table are unknown (see §5.1),
- the *dblink* connection is not operationnal,
- event triggers protecting E-Maj are missing or are disabled.

The *emaj_verify_all()* function can be executed by any role belonging to *emaj_adm* or *emaj_viewer* roles (the *dblink* connection not being tested for the later).

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.

4.11.3 Exporting and importing parameters configurations

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

4.11.3.1 Export a parameters configuration

Two versions of the *emaj_export_parameters_configuration()* function export all the parameters registered in the *emaj_param* table in a JSON structure.

The parameters data can be written to a file with:

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

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

The function returns the number of exported parameters.

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

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

4.11.3.2 Import a parameters configuration

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

A file containing parameters to load can be read with:

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

The file path must be accessible by the PostgreSQL instance.

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

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

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

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

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

The function returns the number of imported parameters.

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

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

4.11.4 Identify the current log table linked to an application table

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

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

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

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

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

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

4.11.5 Purge history data

E-Maj keeps some historical data: traces of elementary operations, E-Maj rollback details, tables groups structure changes (see §5.5). Oldest traces are automatically purged by the extension. But it is also possible to purge these obsolete traces on demand using:

```
SELECT emaj.emaj_purge_histories('<retention.delay>');
```

The *<retention.delay>* parameter is of type *INTERVAL*. It overloads the '*history_retention*' parameter of the *emaj_param* table.

4.11.6 Deactive/reactive event triggers

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

To deactivate the existing event triggers:

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

The function returns the number of deactivated event triggers .

To reactivate existing event triggers:

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

The function returns the number of reactivated event triggers.

4.11.7 Snap tables and sequences of a tables group

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

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

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

The third parameter defines the output files format. It is a character string that matches the precise syntax available for the *COPY TO SQL* statement. Look at the PostgreSQL documentation to get more details about the available options (<https://www.postgresql.org/docs/current/sql-copy.html>).

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

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

New files will overwrite existing files of the same name.

Created files are named with the following pattern:

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

In order to manipulate generated files more easily, any inconvenient in file name characters, namely spaces, “/”, “\”, “\$”, “>”, “<”, “|”, single or double quotes and “*” are replaced by “_”. Beware, these file names adjustment may lead to duplicates, the last generated file overwriting then the previous ones.

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

Files corresponding to tables contain one record per row, in the format corresponding to the supplied parameter. These records are sorted on the primary key ascending order (or on all columns if the table has no primary key). Each row contains all table columns, including generated columns.

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.

4.12 MULTI-GROUPS FUNCTIONS

4.12.1 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.

4.12.2 Functions list

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

Multi-groups functions	Relative mono-group function	§
<i>emaj.emaj_start_groups()</i>	<i>emaj.emaj_start_group()</i>	4.3.2
<i>emaj.emaj_stop_groups()</i>	<i>emaj.emaj_stop_group()</i>	4.3.6
<i>emaj.emaj_set_mark_groups()</i>	<i>emaj.emaj_set_mark_group()</i>	4.3.3
<i>emaj.emaj_rollback_groups()</i>	<i>emaj.emaj_rollback_group()</i>	4.3.4
<i>emaj.emaj_logged_rollback_groups()</i>	<i>emaj.emaj_logged_rollback_group()</i>	4.3.5
<i>emaj.emaj_estimate_rollback_groups()</i>	<i>emaj.emaj_estimate_rollback_group()</i>	4.7.1
<i>emaj.emaj_log_stat_groups()</i>	<i>emaj.emaj_log_stat_group()</i>	4.8.1.1
<i>emaj.emaj_detailed_log_stat_groups()</i>	<i>emaj.emaj_detailed_log_stat_group()</i>	4.8.1.2
<i>emaj.emaj_sequence_stat_groups()</i>	<i>emaj.emaj_sequence_stat_group()</i>	4.8.1.3
<i>emaj.emaj_gen_sql_groups()</i>	<i>emaj.emaj_gen_sql_group()</i>	4.10

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

4.12.3 Syntax for groups array

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

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

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

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

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

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

Both syntax are equivalent.

4.12.4 Other considerations

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

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

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

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

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

4.13 PARALLEL ROLLBACK CLIENT

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

4.13.1 Sessions

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

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

Tables are assigned to sessions so that the estimated session durations be the most balanced as possible.

4.13.2 Prerequisites

The provided tool is coded in perl. It needs that the perl software with the *DBI* and *DBD::Pg* modules be installed on the server that executes the command (which is not necessarily the same as the one that hosts the PostgreSQL instance).

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

4.13.3 Syntax

The syntax is:

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

General options:

- l specifies that the requested rollback is a “*logged rollback*” (see §4.3.5)
- a specifies that the requested rollback is allowed to reach a mark set before an alter group operation (see §4.4)
- c <comment> sets a comment on the rollback operation
- v displays more information about the execution of the processing
- help only displays a command help
- version only displays the software version

Connection options:

- d database to connect to

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

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

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

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

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

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

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

It is possible to monitor the multi-session rollback operations with the same tools as for mono-session rollbacks: *emaj_rollback_activity()* function, the *emajRollbackMonitor* command (See §4.14) or the Emaj_web rollback monitor page. As for mono-session rollbacks, the *dblink_user_password* parameter (See §5.1) must be set in order to get detailed status of the operations progress.

In order to test the *emajParallelRollback* client, the E-Maj extension supplies a test script, *emaj_prepare_parallel_rollback_test.sql*. It prepares an environment with two tables groups containing some tables and sequences, on which some updates have been performed, with intermediate marks. Once this script has been executed under *psql*, the command displayed at the end of the script can be simply run.

4.13.4 Examples

The command:

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

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

The command:

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

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

4.14 ROLLBACK MONITORING CLIENT

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

4.14.1 Prerequisite

The provided tool is coded in perl. It needs that the perl software with the *DBI* and *DBD::Pg* modules be installed on the server that executes the command (which is not necessarily the same as the one that hosts the PostgreSQL instance).

In order to get detailed information about the in-progress rollback operations, it is necessary to set the *dblink_user_password* parameter and give right to execute the *dblink_connect_u* function (See §4.7.2.1).

4.14.2 Syntax

The command has the following syntax:

emajRollbackMonitor.pl [OPTIONS]...

General options:

- i time interval between 2 displays (in seconds, default = 5s)
- n number of displays (default = 1, 0 for infinite loop)
- a maximum time interval for rollback operations to display (in hours, default = 24h)
- l maximum number of completed rollback operations to display (default = 3)
- help only displays a command help
- version only displays the software version

Connection options:

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

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

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

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

4.14.3 Examples

The command:

```
emajRollbackMonitor.pl -i 3 -n 10
```

displays 10 times and every 3 seconds, the list of in progress rollback operations and the list of the at most 3 latest rollback operations completed in the latest 24 hours.

The command:

```
emajRollbackMonitor.pl -a 12 -l 10
```

displays only once the list of in progress rollback operations and the list of at most 10 operations completed in the latest 12 hours.

Display example:

```
E-Maj (version 4.2.0) - Monitoring rollbacks activity
-----
21/03/2023 - 08:31:23
** rollback 34 started at 2023-03-21 08:31:16.777887+01 for groups {myGroup1,myGroup2}
   status: COMMITTED ; ended at 2023-03-21 08:31:16.9553+01
** rollback 35 started at 2023-03-21 08:31:17.180421+01 for groups {myGroup1}
   status: COMMITTED ; ended at 2023-03-21 08:31:17.480194+01
-> rollback 36 started at 2023-03-21 08:29:26.003502+01 for groups {group20101}
   status: EXECUTING ; completion 85 %; 00:00:20 remaining
-> rollback 37 started at 2023-03-21 08:29:16.123386+01 for groups {group20102}
   status: LOCKING ; completion 0 %; 00:22:20 remaining
-> rollback 38 started at 2023-03-21 08:30:16.130833+01 for groups {group20103}
   status: PLANNING ; completion 0 %
```

4.15 CHANGES RECORDING MONITORING CLIENT

E-Maj delivers an external client to run as a command that monitors the tables data changes and the sequences progress.

4.15.1 Prerequisite

The provided tool is coded in *perl*. It needs that the *perl* software with the *DBI* and *DBD::Pg* modules be installed on the server that executes the command (which is not necessarily the same as the one that hosts the PostgreSQL instance).

4.15.2 Syntax

The command syntax is:

emajStat.pl [OPTIONS]...

General options:

--interval	time interval between 2 displays (in seconds, default = 5s)
--iteration	number of display iterations (default = 0 = infinite loop)
--include-groups	regexp to select tables groups to process (default = '.' = all)
--exclude-groups	regexp to exclude tables groups to process (default = '' = no exclusion)
--max-groups	limits the number of tables groups to display (default = 5)
--include-tables	regexp to select tables to process (default = '.' = all)
--exclude-tables	regexp to exclude tables to process (default = '' = no exclusion)
--max-tables	limits the number of tables to display (default = 20)
--include-sequences	regexp to select sequences to process (default = '.' = all)
--exclude-sequences	regexp to exclude sequences to process (default = '' = no exclusion)
--max-sequences	limits the number of sequences to display (default = 20)
--no-cls	do not clear the screen at each display
--sort_since_previous	sorts groups, tables and sequences on changes since the previous display (default = sort on the changes since the latest mark)
--max-relation-name-length	limits the size of full tables and sequences names (default = 32 characters)
--help	only displays a command help
--version	only displays the software version

Connection options:

-d	database to connect to
-h	host to connect to

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

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

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

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

4.15.3 Example

The command:

```
emajStat.pl --interval 30 --max-tables 40 --exclude-tables '\.sav$' --max-sequences 0
```

displays every 30 seconds, cumulated changes of the 5 most active tables groups and the 40 most active tables, tables named with a “*.sav*” suffix being excluded, no sequences being processed.

4.15.4 Display description

Example of display:

```
E-Maj (version 4.5.0) - Monitoring logged changes on database regression
(@127.0.0.1:5412)
-----
2024/08/15 08:12:59 - Logging: groups=2/3 tables=11/11 sequences=4/4 - Changes since
1.004 sec: 0 (0.000 c/s)
Group name + Latest mark + Changes since mark + Changes since prev.
myGroup1 | Multi-1 (2024/08/15 08:12:38) | 359 (17.045 c/s) | 0 (0.000 c/s)
Table name + Group + Changes since mark + Changes since prev.
myschema1.mytbl1 | myGroup1 | 211 (10.018 c/s) | 0 (0.000 c/s)
myschema1.myTbl3 | myGroup1 | 60 ( 2.849 c/s) | 0 (0.000 c/s)
myschema1.mytbl2b | myGroup1 | 52 ( 2.469 c/s) | 0 (0.000 c/s)
myschema1.mytbl2 | myGroup1 | 27 ( 1.282 c/s) | 0 (0.000 c/s)
myschema1.mytbl4 | myGroup1 | 9 ( 0.427 c/s) | 0 (0.000 c/s)
Sequence name + Group + Changes since mark + Changes since prev.
myschema1.mytbl2b_col20_seq | myGroup1 | -5 (-0.237 c/s) | 0 (0.000 c/s)
myschema1.myTbl3_col31_seq | myGroup1 | -20 (-0.950 c/s) | 0 (0.000 c/s)
```

The first title line reminds the *emajStat* client version, the logged on database, and the IP address and port when no socket is used for the connexion.

The second line displays:

- the current date and time,
- the number of tables groups in logging state, the number of tables and sequences assigned to those tables groups,
- the total number of changes recorded since the previous display and the throughput in changes per second.

Then, a tables groups list appears, with:

- the group name,
- the name and timestamp of the latest mark set on the group,
- the cumulated number of changes recorded for all selected tables of the group since the latest mark and the related throughput,
- the cumulated number of changes recorded for all selected tables of the group since the latest display and the related throughput.

By default, this groups table is ordered by the numbers of changes since the latest mark in descending order and then in group names in ascending order. Using the *--sort-since-previous* option the table is sorted first on the number of changes since the previous display. If the number of groups is greater than the *--max-groups* option, only the most active are displayed.

Then, one finds the lists of selected tables and sequences, with:

- the table or sequence name, prefixed with their schema, and potentially truncated to fit the *--max-relation-name-length* option,
- the group name,
- the cumulated number of changes recorded for the table or the number of sequences increments since the latest mark and the related throughput,
- the cumulated number of changes recorded for the table or the number of sequences increments since the latest display and the related throughput.

Both lists are ordered by the same criteria than the tables groups. Similarly, the *--max-tables* and *--max-sequences* options limit the number of displayed tables or sequences.

At the first display or when a tables group structure changes (for instance when a table or sequence is assigned to or removed from their group) or when a mark is set, the statistics about changes since the previous display are masked.

If an E-Maj rollback is performed on a tables group, it may happen that negative numbers of changes and changes per second be displayed.

5 MISCELLANEOUS

5.1 PARAMETERS

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

The *emaj_param* table structure is the following:

Column	Type	Description
<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_numeric</i>	NUMERIC	parameter value, if its type is numeric (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)

Once the *emaj* extension is installed, the *emaj_param* table is empty, all parameters having their default value. But the E-Maj administrator may insert rows to set up other values for any parameter.

Presented in alphabetic order, the existing key values are:

- *alter_log_table* (text) *ALTER TABLE* directive executed at the log table creation ; no *ALTER TABLE* executed by default (see §5.2.2).
- *avg_fkey_check_duration* (interval) default value = 20 µs ; defines the average duration of a *foreign key* value check ; can be modified to better represent the performance of the server that hosts the database (see §4.7.1).
- *avg_row_delete_log_duration* (interval) default value = 10 µs ; defines the average duration of a log row deletion ; can be modified to better represent the performance of the server that hosts the database (see §4.7.1).
- *avg_row_rollback_duration* (interval) default value = 100 µs ; defines the average duration of a row rollback ; can be modified to better represent the performance of the server that hosts the database (see §4.7.1).
- *dblink_user_password* (text) *empty string by default ; format = 'user=<user> password=<password>'; defines the user and password that elementary functions executing E-Maj rollback operations can use to update the internal rollback monitoring tables with autonomous transactions. This is required to monitor the in progress E-Maj rollback operations (see §4.3.4 and §4.3.5).*
- *fixed_dblink_rollback_duration* (interval) default value = 4 ms ; defines an additional cost for each rollback step when a *dblink* connection is used ; can be

modified to better represent the performance of the server that hosts the database (see §4.7.1).

- *fixed_table_rollback_duration* (interval) default value = 1 ms ; defines a fixed rollback cost for any table belonging to a group ; can be modified to better represent the performance of the server that hosts the database (see §4.7.1).
- *fixed_step_rollback_duration* (interval) default value = 2,5 ms ; defines a fixed cost for each rollback step ; can be modified to better represent the performance of the server that hosts the database (see §4.7.1).
- *history_retention* (interval) default value = 1 year ; it can be adjusted to change the retention delay of rows in the *emaj_hist* history table and some other technical tables (see § 5.5) ; a value greater or equal to 100 years is equivalent to infinity.

Below is an example of a SQL statement that defines a retention delay of history table's rows equal to 3 months:

```
INSERT INTO emaj.emaj_param (param_key, param_value_interval) VALUES  
('history_retention','3 months'::interval);
```

Any change in the *emaj-param* table's content is logged into the *emaj_hist* table (cf §5.5).

Only super-user and roles having *emaj_adm* rights can access the *emaj_param* table.

Roles having *emaj_viewer* rights can only access a part of the *emaj_param* table, through the *emaj.emaj_visible_param* view. This view just masks the real value of the *param_value_text* column for the 'dblink_user_password' key.

The *emaj_export_parameters_configuration()* and *emaj_import_parameters_configuration()* functions allow to save the parameters values and restore them (Cf §4.11.3).

