E-Maj Documentation

Version 3.4.0

Philippe Beaudoin

juil. 26, 2020

Aperçu:

1 Introduction 1

2 Concepts 3

3 Architecture 5

4 Démarrage rapide 9

5 Installation du logiciel E-Maj 11

6 Création de l'extension E-Maj dans une base de données 13

7 Mise à jour d'une version E-Maj existante 17

8 Désinstallation d'E-Maj d'une base de données 23

9 Changement de version de PostgreSQL 25

10 Mise en place de la politique d'accès à E-Maj 27

11 Création et suppression des groupes de tables 29

12 Fonctions principales 35

13 Modification des groupes de tables 43

14 Autres fonctions de gestion des groupes 53

15 Fonctions de gestion des marques 59

16 Fonctions statistiques 63

17 Fonctions d'extraction de données 67

18 Autres fonctions 71

19 Fonctions multi-groupes 77

20 Client de rollback avec parallélisme 79

21 Client de suivi des rollbacks 83

22 Paramétrage 85

23 Structure des tables de log 87

24 Fiabilisation du fonctionnement 89

25 Traçabilité des opérations 91

26 Impacts sur l'administration de l'instance et de la base de données 95

27 Sensibilité aux changements de date et heure système 101

28 Performances 103

29 Limites d'utilisation 105

30 Responsabilités de l'utilisateur 107

31 Présentation générale d'Emaj_web 109

32 Installation du client Emaj_web 111

33 Utilisation d'Emaj_web 113

34 Contribuer au développement d'E-Maj 125

35 Liste des fonctions E-Maj 133

36 Contenu de la distribution E-Maj 141

37 Matrice de compatibilité des versions PostgreSQL et E-Maj 143

38 Index et tables 145

CHAPITRE 1

Introduction

1.1 Licence

Cette extension et toute la documentation qui l'accompagne sont distribuées sous licence GPL (GNU - General Public License).

1.2 Objectifs d'E-Maj

E-Maj est l'acronyme français de « *Enregistrement des Mises A Jour* ».

Il répond à deux objectifs principaux :

- E-Maj peut servir à **tracer les mises à jours** effectuées sur le contenu de tables par des traitements. La consultation de ces mises à jour enregistrées offre ainsi une réponse aux besoins d'' « audit des mises à jour ».
- Utilisant ces mises à jour enregistrées, E-Maj est capable de **remettre le contenu d'un ensemble de tables dans un état prédéfini**, sans restauration physique de l'ensemble des fichiers d'une instance (cluster) PostgreSQL, ni rechargement complet de l'ensemble des tables concernées.

En d'autres termes, E-Maj est une extension PostgreSQL qui permet un enregistrement des mises à jour avec une fine granularité et un voyage dans le temps de sous-ensembles de bases de données.

Il constitue ainsi une bonne solution pour :

- positionner à des moments précis des points de sauvegarde sur un ensemble de tables,
- restaurer si nécessaire cet ensemble de tables dans un état stable, sans arrêt de l'instance,
- gérer plusieurs points de sauvegarde, chacun d'eux étant utilisable à tout moment comme point de restauration.

Ainsi, dans un **environnement de production**, E-Maj peut permettre de simplifier l'architecture technique utilisée, en offrant une alternative souple et efficace à des sauvegardes intermédiaires longues (pg_dump) et/ou coûteuses en espace disque (disques miroirs). E-Maj peut également apporter une aide au débogage, en offrant la possibilité d'analyser de façon précise les mises à jour effectuées par un traitement suspect sur les tables applicatives.

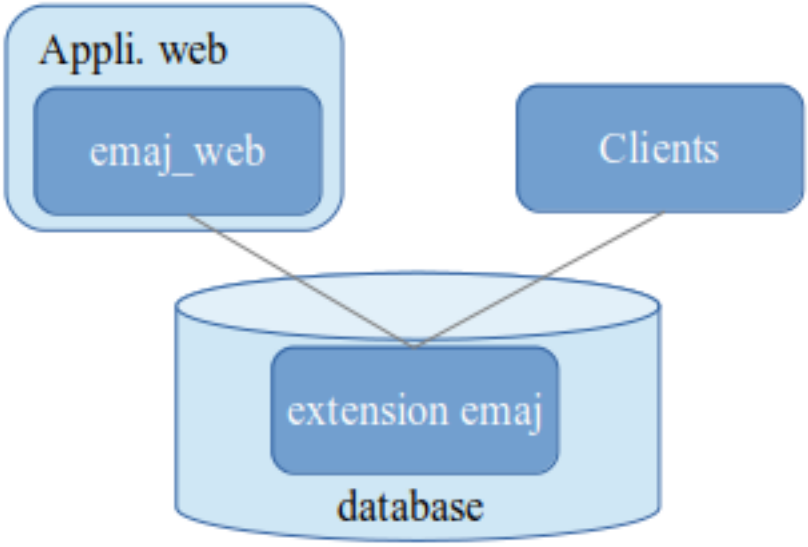
Dans un **environnement de test**, E-Maj permet également d'apporter de la souplesse dans les opérations. Il est ainsi possible de repositionner très facilement les bases de données dans des états stables prédéfinis afin de répéter autant de fois que nécessaire des tests de traitement.

1.3 Principaux composants

E-Maj regroupe en fait plusieurs composants :

- un objet PostgreSQL **extension** créé dans chaque base de données, nommée emaj et contenant quelques tables, fonctions, séquences, ...
- un ensemble de **clients externes** appelables en ligne de commande,
- une **interface graphique web**, **Emaj_web**.

Les clients externes et l'interface graphique font appel aux fonctions de l'extension emaj.



Tous ces composants sont décrits dans cette documentation.

CHAPITRE 2

Concepts

E-Maj s'appuie sur trois concepts principaux.

2.1 Groupe de tables

Le « **groupe de tables** » (*tables group*) représente un ensemble de **tables applicatives** qui vivent au même rythme, c'est-à-dire dont, en cas de besoin, le contenu doit être restauré comme un tout. Il s'agit typiquement de toutes les tables d'une base de données mises à jour par un ou plusieurs traitements. Chaque groupe de tables est défini par un nom unique pour la base de données concernée. Par extension, un groupe de tables peut également contenir des **partitions** de tables partitionnées et des **séquences**. Les tables (incluant les partitions) et séquences qui constituent un groupe peuvent appartenir à des schémas différents de la base de données.

A un instant donné, un groupe de tables est soit dans un état « **actif** » (*LOGGING*), soit dans un état « **inactif** » (*IDLE*). L'état actif signifie que les mises à jour apportées aux tables du groupe sont enregistrées.

Un groupe de tables est soit de type « **ROLLBACKABLE** » (cas standard), soit de type « **AUDIT_ONLY** ». Dans ce second cas, il n'est pas possible de procéder à un rollback du groupe. En revanche, cela permet d'enregistrer à des fins d'observation les mises à jour du contenu de tables ne possédant pas de clé primaire ou de tables de type *UNLOGGED* ou *WITH OIDS*.

2.2 Marque

Une « **marque** » (*mark*) est un point particulier dans la vie d'un groupe de tables correspondant à un état stable des tables et séquences du groupe. Elle est positionnée de manière explicite au travers d'une intervention de l'utilisateur. Une marque est définie par un nom unique au sein du groupe de tables.

2.3 Rollback

L'opération de « **rollback** » consiste à remettre toutes les tables et séquences d'un groupe dans l'état dans lequel elles se trouvaient lors de la pose d'une marque.

Il existe deux types de rollback :

— avec un « **rollback non tracé** » (*unlogged rollback*), aucune trace des mises à jour annulées par l'opération de rollback n'est conservée : il n'y a pas de mémoire de ce qui a été effacé,

— au contraire, dans une opération de « **rollback tracé** » (*logged rollback*), les annulations de mises à jour sont elles-mêmes tracées dans les tables de log, offrant ainsi la possibilité d'annuler l'opération de rollback elle-même.

Notez que cette notion de « *Rollback E-Maj* » est distincte de celle du « *Rollback de transaction* » géré par PostgreSQL.

CHAPITRE 3

Architecture

Pour mener à bien l'opération de rollback sans avoir conservé au préalable une image physique des fichiers de l'instance PostgreSQL, il faut pouvoir enregistrer les mises à jour effectuées sur les tables applicatives de manière à pouvoir les annuler.

Avec E-Maj, cela prend la forme suivante.

3.1 Les requêtes SQL tracées

Les opérations de mises à jour enregistrées concernent les verbes SQL suivants :

- insertions de lignes :
  - INSERT élémentaires (INSERT … VALUES) ou ensemblistes (INSERT … SELECT)
  - COPY … FROM
- mises à jour de lignes :
  - UPDATE
- suppression de lignes :
  - DELETE
- vidage de table :
  - TRUNCATE

Pour les requêtes qui traitent plusieurs lignes, chaque création, modification ou suppression est enregistrée individuellement. Ainsi par exemple, une requête *DELETE FROM <table>* portant sur une table d'1 million de lignes générera l'enregistrement d'1 million de suppressions de ligne.

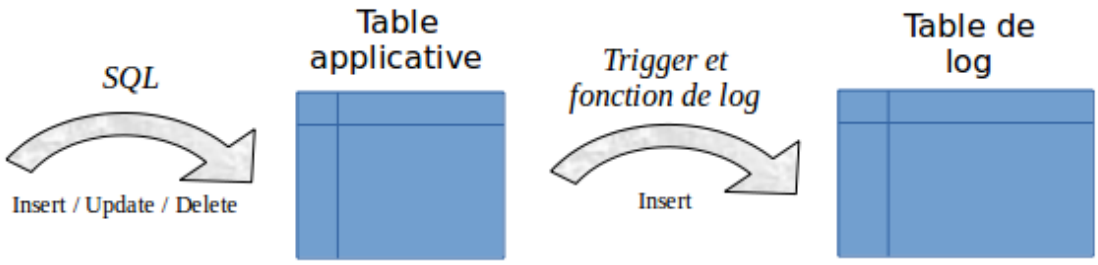
Lors de l'exécution d'un verbe SQL *TRUNCATE*, l'ensemble du contenu de la table est enregistré juste avant son effacement effectif.

3.2 Les objects créés

Pour chaque table applicative sont créés :

- une **table de log** dédiée, qui contient les données correspondant aux mises à jour effectuées,

— un **trigger** et une **fonction** spécifique, permettant, lors de chaque création (*INSERT*, *COPY*), mise à jour (*UPDATE*) ou suppression (*DELETE*) de ligne, d'enregistrer dans la table de log toutes les informations nécessaires à l'annulation ultérieure de l'action élémentaire,
— un autre **trigger** permettant de tracer l'exécution des verbes SQL *TRUNCATE*,
— une **séquence** qui permet de dénombrer très rapidement le nombre de mises à jour enregistrées dans les tables de log entre 2 marques.



Une **table de log** a la même structure que la table applicative correspondante. Elle comprend néanmoins quelques *colonnes techniques supplémentaires*.

Pour le bon fonctionnement d'E-Maj, un certain nombre d'**objets techniques** sont également créés à l'installation de cette extension :

— 18 tables,
— 8 types composites et 3 énumérations,
— 1 vue,
— 2 triggers,
— plus de 150 fonctions, dont plus de 70 directement *appelables par les utilisateurs*,
— 1 séquence, nommée *emaj_global_seq*, permettant d'associer à chaque mise à jour enregistrée dans une table de log quelconque de la base de données un identifiant unique de valeur croissante au fil du temps,
— 1 schéma spécifique, nommé *emaj*, qui contient tous ces objets,
— 2 rôles de type groupe (sans possibilité de connexion) : *emaj_adm* pour administrer les composants E-Maj, et *emaj_viewer* pour uniquement consulter les composants E-Maj,
— 3 triggers sur événement.

Quelques tables techniques dont il peut être utile de connaître la structure sont décrites en détail : *emaj_group_def*, *emaj_param* et *emaj_hist*.

3.3 Les schémas créés

Tous les objets techniques créés lors de l'installation de l'extension sont localisés dans le schéma *emaj*. Seule la fonction associée au trigger sur événement *emaj_protection_trg* appartient au schéma *public*.

Les objets associés aux tables applicatives sont créés dans des schémas nommés *emaj_<schéma>*, où <schéma> est le nom de schéma des tables applicatives.

La création et la suppression de ces **schémas de log** sont gérées exclusivement par les fonctions E-Maj. Ils ne devront PAS contenir d'objets autres que ceux créés par E-Maj.

