« E-Maj » PostgreSQL extension

-

User's Guide

Version: 0.9.0

Last update: 2011 february 27th.

Summary

1 - Introduction....4

1.1 - Document content....4

1.2 - License....4

1.3 - E-Maj's objectives....4

2 - How E-Maj works....6

2.1 - Concepts....6

2.1.1 - Tables Group....6

2.1.2 - Mark....6

2.1.3 - Rollback....6

2.2 - Architecture....7

2.2.1 - Created objects....7

2.2.2 - Norm for E-Maj objects naming....8

3 - How to use E-Maj....9

3.1 - Deletion of a previous version....9

3.2 - E-Maj extension setup....10

3.2.1 - Preliminary operations....10

3.2.2 - E-Maj extension decompression....10

3.2.3 - E-Maj components installation....11

3.2.4 - Changes in postgresql.conf configuration file....11

3.3 - Set-up the E-Maj access policy....12

3.3.1 - E-Maj roles....12

3.3.2 - Giving E-Maj rights....12

3.3.3 - Giving rights on application tables and objects....13

3.3.4 - Synthesis....13

3.4 - Main functions....14

3.4.1 - Operations chain....14

3.4.2 - Define tables groups....15

3.4.3 - Create a tables group....16

3.4.4 - Start a tables group....16

3.4.5 - Set an intermediate mark....17

3.4.6 - Rollback a tables group....17
3.4.7 - Stop a tables group....18
3.4.8 - Drop a tables group....19
3.4.9 - Change a tables group content....19
3.4.10 - Change the structure of an application table....19
3.5 - Secondary functions....21
3.5.1 - Simultaneous rollback and stop of a tables group....21
3.5.2 - Reset log tables of a group....21
3.5.3 - Rename a mark....22
3.5.4 - Delete a mark....22
3.5.5 - Get statistics....22
3.5.6 - Estimate the rollback duration....24
3.5.7 - Check the consistency of E-Maj objects....25
3.5.8 - Forced suppression of a tables group....26
3.5.9 - Snap tables of a group....26
3.6 - Parallel Rollback....28
3.6.1 - Tables sub-groups....28
3.6.2 - Prerequisites....28
3.6.3 - Syntax....29
4 - MISCELLANEOUS....30
4.1 - Parameters....30
4.2 - Traces of operations....32
4.3 - Checks....33
4.4 - The « log_only » mode....33
4.5 - Usage limits....33
4.6 - User's responsability....35
4.6.1 - Defining tables groups content....35
4.6.2 - Call of main functions....35
4.6.3 - Management of application triggers....35
4.7 - phpPgAdmin plugin....36
4.7.1 - General presentation....36
4.7.2 - Using phpPgAdmin plugin....36

1 INTRODUCTION

1.1 DOCUMENT CONTENT

This document is a user's guide for E-Maj PostgreSQL extension.

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

Chapter 3 describes into details how to setup and use E-Maj.

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

1.2 LICENSE

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

1.3 E-MAJ'S OBJECTIVES

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

The main goal of E-Maj is to supply a way to logically restore sets of tables into predefined states, without been obliged to either restore all files of the PostgreSQL instance (cluster) or reload the entire content of the concerned tables.

It brings a good solution to :

- define save points between batch chains on a tables group,
- restore if needed this tables group into a stable state, without stopping the cluster,
- manage several save points during the batch window, each of them being usable at any moment as restore point.

In a production environment, E-Maj may simplify the technical architecture, by offering a smooth and efficient alternative to disk mirroring solutions. It also may reduce the number of PostgreSQL clusters to administer.

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

2 HOW E-MAJ WORKS

2.1 CONCEPTS

E-Maj is built on three main concepts.

2.1.1 Tables Group

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

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

2.1.2 Mark

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

2.1.3 Rollback

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

2.2 ARCHITECTURE

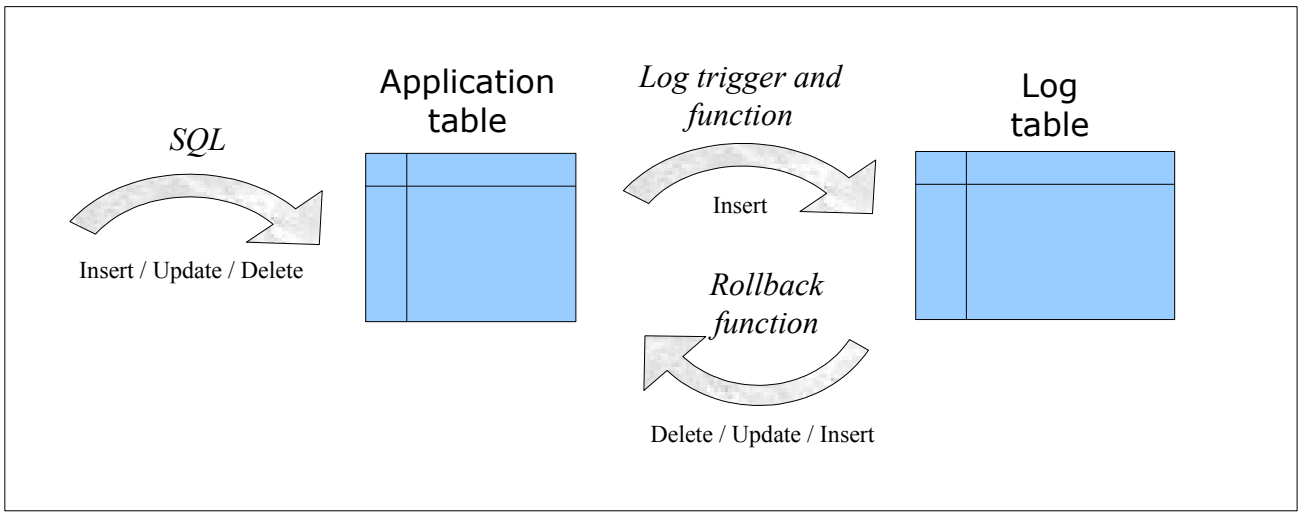
In order to be able to perform a rollback operation without having previously kept a physical image of the PostgreSQL cluster's files, all updates applied on application tables must be recorded, so that they can be cancelled.