5.2 LOG TABLES STRUCTURE

5.2.1 Standart structure

The structure of log tables is directly derived from the structure of the related application tables. The log tables contain the same columns with the same type. But they also have some additional technical columns:

- *emaj_verb* type of the SQL verb that generated the update (INS, UPD, DEL, TRU)
- *emaj_tuple* row version (OLD for DEL, UPD and TRU ; NEW for INS and UPD ; empty string for TRUNCATE events)
- *emaj_gid* log row identifier
- *emaj_changed* log row insertion timestamp
- *emaj_txid* transaction id (the PostgreSQL *txid*) that performed the update
- *emaj_user* connection role that performed the update

When a *TRUNCATE* statement is executed for a table, each row of this table is recorded (with *emaj_verb* = *TRU* and *emaj_tuple* = *OLD*). A row is added, with *emaj_verb* = *TRU*, *emaj_tuple* = "" and the other columns being set to *NULL*. This row is used by the sql scripts generation.

5.2.2 Adding technical columns

It is possible to add one or several technical columns to enrich the traces. These columns value must be set as a default value (a *DEFAULT* clause) associated to a function (so that the log triggers are not impacted).

To add one or several technical columns, a parameter of key '*alter_log_table*' must be inserted into the *emaj_param* table (Cf §5.1). The associated text value must contain an ALTER TABLE clause. At the log table creation time, if the parameter exists, an ALTER TABLE statement with this parameter is executed.

For instance, one can add to log tables a column to record the value of the '*application_name*' connection field with:

```
INSERT INTO emaj.emaj_param (param_key, param_value_text) VALUES
('alter_log_table', 'ADD COLUMN extra_col_apptime TEXT
DEFAULT current_setting("application_name");');
```

Several ADD COLUMN directives may be concatenated, separated by a comma. For instance, to create columns recording the ip adress and port of the connected client:

```
INSERT INTO emaj.emaj_param (param_key, param_value_text) VALUES
('alter_log_table', 'ADD COLUMN emaj_user_ip INET DEFAULT inet_client_addr(),
ADD COLUMN emaj_user_port INT DEFAULT inet_client_port());');
```

To change the structure of existing log tables once the *alter_log_table* parameter has been set, the tables groups must be dropped and then recreated.

5.3 RELIABILITY

Two additional elements help in ensuring the E-Maj reliability: internal checks are performed at some key moments of tables groups life and event triggers can block some risky operations.

5.3.1 Internal checks

When a function is executed to start a tables group, to set a mark or to rollback a tables group, E-Maj performs some checks in order to verify the integrity of the tables groups to process.

These tables group integrity checks verify that:

- the PostgreSQL version at tables group creation time is compatible with the current version,
- each application sequence or table of the group always exists,
- each table of the group has its log table, its log function and its triggers,
- the log tables structure always reflects the related application tables structure, and contains all required technical columns,
- for each table, the generated columns list is unchanged,
- for *ROLLBACKABLE* tables groups, no table has been altered as *UNLOGGED*,
- for *ROLLBACKABLE* tables groups, application tables have their primary key and their structure has not changed.

By using the *emaj_verify_all()* function (§4.11.2), the administrator can perform the same checks on demand on all tables groups.

5.3.2 Event triggers

Installing E-Maj adds 2 event triggers of type “*sql_drop*”:

- *emaj_sql_drop_trg* blocks the drop attempts of:
 - ✓ any E-Maj object (log schema, log table, log sequence, log function and log trigger),
 - ✓ any application table or sequence belonging to a tables group in “*LOGGING*” state,
 - ✓ any primary key of a table belonging to a rollbackable tables group,
 - ✓ any schema containing at least one table or sequence belonging to a tables group in “*LOGGING*” state.
- *emaj_protection_trg* blocks the drop attempts of the *emaj* extension itself and the main *emaj* schema.

Installing E-Maj also adds 1 event trigger of type “*table_rewrite*”:

- *emaj_table_rewrite_trg* blocks any structure change of application or log table.

It is possible to deactivate and reactivate these event triggers thanks to 2 functions: *emaj_disable_protection_by_event_triggers()* and *emaj_enable_protection_by_event_triggers()* (see §4.11.6).

However, the protections do not cover all risks. In particular, they do not prevent any tables or sequences renaming or any schema change. And some other DDL statements altering tables structure do not fire any trigger.

5.4 MANAGEMENT OF GENERATED COLUMNS

As a remainder, PostgreSQL allows to create generated columns, using the *GENERATED ALWAYS AS expression* clause. With the *STORED* attribute, the expression result is physically stored into the table. Without this attribute, the expression result is just dynamically computed at table examination.

5.4.1 Generated columns in log tables

Generated columns of application tables are represented as standart columns in log tables. Their content is set by the log trigger with the same content as the application table columns: the expression result for *STORED* generated columns or the *NULL* value for virtual generated columns.

As a result, in order to visualize the impact of row changes on *STORED* generated columns, it's possible to directly look at the corresponding column in the log table. On the contrary, for virtual generated columns, the expression associated to the column must be used in the SQL statement.

5.4.2 DDL changes on generated columns

It is possible to **change the expression** of a virtual generated column (*ALTER TABLE ... ALTER COLUMN ... SET EXPRESSION ...*) while the table belongs to a tables group. But such an operation is blocked by E-Maj if the column is *STORED*. In order to change the generated column expression of a table that belongs to a tables group, the table must be removed from its group before the expression change and reassigned after, the E-Maj rollback of the table targeting a mark prior the change becoming impossible.

PostgreSQL allows to **drop the expression** of a *STORED* generated column (*ALTER TABLE ... ALTER COLUMN ... DROP EXPRESSION*). Then, the column becomes a standart column and keeps its current data content. This column definition can be changed while the table is assigned to a tables group. But in case of E-Maj rollback, the new expression will be used to feed the application table, even to revert data changes prior the expression change.

It is also possible to **transform** a non generated column into a generated column and vice versa (*ALTER TABLE ... DROP COLUMN ..., ADD COLUMN ...*). But this would cause damages at E-Maj rollback time: at best the operation would fail with an error message, at worse, the column content would be corrupted. Therefore, checks are performed before every mark set and rollback attempt in order to be sure that the generated columns list of each application table is stable.

5.5 TRACES OF OPERATIONS

5.5.1 The emaj_hist table

All operations performed by E-Maj, and that impact in any way a tables group, are traced into a table named *emaj_hist*.

Any user having *emaj_adm* or *emaj_viewer* rights may look at the *emaj_hist* content.

The *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>	TIMESTAMPZ	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:

- *ADJUST_GROUP_PROPERTIES* adjust the *group_has_waiting_changes* column content of the *emaj_group* table
- *ASSIGN_SEQUENCE* sequence assigned to a tables group
- *ASSIGN_SEQUENCES* sequences assigned to a tables group
- *ASSIGN_TABLE* table assigned to a tables group
- *ASSIGN_TABLES* tables assigned to a tables group
- *CLEANUP_RLBK_STATE* cleanup the state of recently completed rollback operations
- *COMMENT_GROUP* comment set on a group
- *COMMENT_MARK_GROUP* comment set on a mark for a tables group
- *COMMENT_ROLLBACK* comment set on an E-Maj rollback
- *CONSOLIDATE_RLBK_GROUP* consolidate a logged rollback operation
- *CREATE_GROUP* tables group creation
- *DBLINK_OPEN_CNX* open a *dblink* connection for a rollback operation
- *DBLINK_CLOSE_CNX* close a *dblink* connection for a rollback operation
- *DELETE_MARK_GROUP* mark deletion for a tables group
- *DISABLE_PROTECTION* desactivate event triggers
- *DROP_GROUP* tables group suppression
- *EMAJ_INSTALL* E-Maj installation or version update

➤ <i>ENABLE_PROTECTION</i>	activate event triggers
➤ <i>EXPORT_GROUPS</i>	export a tables groups configuration
➤ <i>EXPORT_PARAMETERS</i>	export an E-maj parameters configuration
➤ <i>FORCE_DROP_GROUP</i>	tables group forced suppression
➤ <i>FORCE_STOP_GROUP</i>	tables group forced stop
➤ <i>FORGET_GROUP</i>	erase historical traces for a dropped tables group
➤ <i>GEN_SQL_GROUP</i>	generation of a psql script to replay updates for a tables group
➤ <i>GEN_SQL_GROUPS</i>	generation of a psql script to replay updates for several tables groups
➤ <i>IMPORT_GROUPS</i>	import a tables groups configuration
➤ <i>IMPORT_PARAMETERS</i>	import an E-maj parameters configuration
➤ <i>LOCK_GROUP</i>	lock set on tables of a group
➤ <i>LOCK_GROUPS</i>	lock set on tables of several groups
➤ <i>LOCK_SESSION</i>	lock set on tables for a rollback session
➤ <i>MODIFY_TABLE</i>	table properties change
➤ <i>MODIFY_TABLES</i>	tables properties change
➤ <i>MOVE_SEQUENCE</i>	sequence moved to another tables group
➤ <i>MOVE_SEQUENCES</i>	sequences moved to another tables group
➤ <i>MOVE_TABLE</i>	table moved to another tables group
➤ <i>MOVE_TABLES</i>	tables moved to another tables group
➤ <i>PROTECT_GROUP</i>	set a protection against rollbacks on a group
➤ <i>PROTECT_MARK_GROUP</i>	set a protection against rollbacks on a mark for a group
➤ <i>PURGE_HISTORIES</i>	delete from the historical tables the events prior the retention delay
➤ <i>REMOVE_SEQUENCE</i>	sequence removed from its tables group
➤ <i>REMOVE_SEQUENCES</i>	sequences removed from their tables group
➤ <i>REMOVE_TABLE</i>	table removed from its tables group
➤ <i>REMOVE_TABLES</i>	tables removed from their tables group
➤ <i>RENAME_MARK_GROUP</i>	mark rename for a tables group
➤ <i>RESET_GROUP</i>	log tables content reset for a group
➤ <i>ROLLBACK_GROUP</i>	rollback updates for a tables group
➤ <i>ROLLBACK_GROUPS</i>	rollback updates for several tables groups
➤ <i>ROLLBACK_TABLE</i>	rollback updates for one table
➤ <i>ROLLBACK_SEQUENCE</i>	rollback one sequence
➤ <i>SET_MARK_GROUP</i>	mark set on a tables group
➤ <i>SET_MARK_GROUPS</i>	mark set on several tables groups
➤ <i>SNAP_GROUP</i>	snap all tables and sequences for a group
➤ <i>SNAP_LOG_GROUP</i>	snap all log tables for a group
➤ <i>START_GROUP</i>	tables group start
➤ <i>START_GROUPS</i>	tables groups start
➤ <i>STOP_GROUP</i>	tables group stop
➤ <i>STOP_GROUPS</i>	tables groups stop
➤ <i>UNPROTECT_GROUP</i>	remove a protection against rollbacks on a group
➤ <i>UNPROTECT_MARK_GROUP</i>	remove a protection against rollbacks on a mark for a group

The *hist_event* column can take the following values:

- *BEGIN*

➤ DELETED PARAMETER	parameter deleted from <i>emaj_param</i>
➤ END	
➤ EVENT TRIGGER RECREATED	
➤ EVENT TRIGGERS DISABLED	
➤ EVENT TRIGGERS ENABLED	
➤ GROUP CREATED	new tables group created
➤ INSERTED PARAMETER	parameter inserted into <i>emaj_param</i>
➤ LOG DATA TABLESPACE CHANGED	tablespace for the log table modified
➤ LOG INDEX TABLESPACE CHANGED	tablespace for the log index modified
➤ LOG_SCHEMA CREATED	
➤ LOG_SCHEMA DROPPED	
➤ MARK DELETED	
➤ NAMES PREFIX CHANGED	E-Maj names prefix modified
➤ NOTICE	notice message issued by a rollback
➤ PRIORITY CHANGED	priority level modified
➤ SEQUENCE ADDED	sequence added to a logging tables group
➤ SEQUENCE MOVED	sequence moved from one group to another
➤ SEQUENCE REMOVED	sequence removed from a logging tables group
➤ TABLE ADDED	table added to a logging tables group
➤ TABLE MOVED	table moved from one group to another
➤ TABLE REMOVED	table removed from a logging tables group
➤ TABLE REPAIRED	table repaired for E-Maj
➤ TIME STAMP SET	internal time stamp recorded
➤ TRIGGERS TO IGNORE CHANGED	set of application triggers to ignore at rollback time changed
➤ UPDATED PARAMETER	parameter updated in <i>emaj_param</i>
➤ WARNING	warning message issued by a rollback

5.5.2 Other history tables

Several other internal tables store historical data :

- *emaj_version_hist* keeps the trace of the extension version changes;
- *emaj_group_hist* records tables groups creations and drops;
- *emaj_rel_hist* keeps tables and sequences assignments to tables groups;
- *emaj_log_session* records the period of time when the tables groups are enabled (started);
- and several other tables handling E-Maj rollbacks data.

The Emaj_web client is the easiest way to examine these tables content.

5.5.3 Purge obsolete traces

When a tables group is started with reset (*emaj_start_group()* function) or when old marks are deleted (*emaj_delete_before_mark_group()* function), the oldest events are deleted from most historical tables. The events kept are those not older than:

- a parametrised retention delay,
- the oldest mark,
- and the oldest uncompleted rollback operation.

By default, the retention delay for events equals 1 year. But this value can be modified at any time by inserting the *history_retention* parameter into the *emaj_param* table with a SQL statement (see §5.1). If the *history_retention* parameter is set to 100 years or more, no histories purge is executed.

The obsolete traces purge can also be initiated by explicitly calling the *emaj_purge_histories()* function (see §4.11.5). The input parameter of the function defines a retention delay that overloads the *history_retention* parameter of the *emaj_param* table.

In order to schedule purges periodically, it is possible to:

- set the *history_retention* parameter to a very high value (for instance '100 YEARS')
- and schedule purge operations by any mean (*crontab*, *pgAgent*, *pgTimeTable* or any other tool).

5.6 THE E-MAJ ROLLBACK UNDER THE HOOD

5.6.1 Planning and execution

E-Maj rollbacks are complex operations. They can be logged or not, concern one or several tables groups, with or without parallelism, and be launched by a direct SQL function call or by a client. Thus E-Maj rollbacks are splitted into **elementary steps**.

An E-Maj rollback is executed in two phases: a planning phase and an execution phase.

The **planning** phase determines all the needed elementary steps and estimates the execution duration. The estimate is computed for each step by taking into account:

- duration statistics of similar steps for previous rollback operations, stored into the *emaj_rlbk_stat* table
- and predefined parameters of the cost model (see § 5.1).

Then, for parallel rollbacks, elementary steps are assigned to the requested *n* sessions.

The *emaj_estimate_rollback_group()* function (Cf §4.7.1) executes the planning phase and just returns its result, without chaining the execution phase.

The plan produced by the planning phase is recorded into the *emaj_rlbk_plan* table.

The E-Maj rollback **execution** phase just chains the elementary steps of the built plan.

First, a lock of type *EXCLUSIVE* is set on all tables of the rolled back tables group or tables groups, so that any table's content change attempt from another client be blocked.

Then, for each table having changes to revert, the elementary steps are chained. In ascending order:

- preparing application triggers;
- disabling E-Maj triggers;
- deleting or setting as *DEFERRED* foreign keys;
- rolling back the table;
- deleting a content of the log table;
- recreating or resetting the state of foreign keys;
- resetting the state of application triggers;
- re-enabling E-Maj triggers.

The processing of all sequences concerned by the E-Maj rollback is performed by a single elementary step that is scheduled at the beginning of the operation.

For each elementary step, the function that drives the plan execution updates the *emaj_rlbk_plan* table. Reading this table's content may bring interesting information about the way the E-Maj rollback operation has been processed.