3.4 Norme de nommage des objets E-Maj

Pour une table applicative, le nom des objets de log est préfixé par le nom de la table. Ainsi :

— **le nom de la table de log est :** <nom.de.la.table>_log

- **le nom de la fonction de log est :** <nom.de.la.table>_log_fnct
- **le nom de la séquence associée à la table de log est :** <nom.de.la.table>_log_seq

Pour les tables applicatives dont le nom est très long (plus de 50 caractères), le préfixe utilisé pour construire le nom des objets de log est généré pour respecter les règles de nommage de PostgreSQL et éviter tout doublon.

Le nom des tables de log peut porter un suffixe de type « _1 », « _2 », etc. Il s'agit alors d'anciennes tables de logs qui ont été renommées lors d'une modification de groupe de tables.

Le nom des autres **fonctions** E-Maj est aussi normalisé :

- les fonctions dont les noms commencent par *emaj_* sont appelables par les utilisateurs,
- les fonctions dont les noms commencent par _ sont des fonctions internes qui ne doivent pas être appelées directement.

Les **triggers** créés sur les tables applicatives portent tous le même nom :

- *emaj_log_trg* pour les triggers de log,
- *emaj_trunc_trg* pour les triggers de contrôle des verbes *TRUNCATE*.

Le nom des **triggers sur événements** commence par *emaj_* et se termine par *_trg*.

3.5 Les tablespaces utilisés

Lors de l'installation de l'extension, les tables techniques E-Maj sont stockées dans le tablespace par défaut, positionné au niveau de l'instance ou de la database ou explicitement défini pour la session courante.

Il en est de même pour les tables de log et leur index. Mais au travers du *paramétrage des groupes de tables*, il est aussi possible de créer les tables de log et leur index dans des tablespaces spécifiques.

CHAPITRE 4

Démarrage rapide

L'installation d'E-Maj est présentée plus loin en détail. Mais les quelques commandes suivantes permettent de procéder rapidement à une installation et une utilisation sous Linux.

4.1 Installation du logiciel

Pour installer le logiciel E-Maj, connectez-vous à votre compte postgres (ou un autre), téléchargez E-Maj depuis PGXN (https://pgxn.org/dist/e-maj/) et tapez :

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

cd e-maj-<version>/

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

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

Pour plus de détails, ou en cas de problème, allez *ici*.

4.2 Installation de l'extension

Pour installer l'extension emaj dans une base de données, connectez-vous à la base de données cible, en utilisant un rôle super-utilisateur et passez les commandes :

```
create extension emaj cascade;

grant emaj_adm to <role>;
```

Pour les versions de PostgreSQL antérieures à la version 9.6, se référer à ce *chapitre*.

La dernière requête permet de donner les droits d'administration E-Maj à un rôle particulier, Par la suite, vous pourrez utiliser ce rôle pour exécuter les opérations E-Maj sans être connecté comme super-utilisateur.

4.3 Utilisation de l'extension

Vous pouvez maintenant vous connecter à la base de données avec le rôle qui possède les droits d'administration E-Maj.

Il faut tout d'abord créer un groupe de tables vide (ici de type *ROLLBACKABLE*) :

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

On peut alors le garnir de tables et séquences avec des requêtes du type :

```
SELECT emaj.emaj_assign_table('mon_schéma', 'ma_table', 'mon_groupe');
```

pour ajouter une table, ou encore, pour ajouter toutes les tables et les séquences d'un schéma :

```
SELECT emaj.emaj_assign_tables('mon_schéma', '.*', '', 'mon_groupe');

SELECT emaj.emaj_assign_sequences('mon_schéma', '.*', '', 'mon_groupe');
```

Notez que seules les tables ayant une clé primaire sont affectées à un groupe de tables *ROLLBACKABLE*.

Ensuite, l'enchaînement typique de commandes

```
SELECT emaj.emaj_start_group('mon_groupe', 'Mark-1');

[INSERT/UPDATE/DELETE sur les tables du groupe]

SELECT emaj.emaj_set_mark_group('mon_groupe','Mark-2');

[INSERT/UPDATE/DELETE sur les tables du groupe]

SELECT emaj.emaj_set_mark_group('mon_groupe','Mark-3');

[INSERT/UPDATE/DELETE sur les tables du groupe]

SELECT emaj.emaj_rollback_group('mon_groupe','Mark-2');

SELECT emaj.emaj_stop_group('mon_groupe');

SELECT emaj.emaj_drop_group('mon_groupe');
```

permet de « démarrer » le groupe de tables, d'enregistrer les mises à jour en posant des marques intermédiaires, de revenir à l'une d'elles, d'arrêter l'enregistrement et enfin de supprimer le groupe.

Pour plus de précisions, les principales fonctions sont décrites *ici*.

En complément, le client *Emaj_web* peut être installé et utilisé.

CHAPITRE 5

Installation du logiciel E-Maj

5.1 Source de téléchargement

E-Maj est disponible en téléchargement sur le site Internet **PGXN** (https://pgxn.org/dist/e-maj/).

E-Maj et ses compléments sont également disponibles sur le site Internet **github.org** :
- Composants sources (https://github.com/dalibo/emaj)
- Documentation (https://github.com/beaud76/emaj_doc)
- Interface graphique Emaj_web (https://github.com/dalibo/emaj_web)

Pour mémoire, un ancien dépôt (https://github.com/beaud76/emaj_ppa_plugin) contient un plugin pour *phpPgAdmin*. Il n'est plus maintenu à partir de la version 3.0 d'E-Maj.

5.2 Installation standard sur Linux

Téléchargez la dernière version d'E-Maj par un moyen à votre convenance. Si le *client pgxn* est installé, on peut simplement exécuter la commande :

```
pgxn download E-Maj
```

Puis décompressez l'archive avec les commandes suivantes :

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

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

Identifiez la localisation précise du répertoire *SHAREDIR*. Selon l'installation de PostgreSQL, la commande *pg_config –sharedir* peut retourner directement le nom du répertoire. Sinon, rechercher les localisations typiques, telles que :
- */usr/share/postgresql/<pg_version>* pour Debian ou Ubuntu
- */usr/pgsql-<pg_version>/share* pour RedHat ou CentOS

Copiez quelques fichiers vers le répertoire des extensions de la version de PostgreSQL souhaitée. En tant que super-utilisateur ou en préfixant les commandes avec sudo, taper

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

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

La dernière version d'E-Maj est maintenant installée et référencée par PostgreSQL. Le répertoire e-maj-<version> contient l'arborescence *décrite ici*.

5.3 Installation minimale sur Linux

Sur certains environnements (cloud de type DBaaS par exemple), il n'est pas possible d'ajouter des extensions dans le répertoire *SHAREDIR*. Pour ces cas de figure, on peut procéder à une installation minimale.

Téléchargez la dernière version d'E-Maj par un moyen à votre convenance, et décompressez la.

Par exemple, si le client pgxn est installé, exécutez les commandes :

```
pgxn download E-Maj

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

Le répertoire e-maj-<version> généré contient l'arborescence *décrite ici*.

La *création de l'extension* est alors un peu différente.

5.4 Installation sous Windows

Pour installer E-Maj sous Windows, il faut :

- Télécharger l'extension depuis le site pgxn.org,
- Extraire l'arborescence du fichier zip reçu,
- En copier les fichiers *emaj.control* et *sql/emaj–** dans le dossier *share\extension* du dossier d'installation de la version de PostgreSQL (typiquement *c :\Program_Files\PostgreSQL\<version_postgres>*).

5.5 Localisation alternative des scripts SQL d'installation

Le fichier *emaj.control*, positionné dans le répertoire *SHAREDIR/extension* de la version de PostgreSQL, peut contenir une directive indiquant à PostgreSQL le répertoire dans lequel sont localisés les scripts SQL d'installation ou d'upgrade.

Il est donc possible de ne mettre dans ce répertoire *SHAREDIR/extension* que le seul fichier *emaj.control* en créant ce pointeur vers le répertoire de scripts. Pour ce faire, il faut :

- Copier le fichier *emaj.control* fourni dans le répertoire racine de la version décompressée vers le répertoire *SHAREDIR/extension*,
- Adapter la directive *directory* du fichier *emaj.control* pour spécifier le répertoire sql contenant les scripts d'installation d'E-Maj.

CHAPITRE 6

Création de l'extension E-Maj dans une base de données

Si une extension existe déjà dans la base de données, mais dans une ancienne version d'E-Maj, il faut la *mettre à jour*.

La façon standard d'installer E-Maj consiste à créer un objet *EXTENSION* (au sens de PostgreSQL). Pour ce faire, l'utilisateur doit être connecté à la base de données en tant que super-utilisateur.

Pour les environnements pour lesquels il n'est pas possible de procéder ainsi (cas des *installations minimales*), on peut exécuter un script *psql*.

6.1 Opération préliminaire facultative

Les tables techniques de l'extension sont créés dans le tablespace par défaut. Si l'administrateur E-Maj veut stocker les tables techniques dans un tablespace dédié, il peut le créer si besoin et le définir comme tablespace par défaut pour la session :

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

6.2 Création standard de l'EXTENSION emaj

6.2.1 Version PostgreSQL 9.6 et suivantes

L'extension E-Maj peut maintenant être créée dans la base de données, en exécutant la commande SQL :

```
CREATE EXTENSION emaj CASCADE;
```

Le script commence par vérifier que la version de PostgreSQL est supérieure ou égale à la version 9.5, que le rôle qui exécute le script a bien l'attribut *superuser*.

Le script crée alors le schéma *emaj* avec ses tables techniques, ses types et ses fonctions.

Prudence : Le schéma *emaj* ne doit contenir que des objets liés à E-Maj.

S'ils n'existent pas déjà, les 2 rôles *emaj_adm* et *emaj_viewer* sont également créés.

Enfin, le script d'installation examine la configuration de l'instance. Le cas échéant, il affiche un message concernant le paramètre *-max_prepared_statements*.

6.2.2 Version PostgreSQL 9.5

Pour les versions de PostgreSQL antérieures à la version 9.6, la clause *CASCADE* n'existe pas. Les extensions prérequises doivent donc être créées explicitement si elles n'existent pas déjà dans la base de données

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

6.3 Création de l'extension par script

Lorsque la création de l'objet *EXTENSION* emaj n'est pas permise, il est possible de créer tous les composants nécessaires par un script psql

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

où <répertoire_emaj> est le répertoire issu de l'*installation du logiciel* et <version> la version courante d'E-Maj.

Prudence : Il n'est pas indispensable d'avoir de droit super-utilisateur pour exécuter ce script d'installation. Mais si ce n'est pas le cas, le rôle utilisé devra disposer des droits nécessaires pour créer les triggers sur les tables applicatives des futurs groupes de tables.

6.4 Adaptation du fichier de configuration postgresql.conf

Les fonctions principales d'E-Maj posent un verrou sur chacune des tables du groupe traité. Si le nombre de tables constituant le groupe est élevé, il peut s'avérer nécessaire d'augmenter la valeur du paramètre **max_locks_per_transaction** dans le fichier de configuration *postgresql.conf*. Ce paramètre entre dans le dimensionnement de la table en mémoire qui gère les verrous de l'instance. Sa valeur par défaut est de 64. On peut le porter à une valeur supérieure si une opération E-Maj échoue en retournant un message d'erreur indiquant clairement que toutes les entrées de la table des verrous sont utilisées.

De plus, si l'utilisation de l'outil de *rollback en parallèle* est envisagée, il sera probablement nécessaire d'ajuster le paramètre **max_prepared_transaction**.

6.5 Paramétrage d'E-Maj

Un certain nombre de paramètres influence le fonctionnement d'E-Maj. Le détail des paramètres est présenté *ici*.

Cette étape de valorisation des paramètres est optionnelle. Leur valeur par défaut permet à E-Maj de fonctionner correctement.

Néanmoins, si l'administrateur E-Maj souhaite bénéficier du suivi des opérations de rollback, il est nécessaire de créer une ligne dans la table *emaj_param* pour définir la valeur du paramètre **dblink_user_password**.

6.6 Test et démonstration

Il est possible de tester le bon fonctionnement des composants E-Maj installés et d'en découvrir les principales fonctionnalités en exécutant un script de démonstration. Sous *psql*, il suffit d'exécuter le script *emaj_demo.sql* fourni avec l'extension