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

2.2.1 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*), change (*UPDATE*) or suppression (*DELETE*), record into the log table all data needed to potentially cancel later this elementary action,
- a rollback function, that can cancel all or some recorded updates for the table,
- starting from PostgreSQL 8.4, another trigger that blocks any execution of a *TRUNCATE* SQL verb while log triggers are activated.



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

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

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

- 8 tables,
- 2 types,
- 34 technical functions, 15 of them being directly callable by users,
- 1 specific schema, named *emaj*, that contains all these relational objects as well as the log tables and functions,
- 1 tablespace, *tspemaj*, hosting all these technical and log tables,
- 2 roles acting as groups (NOLOGIN): *emaj_adm* to manage E-Maj components, and *emaj_viewer* to only look at E-Maj components.

Some technical tables, whose structure is interesting to know, are described in the coming chapters.

2.2.2 Norm for E-Maj objects naming

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

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

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

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

- the name of the trigger that blocks *TRUNCATE* verb is:
  <schema.name>_<table.name>_emaj_trunc_trg

- the name of the rollback function is:
  <schema.name>_<table.name>_rlbk_fnct

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

3 HOW TO USE E-MAJ

In this chapter, we will describe how to install and then use E-Maj extension.

But if another version of E-Maj is already installed, it must be previously uninstalled.

3.1 DELETION OF A PREVIOUS VERSION

For this operation, the user must log on the concerned database with psql, as a superuser.

If some tables groups in logging state remain, they must be stopped with the *emaj_stop_group()* function (see § 3.4.7).

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

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

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

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

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

This script first drops the remaining tables groups. Then it drops the *emaj* schema with all contained objects.

If *emaj_adm* and *emaj_viewer* roles are not associated to other role any longer, and do not own rights on other tables, they are dropped. Otherwise a message is displayed, inviting to a manual deletion.

The *tspemaj* tablespace is not dropped by the script. To drop it, the only SQL command to execute is:

```
DROP TABLESPACE tspemaj;
```

If E-Maj is installed into several databases of the same PostgreSQL cluster, the *tspemaj* tablespace is used by all E-Maj objects of all databases. So it could only be dropped after the deletion of all E-Maj components of all the cluster's databases.

3.2 E-MAJ EXTENSION SETUP

For this operation, the user must log on the concerned database as a superuser.

3.2.1 Preliminary operations

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

```
CREATE LANGUAGE plpgsql;
```

The second preliminary operation consists in creating the tablespace named *tspemaj* that will be dedicated to E-Maj, except if this tablespace has already been created by a previous installation.

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

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

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

3.2.2 E-Maj extension decompression

E-Maj is available for download on the pgFoundry.org Internet site, at the url: http://pgfoundry.org/projects/emaj/.

The extension is delivered as a single compressed file. To be usable, this file must be decompressed. Under Unix/Linux, a command like the following can be executed:

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

The following files are now available:

- emaj.sql — psql script to install E-Maj components
- test-emaj.sql — psql test script
- uninstall.sql — psql script to uninstall E-Maj components
- readme.txt — extension's documentation
- releaseNotes.txt — release notes
- Emaj.<version>.en.pdf — presentation of E-Maj extension
- emajParallelRollback.php — php tool for parallel rollback
- test-emaj-2.sql — psql test script for parallel rollbacks

3.2.3 E-Maj components installation

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

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

The script creates the *emaj* schema with its 10 technical tables, its 2 types and its 36 functions. If they were not already present, both *emaj_adm* and *emaj_viewer* roles are also created.

It is possible to check whether the installed E-Maj components work fine, by executing under psql the *test-emaj.sql* script.

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

If no error is encountered, the final message

"--- *test-emaj script successfuly completed* ---"

is displayed.

3.2.4 Changes in postgresql.conf configuration file

Main E-Maj functions set a lock on each table of a processed tables group. If some groups contains a large number of tables, it may be necessary to increase the value of the

max_locks_per_transaction parameter in the *postgresql.conf* configuration file. The default value of this parameter equals 64. It can be increased up to a value at least equal to the number of tables of the largest group.

Furthermore, if the parallel rollback tool may be used (see § 3.6), it will be probably necessary to adjust the *max_prepared_transaction* parameter.

3.3 SET-UP THE E-MAJ ACCESS POLICY

A bad usage of E-Maj can break databases integrity. So it is advisable to only authorise its use to particular users.