If the *dblink_user_password* parameter is set and the execution right on the *dblink_connect_u* function has been given to the E-Maj administrator submitting the rollback, the *emaj_rlbk_plan* updates are processed into autonomous transactions, so that it is possible to look at the rollback operation in real time. That's what the *emaj_rollback_activity()* function (§4.11.2) and the *emajRollbackMonitor* (§4.14) and *Emaj_web* (§6.3.12) clients do. If the *dblink* connection is not operational, the *emaj_verify_all()* (§4.11.2) function explains why.

5.6.2 Rolling back a table

Rollbacking a table consists in resetting its content in the state at the time of the E-Maj rollback target mark setting.

In order to optimize the operation and avoid the execution of one SQL statement for each elementary change, a table rollback just executes 4 global SQL statements:

- create and populate a temporary table containing all primary keys to process;
- delete from the table to process all rows corresponding to changes to revert of type INSERT and UPDATE;
- ANALYZE the log table if the rollback is logged and if the number of changes is greater than 1000 (to avoid a poor execution plan of the last statement);
- insert into the table to process the oldest rows content corresponding to the changes to revert of type UPDATE and DELETE.

5.6.3 Foreign keys management

If a table processed by the rollback operation has a foreign key or is referenced by a foreign key belonging to another table, then this foreign key needs to be taken into account for the rollback execution.

Depending on the context, several behaviours exist.

For a given table, if all other tables linked to it by foreign keys belong to the same tables group or tables groups processed by the E-Maj rollback operation, reverting the changes on all tables will safely preserve the referential integrity.

For this first case (which is the most frequent) the table rollback is executed with a *session_replication_role* parameter set to 'replica'. In this mode, no check on foreign keys is performed while updating the table.

On the contrary, if tables are linked to other tables that do not belong to the tables groups processed by the rollback operation or that are not including into any tables groups, then it is essential that the referential integrity be checked.

In this second case, checking the referential integrity is performed:

- either by pushing the checks at the end of the transaction, with a *SET CONSTRAINTS ... DEFERRED* statement executed if the key is declared *DEFERRABLE INITIALLY IMMEDIATE*;
- or by dropping the foreign key before rolling back the table and recreating it after.

The first option is chosen if the foreign key is declared *DEFERRABLE* and does not hold an *ON DELETE* or *ON UPDATE* clause.

FOREIGN KEYS defined on partitionned tables are not supported by E-Maj rollback operations if:

- tables/partitions linked by these keys do not all belong to the same tables groups to process,
- and these keys are of type *IMMEDIATE* or hold *ON DELETE* or *ON UPDATE* clauses.

Indeed, it is impossible to drop and recreate such a foreign key for just a partition. As a workaround:

- foreign keys of type *IMMEDIATE* (the default state) can easily be declared as *DEFERRABLE INITIALLY IMMEDIATE*,
- foreign keys having *ON DELETE* or *ON UPDATE* clauses can be created on each elementary partition.

5.6.4 Other integrity constraints

Tables may hold other integrity constraints: *NOT NULL*, *CHECK*, *UNIQUE* and *EXCLUDE*. But these constraints only concern the content of the table that holds them, without any link with other tables.

During an E-Maj rollback, these constraints are verified by PostgreSQL, immediately at data change, or at the transaction end for *UNIQUE* or *EXCLUDE* constraints that are defined as *DEFERRED*. Considering the way elementary tables are rolled back (Cf §5.6.2), no specific action is performed to support these constraints, and no integrity violation should arise if all these integrity constraints already existed when the rollback target mark was set.

5.6.5 Application triggers management

Triggers belonging to tables to rollback that are not E-Maj triggers are temporarily disabled during the operation. But this default behaviour can be adjusted when assigning a table to a tables group or importing a tables group configuration, by defining a trigger as “not to be disabled at rollback time” (see §4.4.2, §4.4.7, §4.5.6.2 and §5.11.3).

The technical way to disable or not the application triggers depends on the *session_replication_role* parameter value set for each table to rollback.

If *session_replication_role* equals ‘*replica*’, then the enabled triggers at the E-Maj rollback start are not called. If a trigger is declared as ‘not to be disabled’, it is temporarily changed into an *ALWAYS* trigger during the operation.

If *session_replication_role* keeps its default value, enabled triggers to neutralize are just temporarily disabled during the operation.

In a declarative partitionning context, it is possible to create a trigger on a partitionned table. As a result, each partition of the table inherits the trigger. There is no practical issue with this on E-Maj rollbacks. If one wishes to let the trigger enabled during the rollback, it must be declared as such for each partition.

5.7 IMPACTS ON INSTANCE AND DATABASE ADMINISTRATION

5.7.1 Stopping and restarting the instance

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

5.7.1.1 General rule

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

If a transaction with table updates were not committed at instance stop, it would be rolled back during the recovery phase of the instance start, the application tables updates and the log tables updates being cancelled at the same time.

This rule also applies of course to transactions that execute E-Maj functions, like a tables group start or stop, a rollback, a mark deletion,...

5.7.1.2 Sequences rollback

Due to a PostgreSQL constraint, the rollback of application sequences assigned to a tables group is the only operation that is not protected by transactions. That is the reason why application sequences are processed at the very end of the rollback operations (See §4.3.4). (For the same reason, at set mark time, application sequences are processed at the beginning of the operation.)

In case of an instance stop during an E-Maj rollback execution, it is recommended to rerun this rollback just after the instance restart, to ensure that application sequences and tables remain properly synchronised.

5.7.2 Saving and restoring the database



Using E-Maj allows a reduction in the database saves frequency. But E-Maj cannot be considered as a substitute to regular database saves that remain indispensable to keep a full image of databases on an external support.

5.7.2.1 File level saves and restores

When saving or restoring instances at file level, it is essential to save or restore ALL instance files, including those stored on dedicated tablespaces.

After a file level restore, tables groups are in the very same state as at the save time, and the database activity can be restarted without any particular E-Maj operation.

5.7.2.2 Logical saves and restores of entire database

To properly save and restore a database with E-Maj, using *pg_dump*, and *psql* or *pg_restore*, it is essential that both source and restored databases use the **same E-Maj version**. If this is not the case, the content of some technical tables may be not synchronised with their structure. The *emaj_get_version()* function allows to know the current version of the *emaj* extension (Cf §4.11.1).

Regarding stopped tables groups (in *IDLE* state), as log triggers are disabled and the content of related log tables is meaningless, there is no action required to find them in the same state as at save time.

Concerning tables groups in *LOGGING* state at save time, it is important to be sure that log triggers will only be activated after the application tables rebuild. Otherwise, during the tables rebuild, tables updates would also be recorded in log tables!

When using *pg_dump* command for saves and *psql* or *pg_restore* commands for restores, and processing full databases (schema + data), these tools recreate triggers, E-Maj log triggers among them, after tables have been rebuilt. So there is no specific precaution to take.

On the other hand, in case of data only save or restore (i.e. without schema, using *-a* or *--data-only* options), the *--disable-triggers* must be supplied:

- with *pg_dump* (or *pg_dumpall*) with save in *plain* format (and *psql* is used to restore),
- with *pg_restore* command with save in *tar* or *custom* format.

Restoring the database structure generates 2 error messages reporting that the *_emaj_protection_event_trigger_fnct()* function and the *emaj_protection_trg* event trigger already exist:

```
...
ERROR:  function "_emaj_protection_event_trigger_fnct" already exists with same
argument types
...
ERROR:  event trigger "emaj_protection_trg" already exists
...
```

This message display is normal and does not indicate a defective restore. Indeed, both objects are created with the extension and are then detached from it, so that the trigger can block any attempt of the extension drop. As a result, the *pg_dump* tool saves them as independent objects. And when restoring, these objects are created twice, first with the *emaj* extension creation, and then as independent objects, this second attempt generating both error messages.

5.7.2.3 Logical save and restore of partial database

With *pg_dump* and *pg_restore* tools, database administrators can perform on a subset of database schemas or tables.

Restoring a subset of application tables and/or log tables generates a heavy risk of data corruption in case of later E-Maj rollback of concerned tables. Indeed, it is impossible to guarantee in this case that application tables, log tables and internal E-Maj tables that contain essential data for rollback, remain coherent.

If it is necessary to perform partial application tables restores, a drop and recreation of all tables groups concerned by the operation must be performed just after.

The same way, it is strongly recommended to NOT restore a partial *emaj* schema content.

The only case of safe partial restore concerns a full restore of the *emaj* schema content as well as all tables belonging to all groups that are created in the database.

5.7.3 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.

5.7.4 Tables reorganisation

5.7.4.1 Reorganisation of application tables

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

5.7.4.2 Reorganisation of E-Maj tables

The index corresponding to the primary key of each table from E-Maj schemas (neither log tables nor technical tables) is declared “cluster”.



So using E-Maj may have an operational impact regarding the execution of *CLUSTER* SQL commands at database level.

When E-Maj is used in continuous mode (with deletion of oldest marks instead of regular tables groups stop and restart), it is recommended to regularly reorganize E-Maj log tables. This reclaims unused disk space following mark deletions.

5.7.5 Using E-Maj with replication

5.7.5.1 Integrated physical replication

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

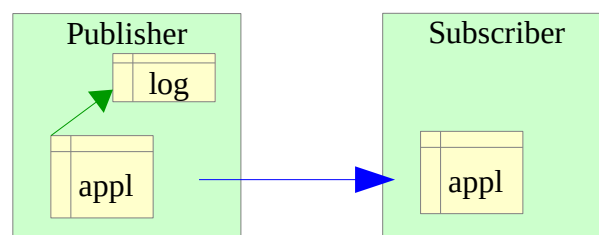
However, because of the way PostgreSQL manages sequences, the sequences' current values may be a little forward on secondary instances than on the primary instance. For E-Maj, this may lightly overestimate the number of log rows in general statistics. But there is no consequence on the data integrity.

5.7.5.2 Integrated logical replication

PostgreSQL includes logical replication mechanisms. The replication granularity is the table. The *publication* object used with the logical replication is quite close to the E-Maj tables group concept, except that a *publication* cannot contain sequences.

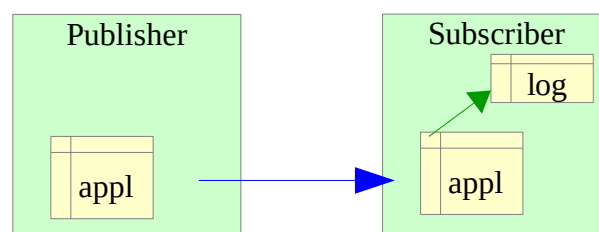
Several cases have to be examined.

Replication of application tables managed by E-Maj



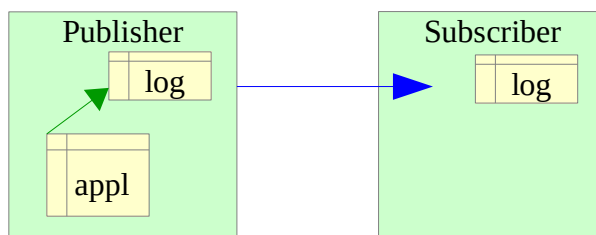
An application table that belongs to a tables group can be replicated. The effect of any rollback operation that may occur would be simply replicated on *subscriber* side, as long as no filter has been applied on replicated SQL verbs types.

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



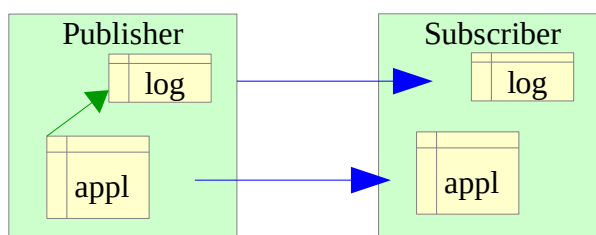
As of E-Maj 4.0, it is possible to include an application table into a tables group, with updates coming from a logical replication flow. But all E-Maj operations (starting/stopping the group, setting marks,...) must of course be executed on the *subscriber* side. An E-Maj rollback operation can be launched once the replication flow has been stopped (to avoid updates conflicts). But then, tables on both *publisher* and *subscriber* sides are not coherent anymore.

Replication of E-Maj log tables



As of E-Maj 4.0, it is technically possible to replicate an E-Maj log table (once found a way to get the DDL that creates the log table – using *pg_dump* for instance). This allows to duplicate or concentrate logs content on another server. But the replicated log table can only be used for log **auditing**. As log sequences are not replicated, these logs cannot be used for other purposes.

Replication of application tables and E-Maj log tables



Application tables and log tables can be simultaneously replicated. But as seen previously, these replicated logs can only be used for **auditing** purpose. E-Maj rollback operations can only be executed on publisher side.

5.7.5.3 Other replication solutions

Using E-Maj with external replication solutions based on triggers like Slony or Londiste, requires some attention... It is probably advisable to avoid replicating log tables and E-Maj technical tables.

5.8 SENSITIVITY TO SYSTEM TIME CHANGE

To ensure the integrity of tables managed by E-Maj, it is important that the rollback mechanism be insensitive to potential date or time change of the server that hosts the PostgreSQL instance.

The date and time of each update or each mark is recorded. But nothing other than sequence values recorded when marks are set, are used to frame operation in time. So rollbacks and mark deletions are insensitive to potential system date or time change.

However, two minor actions may be influenced by a system date or time change:

- the deletion of oldest events in the *emaj_hist* table (the retention delay is a time interval),
- finding the name of the mark immediately preceding a given date and time as delivered by the *emaj_get_previous_mark_group()* function.

5.9 PERFORMANCE

5.9.1 Updates recording overhead

Recording updates in E-Maj log tables has necessarily an impact on the duration of these updates. The global impact of this log on a given processing depends on numerous factors. Among them:

- the part that the update activity represents on the global processing,
- the intrinsic performance characteristics of the storage subsystem that supports log tables.

However, the E-Maj updates recording overhead is generally limited to a few per-cents. But this overhead must be compared to the duration of potential intermediate saves avoided with E-Maj.

5.9.2 E-Maj rollback duration

The duration of an E-Maj rollback depends on several factors, like:

- the number of updates to cancel,
- the intrinsic characteristics of the server and its storage material and the load generated by other activities hosted on the server,
- triggers or foreign keys on tables processed by the rollback operation,
- contentions on tables at lock set time.

To get an order of magnitude of an E-Maj rollback duration, it is possible to use the *emaj_estimate_rollback_group()* and *emaj_estimate_rollback_groups()* functions (See §4.7.1).

5.9.3 Optimizing E-Maj operations

Here are some advice to optimize E-Maj operations:

5.9.3.1 Use tablespaces

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

By setting a specific default tablespace for the session before the tables groups creation, log tables and indexes are created by default into this tablespace, without any additional action.

But through parameters set when calling the `emaj_assign_table()`, `emaj_assign_tables()` and `emaj_modify_table()` functions, it is also possible to specify a tablespace to use for any log table or log index (see §4.2.1.3).

5.9.3.2 Declare foreign keys as DEFERRABLE

Foreign keys can be explicitly declared as *DEFERRABLE* at creation time. If a foreign key links two tables belonging to different tables groups or if one of them doesn't belong to any tables group and if the foreign key has no *ON DELETE* or *ON UPDATE* clause then it is recommended to declare it as *DEFERRABLE*. This will avoid to be dropped and recreated at subsequent E-Maj rollbacks. The foreign key checks of updated rows are just deferred to the end of the rollback function execution, once all log tables are processed. This generally greatly speeds up the rollback operation.

5.9.3.3 Modify memory parameters

Increasing the value of the *work_mem* parameter when performing an E-Maj rollback may bring some performance gains.

If foreign keys have to be recreated by an E-Maj rollback operation, increasing the value of the *maintenance_work_mem* parameter may also help.