```
\i <répertoire_emaj>/sql/emaj_demo.sql
```

Si aucune erreur n'est rencontrée, le script affiche ce message final

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

L'examen des messages affichés par l'exécution du script permet de découvrir les principales fonctionnalités de l'extension. Après l'exécution du script, l'environnement de démonstration est laissé en l'état. On peut alors l'examiner et jouer avec. Pour le supprimer, exécuter la fonction de nettoyage qu'il a généré

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

Ceci supprime le schéma *emaj_demo_app_schema* et les deux groupes de tables *emaj demo group 1* et *emaj demo group 2*.

CHAPITRE 7

Mise à jour d'une version E-Maj existante

7.1 Démarche générale

La première étape consiste à *installer la nouvelle version du logiciel E-Maj*. Conserver l'ancien répertoire E-Maj au moins jusqu'à la fin de la mise à jour. Certains fichiers pourront être utiles.

Il faut également vérifier si des *opérations préliminaires* doivent être exécutées (extensions pré-requises, *tablespace* par défaut).

Ensuite, la procédure de mise à jour de la version d'E-Maj installée dans une base de données dépend de cette version installée.

Pour les versions d'E-Maj antérieures à 0.11.0, il n'existe pas de procédure spécifique de mise à jour. On procédera donc à une simple désinstallation puis réinstallation de l'extension. Cette démarche peut d'ailleurs être utilisée quelle que soit la version d'E-Maj installée. Elle présente néanmoins l'inconvénient de devoir supprimer tous les logs enregistrés, perdant ainsi toute capacité ultérieure de rollback ou d'examen des mises à jour enregistrées.

Pour les versions d'E-Maj installées 0.11.0 et suivantes, il est possible de procéder à une mise à jour sans désinstallation. Suivant la situation, il faut procéder en une ou en plusieurs étapes.

Prudence : A partir de la version 2.2.0, E-Maj ne supporte plus les versions de PostgreSQL antérieures à 9.2. A partir de la version 3.0.0, E-Maj ne supporte plus les versions de PostgreSQL antérieures à 9.5. Si une version antérieure de PostgreSQL est utilisée, il faut la faire évoluer avant de migrer E-Maj dans une version supérieure.

7.2 Mise à jour par désinstallation puis réinstallation

Pour ce type de mise à jour, il n'est pas nécessaire d'utiliser la procédure de *désinstallation complète*. Les tablespaces et les rôles peuvent notamment rester en l'état. En revanche, il peut s'avérer judicieux de sauvegarder quelques données utiles. C'est pourquoi, la démarche suivante est proposée.

7.2.1 Arrêt des groupes de tables

Si certains groupes de tables sont encore actifs, il faut au préalable les arrêter à l'aide de la fonction *emaj_stop_group()* (ou de la fonction *emaj_force_stop_group()* si *emaj_stop_group()* retourne une erreur).

7.2.2 Sauvegarde des données utilisateurs

La configuration complète des groupes de tables existants ainsi que les paramètres E-Maj peuvent être sauvegardés sur un fichier par :

```
SELECT emaj.emaj_export_groups_configuration('<chemin.fichier.1>');

SELECT emaj.emaj_export_parameters_configuration('<chemin.fichier.2>');
```

Si la version E-Maj installée est antérieure à 3.3.0, ces fonctions ne sont pas disponibles. Il faut alors sauvegarder manuellement le contenu des tables techniques *emaj_group_def* et *emaj_param*. On peut par exemple décharger chacune de ces tables sur des fichiers par des commandes *copy*. On peut aussi les dupliquer en dehors du schéma *emaj*.

Si la version courante d'E-Maj contient la fonction *emaj_sync_def_group()* (3.2+), il est prudent d'utiliser cette fonction pour synchroniser d'abord la table *emaj_group_def* avec la configuration actuelle des groupes créés.

Si la version E-Maj installée est une version 3.1.0 ou supérieure, et si l'administrateur E-Maj a enregistré des triggers applicatifs comme « ne devant pas être automatiquement désactivés lors des opérations de rollback E-Maj », il faut également sauver la table *emaj_ignored_app_trigger*.

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

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

CREATE TABLE public.sav_ignored_app_trigger AS SELECT * FROM emaj.emaj_ignored_app_
↪trigger;

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

7.2.3 Suppression et réinstallation d'E-Maj

Une fois connecté en tant que super-utilisateur, il suffit d'enchaîner le script de désinstallation *uninstall.sql* de la version en place puis la création de l'extension.

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

CREATE EXTENSION emaj;
```

NB : avant la version 2.0.0, le script de désinstallation se nommait *uninstall.sql*.

7.2.4 Restauration des données utilisateurs

Si les configurations de groupes de tables et de paramètres ont été exportées dans des fichiers avec les fonctions *emaj_export_parameters_configuration()* et *emaj_export_groups_configuration()*, ces configurations peuvent être rechargées par :

```
SELECT emaj.emaj_import_parameters_configuration('<chemin.fichier.2>', TRUE);

SELECT emaj.emaj_import_groups_configuration('<chemin.fichier.1>');
```

Si les configurations ont été sauvegardées au préalable par déchargement ou copie de tables, ces tables techniques d'E-Maj peuvent être par exemple rechargées avec des requêtes de type INSERT SELECT.

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

INSERT INTO emaj.emaj_ignored_app_trigger SELECT * FROM public.sav_ignored_app_
↪trigger;

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

Une fois les données copiées, les tables ou fichiers temporaires peuvent être supprimés.

7.3 Mise à jour à partir d'une version E-Maj comprise entre 0.11.0 et 1.3.1

Pour les versions comprises entre 0.11.0 et 1.3.1, des **scripts psql de mise à jour** sont livrés. Ils permettent de passer d'une version à la suivante.

Chaque étape peut être réalisée sans toucher aux groupes de tables, ceux-ci pouvant même être actifs au moment du changement de version. Ceci signifie en particulier :

- que des mises à jour de tables peuvent être enregistrées avant puis après le changement de version, sans que les groupes de tables soient arrêtés,
- et donc qu'après le changement de version, un *rollback* à une marque posée avant ce changement de version est possible.

Version source	Version cible	script psql	Durée	Mises à jour concurrentes (1)
0.11.0	0.11.1	emaj-0.11.0-to-0.11.1.sql	Très rapide	Oui
0.11.1	1.0.0	emaj-0.11.1-to-1.0.0.sql	Très rapide	Oui
1.0.0	1.0.1	emaj-1.0.0-to-1.0.1.sql	Très rapide	Oui
1.0.1	1.0.2	emaj-1.0.1-to-1.0.2.sql	Très rapide	Oui
1.0.2	1.1.0	emaj-1.0.2-to-1.1.0.sql	Variable	Non (2)
1.1.0	1.2.0	emaj-1.1.0-to-1.2.0.sql	Très rapide	Oui
1.2.0	1.3.0	emaj-1.2.0-to-1.3.0.sql	Rapide	Oui (3)
1.3.0	1.3.1	emaj-1.3.0-to-1.3.1.sql	Très rapide	Oui

(1) La dernière colonne indique si la mise à jour de la version E-Maj peut être effectuée alors que des tables couvertes par E-Maj sont accédées en mise à jour. Notons que durant la mise à jour, d'éventuelles autres actions E-Maj (pose de marque, rollback,...) sont mises en attentes.

(2) Le passage en 1.1.0 nécessite la transformation des tables de log (ajout d'une colonne). Cela a pour conséquence que :

- même si les groupes de tables peuvent rester actifs, ce changement de version ne peut s'exécuter qu'à un moment où les tables ne sont pas mises à jour par des traitements,
- la durée de l'opération est très variable et dépend essentiellement du volume de données contenu dans les tables de log.

Notez également que les statistiques qu'E-Maj a collectées lors des précédentes opérations de rollback ne sont pas reprises (le fonctionnement des rollbacks est trop différent pour que ces anciennes statistiques soient pertinentes).

(3) Il est recommandé de réaliser le passage en 1.3.0 dans une période de faible activité sur la base de données. En effet, le renommage des triggers E-Maj sur les tables applicatives entraîne la pose de verrous de type *Access Exclusive* qui peuvent entrer en conflit avec d'autres accès.

A la fin de chaque mise à jour le message suivant est affiché :

```
>>> E-Maj successfully upgraded to <nouvelle_version>
```

7.4 Passage d'E-Maj 1.3.1 à une version supérieure

La mise à jour de la version 1.3.1 est spécifique car elle doit gérer le passage d'une installation par script *psql* à une installation par *extension*.

Pour ce faire, il suffit d'exécuter la requête SQL

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

C'est le gestionnaire d'extension de PostgreSQL qui détermine le ou les scripts à exécuter en fonction de la version indiquée comme courante dans le fichier *emaj.control*.

Cette mise à jour ne peut néanmoins pas traiter le cas où au moins un groupe de tables a été créé avec une version de PostgreSQL antérieure à 8.4. Dans ce cas le ou les groupes de tables concernés doivent être supprimés au préalable puis recréés par la suite.

Cette mise à jour n'est pas non plus possible avec les versions PostgreSQL 13 et suivantes. Pour ces versions de PostgreSQL, E-Maj doit être désinstallé puis réinstallé dans sa dernière version.

7.5 Mise à jour d'une version déjà installée comme extension

Une version existante installée comme une *extension* se met à jour par une simple requête :

```
ALTER EXTENSION emaj UPDATE;
```

C'est le gestionnaire d'extension de PostgreSQL qui détermine le ou les scripts à exécuter en fonction de la version installée et de la version indiquée comme courante dans le fichier *emaj.control*.

L'opération est très rapide et ne touche pas aux groupes de tables. Ceux-ci peuvent rester actifs au moment de la mise à jour. Ceci signifie en particulier :

- que des mises à jour de tables peuvent être enregistrées avant puis après le changement de version
- et donc qu'après le changement de version, un *rollback* à une marque posée avant ce changement de version est possible.

Spécificités liées aux versions :

- La procédure de mise à jour d'une version 2.0.1 en version 2.1.0 peut modifier la table *emaj_group_def* pour refléter le fait que le tablespace tspemaj n'est plus automatiquement considéré comme un tablespace par défaut. Si *tspemaj* est effectivement utilisé comme tablespace par défaut pour des groupes de tables créés, le contenu des colonnes *grpdef_log_dat_tsp* et *grpdef_log_idx_tsp* de la table *emaj_group_def* est automatiquement ajusté afin qu'une future opération de suppression puis recréation d'un groupe de tables puisse stocker les tables et index de log dans les mêmes tablespaces. L'administrateur peut revoir ces changements pour être sûr qu'ils correspondent bien à ses souhaits.
- La procédure de mise à jour d'une version 2.2.2 en version 2.2.3 vérifie les valeurs des séquences de log enregistrées. Dans certains cas, elle peut demander une ré-initialisation préalable de certains groupes de tables.

— La procédure de mise à jour d'une version 2.3.1 en version 3.0.0 change la structure des tables de log : les 2 colonnes *emaj_client_ip* et *emaj_client_port* ne sont plus créées. Les tables de log existantes ne sont pas modifiées. Seules les nouvelles tables de log sont impactées. Mais il est possible à l'administrateur *d'ajouter ces deux colonnes*, en utilisant le paramètre *"alter_log_tables"*.
— La procédure de mise à jour d'une version 3.0.0 en version 3.1.0 renomme les objets de log existants. Ceci conduit à une pose de verrou sur chaque table applicative, qui peut entrer en conflit avec des accès concurrents sur les tables. La procédure de mise à jour génère également un message d'alerte indiquant que les changements dans la gestion des triggers applicatifs par les fonctions de rollback E-Maj peuvent nécessiter des modifications dans les procédures utilisateurs.

CHAPITRE 8

Désinstallation d'E-Maj d'une base de données

Pour désinstaller E-Maj d'une base de données, l'utilisateur doit se connecter à cette base avec *psql*, en tant que super-utilisateur.

Si on souhaite supprimer les **rôles** *emaj_adm* et *emaj_viewer*, il faut au préalable retirer les droits donnés sur ces rôles à d'éventuels autres rôles, à l'aide de requêtes SQL *REVOKE*.

```
REVOKE emaj_adm FROM <role.ou.liste.de.rôles>;
REVOKE emaj_viewer FROM <role.ou.liste.de.rôles>;
```

Si ces rôles *emaj_adm* et *emaj_viewer* possèdent des droits d'accès sur des tables ou autres objets relationnels applicatifs, il faut également supprimer ces droits au préalable à l'aide d'autres requêtes SQL *REVOKE*.