3.3.1 E-Maj roles

To use E-Maj, it is possible to log on as superuser. But for safety reason, it is preferable to take advantage of both previously created roles:

- *emaj_adm* is used as administration role ; it can execute all functions [1] and access to all E-Maj tables, with reading and writing rights,
- *emaj_viewer* is used for read only purpose ; it can only execute statistics functions and can only read E-Maj tables.

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

When created, these roles have no connection capability (*NOLOGIN* option and no defined password). 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.

3.3.2 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.

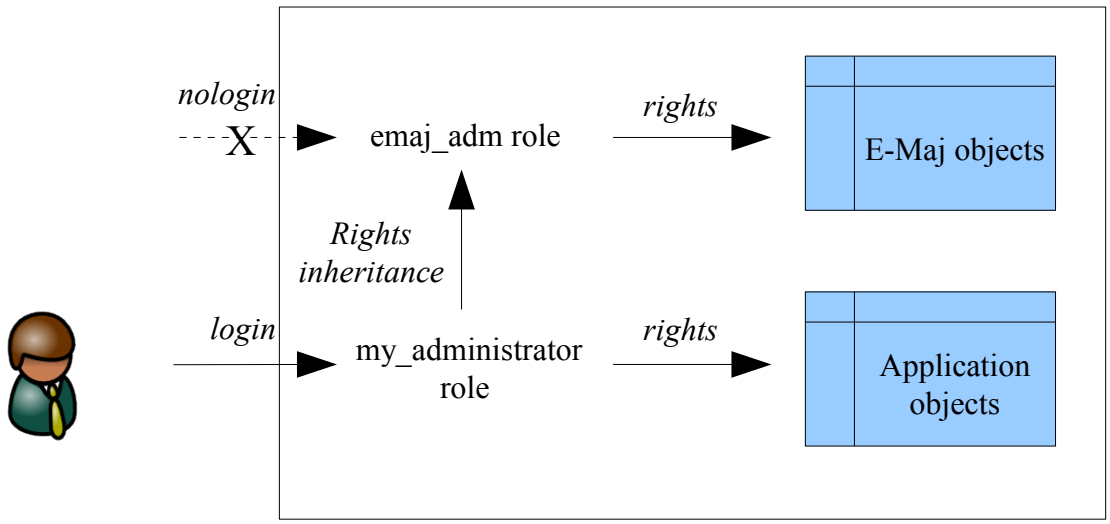
[1] Except for the *emaj_snap_group()* function (see § 3.5.9) which can only be executed by a superuser.

3.3.3 Giving rights on application tables and objects

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

3.3.4 Synthesis

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



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

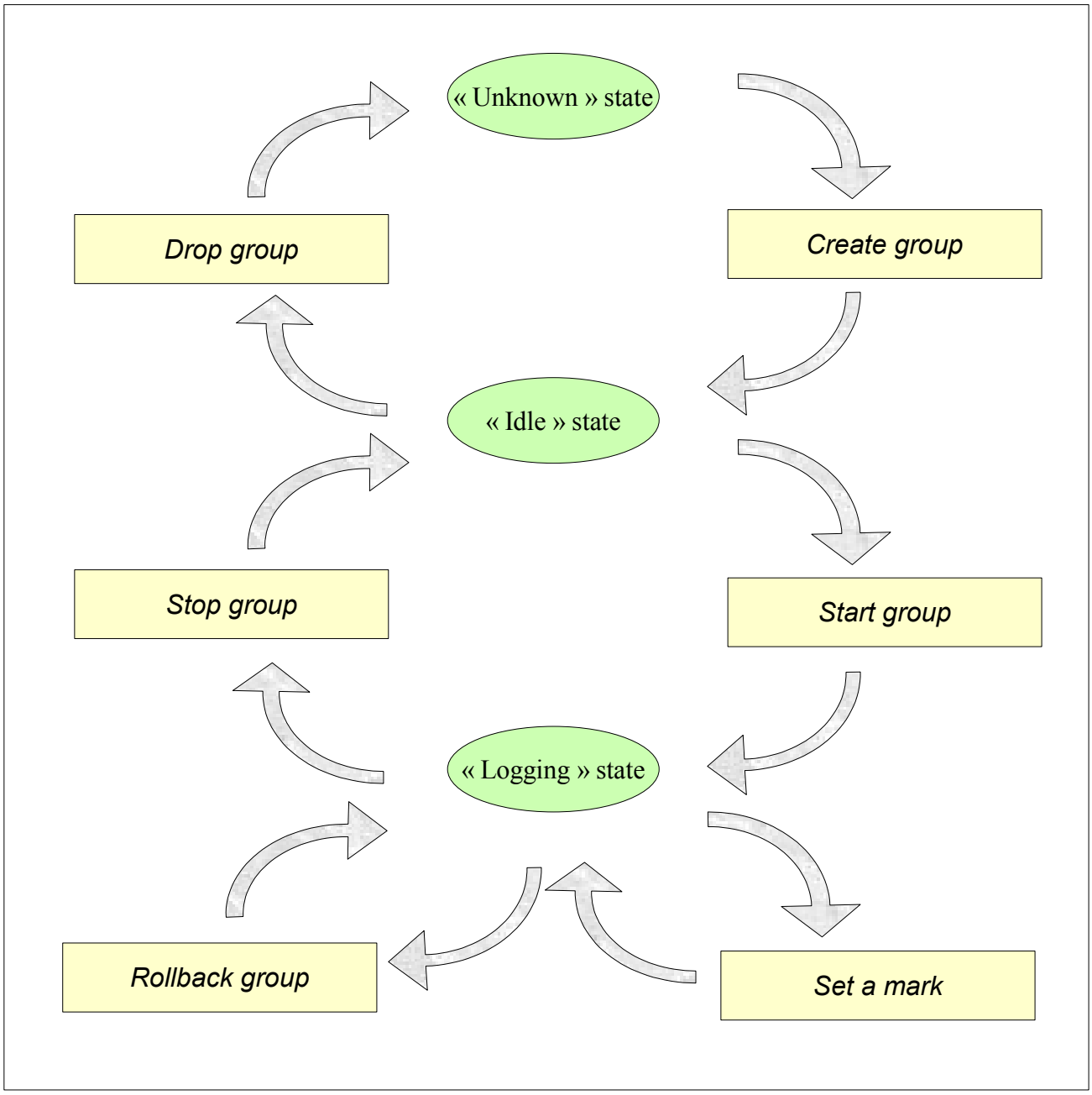
Except when explicitly noticed, the operations presented later can be indifferently executed by a superuser or by a role belonging to the *emaj_adm* group.