If the E-Maj rollback functions are directly called in SQL, these parameters can be previously set at session level, with statements like:

```
SET work_mem = <value>;  
SET maintenance_work_mem = <value>;
```

If the E-Maj rollback operations are executed by a web client, it is also possible to set these parameters at function level, as *superuser*:

```
ALTER FUNCTION emaj._rlbk_tbl(emaj.emaj_relation, BIGINT, BIGINT, INT,  
BOOLEAN) SET work_mem = <value>;  
ALTER FUNCTION emaj._rlbk_session_exec(INT, INT) SET maintenance_work_mem  
= <value>;
```

5.10 USAGE LIMITS

The E-Maj extension usage has some limits.

- The minimum required PostgreSQL version is 12.
- All tables belonging to a “rollbackable” tables group must have an explicit *PRIMARY KEY*. If a table has no explicit *PRIMARY KEY* but has a *UNIQUE* index referencing *NOT NULL* columns, this index should rather be transformed into *PRIMARY KEY*.
- *UNLOGGED* tables can only be members of “audit_only” tables groups.
- *TEMPORARY* tables are not supported by E-Maj.
- In some configurations, *FOREIGN KEYS* defined on partitionned tables are not supported by E-Maj rollback operations (see §5.6.3).
- 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 (see §4.4).

5.11 USER'S RESPONSIBILITY

5.11.1 Defining tables groups content

Defining the content of tables group is essential to guarantee the database integrity. It is the E-Maj administrator's responsibility to ensure that all tables updated by a given operation are really included in a single tables group.

5.11.2 Appropriate call of main functions

emaj_start_group(), *emaj_set_mark_group()*, *emaj_rollback_group()* and *emaj_logged_rollback_group()* functions (and their related multi-groups functions) set explicit locks on tables of the group to be sure that no transactions updating these tables are running at the same time. But it is the user's responsibility to execute these operations “at the right time”, i.e. at moments that really correspond to a stable point in the life of these tables. He must also take care of warning messages that may be reported by E-Maj rollback functions.

5.11.3 Management of application triggers

Triggers may have been created on application tables. It is not rare that these triggers perform one or more updates on other tables. In such a case, it is the E-Maj administrator's responsibility to understand the impact of E-Maj rollback operations on tables concerned by triggers, and if needed, to take the appropriate measures.

By default, E-Maj rollback functions neutralize application triggers during the operation. But the E-Maj administrator can change this behaviour using the “*ignored_triggers*” and

"*ignored_triggers_profiles*" properties of the *emaj_assign_table()*, *emaj_assign_tables()*, *emaj_modify_table()* and *emaj_modify_tables()* functions. (Cf. §4.2.3 and § 4.4.7).

If the trigger simply adjusts the content of the row to insert or update, the logged data contain the final columns values. In case of rollback, the log table contains the right columns content to apply. So the trigger must be disabled at rollback time (the default behaviour), so that it does not disturb the processing

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

- if the updated table belongs to the same tables group, the automatic trigger disabling and the rollback of both tables let them in the expected state,
- if the updated table does not belong to the same tables group, it is essential to analyse the consequences of a rollback operation, in order to avoid a de-synchronisation between both tables. If needed, the triggers can be left enabled. But some other actions may also be required.

For more complex triggers, it is essential to perfectly understand their impacts on E-Maj rollbacks and take any appropriate measure at rollback time.

For parallel rollback operations, a trigger kept enabled that updates other tables from the same tables group, would likely generate a freeze between sessions.

5.11.4 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 the *emaj_param* table may be updated by these users. Any other internal table or sequence update may lead to data corruption.

6 EMAJ_WEB

A web application, *Emaj_web*, makes E-Maj use much easier.

6.1 OVERVIEW

Emaj_web has borrowed to phpPgAdmin its infrastructure (browser, icon trails, database connection, management,...) and some useful functions like browsing the tables content or editing SQL queries.

For databases into which the *emaj* extension has been installed, and if the user is connected with a role that owns the required rights, all E-Maj objects are accessible.

It is then possible to:

- create and setup tables groups,
- see tables groups and manipulate them, depending on their state (drop, start, stop, set or remove a mark, rollback, add or modify a comment),
- list 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 recorded changes (the log tables content) and see their content,
- monitor in progress E-Maj rollback operations, and examine the completed rollbacks,
- check the extension's good health.

6.2 INSTALL THE EMAJ_WEB CLIENT

6.2.1 Prerequisite

Emaj_web requires a web server with a php interpreter, and its *pgsql* and *intl* extensions.

6.2.2 Download the software

The *Emaj_web* application can be downloaded from the following git repository:

https://github.com/dalibo/emaj_web

6.2.3 Configure Emaj-web

The configuration is centralized into a single file: *emaj_web/conf/config.inc.php*. It contains the general parameters of the applications, and the description of the PostgreSQL instances connections.

When the number of instances is large, it is possible to split them into “instances groups”. A group can contain instances or other instance groups.

In order to submit batch rollbacks (i.e. without blocking the use of the browser while the rollback operation is in progress), it is necessary to specify a value for two configuration parameters:

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

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

6.3 USE EMAJ_WEB

6.3.1 Access to Emaj_web and databases

Accessing Emaj_web in a browser displays the welcome page.

To sign in to a database, select the target instance in the left browser or in the ‘*servers*’ tab, and fill the connection identifier and password. Several connections can remain opened simultaneously.

Once connected to a database where the *emaj* extension has been installed, the user interacts with the extension, depending on the role it owns (super-user, *emaj_adm* or *emaj_viewer*).

On the left, the browser tree shows all the configured instances, that may be split into instances groups, and all the databases they contain. By unfolding a database object, the user reaches the E-Maj tables groups and the existing schemas.

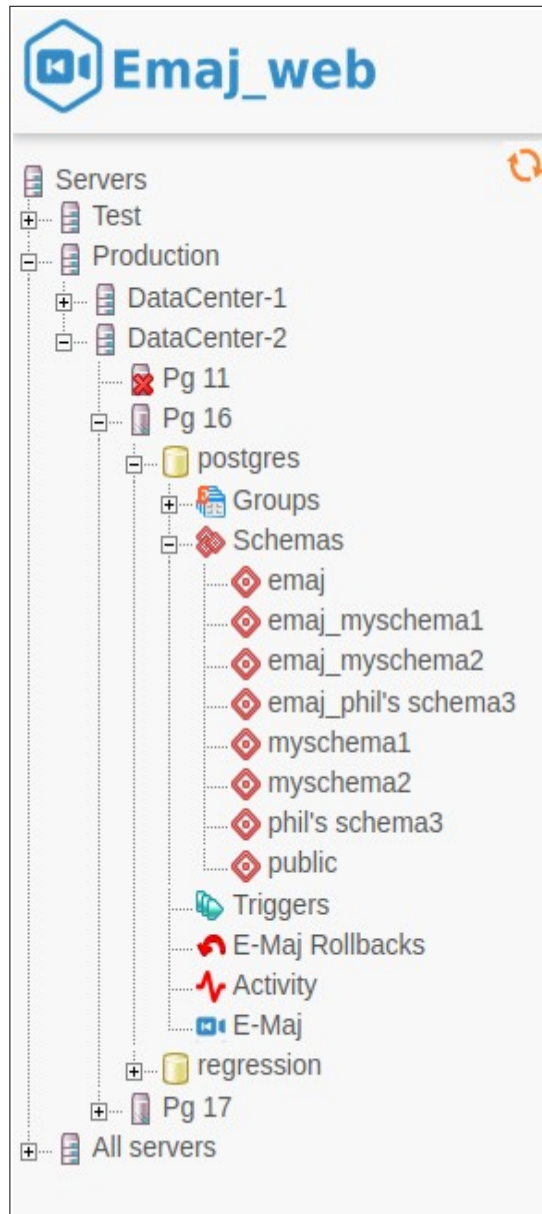


Figure 1 – The browser tree.

Both icons located at the bottom-right (not visible here) allow to adjust the browser width.

6.3.2 Tables groups lists

By selecting a database, the user reaches a page that lists all tables groups created in this database.

03/02/2025 - 08:29:41
User postgres
English
Emaj_web > Pg 16 > postgres
SQL | History

Groups
Schemas
Triggers
E-Maj Rollbacks
Activity
E-Maj

Tables groups in "LOGGING" state (x 2)

	Group	Created at	Tables	Sequences	Type	Marks	Actions	Comment
<input type="checkbox"/>	myGroup1	03 Feb 2025 08:14:25	5	1		3		Useless comm...
<input type="checkbox"/>	myGroup2	03 Feb 2025 08:14:25	4	2		4		

Select
Actions on objects (0)
All / Visible / None / Invert

Tables groups in "IDLE" state (x 1)

	Group	Created at	Tables	Sequences	Type	Marks	Actions	Comment
<input type="checkbox"/>	phil's group#3	03 Feb 2025 08:14:25	2	1		0		

Select
Actions on objects (0)
All / Visible / None / Invert

New group
Export
Import

Old dropped tables groups

No old dropped tables groups.

Figure 2 – Tables groups list.

This page displays two lists:

- the tables groups in *LOGGING* state
- the tables groups in *IDLE* state.

For each created tables group, the following attributes are displayed:

- its creation date and time,
- the number of application tables and sequences it contains,
- its type ("*ROLLBACKABLE*" or "*AUDIT_ONLY*", protected against rollback or not),
- the number of marks it owns,
- its associated comment, if any.

For each tables group, several buttons are available so that the user can perform any possible action, depending on the group state.

Below, three buttons allow to create a new tables group, to export or import a tables groups configuration to or from a local file.

Lastly, a list of dropped tables groups is displayed.

6.3.3 Some details about the user interface

The page headers contain:

- some information regarding the current connection,
- 3 links to reach the SQL statements editor, the history of submitted statements and to logout the current connection,
- a combo box to select the language used by the user interface,
- a breadcrumb trail,
- and a button to directly go to the page bottom.

The user can navigate in Emaj_web functions using four icon bars: one for the general purpose functions, two similar bars for the functions concerning respectively a single tables group or a single table, and the last for the functions concerning a single sequence.



Figure 3 – Main icons bar.

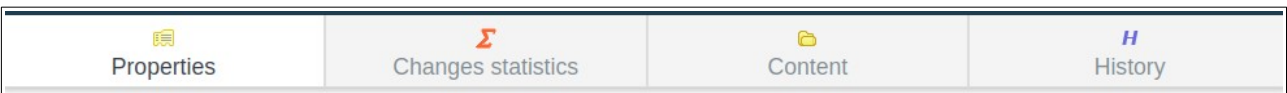


Figure 4 – Tables groups and tables icons bar.

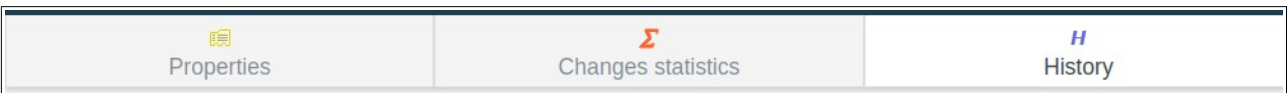


Figure 5 – Sequences icons bar.

For *emaj_viewer* roles, some icons are not visible.

On most tables, it is possible to dynamically sort displayed rows, using small vertical arrows on the right of column titles.

On most tables too, an icon located at the left of the header row, let show or hide input fields that can be used to dynamically filter displayed rows.

Tables groups in "LOGGING" state (x 2)

	Group ▾	Created at ▾	Tables ▾	Sequences ▾	Type ▾	Marks ▾	Actions	Comment ▾
<input type="checkbox"/>	myGroup1	03 Feb 2025 08:14:25	5	1		3		Useless comm...
<input type="checkbox"/>	myGroup2	03 Feb 2025 08:14:25	4	2		4		

Select

Actions on objects (0)

All / Visible / None / Invert

*Figure 6 – Filtering the tables groups in logging state.
Here, only tables groups whose name contains “my” and having more than 2 marks are displayed, sorted in descending order by number of tables.*

Some tables allow to perform actions on several objects at once. In this case, the user selects the objects with the checkboxes on the first column of the table and choose the action to perform among the available buttons under the table.

Columns containing comments have a limited size. But the full comment content is visible in tooltip when the mouse goes over the cell.

Cells containing event timestamps or durations show a full data content in tooltip.

6.3.4 Tables group details

From the tables groups list page, it is possible to get more information about a particular tables group by clicking on its name. This page is also accessible with the “Properties” icon of the groups bar and through the left browsing tree.

03/02/2025 - 08:41:47 User postgres English ▼

Emaj_web > Pg 16 > postgres > myGroup1 SQL | History ▼

Properties

Changes statistics

Content

History

Tables group "myGroup1" properties

Created at	Type	Tables	Sequences	State	Started at	Marks	Log size
03 Feb 2025 08:14:25		5	1		03 Feb 2025 08:14:25	3	144 kB

Comment: Useless comment!

Set a mark

Protect

Stop

Set a comment

Tables group "myGroup1" marks

		Mark	State	Set at	Row changes			Actions	Comment
					Number	Cumulated			
<input type="checkbox"/>		MARK3		Mon 03 Feb 08:14:26	0	0			
<input type="checkbox"/>		MARK2		Mon 03 Feb 08:14:25	7	7			
<input type="checkbox"/>		MARK1		Mon 03 Feb 08:14:25	19	26			End of 1st prog...

Select

Actions on objects (0)

All / Visible / None / Invert

Figure 7 – Details of a tables group

A first line repeats information already displayed on the groups list (number of tables and sequences, type, state and number of marks). It also shows the disk space used by its log tables.

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

Next is a set of buttons to execute actions depending on the group's state.

Then, the user can see the list of all marks that have been set on the group, the most recent being ahead. For each of them, the following is displayed:

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

For each mark, several buttons are available to perform the actions permitted by the mark's state.

6.3.5 Statistics

Using the “Changes *statistics*” tab of the group's bar, one gets statistics about recorded changes for tables or sequences, for the selected tables group and a given time interval. This time interval is defined as either two marks or a mark and the current state.

Three types of statistics can be produced:

- a number of changes estimate for each table,
- a number of increments estimate and properties changes per sequence,
- a precise numbering of changes per table, statement type (*INSERT/UPDATE/DELETE/TRUNCATE*) and role.

The figure below shows an example of detailed statistics for tables.