Bien qu'installé en standard avec une requête *CREATE EXTENSION*, E-Maj ne peut se désinstaller par une simple requête *DROP EXTENSION*. Un trigger sur événement bloque d'ailleurs l'exécution d'une telle requête.

Pour désinstaller E-Maj, il faut simplement exécuter le **script fourni** *emaj_uninstall.sql*.

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

Ce script effectue les actions suivantes :

- iI exécute les éventuelles fonctions de nettoyage créées par l'exécution des scripts de démonstration ou de test fournis,
- il arrête les groupes de tables encore actifs, s'il y en a,
- il supprime les groupes de tables existants, supprimant en particulier les triggers sur les tables applicatives,
- il supprime l'extension et le schéma principal *emaj*,
- il supprime les rôles *emaj_adm* et *emaj_viewer* s'ils ne sont pas associés à d'autres rôles ou à d'autres bases de données de l'instance et ne possèdent pas de droits sur d'autres tables.

L'exécution du script de désinstallation affiche ceci :

```
$ psql ... -f sql/emaj_uninstall.sql
>>> Starting E-Maj uninstallation procedure...
SET
psql:sql/emaj_uninstall.sql:203: WARNING:  emaj_uninstall: emaj_adm and emaj_viewer␣
↪roles have been dropped.
```

(suite sur la page suivante)

(suite de la page précédente)

```
DO
SET
>>> E-maj successfully uninstalled from this database
```

CHAPITRE 9

Changement de version de PostgreSQL

9.1 Changement de version mineure de PostgreSQL

Le changement de version mineure de PostgreSQL se limitant à un simple remplacement des binaires du logiciel, il n'y a aucune contrainte particulière concernant E-Maj.

9.2 Changement simultané de version majeure de PostgreSQL et de version d'E-Maj

Un changement de version majeure de PostgreSQL peut être l'occasion de changer de version d'E-Maj. Mais dans ce cas, l'environnement E-Maj est à recréer complètement et les anciens objets (groupes de tables, logs, marques,...) ne sont pas réutilisables.

9.3 Changement de version majeure de PostgreSQL avec conservation de l'existant E-Maj

Il est néanmoins possible de conserver l'existant E-Maj (groupes de tables, logs, marques,...) lors d'un changement de version majeure de PostgreSQL. Les groupes de tables peuvent même rester actifs lors de l'opération.

Mais ceci nécessite que 2 conditions soient remplies :

- les ancienne et nouvelle instances doivent utiliser la même version d'E-Maj,
- un script d'adaptation doit être exécuté post migration, avant d'utiliser à nouveau E-Maj.

Il est naturellement possible de procéder au changement de version E-Maj avant ou après le changement de version PostgreSQL.

Si le changement de version PostgreSQL s'effectue avec un déchargement et rechargement des données et si les groupes de tables peuvent être arrêtés, une purge des tables de log, en utilisant la fonction *emaj_reset_group()*, peut permettre de diminuer la quantité de données à manipuler et donc d'accélérer l'opération.

9.4 Script d'adaptation post migration

Dans certains cas de figure, un changement de version majeure de PostgreSQL impacte le contenu même de l'extension E-Maj. Un script est donc fourni pour gérer ces cas.

Après chaque changement de version majeure de PostgreSQL, il faut exécuter avec *psql* et en tant que super-utilisateur :

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

Dans les versions E-Maj 2.0.0 et suivantes, ce script ne fait que créer les éventuels triggers sur événement manquants :

- ceux qui apparaissent en version 9.3 et qui protègent contre la suppression de l'extension elle-même et contre la suppression d'objets E-Maj (tables de log, fonctions, . . .),
- celui qui apparaît en version 9.5 et qui protège contre les changements de structure des tables de log.

Le script peut-être lancé plusieurs fois de suite sur une même version, seule la première exécution modifiant l'environnement.

CHAPITRE 10

Mise en place de la politique d'accès à E-Maj

Une mauvaise utilisation d'E-Maj peut mettre en cause l'intégrité des bases de données. Aussi convient-il de n'autoriser son usage qu'à des utilisateurs qualifiés et clairement identifiés comme tels.

10.1 Les rôles E-Maj

Pour utiliser E-Maj, on peut se connecter en tant que super-utilisateur. Mais pour des raisons de sécurité, il est préférable de tirer profit des deux rôles créés par la procédure d'installation :

- **emaj_adm** sert de rôle d'administration ; il peut exécuter toutes les fonctions et accéder à toutes les tables d'E-Maj, en lecture comme en mise à jour,
- **emaj_viewer** sert pour des accès limités à de la consultation ; il ne peut exécuter que des fonctions de type statistique et n'accède aux tables d'E-Maj qu'en lecture.

Tous les droits attribués à *emaj_viewer* le sont aussi à *emaj_adm*.

Mais lors de leur création, ces deux rôles ne se sont pas vus attribuer de capacité de connexion (aucun mot de passe et option *NOLOGIN* spécifiés). Il est recommandé de NE PAS leur attribuer cette capacité de connexion. A la place, il suffit d'attribuer les droits qu'ils possèdent à d'autres rôles par des requêtes SQL de type *GRANT*.

10.2 Attribution des droits E-Maj

Pour attribuer à un rôle donné tous les droits associés à l'un des deux rôles *emaj_adm* ou *emaj_viewer*, et une fois connecté en tant que super-utilisateur pour avoir le niveau de droit suffisant, il suffit d'exécuter l'une des commandes suivantes :

```
GRANT emaj_adm TO <mon.rôle.administrateur.emaj>;
GRANT emaj_viewer TO <mon.rôle.de.consultation.emaj>;
```

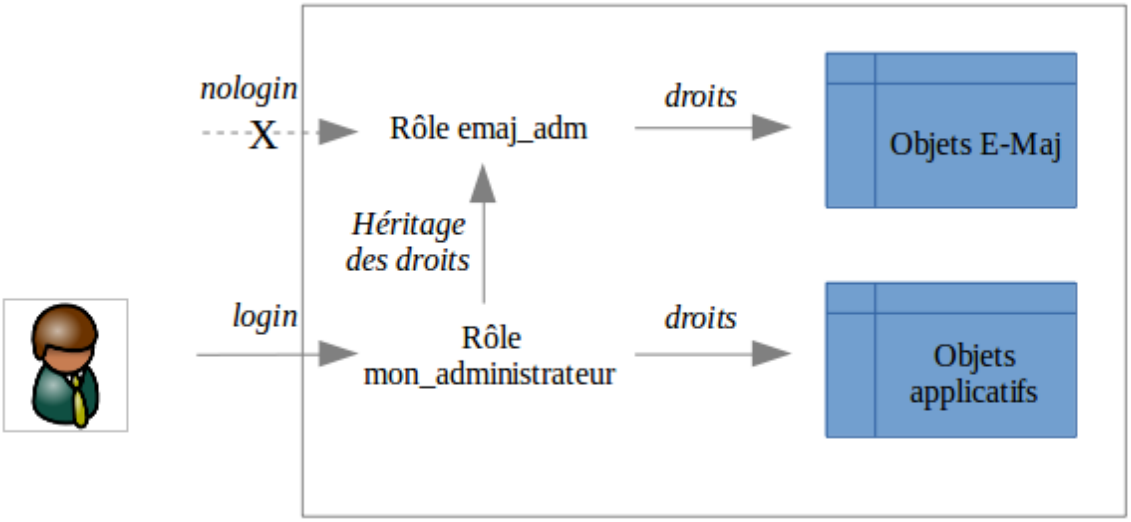
Naturellement, plusieurs rôles peuvent se voir attribuer les droits *emaj_adm* ou *emaj_viewer*.

10.3 Attribution des droits sur les tables et objets applicatifs

Pour qu'un administrateur E-Maj puisse également accéder à des tables ou à d'autres objets applicatifs (schémas, séquences, vues, fonctions,...), on peut attribuer aux rôles *emaj_adm* ou *emaj_viewer* des droits d'accès à ces objets. Mais il est préférable d'affecter ces droits directement et uniquement aux rôles qui héritent des droits d'*emaj_adm* ou *emaj_viewer*, en ne laissant à ces derniers que des droits sur les tables et objets E-Maj.

10.4 Synthèse

Le schéma ci-dessous symbolise l'attribution recommandée des droits pour un administrateur E-Maj.



Bien évidemment, ce schéma s'applique également au rôle *emaj_viewer*.

Sauf indication contraire, les opérations qui suivent vont pouvoir être exécutées indifféremment avec un rôle super-utilisateur ou un rôle du groupe *emaj_adm*.

CHAPITRE 11

Création et suppression des groupes de tables

11.1 Principes de configuration des groupes de tables

Configurer un groupe de tables consiste à :
- définir les caractéristiques du groupe de tables,
- définir les tables et les séquences à assigner au groupe de tables,
- optionnellement, définir quelques propriétés spécifiques à chaque table.

11.1.1 Le groupe de tables

Un groupe de tables est identifié par son nom. Le **nom** doit donc être unique pour la base de données concernée. Un nom de groupe de tables doit contenir au moins un caractère. Il peut contenir des espaces et/ou des caractères de ponctuation. Mais il est recommandé d'éviter les caractères virgule, guillemet simple ou double.

Il faut également spécifier à sa création si le groupe de tables est de type *ROLLBACKABLE ou AUDIT_ONLY*. Notons que cette caractéristique du groupe de tables ne peut être modifiée après la création du groupe. Pour la changer, il faut supprimer puis recréer le groupe de tables.

11.1.2 Les tables et séquences à assigner

Un groupe de tables peut contenir des tables et/ou des séquences d'un ou plusieurs schémas.

Toutes les tables d'un schéma n'appartiennent pas nécessairement au même groupe. Certaines peuvent appartenir à des groupes différents. D'autres peuvent n'être affectées à aucun groupe.

Mais à **un instant donné**, une table ou une séquence ne peut être affectée qu'à au plus **un seul groupe** de tables.

Prudence : Pour garantir l'intégrité des tables gérées par E-Maj, il est fondamental de porter une attention particulière à cette phase de définition des groupes de tables. Si une table était manquante, son contenu se trouverait bien sûr désynchronisé après une opération de *rollback* E-Maj sur le groupe de tables auquel elle aurait dû appartenir.

En particulier, lors de la création ou de la suppression de tables applicatives, il est important de tenir à jour la configuration des groupes de tables.

Toute table appartenant à un groupe de tables non créé en mode *AUDIT_ONLY* doit posséder une clé primaire explicite (clause *PRIMARY KEY* des *CREATE TABLE* ou *ALTER TABLE*).

E-Maj gère les partitions élémentaires de tables partitionnées créées avec le DDL déclaratif (à partir de PostgreSQL 10). Elles sont gérées comme n'importe quelle autre table. En revanche, comme les tables mères restent toujours vides, E-Maj refuse qu'elles soient assignées à un groupe de tables. Toutes les partitions d'une même table partitionnée n'ont pas nécessairement besoin d'être couvertes par E-Maj. Des partitions d'une même table partitionnée peuvent être affectées à des groupes de tables différents.

De par leur nature, les tables temporaires (*TEMPORARY TABLE*) ne peuvent être supportées par E-Maj. Et les tables de type *UNLOGGED* ou *WITH OIDS* ne peuvent appartenir qu'à un groupe de tables de type *AUDIT_ONLY*.

Si une séquence est associée à une table applicative, il est recommandé de l'assigner au même groupe que sa table. Ainsi, lors d'une opération de rollback E-Maj, elle sera remise dans l'état où elle se trouvait lors de la pose de la marque servant de référence au rollback. Dans le cas contraire, l'opération de Rollback E-Maj provoquera simplement un trou dans la suite de valeurs de la séquence.

Les tables de log E-Maj et leur séquence NE doivent PAS être référencées dans un groupe de tables.

11.1.3 Propriétés spécifiques aux tables

Il existe 3 propriétés spécifiques aux tables affectées à un groupe de tables :
- niveau de priorité,
- tablespace pour les données des tables de log,
- tablespace pour les index des tables de log.

Le niveau de **priorité** est un entier (INTEGER). Par défaut, il prend la valeur NULL, Il correspond à l'ordre dans lequel les tables seront traitées par les principales fonctions d'E-Maj. Ceci peut-être en particulier utile pour faciliter la pose des verrous. En effet, en posant les verrous sur les tables dans le même ordre que les accès applicatifs typiques, on peut limiter le risque de deadlock. Les fonctions E-Maj traitent les tables dans l'ordre croissant de priorité, les valeurs *NULL* étant traitées en dernier. Pour un même niveau de priorité, les tables sont traitées dans l'ordre alphabétique de nom de schéma puis de nom de table.