3.4 MAIN FUNCTIONS

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

3.4.1 Operations chain

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



3.4.2 Define tables groups

The content of tables groups E-Maj will manage has to be defined by populating the *emaj.emaj_group_def* table. One row has to be inserted into this table for each application table or sequence to include into a tables group. This *emaj.emaj_group_def* table contains 3 columns of type TEXT:

- *grpdef_group* : tables group name
- *grpdef_schema* : name of the schema containing the application table or sequence
- *grpdef_tblseq* : application table or sequence name

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

A table or a sequence of a given schema can be affected to, at most, one tables group. 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.

Except in a particular working mode (see § 4.4), E-Maj can only manage tables for which a primary key has been explicitly defined (*PRIMARY KEY* clause in *CREATE TABLE* or *ALTER TABLE*).

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

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

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

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

3.4.3 Create a tables group

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

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

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

For each table of the group, this function creates the associated log table, both functions for logging and rollbacking as well as the log trigger.

The *emaj_create_group()* function also checks the existence of application triggers on any tables of the group. If a trigger exists on a table of the group, a message is returned, suggesting the user to verify that this trigger does not update any tables that would not belong to the group.

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

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

A tables group can be suppressed by the *emaj_drop_group()* function (see § 3.4.8).

3.4.4 Start a tables group

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

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

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

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

To be sure that no transaction implying any table of the group is currently running, the *emaj_start_group()* function explicitly set an *EXCLUSIVE* lock on each table of the group. If

transactions accessing these tables are running, this can lead to *deadlock*. In such a case, the function may need to be rerun.

Before activating log triggers that will log updates into log tables, all log tables are emptied.

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

3.4.5 Set an intermediate mark

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

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

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

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

The *emaj_set_mark_group()* function records the identity of the new mark, with the state of the application sequences belonging to the group, as well as the state of the log sequences associated to each table of the group.

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

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

3.4.6 Rollback a tables group

If it is necessary to reset tables and sequences of a group in the state they were when a mark was set, the following SQL statement needs to be executed:

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

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

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

To be sure that no transaction implying any table of the group is currently running, and that no other transaction will be able to access any table during the rollback operation, the *emaj_rollback_group()* function explicitly set an *EXCLUSIVE* lock on each table of the group. If transactions accessing these tables are running, this can lead to *deadlock*. In such a case, the function may need to be rerun.

When the rollback operation is completed, are deleted:

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

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

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

3.4.7 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>');
```

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

Stopping a tables group simply deactivate log triggers of application tables of the group. The locks set can lead to *deadlock*. In such a case, the function may need to be rerun.

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

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

When a group is stopped, its state becomes « *IDLE* » again.

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

3.4.8 Drop a tables group

To drop a tables group previously created by the *emaj_create_group()* function, this group must be already in idle state. If it is not the case, the *emaj_stop_group()* function has to be used (see § 3.4.7).

Then, just execute the SQL command:

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

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

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

The locks set by this operation can lead to *deadlock*. In such a case, the function may need to be rerun.

3.4.9 Change a tables group content

The content of a tables group may vary. It may be necessary to add new tables or sequences into a group, or suppress some others from a group. In that case, the following operations must be chained:

- stop the group if it is in logging state, using the *emaj_stop_group()* function,
- drop the group with the *emaj_drop_group()* function,
- adapt the content of the *emaj_group_def* table to reflect the desired change in the group's content,
- recreate the group with the *emaj_create_group()* function.

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

3.4.10 Change the structure of an application table

If the structure of an application table belonging to a tables group changes (addition or suppression of a column, change in a column type), it is mandatory to drop the related tables group, using the *emaj_drop_group()* function, and then to recreate it, using the *emaj_create_group()* function. Indeed, these evolutions will have an impact on the associated log table structure.

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

3.5 SECONDARY FUNCTIONS

In addition to the main functions previously presented, E-Maj has some other functions that makes its usage easier.

3.5.1 Simultaneous rollback and stop of a tables group

If the E-Maj administrator wants to rollback a table group and to stop this same group just after, it is possible to group actions performed by both *emaj_rollback_group()* and *emaj_stop_group()* functions into a single operation.

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

As with *emaj_rollback_group()* function, this function returns the number of tables and sequences that have been effectively modified by the rollback operation.

Using the *emaj_rollback_and_stop_group()* function instead of both usual functions saves time. Indeed, the group being stopped, no rollback will be possible after the completion of the function. As a consequence, the *emaj_rollback_and_stop_group()* function do not erase the content of log tables corresponding to the cancelled updates. The time saved will be all the more significant as the rollback cancels a large number of updates.

3.5.2 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.

3.5.3 Rename a mark

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

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

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

The function does not return any data.

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

3.5.4 Delete a mark

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

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

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

The function does not return any data.

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.

3.5.5 Get statistics

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

- *emaj_log_stat_group()* quickly delivers, for each table of a group, the number of updates that have been recorded in the related log tables, either between 2 marks or since a particular mark,

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

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

Both functions return a set of rows, on which it is possible to execute SQL statements.

For instance, it is possible to get full global statistics with this SQL statement:

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

The returned array, whose type is named *emaj.emaj_log_stat_type*, contains the following columns:

- *stat_group* : tables group name (type TEXT),
- *stat_schema* : schema name (type TEXT),
- *stat_table* : table name (type TEXT),
- *stat_rows* : number of updates recorded into the related log table (type BIGINT)

A NULL value supplied as start mark represents the oldest mark.

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

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

It is possible to easily execute more precise requests on these statistics. For instance, once the *test-emaj-2.sql* test script has been executed, it is possible to get the number of database updates by application schema, with a statement like:

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

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

But 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 or NULL>', '<end.mark or NULL>');
```