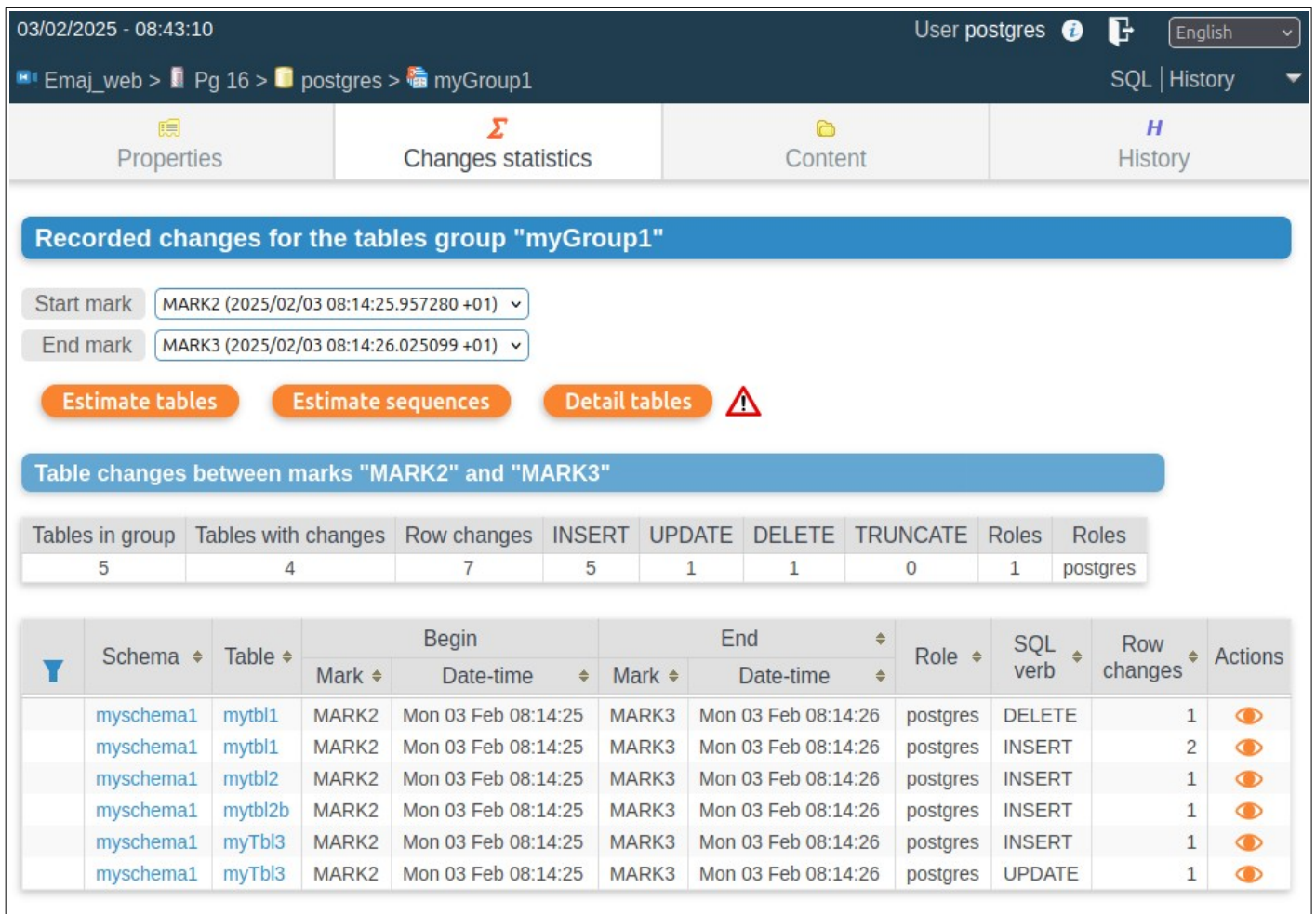


Figure 8 – Detailed statistics about updates recorded between two marks

The displayed page contains a first line returning global counters.

On each line of the statistics table, the user can click on a button to easily look at the log tables content. A click on this button opens a window to set the SQL generation parameters. Then, the generated SQL statement is displayed into the SQL editor window so that the user can adjust it before execution to better fit his needs.

Generate the changes dump SQL statement

Table = "myschema1.mytbl1"

Tables group = "myGroup1"

Marks interval = "MARK2" - "MARK3"

Consolidation

☒ None
☐ Partial
☐ Full

SQL verbs

☐ INSERT
☐ UPDATE
☒ DELETE
☐ TRUNCATE

All

Roles

Known roles: postgres

All Delete

E-Maj technical columns

☒ emaj_verb
☒ emaj_tuple
☒ emaj_gid
☒ emaj_changed
☒ emaj_txid
☒ emaj_user

All Minimum

Columns order

☒ As the log table
☐ Primary key ahead

Rows order

☒ Chronological
☐ Primary key

Generate SQL

Reset

Figure 9 – Form to generate the SQL statement displaying table changes

6.3.6 Tables group content

Using the “Content” tab of the group’s bar, it is possible to get a summary of a tables group content.

For each table belonging to the group, the displayed sheet shows its E-Maj characteristics, as well as the disk space used by its log table and index.

03/02/2025 - 08:48:15

User postgres

English

Emaj_web > Pg 16 > postgres > myGroup1

SQL | History

Properties

Changes statistics

Content

History

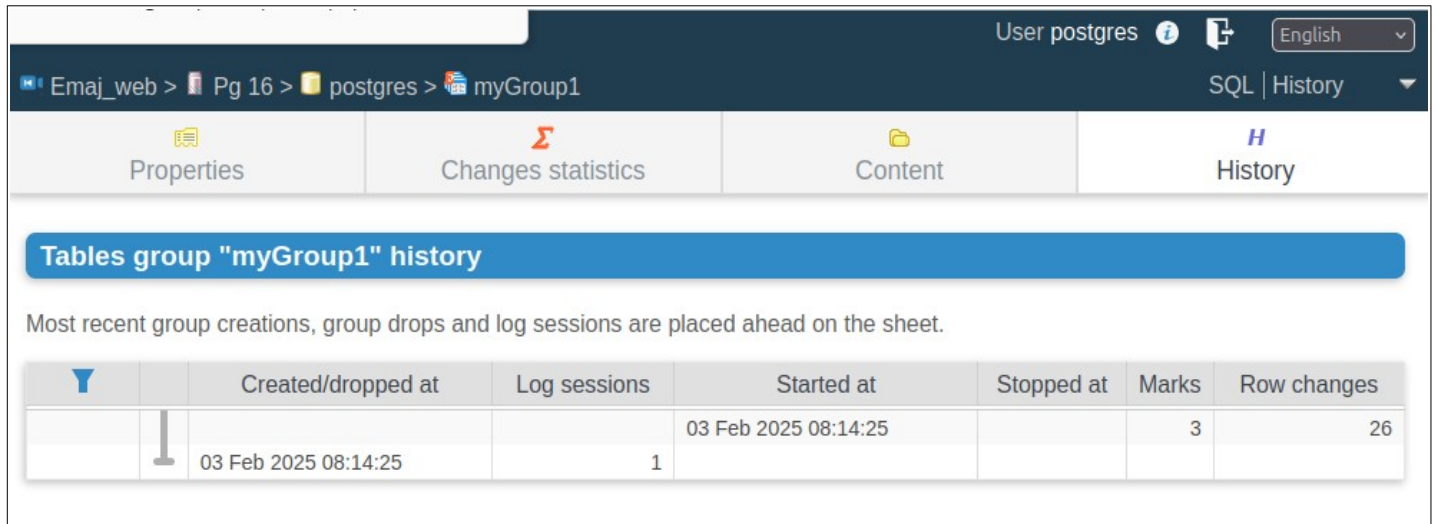
Current content of the tables group "myGroup1"

	Type	Schema	Name	Since	Priority	Tablespace		Current log table	Log size
						Log tables	Log indexes		
		myschema1	mytbl1	03 Feb 2025 08:14:25	20			emaj_myschema1.mytbl1_log	32 kB
		myschema1	mytbl2	03 Feb 2025 08:14:25				emaj_myschema1.mytbl2_log	32 kB
		myschema1	mytbl2b	03 Feb 2025 08:14:25				emaj_myschema1.mytbl2b_log	24 kB
		myschema1	myTbl3	03 Feb 2025 08:14:25	10			emaj_myschema1.myTbl3_log	24 kB
		myschema1	myTbl3_col31_seq	03 Feb 2025 08:14:25					
		myschema1	mytbl4	03 Feb 2025 08:14:25	20			emaj_myschema1.mytbl4_log	32 kB

Figure 10 – A tables group’s content.

6.3.7 Tables group history

The “History” tab displays the periods of time when the tables group exists and the periods of time when the group is in LOGGING state (*log sessions*). The historical depth depends on the *history_retention* parameter.



The screenshot shows the 'History' tab of the E-Maj interface. The breadcrumb navigation is 'Emaj_web > Pg 16 > postgres > myGroup1'. The top bar shows 'User postgres' and a language dropdown set to 'English'. The 'History' tab is selected, showing a table titled 'Tables group "myGroup1" history'. Below the title, a note states: 'Most recent group creations, group drops and log sessions are placed ahead on the sheet.' The table has columns: 'Created/dropped at', 'Log sessions', 'Started at', 'Stopped at', 'Marks', and 'Row changes'. A single row is visible, representing a log session that started on 03 Feb 2025 at 08:14:25, with 1 log session, 3 marks, and 26 row changes.

	Created/dropped at	Log sessions	Started at	Stopped at	Marks	Row changes
1	03 Feb 2025 08:14:25	1	03 Feb 2025 08:14:25		3	26

Figure 11 – A tables group’s history.

6.3.8 Schemas and tables groups configuration

The “schemas” tab displays the list of schemas contained in the database.

By selecting one of them, two additional lists are displayed: the tables and the sequences contained by this schema.

For both lists, the E-Maj properties and some general properties of each object become visible. Some action buttons allow to reach their properties or content and manage their assignment to tables groups.

Emaj_web > Pg 16 > postgres > myschema1

User postgres English

SQL | History

Groups

Schemas

Triggers

E-Maj Rollbacks

Activity

E-Maj

All schemas (x 8)

	Schema	E-Maj?	Owner	Comment
	emaj		postgres	Contains all E-Maj related objec...
	emaj_myschema1		emaj_adm	
	emaj_myschema2		emaj_adm	
	emaj_phil's schema3		emaj_adm	
	myschema1		postgres	
	myschema2		postgres	
	phil's schema3		postgres	
	public		pg_database_owner	standard public schema

Schema "myschema1" tables (x 5)

	Table	Actions	Group	Priority	Tablespace		Owner	Tablespace	Estimated row count	Comment	
					Log tables	Log indexes					
<input type="checkbox"/>	myTbl3			myGroup1	10			postgres			
<input type="checkbox"/>	mytbl1			myGroup1	20			postgres			
<input type="checkbox"/>	mytbl2			myGroup1				postgres			
<input type="checkbox"/>	mytbl2b			myGroup1				postgres			
<input type="checkbox"/>	mytbl4			myGroup1	20			postgres			

Select

Actions on objects (0)

All / Visible / None / Invert

Schema "myschema1" sequences (x 2)

Figure 12 – Schema content and tables groups configuration.

By clicking on a table or sequence name or on an eye icon, one gets details about the table or sequence characteristics and content.

6.3.9 Table details

By selecting a table from a schema tables list, one reaches a first “Properties” tab that shows the current E-Maj properties, the table structure, and its triggers.

E-Maj properties like tables group assignment, move or removal are available through appropriate buttons.

18/08/2025 - 09:17:47
User postgres
English

Emaj_web > Pg 16 > postgres > myschema1 > mytbl1
SQL | History

Properties

Changes statistics

Content

History

Table "myschema1.mytbl1"

E-Maj properties

Group	Since	Priority	Tablespace	
			Log tables	Log indexes
myGroup1	18 Aug 2025 08:48:57	20		

Update
Move
Remove

Table structure

	#	Column	Type	Not Null	Valuation		Constraints	Comment
					Type	Expression		
	1	col11	numeric(7,0)	✓			✓	
	2	col12	character(10)	✓			✓	
	3	col13	bytea					

Triggers

Exec. order	Trigger	Level	Triggering event	Called function	State	E-Maj?	Auto disable	Actions
1	emaj_trunc_trg	STATEMENT	BEFORE TRUNCATE	emaj_truncate_trigger_fnct	Enabled Always	✓		
2	emaj_log_trg	ROW	AFTER INSERT OR DELETE OR UPDATE	emaj_myschema1.mytbl1_log_fnct	Enabled Always	✓		

Figure 13 – Table properties.

Thanks to the specific table tabs bar three other functions are available.

The “Change statistics” tab offers statistics about changes registered for the table, on a specified time frame, like estimated statistics about tables groups. However, they differ in several ways:

- a single table is processed,
- but one gets one statistic row per elementary marks interval,
- the number of executed E-Maj rollbacks is also reported for each marks interval.

Alike for tables groups, an “eye” icon allows to look at changes details.

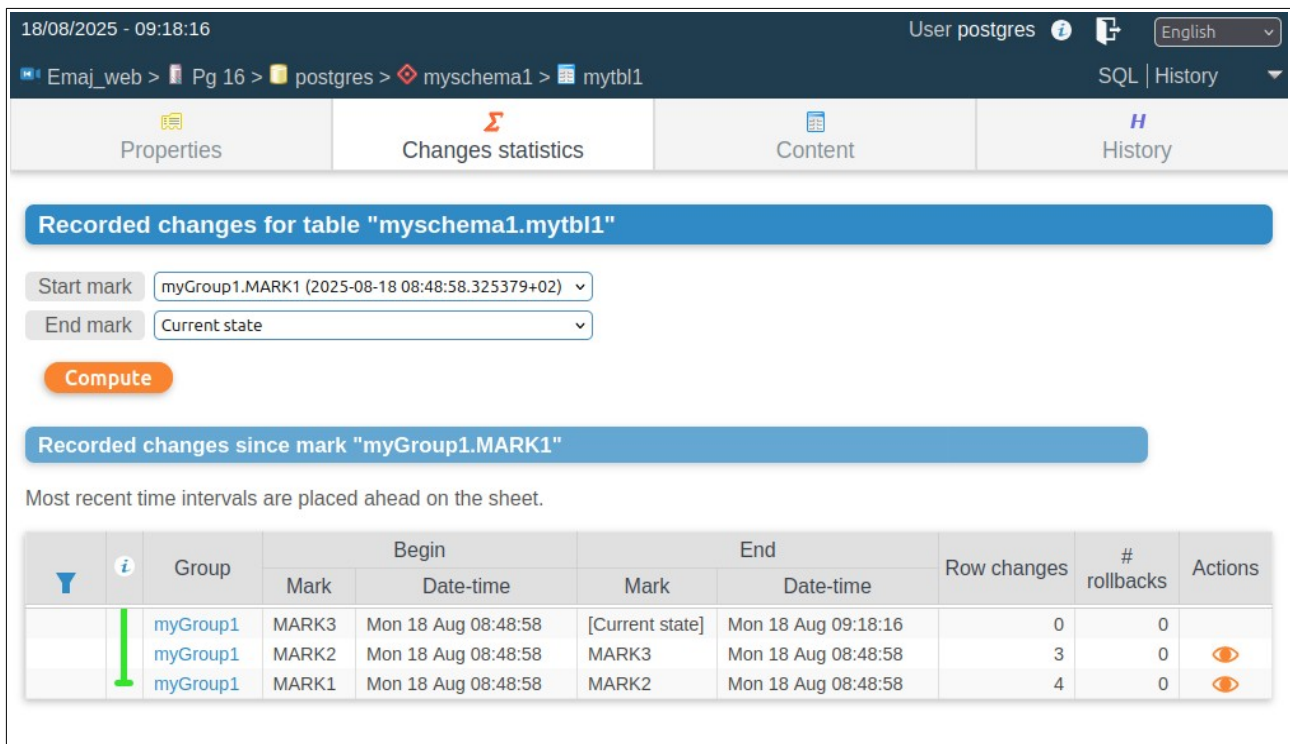


Figure 14 – Table changes statistics.

The “Content” tab allows to visualize the current table rows.

Lastly, the “History” tab shows the E-Maj properties evolutions of the table, and in particular its tables groups assignments.