Pour optimiser les performances des installations E-Maj comportant un très grand nombre de tables, il peut s'avérer intéressant de répartir les tables de log et leur index dans plusieurs tablespaces. Deux propriétés sont donc disponibles pour spécifier :
- un nom de **tablespace** à utiliser pour la **table de log** d'une table applicative,
- un nom de **tablespace** à utiliser pour l'**index** de la table de log.

Par défaut, ces propriétés prennent la valeur *NULL*, indiquant l'utilisation du tablespace par défaut de la session courante.

11.2 Création des groupes de tables

Il existe deux grandes façons de gérer la structure des groupes de tables :
- gérer l'affectation des tables et séquences dans les groupes de manière dynamique, au travers des fonctions dédiées à cet effet,
- décrire la liste des groupes, avec les tables et séquences qu'ils contiennent, dans une table de configuration, *emaj_group_def*.

Bien que très différentes, les deux méthodes sont *combinables entre elles*.

11.2.1 Méthode « constitution dynamique »

La création d'un groupe de tables s'opère en plusieurs étapes. Dans un premier temps, on crée un groupe de tables vide. Puis on va y assigner des tables et des séquences.

Création d'un groupe de tables vide

Pour créer un groupe de tables vide, il suffit d'exécuter la requête SQL suivante

```
SELECT emaj.emaj_create_group('<nom.du.groupe>', <est.rollbackable>, <est.vide>);
```

Le second paramètre, de type booléen, indique si le groupe est de type *ROLLBACKABLE* avec la valeur vrai ou de type *AUDIT_ONLY* avec la valeur fausse. Si le second paramètre n'est pas fourni, le groupe à créer est considéré comme étant de type *ROLLBACKABLE*.

Notez que le troisième paramètre doit être explicitement valorisé à « vrai ».

Le groupe peut être référencé dans la table *emaj_group_def*. Mais dans ce cas, le contenu de la table emaj_group_def est simplement ignoré.

La fonction retourne le nombre de tables et de séquences contenues dans le groupe.

Assignation de tables et séquences dans un groupe de tables

Six fonctions permettent d'ajouter dynamiquement des tables ou des séquences dans un groupe de tables.

Pour **ajouter une ou plusieurs tables** dans un groupe de tables :

```
SELECT emaj.emaj_assign_table('<schéma>', '<table>', '<nom.du.groupe>' [,'<propriétés>
↪' [,'<marque>']]);
```

ou

```
SELECT emaj.emaj_assign_tables('<schéma>', '<tableau.de.tables>', '<nom.du.groupe>' [,
↪'<propriétés>' [,'<marque>']] );
```

ou

```
SELECT emaj.emaj_assign_tables('<schéma>', '<filtre.de.tables.à.inclure>', '<filtre.
↪de.tables.à.exclure>', '<nom.du.groupe>' [,'<propriétés>' [,'<marque>']] );
```

Pour **ajouter une ou plusieurs séquences** dans un groupe de tables :

```
SELECT emaj.emaj_assign_sequence('<schéma>', '<séquence>', '<nom.du.groupe>' [,'
↪<marque>']);
```

ou

```
SELECT emaj.emaj_assign_sequences('<schéma>', '<tableau.de.séquences>', '<nom.du.
↪groupe>' [,'<marque>'] );
```

ou

```
SELECT emaj.emaj_assign_sequences('<schéma>', '<filtre.de.séquences.à.inclure>', '
↪<filtre.de.séquences.à.exclure>', '<nom.du.groupe>' [,'<marque>'] );
```

Pour les fonctions traitant plusieurs tables ou séquences en une seule opération, la liste des tables ou séquences à traiter est :

- soit fournie par un paramètre de type tableau de TEXT,
- soit construite à partir de deux expressions rationnelles fournies en paramètres.

Un tableau de *TEXT* est typiquement exprimé avec une syntaxe du type :

```
ARRAY['élément1', 'élément2', ...]
```

Les deux expressions rationnelles suivent la syntaxe *POSIX* (se référer à la documentation PostgreSQL pour plus de détails). La première définit un filtre de sélection des tables dans le schéma, La seconde définit un filtre d'exclusion appliqué sur les tables sélectionnées. Quelques exemples de filtres.

Pour sélectionner toutes les tables ou séquences du schéma *mon_schema* :

```
'mon_schema', '.*', ''
```

Pour sélectionner toutes les tables de ce schéma, et dont le nom commence par *"tbl"* :

```
'mon_schema', '^tbl.*', ''
```

Pour sélectionner toutes les tables de ce schéma, et dont le nom commence par *"tbl"*, à l'exception de celles dont le nom se termine par *"_sav"* :

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

Les fonctions d'assignation à un groupe de tables construisant leur sélection à partir des deux expressions rationnelles tiennent compte du contexte des tables ou séquences concernées. Ne sont pas sélectionnées par exemple : les tables ou séquences déjà affectées à un groupe, les tables sans clé primaire pour un groupe de tables *rollbackable* ou celles déclarées *UNLOGGED*.

Le paramètre <*propriétés*> des fonctions d'ajout de tables à un groupe de tables est optionnel. Il permet de préciser les propriétés spécifiques pour la ou les tables. De type JSONB. on peut le valoriser ainsi :

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

où :

- <n> est le niveau de priorité pour la ou les tables
- <xxx> est le nom du tablespace pour les tables de log
- <yyy> est le nom du tablespace pour les index de log

Si une des propriétés n'est pas valorisée dans le paramètre *JSONB*, sa valeur est considérée comme *NULL*.

Si des tablespaces spécifiques pour les tables de log ou pour leurs index sont référencés, ceux-ci doivent exister au préalable.

Pour toutes les fonctions, un verrou exclusif est posé sur chaque table du ou des groupes de tables concernés, afin de garantir la stabilité des groupes durant ces opérations.

Toutes ces fonctions retournent le nombre de tables ou séquences ajoutées au groupe de tables.

Les fonctions d'assignation de tables dans un groupe de tables créent les tables de log, les fonctions et triggers de log, ainsi que les triggers traitant les exécutions de requêtes SQL *TRUNCATE*. Elles créent également les éventuels schémas de log nécessaires.

11.2.2 Méthode « table de configuration »

Il s'agit de la méthode initialement disponible avec E-Maj.

La création d'un groupe de tables s'opère en 2 temps :

- définition de la configuration du groupe dans la table *emaj.emaj_group_def*,
- création proprement dite du groupe de tables.

Alimentation de la table emaj_group_def

Le contenu du ou des groupes de tables à créer se définit en garnissant la table **emaj.emaj_group_def**. Il faut insérer dans cette table une ligne par table ou séquence applicative à intégrer dans un groupe. Cette table *emaj_group_def* a la structure suivante :

Colonne	Type	Description
grpdef_group	TEXT	nom du groupe de tables
grpdef_schema	TEXT	nom du schéma contenant la table ou la séquence applicative
grpdef_tblseq	TEXT	nom de la table ou de la séquence applicative
grpdef_priority	INT	niveau de priorité de la table dans les traitements E-Maj (optionnel)
grpdef_log_dat_tsp	TEXT	nom du tablespace hébergeant la table de log (optionnel)
grpdef_log_idx_tsp	TEXT	nom du tablespace hébergeant l'index de la table de log (optionnel)

L'administrateur peut alimenter cette table par tout moyen usuel : verbe SQL *INSERT*, verbe SQL *COPY*, commande *psql \copy*, outil graphique, etc.

Le contenu de la table *emaj_group_def* est sensible à la casse. Les noms de schéma, de table, de séquence et de tablespace doivent correspondre à la façon dont PostgreSQL les enregistre dans son catalogue. Ces noms sont le plus souvent en minuscule. Mais si un nom est encadré par des double-guillemets dans les requêtes SQL, car contenant des majuscules ou des espaces, alors il doit être enregistré dans la table *emaj_group_def* avec ces mêmes majuscules et espaces.

Attention, le contenu de la table *emaj_group_def* est altéré par les fonctions *emaj_import_groups_configuration()*.

Création du groupe de tables

Une fois la constitution d'un groupe de tables définie, E-Maj peut créer ce groupe. Pour ce faire, il suffit d'exécuter la requête SQL suivante

```
SELECT emaj.emaj_create_group('<nom.du.groupe>', <est.rollbackable>);
```

ou encore, dans sa forme abrégée

```
SELECT emaj.emaj_create_group('<nom.du.groupe>');
```

Le second paramètre, de type booléen, indique si le groupe est de type *ROLLBACKABLE* avec la valeur vrai ou de type *AUDIT_ONLY* avec la valeur fausse. Si le second paramètre n'est pas fourni, le groupe à créer est considéré comme étant de type *ROLLBACKABLE*.

La fonction retourne le nombre de tables et de séquences contenues dans le groupe.

Pour chaque table du groupe, cette fonction crée la table de log associée, la fonction et le trigger de log, ainsi que le trigger traitant les exécutions de requêtes SQL *TRUNCATE*.

La fonction crée également les schémas de log nécessaires.

En revanche, si des tablespaces pour les tables de log ou pour leurs index sont référencés, ceux-ci doivent déjà exister avant l'exécution de la fonction.

La fonction *emaj_create_group()* contrôle également l'existence de « triggers applicatifs » sur les tables du groupe. Si un trigger existe sur une table du groupe, un message est retourné incitant l'utilisateur à vérifier l'impact du trigger lors des éventuels rollbacks E-Maj.

Si une séquence du groupe est associée à une colonne soit de type *SERIAL* ou *BIGSERIAL* soit définie avec une clause *GENERATED AS IDENTITY*, et que sa table d'appartenance ne fait pas partie du groupe, la fonction génère également un message de type *WARNING*.

Une forme particulière de la fonction permet de créer un groupe de tables vide, c'est à dire ne contenant à sa création aucune table ni séquence

```
SELECT emaj.emaj_create_group('<nom.du.groupe>', <est.rollbackable>, <est.vide>);
```

Le troisième paramètre prend la valeur *faux* par défaut. Si le paramètre est valorisé à *vrai*, le groupe peut être référencé dans la table *emaj_group_def*. Mais dans ce cas, le contenu de la table *emaj_group_def* est ignoré. Une fois créé, un groupe vide peut ensuite être peuplé, à l'aide de la fonction *emaj_alter_group()* ou des fonctions d'*ajustement dynamique des groupes de tables*.

Toutes les actions enchaînées par la fonction *emaj_create_group()* sont exécutées au sein d'une unique transaction. En conséquence, si une erreur survient durant l'opération, toutes les tables, fonctions et triggers déjà créés par la fonction sont annulées.

En enregistrant la composition du groupe dans une autre table interne (*emaj_relation*), la fonction *emaj_create_group()* en fige sa définition pour les autres fonctions E-Maj, même si le contenu de la table *emaj_group_def* est modifié entre temps.

Un groupe créé peut être modifié par la fonction *emaj_alter_group()*.

11.3 Suppression d'un groupe de tables

Pour supprimer un groupe de tables créé au préalable par la fonction *emaj_create_group()*, il faut que le groupe de tables à supprimer soit déjà arrêté. Si ce n'est pas le cas, il faut d'abord utiliser la fonction *emaj_stop_group()*.

Ensuite, il suffit d'exécuter la commande SQL

```
SELECT emaj.emaj_drop_group('<nom.du.groupe>');
```

La fonction retourne le nombre de tables et de séquences contenues dans le groupe.

Pour ce groupe de tables, la fonction *emaj_drop_group()* supprime tous les objets qui ont été créés par les fonctions d'assignation ou par la fonction *emaj_create_group()* : tables, séquences, fonctions et triggers de log.

Les éventuels schémas de log qui deviennent inutilisés sont également supprimés.

La pose de verrous qu'entraîne cette opération peut se traduire par la survenue d'une étreinte fatale (*deadlock*). Si la résolution de l'étreinte fatale impacte la fonction E-Maj, le *deadlock* est intercepté et la pose de verrou est automatiquement réitérée, avec un maximum de 5 tentatives.