The returned array, whose type is named *emaj.emaj_detailed_log_stat_type*, contains the following columns:

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

A NULL value supplied as start mark represents the oldest mark.

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

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

It is possible to easily execute more precise requests on these statistics. For instance, once the *test-emaj-2.sql* test script has been executed, it is possible to get the number of updates for a given table, here mytbl1, per SQL verb, using a statement like:

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

3.5.6 Estimate the rollback duration

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

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

The function returns an *INTERVAL* data.

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,
- 4 generic parameters (see § 4.1) 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 4.1. 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.

3.5.7 Check the consistency of E-Maj objects

A function is also available to check the consistency between all E-Maj objects created In the *emaj* schema on one side and all tables and sequences referenced in all created tables groups in the other side. This function can be called with the following SQL statement:

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

The function checks :

- each log table existing in the *emaj* schema corresponds to a table referenced into the *emaj_group* table, the internal table that describes the composition of created groups,
- each log or rollback function that exists in the *emaj* schema corresponds to a table referenced in this same *emaj_group* table.

The function returns a set of rows describing the detected discrepancies. If no error has been detected, the function returns a unique row containing the following message:

'*No error encountered*'

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.

3.5.8 Forced suppression of a tables group

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

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

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

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

3.5.9 Snap tables of a group

In order to test E-Maj extension, and in particular rollback functions, it may be needed to take images of all tables belonging to group to be able to compare them. It is possible to dump tables of a group with:

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

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.

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

Created files name has the following pattern:

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

As the files are internally created using *COPY* commands, the *emaj_snap_group()* function can only be called by a superuser. An *emaj_adm* role cannot execute this function.

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

Files corresponding to tables contain one record per row, with the default format of the *COPY* command. These records are sorted in ascending order of the primary key.

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

Thanks to this function, a simple test of 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.

3.6 PARALLEL ROLLBACK

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

3.6.1 Tables sub-groups

To run a rollback in parallel, E-Maj spreads tables and sequences to process for a tables group into « sub-groups ». Each sub-group is then processed in each own corridor.

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

To built most balanced sub-groups as possible, E-Maj takes into account:

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

3.6.2 Prerequisites

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

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

3.6.3 Syntax

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

emajParallelRollback.php -g <group.name> -m <mark> -s <number.of.sub-groups> [OPTIONS]...

General options:

-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.

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

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

To let the rollback operation work, the tables group must be in logging state.

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

In order to test the *emajParallelRollback.php* command, E-Maj extension supply a test script, *test-emaj-2.sql*. It prepares an environment with a tables group 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 MISCELLANEOUS

4.1 PARAMETERS

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

emaj_param table strusture is the following:

- *param_key* keyword identifying the parameter
- *param_value_text* parameter value, if its type is text (otherwise NULL)
- *param_value_int* parameter value, if its type is integer (otherwise NULL)
- *param_value_boolean* parameter value, if its type is boolean (otherwise NULL)
- *param_value_interval* parameter value, if its type is time interval (otherwise NULL)

Presented in alphabetic order, the existing key values are:

- *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 § 3.5.6).
- *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 § 3.5.6).
- *fixed_table_rollback_duration* (interval) default value = 5 ms ; defines a fixed rollback cost for any table belonging to a group ; can be modified to better represent the performance of the server that hosts the database (see § 3.5.6).
- *fixed_table_with_rollback_duration* (interval) default value = 2,5 ms ; defines an additional fixed rollback cost for any table that effectively has updates to rollback ; can be modified to better represent the performance of the server that hosts the database (see § 3.5.6).
- *history_retention* (interval) default value = 1 mois ; it can be adjusted to change the retention delay of rows in the *emaj_hist* E-Maj history table (see § 4.2),
- *log_only* (boolean) default value = *false* ; can be set to *true* so that log be activated without any rollback capability (see § 4.4),
- *version* (text) value defined at E-Maj initialisation time ; it must not be modified.