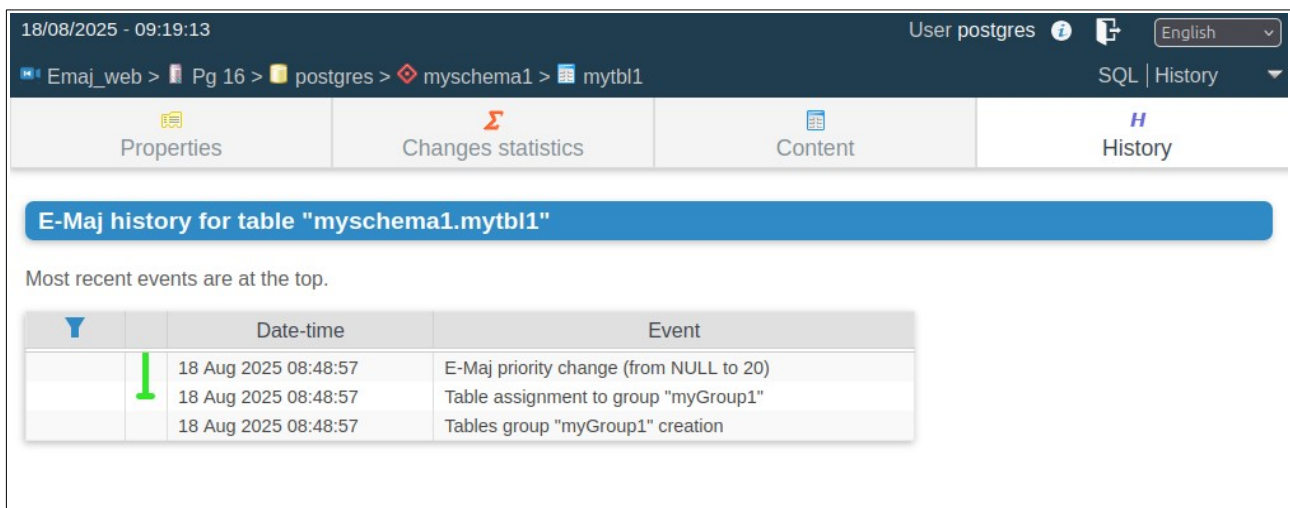


Figure 15 – E-Maj history for a table.

6.3.10 Sequence details

Regarding sequences, the “Properties” tab shows the E-Maj properties and the characteristics of a single sequence. Some buttons allow to modify the sequence E-Maj properties (tables group assignment, move or removal).

Sequence "myschema1.myTbl3_col31_seq"

E-Maj properties

Group	Since
myGroup1	18 Aug 2025 08:48:57

Move Remove

Sequence properties

Last value	Will increment last value before returning next value (is_called)?	Start value	Min value	Max value	Increment	Can cycle?	Cache size	Log count
13	Yes	1	1	10000000	3	No	1	32

Figure 16 – Sequence properties.

Thanks to the sequence specific tabs bar, two other functions are available.

The “Changes statistics” tab returns statistics about the sequence state evolution on a specified time frame, with, for each elementary marks interval:

- the number of sequence increments,
- a flag indicating whether another sequence property has changed (min value, max value, increment, etc),
- the number of executed E-Maj rollbacks.

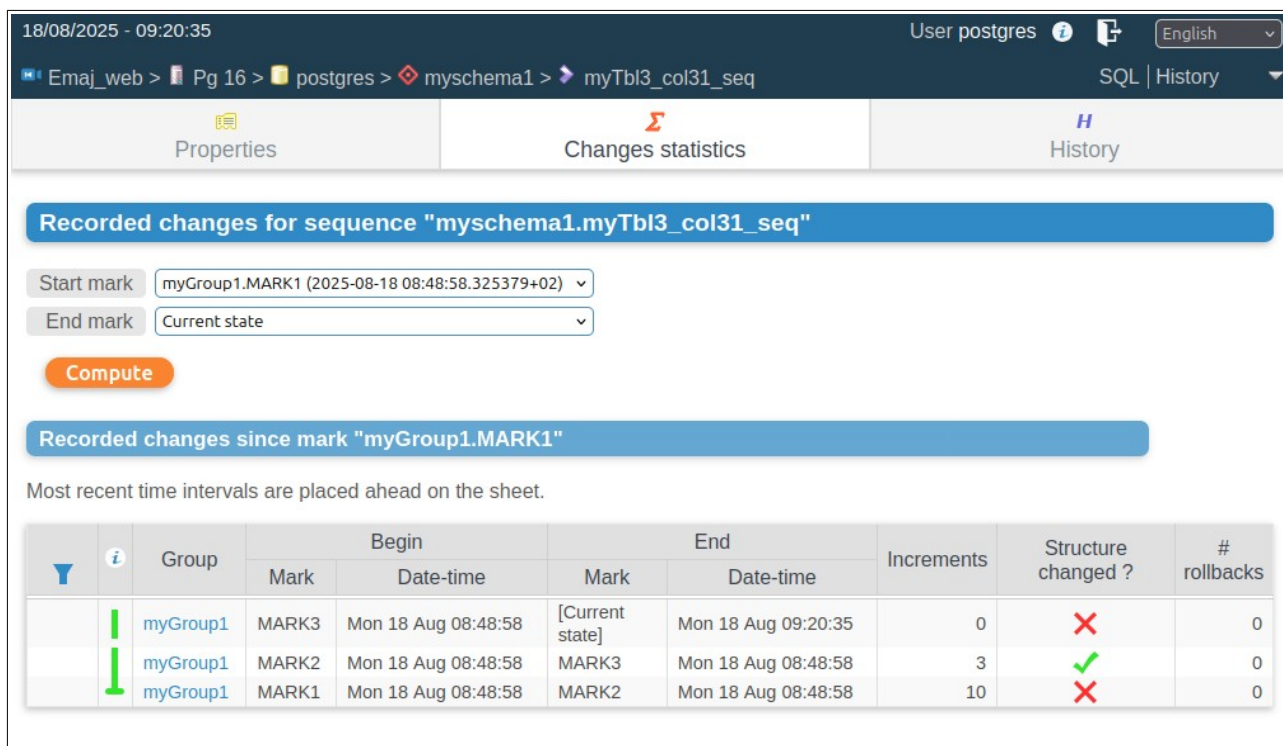


Figure 17 – Sequence statistics.

Lastly, the “History” tab shows the sequence E-Maj characteristics changes, in particular the assignment into tables groups.

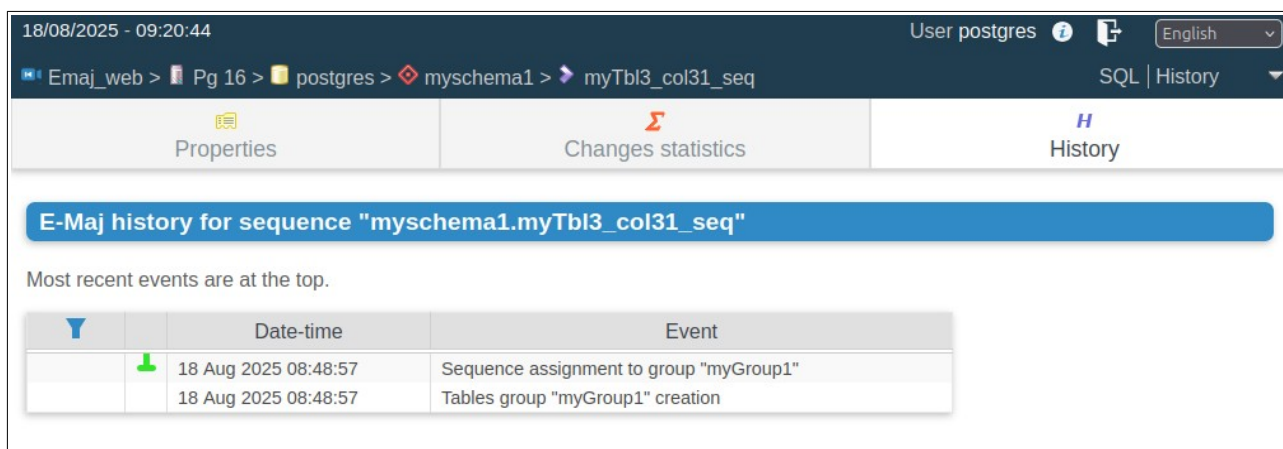


Figure 18 – Sequence E-Maj history.

6.3.11 Triggers

The “Triggers” tab lists the application triggers (those not linked to E-Maj), with their main characteristics.

A button allows to switch their de-activation mode at E-Maj rollback time.

03/02/2025 - 08:53:28

User postgres

English

Emaj_web > Pg 16 > postgres

SQL | History

Groups

Schemas

Triggers

E-Maj Rollbacks

Activity

E-Maj

Application triggers (x 1)

	Schema	Table	Trigger	Level	Triggering event	Called function	State	Auto disable	Actions
<input type="checkbox"/>	myschema1	mytbl2	mytbl2trg	ROW	AFTER INSERT OR DELETE OR UPDATE	myschema1.mytbl2trgfct	Enabled	ON	OFF

Select

Actions on objects (0)

All / Visible / None / Invert

OFF ON

Figure 19 – Application triggers list.

6.3.12 Monitoring rollback operations

Using the “Rollback operations” tab of the main bar, users can monitor the rollback operations. Three different lists are displayed:

- in progress rollback operations, with the characteristics of the rollback operations and estimates of the percentage of the operation already done and of the remaining duration,
- the completed operations,
- logged rollback operations that are consolidable.

For each consolidable rollback, a button allows to effectively consolidate the operation.

User postgres

English

Emaj_web > Pg 16 > postgres

SQL | History

Groups

Schemas

Triggers

E-Maj Rollbacks

Activity

E-Maj

In progress E-Maj rollbacks (x 1)

	Rlbk Id.	Groups	State	Rollback start	Current duration	Estimated remaining	% completed	Target mark	Logged ?	Sessions	Actions	Comments
	4	myGroup2	LOCKING	Mon 03 Feb 09:15:08	2:05.869	0.067186	0	MARK1	Yes	1		

Completed E-Maj rollbacks (x 3)

	Rlbk Id.	Groups	State	Rollback start	Rollback end	Duration	Target mark	Logged ?	Sessions	Actions	Comment
	3	myGroup2		Mon 03 Feb 08:14:26	Mon 03 Feb 08:14:26	0.185248	MARK3	Yes	1		
	2	myGroup2		Mon 03 Feb 08:14:26	Mon 03 Feb 08:14:26	0.164506	tmp_mark	No	1		
	1	myGroup2		Mon 03 Feb 08:14:26	Mon 03 Feb 08:14:26	0.188931	MARK1	Yes	1		

Consolidable E-Maj logged rollbacks (x 1)

	Group	Target mark		Row changes	Intermediate marks	End rollback mark		Actions
		Name	Set at			Name	Set at	
	myGroup2	MARK3	Mon 03 Feb 08:14:26	6	1	RLBK_3_DONE	Mon 03 Feb 08:14:26	

Figure 20 – Rollback operations monitoring.

Clicking on a *rollback* identifier in one of these tables displays a page that shows information details about the selected in progress or completed operation.

More precisely, are displayed:

- the rollback properties,
- its progress,
- the final report returned to the user, when the operation is completed,
- the detail of the operation plan, showing each elementary step, with its duration and optionally estimates computed by E-Maj at the operation initialisation,
- and information about rollback sessions.

03/02/2025 - 09:20:08
User postgres
English

Email_web > Pg 16 > postgres > 3
SQL | History

Groups
Schemas
Triggers
E-Maj Rollbacks
Activity
E-Maj

< #2
E-Maj rollback #3
Back to the list
#4 >

Properties

Groups	Target mark	Set at	Logged ?	Sessions	Tables to process	Sequences to process
myGroup2	MARK3	Mon 03 Feb 08:14:26	Yes	1	2 / 4	1 / 2

Set a comment

Rollback progress

State	Rollback start	Rollback end	Global duration	Planning duration	Locking duration
COMMITTED	Mon 03 Feb 08:14:26	Mon 03 Feb 08:14:26	0.185248	0.080805	0.005684

Execution report

Message

- Rollback id = 3.
- 2 / 4 tables effectively processed.
- 1 / 2 sequences effectively processed.

Elementary steps (x 3)

#	Schema	Table	Step	Session	Start time	Actual		Estimates		Q	Method
						Duration	Quantity	Duration	Quantity		
1			Rollback sequences	1	08:14:26.610	0.014709	2	0.010306	2		STAT
2	myschema2	mytbl1	Rollback the table	1	08:14:26.631	0.010664	2	0.007200	2		STAT+
3	myschema2	myTbl3	Rollback the table	1	08:14:26.647	0.009688	1	0.006889	2		STAT+

Sessions (x 1)

Session	Start time	End time	Duration	Transaction id.
1	Mon 03 Feb 08:14:26	Mon 03 Feb 08:14:26	0.124159	372940

Figure 21 – Details of a Rollback operation.

6.3.13 E-Maj activity

The “Activity” tab allows to monitor in real time the activity of E-Maj log triggers.

03/02/2025 - 09:36:34
User postgres
English

Emaj_web > Pg 16 > postgres
SQL | History

Groups
Schemas
Triggers
E-Maj Rollbacks
Activity
E-Maj

E-Maj changes activity

Reset

Groups
Tables
Sequences

Include regexp
Exclude regexp
Maximum #rows
Main sort criteria

Refresh
Auto refresh

Global activity

Since	Row changes	Changes / sec
22.654 s	0	0.000

Logging groups (x 2/3)

Group	Latest mark	Set at	Since latest mark		Since previous display	
			Row changes	Changes / sec	Row changes	Changes / sec
myGroup1	MARK3	2025/02/03 08:14:26	0	0.000	0	0.000
myGroup2	RLBK_3_DONE	2025/02/03 08:14:26	0	0.000	0	0.000

Tables in logging groups (x 9/11)

Schema	Table	Group	Since latest mark		Since previous display	
			Row changes	Changes / sec	Row changes	Changes / sec
myschema1	myTbl3	myGroup1	0	0.000	0	0.000
myschema1	mytbl1	myGroup1	0	0.000	0	0.000
myschema1	mytbl2	myGroup1	0	0.000	0	0.000
myschema1	mytbl2b	myGroup1	0	0.000	0	0.000
myschema1	mytbl4	myGroup1	0	0.000	0	0.000
myschema2	myTbl3	myGroup2	0	0.000	0	0.000
myschema2	mytbl1	myGroup2	0	0.000	0	0.000
myschema2	mytbl2	myGroup2	0	0.000	0	0.000
myschema2	mytbl4	myGroup2	0	0.000	0	0.000

Sequences in logging groups (x 3/4)

Schema	Sequence	Group	Since latest mark		Since previous display	
			Row changes	Changes / sec	Row changes	Changes / sec
myschema1	myTbl3_col31_seq	myGroup1	-4.333333333333333	-0.001	0	0.000
myschema2	myTbl3_col31_seq	myGroup2	-1	-0.000	0	0.000
myschema2	myseq1	myGroup2	-1000	-0.202	0	0.000

Figure 22 – E-Maj activity.

At the page top, the form allows to define the monitoring wishes, by default the 5 most active tables groups, the 20 most active tables and the 20 most active sequences. Regular expressions may be set to filter (include or exclude) tables groups, tables and sequences.

The activity indicators are displayed under the form:

- at a global level (all groups, tables and sequences),

E-Maj Reference Guide – version 4.7.1

Page 145 / 167

- for the selected tables groups,
- for the selected tables of selected tables groups,
- for the selected sequences of selected tables groups.

Four activity indicators are computed:

- the number of changes since the latest mark set for the tables group,
- the throughput since the latest mark, in number of changes per second,
- the number of changes since the last display,
- the throughput since the last display.

Groups, tables and sequences are displayed and filtered in descending order by the number of changes since either the last display (by default) or the latest mark.

The page can be refreshed either manually or automatically. The automatic refresh delay is set in the `Emaj_web` configuration (10 seconds by default).

6.3.14 E-Maj environment state

By selecting the “*E-Maj*” tab of the main bar, the user reaches an overview of the E-Maj environment state.

First, the installed PostgreSQL and E-Maj versions are displayed,

If the user is connected with a “*superuser*” role, some buttons allow to create, update or drop the *emaj* extension, depending on the context.

Then the disk space used by E-Maj (log tables, technical tables and their indexes), and the part of the global database space it represents are displayed.

Next, the environment integrity is checked; the result of the *emaj_verify_all()* function execution is displayed.

The page ends with a list of the extension parameters value, be they present in the *emaj_param* table or set to their default value.

Two buttons allow to export and import parameters configurations to or from a local file.