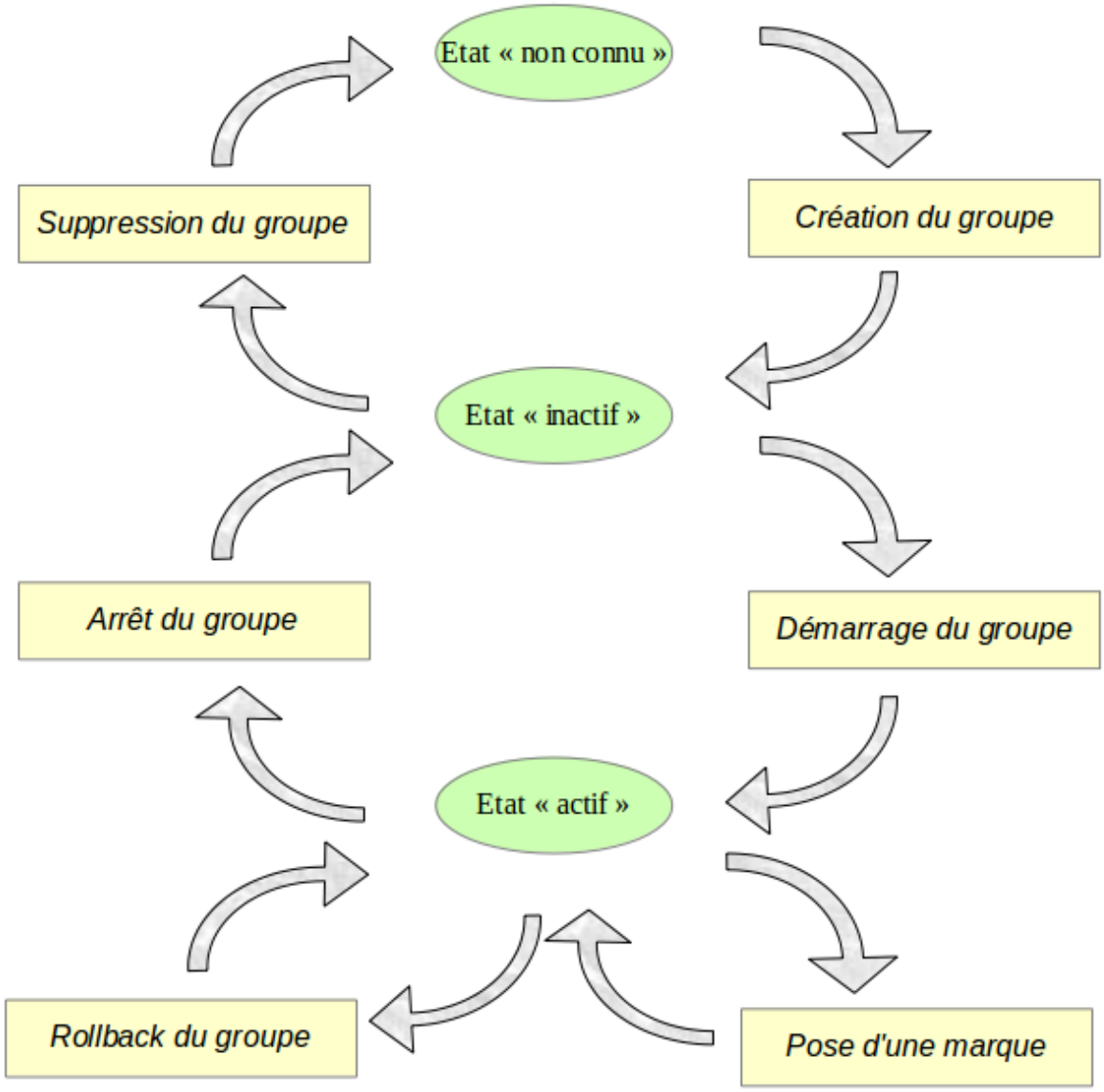
CHAPITRE 12

Fonctions principales

Avant de décrire chacune des principales fonctions d'E-Maj, il est intéressant d'avoir un aperçu global de l'enchaînement typique des opérations.

12.1 Enchaînement des opérations

L'enchaînement des opérations possibles pour un groupe de tables peut se matérialiser par ce synoptique.



12.2 Démarrage d'un groupe de tables

Démarrer un groupe de table consiste à activer l'enregistrement des mises à jour des tables du groupe. Pour ce faire, il faut exécuter la commande

```
SELECT emaj.emaj_start_group('<nom.du.groupe>'[, '<nom.de.marque>' [, <effacer.
↪anciens.logs?>]]);
```

Le groupe de tables doit être au préalable à l'état inactif.

Le démarrage du groupe de tables créé une première marque.

S'il est spécifié, le nom de la marque initiale peut contenir un caractère générique "%". Ce caractère est alors remplacé

par l'heure courante, au format *hh.mn.ss.mmmm*,

Si le paramètre représentant la marque n'est pas spécifié, ou s'il est vide ou *NULL*, un nom est automatiquement généré : « *START_%* », où le caractère "*%*" représente l'heure courante, au format *hh.mn.ss.mmmm*.

Le paramètre <*anciens.logs.à.effacer*> est un booléen optionnel. Par défaut sa valeur est égal à vrai (true), ce qui signifie que les tables de log du groupe de tables sont purgées de toutes anciennes données avant l'activation des triggers de log. Si le paramètre est explicitement positionné à « faux » (false), les anciens enregistrements sont conservés dans les tables de log. De la même manière, les anciennes marques sont conservées, même si ces dernières ne sont alors plus utilisables pour un éventuel rollback (des mises à jour ont pu être effectuées sans être tracées alors que le groupe de tables était arrêté).

La fonction retourne le nombre de tables et de séquences contenues dans le groupe.

Pour être certain qu'aucune transaction impliquant les tables du groupe n'est en cours, la fonction *emaj_start_group()* pose explicitement sur chacune des tables du groupe un verrou de type *SHARE ROW EXCLUSIVE*. Si des transactions accédant à ces tables sont en cours, ceci peut se traduire par la survenue d'une étreinte fatale (*deadlock*). Si la résolution de l'étreinte fatale impacte la fonction E-Maj, le deadlock est intercepté et la pose de verrou est automatiquement réitérée, avec un maximum de 5 tentatives.

La fonction procède également à la purge des événements les plus anciens de la table technique *emaj_hist*.

A l'issue du démarrage d'un groupe, celui-ci devient actif (« *LOGGING* »).

Plusieurs groupes de tables peuvent être démarrés en même temps, en utilisant la fonction *emaj_start_groups()*

```
SELECT emaj.emaj_start_groups('<tableau.des.groupes>'[, '<nom.de.marque>' [, <effacer.
↪anciens.logs?>]]);
```

Plus d'information sur les *fonctions multi-groupes*.

12.3 Pose d'une marque intermédiaire

Lorsque toutes les tables et séquences d'un groupe sont jugées dans un état stable pouvant servir de référence pour un éventuel *rollback*, une marque peut être posée. Ceci s'effectue par la requête SQL suivante

```
SELECT emaj.emaj_set_mark_group('<nom.du.groupe>'[, '<nom.de.marque>']);
```

Le groupe de tables doit être à l'état actif.

Une marque de même nom ne doit pas déjà exister pour le groupe de tables.

Le nom de la marque peut contenir un caractère générique "%". Ce caractère est alors remplacé par l'heure courante, au format *hh.mn.ss.mmmm*,

Si le paramètre représentant la marque n'est pas spécifié ou s'il est vide ou *NULL*, un nom est automatiquement généré : « *MARK_%* », où le caractère "*%*" représente l'heure courante, au format *hh.mn.ss.mmmm*.

La fonction retourne le nombre de tables et de séquences contenues dans le groupe.

La fonction *emaj_set_mark_group()* enregistre l'identité de la nouvelle marque, avec l'état des séquences applicatives appartenant au groupe, ainsi que l'état des séquences associées aux tables de log. Les séquences applicatives sont traitées en premier, pour enregistrer leur état au plus près du début de la transaction, ces séquences ne pouvant pas être protégées des mises à jour par des verrous.

Il est possible d'enregistrer deux marques consécutives sans que des mises à jour de tables aient été enregistrées entre ces deux marques.

La fonction *emaj_set_mark_group()* pose des verrous de type « *ROW EXCLUSIVE* » sur chaque table du groupe. Ceci permet de s'assurer qu'aucune transaction ayant déjà fait des mises à jour sur une table du groupe n'est en cours.

Néanmoins, ceci ne garantit pas qu'une transaction ayant lu une ou plusieurs tables avant la pose de la marque, fasse des mises à jours après la pose de la marque. Dans ce cas, ces mises à jours effectuées après la pose de la marque seraient candidates à un éventuel rollback sur cette marque.

Une marque peut être posée sur plusieurs groupes de tables même temps, en utilisant la fonction *emaj_set_mark_groups()*

```
SELECT emaj.emaj_set_mark_groups('<tableau.des.groupes>'[, '<nom.de.marque>']);
```

Plus d'information sur les *fonctions multi-groupes*.

12.4 Rollback simple d'un groupe de tables

S'il est nécessaire de remettre les tables et séquences d'un groupe dans l'état dans lequel elles se trouvaient lors de la prise d'une marque, il faut procéder à un rollback. Pour un rollback simple (« *unlogged* » ou « *non tracé* »), il suffit d'exécuter la requête SQL suivante

```
SELECT * FROM emaj.emaj_rollback_group('<nom.du.groupe>', '<nom.de.marque>', <est_
↪altération_groupe_permise>);
```

Le groupe de tables doit être à l'état actif et la marque indiquée doit être toujours « active », c'est à dire qu'elle ne doit pas être marquée comme logiquement supprimée.

Le mot clé "*EMAJ_LAST_MARK*" peut être utilisé comme nom de marque pour indiquer la dernière marque posée.

Le 3ème paramètre est un booléen qui indique si l'opération de rollback peut cibler une marque posée antérieurement à une opération de *modification du groupe de tables*. Selon leur nature, les modifications de groupe de tables effectuées alors que ce dernier est en état *LOGGING* peuvent être ou non automatiquement annulées. Dans certains cas, cette annulation peut être partielle. Par défaut, ce paramètre prend la valeur *FAUX*.

La fonction retourne un ensemble de lignes comportant un niveau de sévérité pouvant prendre les valeurs « *Notice* » ou « *Warning* », et un texte de message. La fonction retourne une ligne de type « *Notice* » indiquant le nombre de tables et de séquences effectivement modifiées par l'opération de rollback. Des lignes de types « *Warning* » peuvent aussi être émises dans le cas où des opérations de modification du groupe de tables ont du être traitées par le rollback.

Pour être certain qu'aucune transaction concurrente ne mette à jour une table du groupe pendant toute la durée du rollback, la fonction *emaj_rollback_group()* pose explicitement un verrou de type *EXCLUSIVE* sur chacune des tables du groupe. Si des transactions accédant à ces tables en mise à jour sont en cours, ceci peut se traduire par la survenue d'une étreinte fatale (deadlock). Si la résolution de l'étreinte fatale impacte la fonction E-Maj, le deadlock est intercepté et la pose de verrou est automatiquement réitérée, avec un maximum de 5 tentatives. En revanche, les tables du groupe continuent à être accessibles en lecture pendant l'opération.

Si des tables du groupe à traiter possèdent des *triggers* (déclencheurs), ceux-ci sont, par défaut, temporairement désactivés pendant l'opération. Avec la fonction *emaj_ignore_app_trigger()*, on peut enregistrer des triggers comme « ne devant pas être automatiquement désactivés lors du rollback » (plus de détails *ici*).

Si une table impactée par le rollback possède une clé étrangère (*foreign key*) ou est référencée dans une clé étrangère appartenant à une autre table, alors la présence de cette clé étrangère est prise en compte par l'opération de rollback. Si le contrôle des clés créées ou modifiées par le rollback ne peut être différé en fin d'opération (contrainte non déclarée *DEFERRABLE*), alors cette clé étrangère est supprimée en début de rollback puis recréée en fin de rollback.

Lorsque le volume de mises à jour à annuler est important et que l'opération de rollback est longue, il est possible de suivre l'avancement de l'opération à l'aide de la fonction *emaj_rollback_activity()* ou du client *emajRollbackMonitor.php*.

A l'issue de l'opération de rollback, se trouvent effacées :

— les données des tables de log qui concernent les mises à jour annulées,

— toutes les marques postérieures à la marque référencée dans la commande de rollback.

Les opérations de rollback sont historisées dans la table *emaj_rlbk*. L'état final des opérations de rollback est accessible dans les colonnes *rlbk_status* et *rlbk_msg* de cette table *emaj_rlbk*.