Below is an example of SQL statement modifying the retention delay of rows in the history (here, the duration is set to 15 days):

```
UPDATE emaj.emaj_param SET param_value_interval = '15 days'::interval
WHERE param_key = 'history_retention';
```

It is also possible to modify a parameter value using any graphic tool such as PgAdmin or phpPgAdmin.

4.2 TRACES OF OPERATIONS

All operations performed by E-Maj, and that impact in any way a tables group, are traced into a table named *emaj_hist*.

emaj_hist table structure is the following:

- *hist_id* serial number identifying a row in this history table
- *hist_datetime* recording date and time of the row
- *hist_function* function associated to the traced event
- *hist_event* event kind
- *hist_object* object name related to the event (group, table or sequence)
- *hist_wording* additional comments
- *hist_user* role whose action has generated the event

The *hist_function* column can take the following values:

- *EMAJ_INIT* E-Maj initialisation
- *CREATE_GROUP* tables group creation
- *DROP_GROUP* tables group suppression
- *FORCE_DROP_GROUP* tables group forced suppression
- *START_GROUP* tables group start
- *STOP_GROUP* tables group stop
- *LOCK_GROUP* lock set on tables of a group
- *LOCK_SUBGROUP* lock set on tables of a sub-group
- *SET_MARK_GROUP* mark set on a tables group
- *DELETE_MARK_GROUP* mark deletion for a tables group
- *RENAME_MARK_GROUP* mark rename for a tables group
- *ROLLBACK_GROUP* rollback updates for a tables group
- *RESET_GROUP* log tables content reset for a group
- *ROLLBACK_TABLE* rollback updates for one table
- *ROLLBACK_SEQUENCE* rollback one sequence

the *hist_event* column can take the following values:

- *BEGIN*
- *END*
- *MARK DELETED*

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

When a tables group is started (*emaj_start_group()* function), the oldest rows are deleted from *emaj_hist* tables. The retention delay for history rows is defined by a parameter, *history_retention*, stored in the *emaj_param* table (see § 4.1). While its default value is 1 month, this retention delay can be adjusted at any time, with a SQL statement.

4.3 CHECKS

When a function is executed to start a tables group, to set a mark or to rollback a tables group, E-Maj performs a certain number of checks in order to verify the integrity of the tables group to process.

These tables group integrity verifications include:

- check each application sequence or table of the group always exists,
- check each table of the group has its log table, its log and rollback functions and its triggers,
- checks that the log tables structure always reflects the related application tables structure.

4.4 THE « LOG_ONLY » MODE

It may be interesting to partially test how E-Maj works, with tables not having explicit *primary key* defined in PostgreSQL catalogue. To do this, it is possible to switch E-Maj in a « log_only » mode, via a parameter. In this mode, tables updates can be recorded but, because of primary key lack, no rollback is possible.

To swith E-Maj into this mode, just set the value of the '*log_only*' parameter to *TRUE* (see § 4.1).

Note that this parameter impacts the behaviour of all tables groups of the database where E-Maj extension has been installed.

4.5 USAGE LIMITS

The E-Maj extension usage has some limits.

- The minimum required PostgreSQL version is 8.2.
- All tables belonging to a tables group must have an explicit *PRIMARY KEY*.
- For a table declared in a group, the sum of the schema name length and the table name length can not exceed 52 characters.
- The maximum number of rows that a log table can contain is not unlimited. But this limit is very high (4.611.686.018.427.387.903 !).
- The schema named "*emaj*" is created at E-Maj initialisation. If its name should be changed, the *emaj.sql* scripts, as well as test scripts and the *emajParallelRollback.php* command should be adapted consequently.
- The tablespace named "*tspemaj*" and containing all tables from *emaj* schema must be created before to install the extension. If its name should be changed, *emaj.sql* script should be adapted consequently.

- If a *TRUNCATE* SQL verb is executed on an application table belonging to a group, E-Maj is not able to reset this table in a previous state. Indeed, when a *TRUNCATE* is executed, no trigger is executed at each row deletion. Starting from 8.4 PostgreSQL version, a trigger, created by E-Maj, blocks any *TRUNCATE* statement on any table belonging to a tables group in logging state. For older PostgreSQL versions, this detection is not possible.
- If a DDL operation is executed on an application table belonging to a tables group, E-Maj is not able to reset the table in its previous state.

To understand this last point, it may be interesting to understand the consequences of a DDL operation on the way E-Maj works, depending on the kind of executed operation.

- If a new table were created, it would be able to enter into a group's definition until this group be stopped, dropped and then recreated.
- If a table belonging to a group in logging state were dropped, there would be no way for E-Maj to recover it's structure and its content.
- For a table belonging to a tables group in logging state, adding or deleting a column would generate an error at the next *INSERT/UPDATE/DELETE* SQL verb execution.
- For a table belonging to a tables group in logging state, renaming a column would not generate any error at further log recording. But the checks that E-Maj performs would block any attempt to rollback the related group.
- For a table belonging to a tables group in logging state, changing the type of a column would lead to an inconsistency between the application table and the log table. But, depending on the change of data type applied, updates logging could either work or not. Furthermore, data could be corrupted, for instance in case of increased data length not propagated in log tables. Anyway, due to the checks performed by E-Maj, any attempt to rollback the related group would then fail.
- However, it is possible to create, modify or drop indexes, rights or constraints for a table belonging to a tables group in logging state. But or course, cancelling these changes could not be done by E-Maj.