User postgres

English

Emaj_web > Pg 16 > postgres

SQL | History

Groups

Schemas

Triggers

E-Maj Rollbacks

Activity

E-Maj

Versions

PostgreSQL version: 16.6

E-Maj version: 4.6.0 (installed as extension)

Emaj_web version: 4.6

"emaj" extension management

Drop extension

E-Maj environment characteristics

Disk space used by the E-Maj environment: 448 kB = 3.5% of the current database.

E-Maj environment consistency

Diagnostics

✓

No error detected

Extension parameters

General parameters

History retention delay1 YEAR (def)

Dblink connection string#####...

Log tables structure change(def)

E-Maj rollback cost model parameters

Rollback step fixed cost2500 μs (def)

Dblink overhead for a rollback step4000 μs (def)

Table rollback fixed cost1000 μs (def)

Average cost of an elementary update rollback100 μs (def)

Average cost of an elementary log table deletion10 μs (def)

Average foreign key check cost20 μs (def)

Export

Import

Figure 23 – The E-Maj environment state.

7 CONTRIBUTE TO THE E-MAJ DEVELOPMENT

Any contribution to the development and the improvement of the E-Maj extension is welcome. This chapter gives some information to make these contributions easier.

7.1 BUILD THE E-MAJ ENVIRONMENT

The E-Maj extension repository is hosted on the *github* site: <https://github.com/dalibo/emaj>

7.1.1 Clone the E-Maj repository

So the first action to perform is to locally clone this repository on his/her own computer. This can be done by using the functionalities of the *github* web interface or by typing the shell command:

```
git clone https://github.com/dalibo/emaj.git
```

7.1.2 Description of the E-Maj tree

So one has a full directory tree (except the web clients). It contains all directories and files described in the appendix 8.2, except the *doc* directory content that is separately maintained (see below).

The main directory also contains the following components:

- the *tar.index* file that is used to build the tarball of the E-Maj version distributed on *pgxn.org*
- the *docs* directory with all sources of the online documentation (see §7.4)
- in the *sql* directory:
 - the file *emaj--devel.sql*, source of the extension in its current version
 - the source of the previous version *emaj--<previous_version>.sql*
 - a *emaj_prepare_emaj_web_test.sql* script that prepares an E-Maj environment to test the *Emaj_web* client
- a *test* directory containing all components used to test the extension (see §7.3)
- a *tools* directory containing some ... tools.

7.1.3 Setting tools parameters

The tools stored in the *tools* directory need some parameters to be set, depending on his/her own environment. A parameter system covers some tools. For the others, the *tools/README* file details the changes to apply.

7.1.3.1 Créating the `emaj_tools.env` file

The parameters that may be modified are grouped into the `tools/emaj_tools.env` file, which is called by `tools/emaj_tools.profile`.

The repository contains a file `tools/emaj_tools.env-dist` that may be used as a template to create the `emaj_tools.env` file.

The `emaj_tools.env` file must contain:

- the list of PostgreSQL versions that are supported by the current E-Maj version and for which a PostgreSQL instance exists for tests (`EMAJ_USER_PGVER` variable),
- for each PostgreSQL version used for the tests, 6 variables describing the location of binaries, the main directory of the related instance, the role and the ip-port to be used for the connection to the instance.

7.2 CODING

7.2.1 Versionning

The version currently under development is named `devel`.

Regularly and when it is justified, a new version is created. Its name has a X.Y.Z pattern.

The `create_version` tool assists in creating this version. It is only used by the E-Maj maintainers. So its use is not described here.

7.2.2 Coding rules

Coding the `emaj--devel.sql` script must follow these rules:

- script structure: after some checks about the execution conditions that must be met, the objects are created in the following order: roles, enumerated types, sequences, tables (with their indexes and constraints), composite types, E-Maj parameters, low level functions, elementary functions that manage tables and sequences, functions that manage tables groups, general purpose functions, event triggers, grants, additional actions for the extensions. The script ends with some final operations.
- all objects are created in the `emaj` schema, except the `_emaj_protection_event_trigger_fnct()` function, created in the `public` schema,
- tables and sequences names are prefixed by `'emaj_'`
- functions names are prefixed by `'emaj_'` when they are usable by end users, or by `'_'` for internal functions,
- the internal tables and the functions callable by end users must have a comment,
- the language keywords are in upper case, objects names are in lower case,
- the code is indented with 2 space characters,
- lines must not contain tab characters, must not be longer than 140 characters long and must not end with spaces,

- in the functions structure, the code delimiters must contain the function name surrounded with a \$ character (or *\$do\$* for code blocks),
- variables names are prefixed with 'v_' for simple variables, 'p_' for functions parameters or 'r_' for *RECORD* type variables,
- the code must be compatible with all PostgreSQL versions supported by the current E-Maj version. When this is strictly necessary, the code may be differentiated depending on the PostgreSQL version.

A perl script, *tools/check_code.pl* performs some checks on the code format of the script that creates the extension. It also detects unused variables. This script is directly called in non-regression tests scenarios (Cf §7.3).

7.2.3 Version upgrade script

E-Maj is installed into a database as an extension. The E-Maj administrator must be able to easily upgrade the extension version (cf §3.4). So an upgrade script is provided for each version, that upgrades from the previous version to the next version. It is named *emaj--<previous_version>--devel.sql*.

The development of this script follows these rules:

- Develop/maintain the upgrade script at the same time as the main *emaj--devel.sql* script, so that the tests of a change include upgrade version cases,
- Apply the same coding rules as for the main script,
- As far as possible, ensure that the upgrade operation is able to process tables groups in logging state, without losing the capability to perform E-Maj rollbacks on marks set prior the version upgrade.

At the beginning of a version, the upgrade script is built using a template (the file *tools/emaj_upgrade.template*).

As the development goes on, a perl script helps to synchronize the creation/deletion/replacement of functions. It compares the *emaj--devel.sql* script and the script that creates the previous version and updates the *emaj--<previous_version>--devel.sql* script. To let it work properly, it is essential to keep both tags that frame the part of the script that describes functions.

After having adapted the parameters (see the *TOOLS/README* file), just submit:

```
perl tools/sync_fct_in_upgrade_script.pl
```

The other parts of the script must be coded manually. If the structure of an internal table is changed, the table content must be migrated (scripts for prior version upgrade can be used as examples).

7.3 TESTING

Through the rollback functions, the E-Maj extension updates database content. So the reliability is a key characteristics. For this reason, it is essential to pay a great attention to the tests.

7.3.1 Create PostgreSQL instances

The ideal is to be able to test E-Maj with all PostgreSQL versions that are supported by the extension

The *tools/create_cluster.sh* script helps in creating a test instance. Its content may show the characteristics of the instance to create. It can also be executed (after parameters setting as indicated in *tools/README*):

```
tools/create_cluster.sh <PostgreSQL_major_version>
```

7.3.2 Install software dependencies

Testing the clients may require to install some additional software components:

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

7.3.3 Execute non regression tests

A solid test environment is supplied in the repository. It contains:

- a test tool,
- test scenarios,
- expected results.

7.3.3.1 The test scenarios

The test system contains 5 scenarios:

- a full standart scenario,
- the same scenario but installing the extension with the *emaj-devel.sql* script provided for cases when a “CREATE EXTENSION emaj” statement is not possible,
- the same scenario but installing the extension from the previous version with an immediate upgrade into the current version,
- a shorter scenario but with an upgrade from the previous extension version to the current one while tables groups are in logging state,
- a similar scenario but with an upgrade from the oldest E-Maj version that is available for the oldest supported Postgres version.

These scenarios call *psql* scripts, all located into the *test/sql* directory. The scripts chain E-Maj function calls in different contexts, and SQL statements to prepare or check the results.

At the end of scripts, internal sequences are often reset, so that a single function call insertion does not produce impacts in the next scripts results.

The *psql* test scripts must be maintained in the same time as the extension source.

7.3.3.2 The expected results

For each *psql* script, the test tool produces a result file. These files are distinguished from a PostgreSQL version to another. They are located in the *test/<PostgreSQL_version>/results* directory.

At the end of a run, the test tool compares these files with a reference located into the *test/<PostgreSQL_version>/expected* directory.

Unlike for files in the *test/<PostgreSQL_version>/results* directory, files in the *test/<PostgreSQL_version>/expected* directory belong to the git repository. They must always remain consistent with the source of the extension and the *psql* test scripts.

7.3.3.3 The test tool

The test tool, *regress.sh*, combines all test functions.

Before using it, it is necessary to:

- have the PostgreSQL instances to be used already created and the *tools/emaj_tools.env* file already setup
- manually create the *test/<PostgreSQL_version>/results* directories.

The test tool can be launched with the command:

```
tools/regress.sh
```

As it starts with a copy of the *emaj.control* file into the *SHAREDIR/extension* directory of each configured PostgreSQL version, it may ask for the password of the Linux account to be able to execute *sudo* commands. It also automatically generates the *emaj-devel.sql* script used to create the extension with *psql*.

It then displays the list of test functions in a menu. Just enter the letter corresponding to the chosen test.

The test functions are:

- standart tests for each configured PostgreSQL version,
- the tests with the installation of the previous version followed by an upgrade,
- the tests with the installation of the version with the *emaj-devel.sql* script,
- the tests with an E-Maj version upgrade while tables groups are in logging state,
- tests chaining a database save with *pg_dump* and a restore, with different PostgreSQL versions,
- a PostgreSQL upgrade version test using *pg_upgrade* with a database containing the E-Maj extension.

It is important to execute the four first sets of tests for each E-Maj change.

7.3.3.4 Validate results

After having executed a *psql* script, *regress.sh* compares the outputs of the run with the expected outputs and reports the comparison result with the words 'ok' or 'FAILED'.

Here is an example of the display issued by the test tool (in this case with the scenario chaining the installation and a version upgrade, and with a detected difference):

```
Run regression test
===== dropping database "regression" =====
DROP DATABASE
===== creating database "regression" =====
CREATE DATABASE
ALTER DATABASE
===== running regression test queries =====
test install_upgrade      ... ok
test setup                ... ok
test create_drop          ... ok
test start_stop           ... ok
test mark                 ... ok
test rollback             ... ok
test stat                 ... ok
test misc                 ... ok
test verify               ... ok
test alter                ... ok
test alter_logging        ... ok
test viewer               ... ok
test adm1                 ... ok
test adm2                 ... ok
test adm3                 ... ok
test client               ... ok
test check                ... FAILED
test cleanup              ... ok

=====
1 of 18 tests failed.
=====
```

The differences that caused some tests to fail can be viewed in the file `"/home/postgres/proj/emaj/test/18/regression.diffs"`. A copy of the test summary that you see above is saved in the file `"/home/postgres/proj/emaj/test/18/regression.out"`.

When at least one script fails, it is important to closely analyze the differences, by reviewing the *test/<PostgreSQL_version>/regression.diffs* file content, and check that the differences are directly linked to changes applied in the extension source code or in the test scripts.

Once the reported differences are considered as valid, the content of the *test/<PostgreSQL_version>/result* directories must be copied into the *test/<PostgreSQL_version>/expected* directories. A shell script processes all PostgreSQL versions in a single command:

```
sh tools/copy2Expected.sh
```

It may happen that some test outputs do not match the expected outputs, due to differences in the PostgreSQL behaviour from one run to another. Repeating the test allows to check these cases.

7.3.4 Test coverage

7.3.4.1 Functions test coverage

The PostgreSQL test instances are configured to count the functions executions. The *check.sql* test script displays the functions execution counters. It also displays E-Maj functions that have not been executed.

7.3.4.2 Error messages test coverage

A perl script extracts error and warning messages coded in the *sql/emaj--devel.sql* file. It then extracts the messages from the files of the *test/10/expected* directory. It finally displays error or warning messages that are not covered by tests.

The script can be run with the command:

```
perl tools/check_error_messages.pl
```

Some messages are known to not be covered by tests (for instance internal errors that are hard to reproduce). These messages, coded in the perl script, are excluded from the final report.

7.3.5 Evaluate the performances

The *tools/performance* directory contains some shell scripts helping in measuring performances. As the measurement results totally depend on the platform and the environment used, no reference results are supplied.

The scripts cover the following domains:

- *dump_changes/dump_changes_perf.sh* measures the performances of changes dump operations, with various consolidation levels;
- *large_group/large_group.sh* evaluates the behaviour of groups containing a large number of tables;
- *log_overhead/pgbench.sh* evaluates the log mechanism overhead, using *pgbench*;
- *rollback/rollback_perf.sh* evaluates the E-Maj rollback performances with different tables profiles.

For all these files, some variables have to be configured at the beginning of the scripts.

7.4 DOCUMENTING

A *LibreOffice* format documentation is managed by the maintainers. It has its own *github* repository: *emaj_doc*. Thus the *doc* directory of the main repository remains empty.

The online documentation is managed by *sphinx*. It is located in the *docs* directory.

To install *sphinx*, refer to the *docs/README.rst* file.

The documentation exists in two languages, English and French. Depending on the languages, document sources are located in */docs/en* and */docs/fr*. These documents are in *ReStructured Text* format.

To compile the documentation for a language, set the current directory to *docs/<language>* and execute the command:

```
make html
```

When there is no compilation error anymore, the documentation becomes available locally on a browser, by opening the *docs/<language>/_build/html/index.html* file.

The documentation on the *readthedocs.org* site is automatically updated as soon as the main *github* repository is updated.

7.5 SUBMITTING A PATCH

Patches can be proposed to the E-Maj maintainers through *Pull Requests* on the *github* site.

Before submitting a patch, it may be useful to create an *issue* on *github*, in order to start a discussion with the maintainers and help in working on the patch.

7.6 CONTRIBUTING TO EMAJ_WEB

The web client development is managed in a separate project, even though it is linked to the *emaj* extension. Changes in the extension may need changes in the client, in particular:

- when the API provided by the extension changes;
- to allow the web client users to take benefit from new features added to the extension.

In the first case, both changes must be synchronized.

The project is maintained in the *github* repository: https://github.com/dalibo/emaj_web

It is important to keep in mind that the web client interfaces *emaj* extensions that may be in different versions. The *libraries/version.inc.php* file defines the usable versions ranges.

8 APPENDIX

8.1 E-MAJ FUNCTIONS LIST

The E-Maj functions that are available to users can be grouped into 3 categories. They are listed below, in alphabetic order. They are all callable by roles having *emaj_adm* privileges. The charts also specify those callable by *emaj_viewer* roles.