Il est alors possible de poursuivre les traitements de mises à jour, de poser ensuite d'autres marques et éventuellement de procéder à un nouveau rollback sur une marque quelconque.

Prudence : Par nature, le repositionnement des séquences n'est pas « annulable » en cas de rollback de la transaction incluant l'exécution de la fonction *emaj_rollback_group()*. Pour cette raison, le traitement des séquences applicatives est toujours effectué après celui des tables. Néanmoins, même si le temps de traitement des séquences est très court, il n'est pas impossible qu'un problème surgisse lors de cette dernière phase. La relance de la fonction *emaj_rollback_group()* mènera à bien l'opération de manière fiable. Mais si cette fonction n'était pas ré-exécutée immédiatement, il y aurait risque que certaines séquences aient été repositionnées, contrairement aux tables et à d'autres séquences.

Plusieurs groupes de tables peuvent être « rollbackés » en même temps, en utilisant la fonction *emaj_rollback_groups()*

```
SELECT * FROM emaj.emaj_rollback_groups('<tableau.des.groupes>', '<nom.de.marque>',
↪<est_altération_groupe_permise>);
```

La marque indiquée doit strictement correspondre à un même moment dans le temps pour chacun des groupes listés. En d'autres termes, cette marque doit avoir été posée par l'appel d'une même fonction *emaj_set_mark_groups()*.

Plus d'information sur les *fonctions multi-groupes*.

Une ancienne version de ces fonctions ne comportait pas de troisième paramètre et retournait un simple entier correspondant au nombre de tables et séquences effectivement traitées :

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

SELECT emaj.emaj_rollback_groups('<tableau.des.groupes>', '<nom.de.marque>');
```

Ces 2 fonctions sont dépréciées et sont amenées à être supprimées dans une prochaine version.

12.5 Rollback annulable d'un groupe de tables

Une autre fonction permet d'exécuter un rollback de type « *logged* », Dans ce cas, les triggers de log sur les tables applicatives ne sont pas désactivés durant le rollback, de sorte que durant le rollback les mises à jours de tables appliquées sont elles-mêmes enregistrées dans les tables de log. Ainsi, il est ensuite possible d'annuler le rollback ou, en quelque sorte, de « rollbacker le rollback ».

Pour exécuter un « *logged rollback* » sur un groupe de tables, il suffit d'exécuter la requête SQL suivante :

```
SELECT * FROM emaj.emaj_logged_rollback_group('<nom.du.groupe>', '<nom.de.marque>',
↪<est_altération_groupe_permise>);
```

Les règles d'utilisation sont les mêmes que pour la fonction *emaj_rollback_group()*,

Le groupe de tables doit être en état démarré (*LOGGING*) et la marque indiquée doit être toujours « active », c'est à dire qu'elle ne doit pas être marquée comme logiquement supprimée (*DELETED*).

Le mot clé "EMAJ_LAST_MARK" peut être utilisé comme nom de marque pour indiquer la dernière marque posée.

Le 3ème paramètre est un booléen qui indique si l'opération de rollback peut cibler une marque posée antérieurement à une opération de *modification du groupe de tables*. Selon leur nature, les modifications de groupe de tables effectuées

alors que ce dernier est en état *LOGGING* peuvent être ou non automatiquement annulées. Dans certains cas, cette annulation peut être partielle. Par défaut, ce paramètre prend la valeur *FAUX*.

La fonction retourne un ensemble de lignes comportant un niveau de sévérité pouvant prendre les valeurs « *Notice* » ou « *Warning* », et un texte de message. La fonction retourne une ligne de type « *Notice* » indiquant le nombre de tables et de séquences effectivement modifiées par l'opération de rollback. Des lignes de types « *Warning* » peuvent aussi être émises dans le cas où des opérations de modification du groupe de tables ont du être traitées par le rollback.

Pour être certain qu'aucune transaction concurrente ne mette à jour une table du groupe pendant toute la durée du rollback, la fonction *emaj_logged_rollback_group()* pose explicitement un verrou de type *EXCLUSIVE* sur chacune des tables du groupe. Si des transactions accédant à ces tables en mise à jour sont en cours, ceci peut se traduire par la survenue d'une étreinte fatale (*deadlock*). Si la résolution de l'étreinte fatale impacte la fonction E-Maj, le *deadlock* est intercepté et la pose de verrou est automatiquement réitérée, avec un maximum de 5 tentatives. En revanche, les tables du groupe continuent à être accessibles en lecture pendant l'opération.

Si des tables du groupe à traiter possèdent des *triggers* (déclencheurs), ceux-ci sont, par défaut, temporairement désactivés pendant l'opération. Avec la fonction *emaj_ignore_app_trigger()*, on peut enregistrer des triggers comme « ne devant pas être automatiquement désactivés lors du rollback » (plus de détails *ici*).

Si une table impactée par le rollback possède une clé étrangère (*foreign key*) ou est référencée dans une clé étrangère appartenant à une autre table, alors la présence de cette clé étrangère est prise en compte par l'opération de rollback. Si le contrôle des clés créées ou modifiées par le rollback ne peut être différé en fin d'opération (contrainte non déclarée *DEFERRABLE*), alors cette clé étrangère est supprimée en début de rollback puis recréée en fin de rollback.

Contrairement à la fonction *emaj_rollback_group()*, à l'issue de l'opération de rollback, les données des tables de log qui concernent les mises à jour annulées, ainsi que les éventuelles marques postérieures à la marque référencée dans la commande de rollback sont conservées.

De plus, en début et en fin d'opération, la fonction pose automatiquement sur le groupe deux marques, nommées :

- "*RLBK_<marque.du.rollback>_<heure_du_rollback>_START*"
- "*RLBK_<marque.du.rollback>_<heure_du_rollback>_DONE*"

où *<heure_du_rollback>* représente l'heure de début de la transaction effectuant le rollback, exprimée sous la forme « *heures.minutes.secondes.millisecondes* ».

Lorsque le volume de mises à jour à annuler est important et que l'opération de rollback est longue, il est possible de suivre l'avancement de l'opération à l'aide de la fonction *emaj_rollback_activity()* ou du client *emajRollbackMonitor.php*.

Les opérations de rollback sont historisées dans la table *emaj_rlbk*. L'état final des opérations de rollback est accessible dans les colonnes *rlbk_status* et *rlbk_msg* de cette table *emaj_rlbk*.

A l'issue du rollback, il est possible de poursuivre les traitements de mises à jour, de poser d'autres marques et éventuellement de procéder à un nouveau rollback sur une marque quelconque, y compris la marque automatiquement posée en début de rollback, pour annuler ce dernier, ou encore une ancienne marque postérieure à la marque utilisée pour le rollback. oDes rollbacks de différents types (*logged* / *unlogged*) peuvent être exécutés en séquence. on peut ainsi procéder à l'enchaînement suivant :

```
* Pose de la marque M1
* ...
* Pose de la marque M2
* ...
* Logged rollback à M1 (générant les marques *RLBK_M1_<heure>_STRT*, puis *RLBK_M1_
↪<heure>_DONE*)
* ...
* Rollback à RLBK_M1_<heure>_DONE (pour annuler le traitement d'après rollback)
* ...
* Rollback à RLBK_M1_<heure>_STRT (pour finalement annuler le premier rollback)
```

Une *fonction de « consolidation »* de « *rollback tracé* » permet de transformer un rollback annulable en rollback simple.

Plusieurs groupes de tables peuvent être « rollbackés » en même temps, en utilisant la fonction *emaj_logged_rollback_groups()*

```
SELECT * FROM emaj.emaj_logged_rollback_groups ('<tableau.des.groupes>', '<nom.de.
↪marque>', <est_altération_groupe_permise>);
```

La marque indiquée doit strictement correspondre à un même moment dans le temps pour chacun des groupes listés. En d'autres termes, cette marque doit avoir été posée par l'appel d'une même fonction *emaj_set_mark_groups()*.

Plus d'information sur les *fonctions multi-groupes*.

Une ancienne version de ces fonctions ne comportait pas de troisième paramètre et retournait un simple entier correspondant au nombre de tables et séquences effectivement traitées :

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

SELECT emaj.emaj_logged_rollback_groups('<tableau.des.groupes>', '<nom.de.marque>');
```

Ces 2 fonctions sont dépréciées et sont amenées à être supprimées dans une prochaine version.

12.6 Arrêt d'un groupe de tables

Lorsqu'on souhaite arrêter l'enregistrement des mises à jour des tables d'un groupe, il est possible de désactiver le log par la commande SQL

```
SELECT emaj.emaj_stop_group('<nom.du.groupe>'[, '<nom.de.marque'>]);
```

La fonction retourne le nombre de tables et de séquences contenues dans le groupe.

La fonction pose automatiquement une marque correspondant à la fin de l'enregistrement. Si le paramètre représentant cette marque n'est pas spécifié ou s'il est vide ou *NULL*, un nom est automatiquement généré : « *STOP_%* », où le caractère "*%*" représente l'heure courante, au format *hh.mn.ss.mmmm*.

L'arrêt d'un groupe de table désactive simplement les triggers de log des tables applicatives du groupe. La pose de verrous de type *SHARE ROW EXCLUSIVE* qu'entraîne cette opération peut se traduire par la survenue d'une étreinte fatale (*deadlock*). Si la résolution de l'étreinte fatale impacte la fonction E-Maj, le deadlock est intercepté et la pose de verrou est automatiquement réitérée, avec un maximum de 5 tentatives.

En complément, la fonction *emaj_stop_group()* passe le statut des marques à l'état « supprimé ». Il n'est dès lors plus possible d'exécuter une commande de rollback, même si aucune mise à jour n'est intervenue sur les tables entre l'exécution des deux fonctions *emaj_stop_group()* et *emaj_rollback_group()*.

Pour autant, le contenu des tables de log et des tables internes d'E-Maj peut encore être visualisé.

A l'issue de l'arrêt d'un groupe, celui-ci redevient inactif.

Exécuter la fonction *emaj_stop_group()* sur un groupe de tables déjà arrêté ne génère pas d'erreur. Seul un message d'avertissement est retourné.

Plusieurs groupes de tables peuvent être arrêtés en même temps, en utilisant la fonction *emaj_stop_groups()*

```
SELECT emaj.emaj_stop_groups('<tableau.des.groupes>'[, '<nom.de.marque'>]);
```

Plus d'information sur les *fonctions multi-groupes*.

CHAPITRE 13

Modification des groupes de tables

13.1 Généralités

Plusieurs types d'événements peuvent rendre nécessaire la modification d'un groupe de tables :

- la composition du groupe de tables change, avec l'ajout ou la suppression de tables ou de séquence dans le groupe,
- un des paramètres liés à une table change dans la configuration E-Maj (priorité, tablespace,...),
- une ou plusieurs tables applicatives appartenant au groupe de tables voient leur structure évoluer (ajout ou suppression de colonnes, changement de type de colonne,...),
- une table ou une séquence change de nom ou de schéma.

Lorsque la modification touche un groupe de tables en état *LOGGING*, il peut être nécessaire de sortir temporairement la table ou séquence concernée de son groupe de tables, avec des impacts sur les éventuelles opérations postérieures de rollback E-Maj.

Le plus souvent, en fonction de l'utilisation ou non de la *table de configuration emaj_group_def*, les modifications peuvent être réalisées :

- soit en dynamique, au travers de fonctions dédiées,
- soit en modifiant le contenu de la table *emaj_group_def* , puis en exécutant une fonction *emaj_alter_group()*,

Le tableau suivant liste les actions possibles, en fonction de la méthode choisie.

Actions	Ajustement dynamique	Ajustement par emaj_group_def
Ajouter une table/séquence à un groupe	Fonctions dédiées	emaj_alter_group()
Supprimer une table/séquence d'un groupe	Fonctions dédiées	emaj_alter_group()
Déplacer une table/séquence vers un autre groupe	Fonctions dédiées	emaj_alter_group()
Changer le tablespace de la table ou de l'index de log	Fonctions dédiées	emaj_alter_group()
Changer la priorité E-Maj d'une table	Fonctions dédiées	emaj_alter_group()
Réparer une table	Sortie du groupe + Ajout dans le groupe	emaj_alter_group()
Renommer une table	Sortie du groupe + ALTER TABLE + Ajout	Sortie du groupe + ALTER TABLE + Ajout
Renommer une séquence	Sortie du groupe + ALTER SEQUENCE + Ajout	Sortie du groupe + ALTER SEQUENCE + Ajout
Changer le schéma d'une table	Sortie du groupe + ALTER TABLE + Ajout	Sortie du groupe + ALTER TABLE + Ajout
Changer le schéma d'une séquence	Sortie du groupe + ALTER SEQUENCE + Ajout	Sortie du groupe + ALTER SEQUENCE + Ajout
Renommer une colonne d'une table	Sortie du groupe + ALTER TABLE + Ajout	Sortie du groupe + ALTER TABLE + Ajout
Changer la structure d'une table	Sortie du groupe + ALTER TABLE + Ajout	Sortie du groupe + ALTER TABLE + Ajout
Autres formes d'ALTER TABLE	Sans impact E-Maj	
Autres formes d'ALTER SEQUENCE	Sans impact E-Maj	

Les modifications de composition de groupes de tables en état *LOGGING* peuvent avoir des conséquences sur les opérations de rollback E-Maj ou de génération de scripts SQL (voir plus bas).