4.6 USER'S RESPONSABILITY

4.6.1 Defining tables groups content

Defining the content of tables group is essential to guarantee the database integrity. It is the E-Maj administrator's responsibility to ensure that all tables updated by a given processing are really included in a single tables group.

4.6.2 Call of main functions

emaj_start_group(), *emaj_set_mark_group()*, *emaj_rollback_group()* and *emaj_rollback_and_stop_group()* functions set explicit locks on tables of the group to be sure that no transactions updating these tables are running at the same time. But it is the user's responsibility to execute these operations “at the right time”, i.e. at moments that really correspond to stable point in the life of these tables.

4.6.3 Management of application triggers

Triggers may have been created on application tables. It is not rare that these triggers performs one or several updates on other tables. In such a case, it is the E-Maj administrator's responsibility to understand the impact of rollback operations on tables concerned by triggers, and if needed to take the appropriate measures.

If the trigger simply adjust the content of the row to insert or update, the logged data will contain the final value of columns. So the rollback would reset the old values without any problem. But may be it would be necessary to deactivate such a trigger during a rollback operation.

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

- if the updated table belong to the same tables group, it would be necessary to deactivate the trigger during a rollback operation, so that E-Maj and only E-Maj performs the updates required by the rollback operation,
- if the updated table does not belong to the same tables group, it is essential to analyse the consequences of a rollback operation, in order to avoid a de-synchronisation between both tables. In such a case, only deactivating the trigger may not be sufficient.

4.7 PHPPGADMIN PLUGIN

A plugin for the phpPgAdmin 5 administration tool is also available.

4.7.1 General presentation

For databases into which E-Maj extension has been installed, and if the user is connected with a role that owns the required rights, e-Maj objects are accessible. It is then possible to:

- see the list of tables groups and perform any possible action, depending on groups state (start, stop, set or remove a mark, rollback),
- see the list of the marks that have been set for a group, and perform any possible action (mark deletion or renaming),
- get all kind of statistics about log tables content.

4.7.2 Using phpPgAdmin plugin

On figure 1 below, once being connected with a proper role to a database where E-Maj extension has been installed, a new icon is easily visible in the horizontal database icons tab. Obviously, the *emaj* schema appears in schemas list.

In the tree on the left, a new E-Maj object also appears. Opening it shows the list of created tables groups.

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

By clicking on one of the E-Maj icons, the user can see the list of tables group created in this database. Figure 2 shows the myAppll1 group created by *test-emaj-2.sql* script.

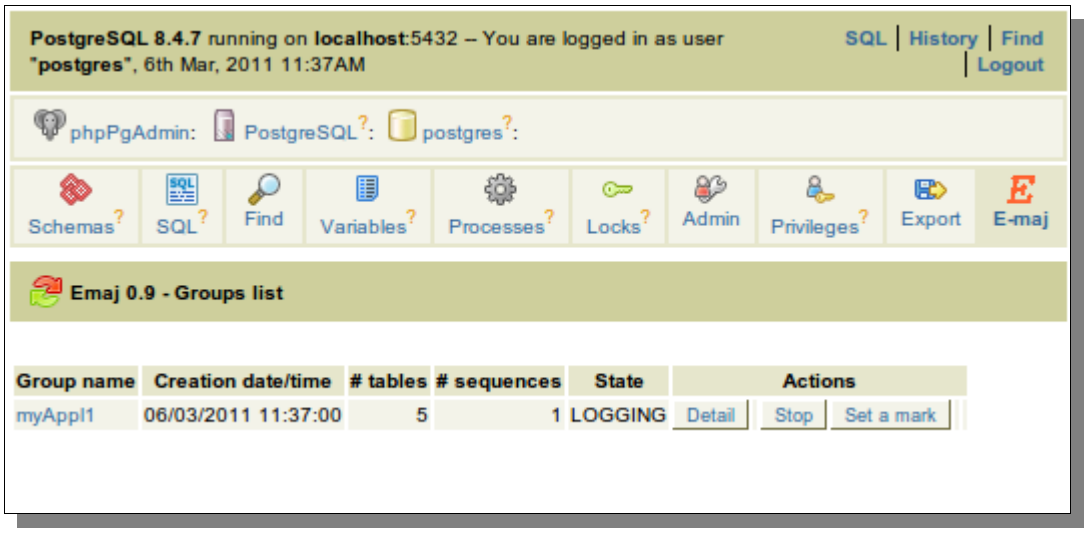


Figure 2 – List of the tables groups created in the database

For each tables group, are displayed the following attributes:

- its creation date and time,
- the number of application tables and sequences it contains,
- its state (« LOGGING » or « IDLE »).

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

By clicking on a « Detail » button, it is possible to get more information about a particular tables group.

Figure 3 shows the detail information for a tables group.