8.1.1 Tables or sequences level functions

Functions	Input parameters	Output data	Callable by <i>emaj_viewer</i>	Ref.
emaj_assign_sequence	schema TEXT sequence TEXT group TEXT [properties JSONB] [mark TEXT]	1 INT		§ 4.4.2
emaj_assign_sequences	schema TEXT sequences.array TEXT[] group TEXT [properties JSONB] [mark TEXT]	#.séquences INT		§ 4.4.2
emaj_assign_sequences	schema TEXT sequences.to.include.filter TEXT sequences.to.exclude.filter TEXT group TEXT [properties JSONB] [mark TEXT]	#.séquences INT		§ 4.4.2
emaj_assign_table	schema TEXT table TEXT groupe TEXT [properties JSONB] [mark TEXT]	1 INT		§ 4.4.2
emaj_assign_tables	schema TEXT tables.array TEXT[] group TEXT [properties JSONB] [mark TEXT]	#.tables INT		§ 4.4.2
emaj_assign_tables	schema TEXT tables.to.include.filter TEXT tables.to.exclude.filter TEXT group TEXT [properties JSONB] [mark TEXT]	#.tables INT		§ 4.4.2
emaj_get_current_log_table	schema TEXT	(log.schema TEXT,	Yes	§ 4.11.4

Functions	Input parameters	Output data	Callable by emaj_viewer	Ref.
	table TEXT	log.table TEXT)		
emaj_log_stat_table	schema TEXT table TEXT [start.date-time TIMESTAMP TZ] [end.date-time TIMESTAMP TZ]	SETOF emaj_log_stat_table_t ype	Yes	§ 4.8.2.1
emaj_log_stat_table	schema TEXT table TEXT start.group TEXT start.mark TEXT [end.group TEXT end.mark TEXT]	SETOF emaj_log_stat_table_t ype	Yes	§ 4.8.2.1
emaj_log_stat_sequence	schema TEXT sequence TEXT [start.date-time TIMESTAMP TZ] [end.date-time TIMESTAMP TZ]	SETOF emaj_log_stat_sequen ce_type	Yes	§ 4.8.2.2
emaj_log_stat_sequence	schema TEXT sequence TEXT start.group TEXT start.mark TEXT [end.group TEXT end.mark TEXT]	SETOF emaj_log_stat_sequen ce_type	Yes	§ 4.8.2.2
emaj_modify_table	schema TEXT table TEXT properties JSONB [mark TEXT]	#.tables INT		§ 4.4.7
emaj_modify_tables	schema TEXT tables.array TEXT[] properties JSONB [mark TEXT]	#.tables INT		§ 4.4.7
emaj_modify_tables	schema TEXT tables.to.include.filter TEXT tables.to.exclude.filter TEXT properties JSONB [mark TEXT]	#.tables INT		§ 4.4.7
emaj_move_sequence	schema TEXT sequence TEXT new.group TEXT [mark TEXT]	1 INT		§ 4.4.6
emaj_move_sequences	schema TEXT sequences.array TEXT[] new.group TEXT [mark TEXT]	#.sequences INT		§ 4.4.6
emaj_move_sequences	schema TEXT sequences.to.include.filter TEXT sequences.to.exclude.filter	#.sequences INT		§ 4.4.6

Functions	Input parameters	Output data	Callable by emaj_viewer	Ref.
	TEXT new.group TEXT [mark TEXT]			
emaj_move_table	schema TEXT table TEXT new.group TEXT [mark TEXT]	1 INT		§ 4.4.5
emaj_move_tables	schema TEXT tables.array TEXT[] new.group TEXT [mark TEXT]	#.tables INT		§ 4.4.5
emaj_move_tables	schema TEXT tables.to.include.filter TEXT tables.to.exclude.filter TEXT new.group TEXT [mark TEXT]	#.tables INT		§ 4.4.5
emaj_remove_sequence	schema TEXT sequence TEXT [mark TEXT]	1 INT		§ 4.4.4
emaj_remove_sequences	schema TEXT sequences.array TEXT[] [mark TEXT]	#.sequences INT		§ 4.4.4
emaj_remove_sequences	schema TEXT sequences.to.include.filter TEXT sequences.to.exclude.filter TEXT [mark TEXT]	#.sequences INT		§ 4.4.4
emaj_remove_table	schema TEXT table TEXT [mark TEXT]	1 INT		§ 4.4.3
emaj_remove_tables	schema TEXT tables.array TEXT[] [mark TEXT]	#.tables INT		§ 4.4.3
emaj_remove_tables	schema TEXT tables.to.include.filter TEXT tables.to.exclude.filter TEXT [mark TEXT]	#.tables INT		§ 4.4.3

8.1.2 Groups level functions

Functions	Input parameters	Output data	Callable by emaj_viewer	Ref.
emaj_comment_group	group TEXT comment TEXT	-		§ 4.5.2

Functions	Input parameters	Output data	Callable by emaj_viewer	Ref.
emaj_comment_mark_group	group TEXT mark TEXT comment TEXT	-		§ 4.6.1
emaj_consolidate_rollback_group	group TEXT end.rollback.mark TEXT	# tables.and.seq INT		§4.7.1
emaj_create_group	group TEXT [is.rollbackable BOOLEAN] [comment TEXT]	1 INT		§ 4.2.2
emaj_delete_before_mark_group	group TEXT mark TEXT	#.deleted.marks INT		§ 4.6.5
emaj_delete_mark_group	group TEXT mark TEXT	1 INT		§ 4.6.4
emaj_detailed_log_stat_group	group TEXT start.mark TEXT end.mark TEXT	SETOF emaj_detailed_log_stat_type	Yes	§ 4.8.1.2
emaj_detailed_log_stat_groups	groups.array TEXT[] start.mark TEXT end.mark TEXT	SETOF emaj_log_stat_type	Yes	§ 4.8.1.2
emaj_does_exist_group	group TEXT	BOOLEAN	Yes	§ 4.5.7
emaj_does_exist_mark_group	group TEXT mark TEXT	BOOLEAN	Yes	§ 4.5.7
emaj_drop_group	group TEXT	#.tables.and.seq INT		§ 4.2.4
emaj_dump_changes_group	group TEXT start.mark TEXT end.mark TEXT options.list TEXT tables.seq.array TEXT[] output.directory TEXT	msg.#.files TEXT		§4.9.4
emaj_estimate_rollback_group	group TEXT mark TEXT is.logged BOOLEAN	duration INTERVAL	Yes	§ 4.7.1
emaj_estimate_rollback_groups	groups.array TEXT[] mark TEXT is.logged BOOLEAN	duration INTERVAL	Yes	§ 4.7.1
emaj_force_drop_group	group TEXT	#.tables.and.seq INT		§4.5.5
emaj_force_stop_group	group TEXT	#.tables.and.seq INT		§ 4.5.4
emaj_forget_group	group TEXT	#.erased.traces INT		§ 4.5.8
emaj_gen_sql_dump_changes_group	group TEXT start.mark TEXT end.mark TEXT options.list TEXT tables.seq.array TEXT[]	msg.#.statements TEXT	Yes	§4.9.5
emaj_gen_sql_dump_changes_group	group TEXT start.mark TEXT end.mark TEXT options.list TEXT	msg.#.statements TEXT		§4.9.5

Functions	Input parameters	Output data	Callable by emaj_viewer	Ref.
	tables.seq.array TEXT[] output.directory TEXT			
emaj_gen_sql_group	group TEXT start.mark TEXT end.mark TEXT output.file.path TEXT [tables.seq.array TEXT[]]	#.gen.statements BIGINT		§ 4.10
emaj_gen_sql_groups	groups.array TEXT[] start.mark TEXT end.mark TEXT output.file.path TEXT [tables.seq.array TEXT[]]	#.gen.statements BIGINT		§ 4.10
emaj_get_previous_mark_group	group TEXT date.time TIMESTAMP TZ	mark TEXT	Yes	§ 4.6.2
emaj_get_previous_mark_group	group TEXT mark TEXT	mark TEXT	Yes	§ 4.6.2
emaj_is_logging_group	group TEXT	BOOLEAN	Yes	§ 4.5.7
emaj_log_stat_group	group TEXT start.mark TEXT end.mark TEXT	SETOF emaj_log_stat_type	Yes	§ 4.8.1.1
emaj_log_stat_groups	groups.array TEXT[] start.mark TEXT end.mark TEXT	SETOF emaj_log_stat_type	Yes	§ 4.8.1.1
emaj_logged_rollback_group	group TEXT mark TEXT [is.alter.group.allowed BOOLEAN] [commentaire TEXT]	SETOF (severity TEXT, message TEXT)		§ 4.3.5
emaj_logged_rollback_groups	groups.array TEXT[] mark TEXT [is.alter.group.allowed BOOLEAN] [commentaire TEXT]	SETOF (severity TEXT, message TEXT)		§ 4.3.5
emaj_protect_group	group TEXT	0/1 INT		§ 4.5.3
emaj_protect_mark_group	group TEXT mark TEXT	0/1 INT		§ 4.6.6
emaj_rename_mark_group	group TEXT mark TEXT new.name TEXT	-		§ 4.6.3
emaj_reset_group	group TEXT	#.tables.and.seq INT		§ 4.5.1
emaj_rollback_group	group TEXT mark TEXT [is_alter_group_allowed BOOLEAN] [commentaire TEXT]	SETOF (severity TEXT, message TEXT)		§ 4.3.4
emaj_rollback_groups	groups.array TEXT[]	SETOF (severity		§ 4.3.4

Functions	Input parameters	Output data	Callable by emaj_viewer	Ref.
	mark TEXT [is_alter_group_allowed BOOLEAN] [commentaire TEXT]	TEXT, message TEXT)		
emaj_sequence_stat_group	group TEXT start.mark TEXT end.mark TEXT	SETOF emaj_log_stat_type	Yes	§ 4.8.1.3
emaj_sequence_stat_groups	groups.array TEXT[] start.mark TEXT end.mark TEXT	SETOF emaj_log_stat_type	Yes	§ 4.8.1.3
emaj_set_mark_group	group TEXT [mark TEXT] [comment TEXT]	#.tables.and.seq INT		§ 4.3.3
emaj_set_mark_groups	groups.array TEXT[] [mark TEXT] [comment TEXT]	#.tables.and.seq INT		§ 4.3.3
emaj_snap_group	group TEXT directory TEXT copy.options TEXT	#.tables.and.seq INT		§ 4.11.7
emaj_start_group	group TEXT [mark TEXT] [reset.log BOOLEAN]	#.tables.and.seq INT		§ 4.3.2
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		§ 4.3.6
emaj_stop_groups	groups.array TEXT[] [mark TEXT]	#.tables.and.seq INT		§ 4.3.6
emaj_unprotect_group	group TEXT	0/1 INT		§ 4.5.3
emaj_unprotect_mark_group	group TEXT mark TEXT	0/1 INT		§ 4.6.6

8.1.3 General purpose functions

Functions	Input parameters	Output data	Callable by emaj_viewer	Ref.
emaj_cleanup_rollback_state	-	# rollback INT		§ 4.7.1
emaj_comment_rollback	rollback.id INT, comment TEXT	-		§ 4.7.3
emaj_disable_protection_by_event_triggers	-	# triggers INT		§ 4.11.6
emaj_drop_extension	-	-		§ 3.5.1
emaj_enable_protection_by_event_triggers	-	# triggers INT		§ 4.11.6

Functions	Input parameters	Output data	Callable by emaj_viewer	Ref.
emaj_export_groups_configuration	NULL [, groups.array TEXT[]]	configuration JSON		§ 4.5.6.1
emaj_export_groups_configuration	file TEXT [, groups.array TEXT[]]	#.groups INT		§ 4.5.6.1
emaj_export_parameters_configuration	-	parameters JSON		§4.11.3.1
emaj_export_parameters_configuration	file.path TEXT	# parameters INT		§ 4.11.3.1
emaj_get_consolidable_rollbacks	-	SETOF emaj_consolidable_rollback_type	Yes	§ 4.7.1
emaj_get_version	-	version TEXT	Yes	§ 4.11.1
emaj_import_groups_configuration	groups JSON [, groups.array TEXT[]] [, alter.logging.groups BOOLEAN] [, mark TEXT]	#.groups INT		§ 4.5.6.1
emaj_import_groups_configuration	file TEXT [, groups.array TEXT[]] [, alter.logging.groups BOOLEAN] [, mark TEXT]	#.groups INT		§ 4.5.6.1
emaj_import_parameters_configuration	parameters JSON, [delete.current.conf BOOLEAN]	# parameters INT		§ 4.11.3.2
emaj_import_parameters_configuration	file.path TEXT, [delete.current.conf BOOLEAN]	# parameters INT		§ 4.11.3.2
emaj_purge_histories	Retention.delay INTERVAL	-		§ 4.11.5
emaj_rollback_activity	-	SETOF emaj_rollback_activity_type	Yes	§ 4.7.1
emaj_verify_all	-	SETOF TEXT	Yes	§ 4.11.2

8.2 E-MAJ DISTRIBUTION CONTENT

Once installed (see §3), an E-Maj version contents the following files.

sql/emaj--<version>.sql	installation script of the extension
sql/emaj-<version>.sql	alternate psql installation script
sql/emaj--4.7.0--4.7.1.sql	extension upgrade script from 4.7.0 to 4.7.1
sql/emaj--4.6.0--4.7.0.sql	extension upgrade script from 4.6.0 to 4.7.0
sql/emaj--4.5.0--4.6.0.sql	extension upgrade script from 4.5.0 to 4.6.0
sql/emaj--4.4.0--4.5.0.sql	extension upgrade script from 4.4.0 to 4.5.0
sql/emaj--4.3.1--4.4.0.sql	extension upgrade script from 4.3.1 to 4.4.0
sql/emaj--4.3.0--4.3.1.sql	extension upgrade script from 4.3.0 to 4.3.1
sql/emaj--4.2.0--4.3.0.sql	extension upgrade script from 4.2.0 to 4.3.0
sql/emaj--4.1.0--4.2.0.sql	extension upgrade script from 4.1.0 to 4.2.0
sql/emaj--4.0.1--4.1.0.sql	extension upgrade script from 4.0.1 to 4.1.0
sql/emaj--4.0.0--4.0.1.sql	extension upgrade script from 4.0.0 to 4.0.1
sql/emaj--3.4.0--4.0.0.sql	extension upgrade script from 3.4.0 to 4.0.0
sql/emaj--3.3.0--3.4.0.sql	extension upgrade script from 3.3.0 to 3.4.0
sql/emaj--3.2.0--3.3.0.sql	extension upgrade script from 3.2.0 to 3.3.0
sql/emaj--3.1.0--3.2.0.sql	extension upgrade script from 3.1.0 to 3.2.0
sql/emaj--3.0.1--3.1.0.sql	extension upgrade script from 3.0.0 to 3.1.0
sql/emaj--2.3.1--3.0.0.sql	extension upgrade script from 2.3.1 to 3.0.0
sql/emaj_demo.sql	psql E-Maj demonstration script
sql/emaj_prepare_parallel_rollback_test.sql	psql test script for parallel rollbacks
sql/emaj_uninstall.sql	psql script to uninstall the E-Maj components
README.md	reduced extension's documentation
CHANGES.md	change log
LICENSE	information about E-Maj license
AUTHORS.md	who are the authors
META.json	technical data for PGXN
emaj.control	extension control file used by the integrated extensions management
Makefile	E-Maj install/uninstall description for the <i>make</i> command
doc/Emaj.<version>_doc_en.pdf	English version of the full E-Maj

	documentation
doc/Emaj.<version>_doc_fr.pdf	French version of the full E-Maj documentation
doc/Emaj.<version>_pres_en.odp	English version of the E-Maj presentation
doc/Emaj.<version>_pres_fr.odp	French version of the E-Maj presentation
doc/Emaj.<version>_pres_en.pdf	English version of the E-Maj presentation (pdf version)
doc/Emaj.<version>_pres_fr.pdf	French version of the E-Maj presentation (pdf version)
client/emajParallelRollback.php	php tool to spawn parallel rollbacks
client/emajParallelRollback.pl	perl tool to spawn parallel rollbacks
client/emajRollbackMonitor.php	php tool to monitor rollbacks
client/emajRollbackMonitor.pl	perl tool to monitor rollbacks
client/emajStat.pl	perl tool to monitor changes recording

8.3 POSTGRESQL AND E-MAJ VERSIONS COMPATIBILITY MATRIX

PostgreSQL versions		E-Maj versions	
Min	Max	Min	Date
12	18	4.7.0	09/2025
11	17	4.5.0	09/2024
11	16	4.2.0	04/2023
9.5	15	4.1.0	10/2022
9.5	14	3.3.0	03/2020
9.5	12	3.1.0	06/2019
9.5	11	3.0.0	03/2019
9.2	11	2.3.1	09/2018
9.2	10	2.3.0	07/2018
9.1	10	2.1.0	08/2017
9.1	9.6	2.0.0	11/2016
8.3	9.6	1.3.1	09/2016
8.3	9.5	1.2.0	01/2016
8.3	9.3	1.1.0	10/2013
8.2	9.2	0.11.1	07/2012
8.2	9.1	0.10.0	11/2011
8.2	9.0	0.8.0	10/2010
8.2	8.4	0.4.0	01/2010