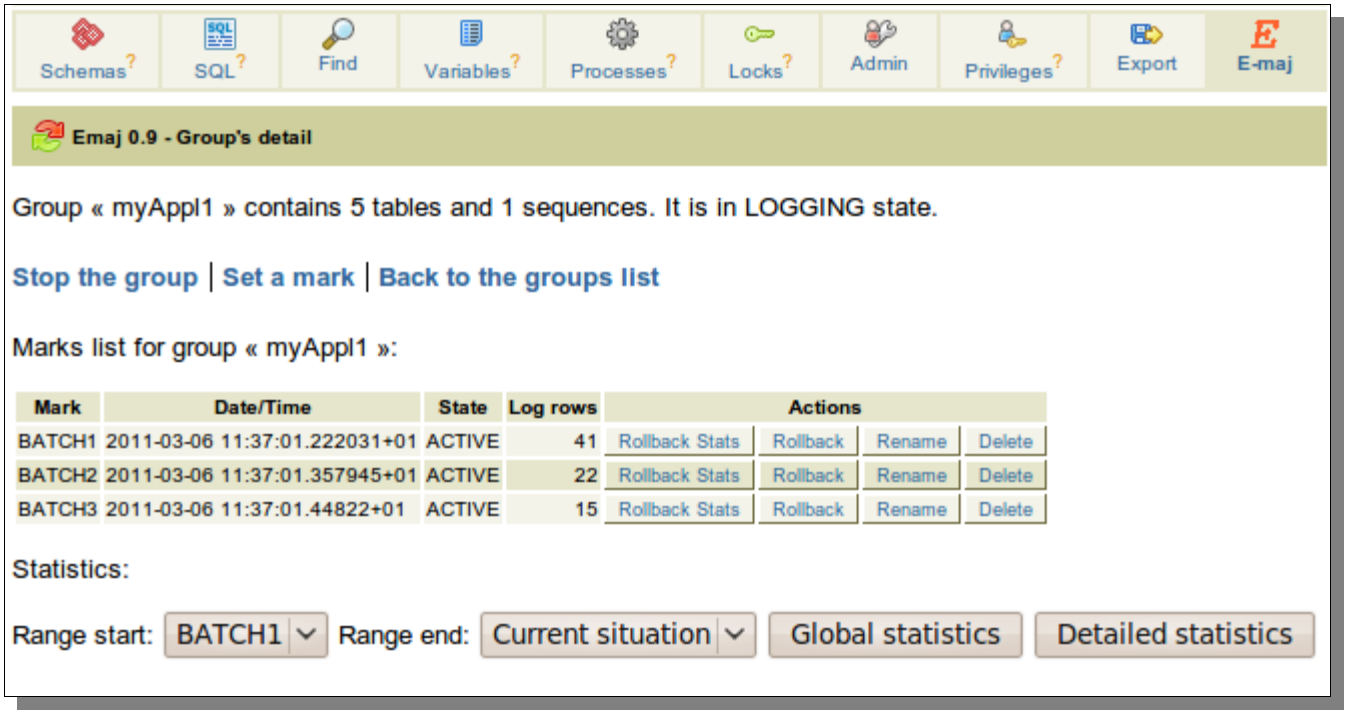


Figure 3 – *Detail of a tables group*

A first line repeats information already displayed on the groups list: number of tables and sequences, and state.
A second line presents the possible actions to perform on this group.

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

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

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

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

Figure 4 partly shows a page obtained by clicking on the « Rollback Stat » button of a mark.

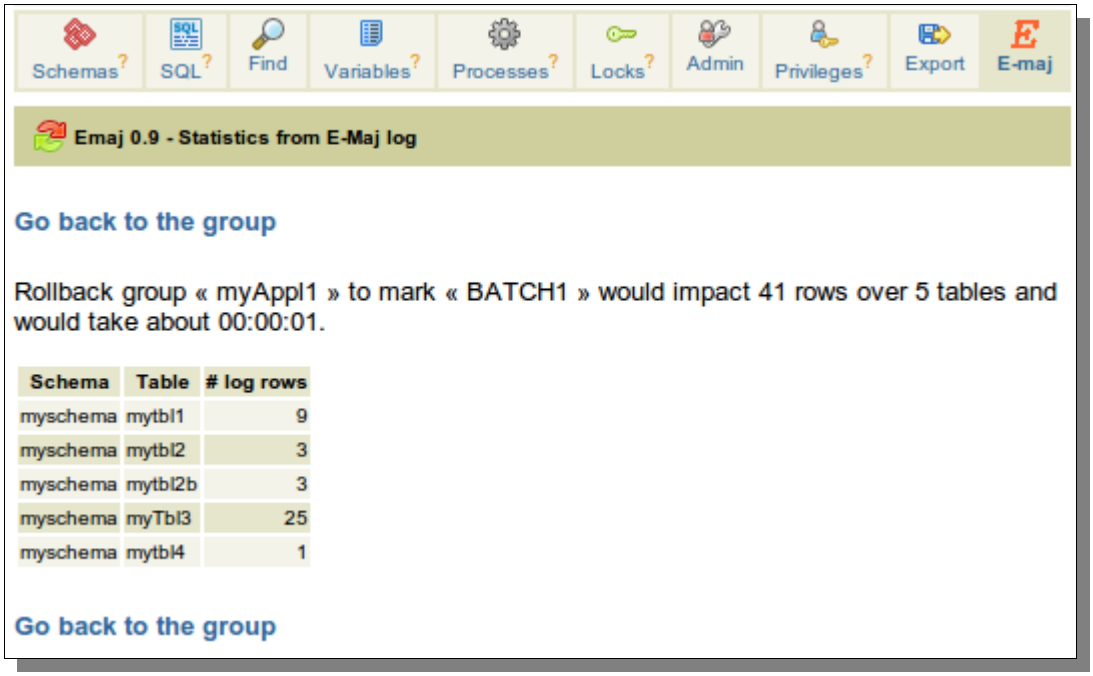


Figure 4 – Before rollback statistics

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

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