D'une manière générale, même si le groupe de tables est en état *LOGGING*, une opération de rollback E-Maj ciblant une marque antérieure à une modification de groupes de tables ne procède PAS automatiquement à une annulation de ces changements. Néanmoins, l'administrateur a la possibilité d'appliquer lui même les changements permettant de remettre uns structure de groupe de tables à un état antérieur.

13.2 Méthode « Ajustement dynamique »

Quelques fonctions permettent d'ajuster dynamiquement le contenu des groupes de tables sans modification de la table *emaj_group_def*.

13.2.1 Ajouter des tables ou des séquences à un groupe

Les fonctions d'*assignation d'une ou plusieurs tables ou séquences* à un groupe de tables, utilisées pour la création des groupes, sont utilisables également au cours de la vie du groupe.

Lors de l'exécution des fonctions, les groupes de tables concernés peuvent être en état *IDLE* ou *LOGGING*.

Lorsque le groupe de tables est actif (état *LOGGING*), un verrou exclusif est posé sur chaque table du groupe.

Lorsque le groupe de table est actif, une marque est également posée. Son nom prend la valeur du dernier paramètre fourni lors de l'appel de la fonction. Ce paramètre est optionnel. S'il n'est pas fourni, le nom de la marque posée est

généré avec un préfixe *ASSIGN*.

13.2.2 Retirer des tables de leur groupe de tables

Les 3 fonctions suivantes permettent de retirer une ou plusieurs tables de leur groupe de tables :

```
SELECT emaj.emaj_remove_table('<schéma>', '<table>' [,'<marque>'] );
```

ou

```
SELECT emaj.emaj_remove_tables('<schéma>', '<tableau.de.tables>' [,'<marque>'] );
```

ou

```
SELECT emaj.emaj_remove_tables('<schéma>', '<filtre.de.tables.à.inclure>', '<filtre.
↪de.tables.à.exclure>' [,'<marque>'] );
```

Leur fonctionnement est identique aux fonctions d'assignation de tables.

Quand plusieurs tables sont sorties, celles-ci ne proviennent pas nécessairement d'un même groupe de tables d'origine.

Lorsque le ou les groupes de tables d'origine sont actifs et que la marque n'est pas fournie en paramètre, le nom de la marque posée est généré avec un préfixe *REMOVE*.

13.2.3 Retirer des séquences de leur groupe de tables

Les 3 fonctions suivantes permettent de retirer une ou plusieurs séquences de leur groupe de tables :

```
SELECT emaj.emaj_remove_sequence('<schéma>', '<séquence>' [,'<marque>'] );
```

ou

```
SELECT emaj.emaj_remove_sequences('<schéma>', '<tableau.de.séquences>' [,'<marque>']␣
↪);
```

ou

```
SELECT emaj.emaj_remove_sequences('<schéma>', '<filtre.de.séquences.à.inclure>', '
↪<filtre.de.séquences.à.exclure>' [,'<marque>'] );
```

Leur fonctionnement est identique aux fonctions d'assignation de séquences.

Quand plusieurs séquences sont sorties, celles-ci ne proviennent pas nécessairement d'un même groupe de tables d'origine.

Lorsque le groupe de tables est actif et que la marque n'est pas fournie en paramètre, le nom de la marque posée est généré avec un préfixe *REMOVE*.

13.2.4 Déplacer des tables vers un autre groupe de tables

3 fonctions permettent de déplacer une ou plusieurs tables vers un autre groupe de tables :

```
SELECT emaj.emaj_move_table('<schéma>', '<table>', 'nouveau.groupe' [,'<marque>'] );
```

ou

```
SELECT emaj.emaj_move_tables('<schéma>', '<tableau.de.tables>', 'nouveau.groupe' [,'
↪<marque>'] );
```

ou

```
SELECT emaj.emaj_move_tables('<schéma>', '<filtre.de.tables.à.inclure>', '<filtre.de.
↪tables.à.exclure>', 'nouveau.groupe' [,'<marque>'] );
```

Quand plusieurs tables sont déplacées, celles-ci ne proviennent pas nécessairement d'un même groupe de tables d'origine.

Lorsque le ou les groupes de tables d'origine sont actifs et que la marque n'est pas fournie en paramètre, le nom de la marque posée est généré avec un préfixe *MOVE*.

13.2.5 Déplacer des séquences vers un autre groupe de tables

3 fonctions permettent de déplacer une ou plusieurs séquences vers un autre groupe de tables :

```
SELECT emaj.emaj_move_sequence('<schéma>', '<séquence>', 'nouveau.groupe' [,'<marque>
↪'] );
```

ou

```
SELECT emaj.emaj_move_sequences('<schéma>', '<tableau.de.séquences>', 'nouveau.groupe
↪' [,'<marque>'] );
```

ou

```
SELECT emaj.emaj_move_sequences('<schéma>', '<filtre.de.séquences.à.inclure>', '
↪<filtre.de.séquences.à.exclure>', 'nouveau.groupe' [,'<marque>'] );
```

Quand plusieurs séquences sont déplacées, celles-ci ne proviennent pas nécessairement d'un même groupe de tables d'origine.

Lorsque le groupe de tables est actif et que la marque n'est pas fournie en paramètre, le nom de la marque posée est généré avec un préfixe *MOVE*.

13.2.6 Modifier les propriétés de tables

3 fonctions permettent de modifier les propriétés d'une table ou de plusieurs tables d'un même schéma

```
SELECT emaj.emaj_modify_table('<schéma>', '<table>', '<propriétés.modifiées>' [,'
↪<marque>']);
```

ou

```
SELECT emaj.emaj_modify_tables('<schéma>', '<tableau.de.tables>', '<propriétés.
↪modifiées>' [,'<marque>']);
```

ou

```
SELECT emaj.emaj_modify_tables('<schéma>', '<filtre.de.tables.à.inclure>', '<filtre.
↪de.tables.à.exclure>','<propriétés.modifiées>' [,'<marque>']);
```

Le paramètre <propriétés.modifiées> est de type JSONB. Ses champs élémentaires sont les mêmes que pour le paramètre <propriétés> des *fonctions d'assignation de tables*. Mais ce paramètre <propriétés.modifiées> ne contient que les propriétés … à modifier. Les propriétés non valorisées restent inchangées. On peut affecter la valeur par défaut d'une propriété en la valorisant avec un *NULL* (le null *JSON*).

Les fonctions retournent le nombre de tables ayant subi au moins une modification de propriété.

Lorsque le groupe de tables est actif et que la marque n'est pas fournie en paramètre, le nom de la marque posée est généré avec un préfixe *MODIFY*.

13.3 Modification par ajustement de la table emaj_group_def

Pour éviter de devoir supprimer puis recréer complètement un ou plusieurs groupes de tables après modification de la table *emaj_group_def*, une fonction permet de traiter uniquement les impacts de ces modifications :

```
SELECT emaj.emaj_alter_group('<nom.du.groupe>' [,'<marque>']);
```

La fonction retourne le nombre de tables et de séquences dorénavant contenues dans le groupe de tables.

La fonction *emaj_alter_group()* recrée également les objets E-Maj qui pourraient manquer (table de log, fonction, . . .). Elle supprime et/ou crée les schémas de log, en fonction des besoins.

Si le groupe de table est en état *IDLE*, le contenu de ses tables de log est purgé.

Si le groupe de table est en état *LOGGING*, la fonction :
- pose un verrou de type ROW EXCLUSIVE sur chaque table applicative du groupe de tables,
- pose une marque dont le nom peut être fourni en paramètre.

Si le paramètre représentant la marque n'est pas spécifié, ou s'il est vide ou *NULL*, un nom est automatiquement généré : *ALTER_%*, où le caractère "%" représente l'heure courante, au format « hh.mn.ss.mmmm ».

Toutes les actions enchaînées par la fonction *emaj_alter_group()* sont exécutées au sein d'une unique transaction. En conséquence, si une erreur survient durant l'opération, le groupe de tables se retrouve dans son état initial.

Il est possible d'anticiper la mise à jour de la table *emaj_group_def*, alors que le groupe de tables est encore actif. Cette mise à jour ne prendra bien sûr effet qu'à l'issue de l'exécution de la fonction *emaj_alter_group()*.

Plusieurs groupes de tables peuvent être modifiés en même temps, en utilisant la fonction *emaj_alter_groups()* :

```
SELECT emaj.emaj_alter_groups('<tableau.des.groupes>' [,'<marque>']);
```

Cette fonction permet notamment de déplacer une table ou une séquence d'un groupe de tables à un autre dans une même opération.

La syntaxe de représentation des tableaux de groupes de tables est présentée *ici*.

13.4 Incidence des ajouts ou suppressions de tables et séquences dans un groupe en état *LOGGING*

Prudence : Quand une table ou une séquence est détachée de son groupe de tables, toute opération de rollback ultérieure sur ce groupe sera sans effet sur cet objet.

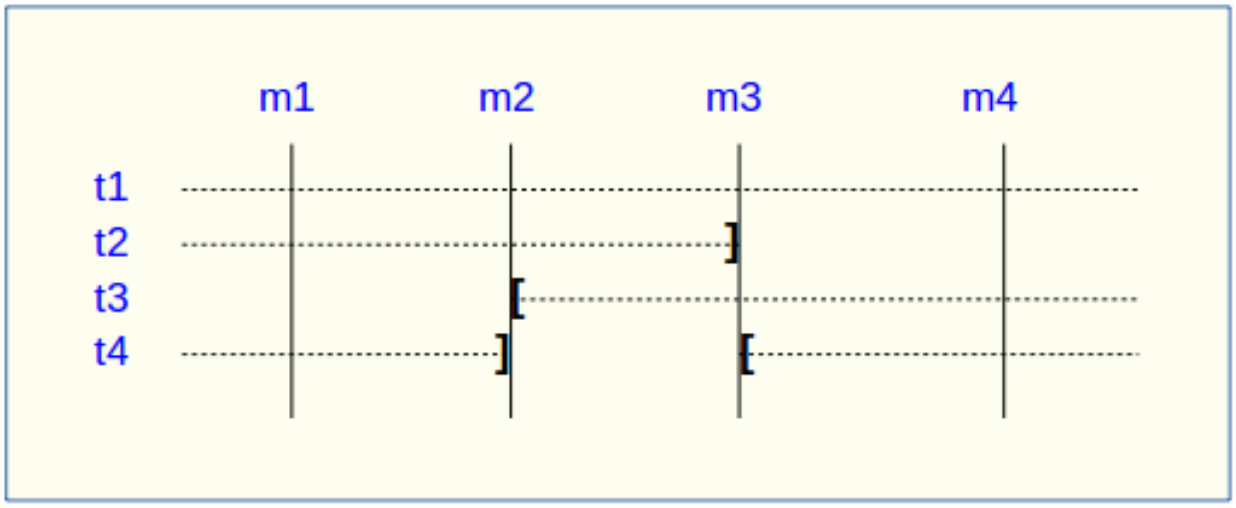
Une fois la table ou la séquence applicative décrochée de son groupe de tables, elle peut être modifiée (*ALTER*) ou supprimée (*DROP*). Les historiques liés à l'objet (logs, trace des marques,. . .) sont conservés pour examen éventuel.

Ils restent néanmoins associés à l'ancien groupe d'appartenance de l'objet. Pour éviter toute confusion, les tables de log sont renommées, avec l'ajout dans le nom d'un suffixe numérique. Ces logs et traces des marques ne seront supprimés que par les opérations de *réinitialisation du groupe de tables* ou par les *suppressions des plus anciennes marques* du groupe.

Prudence : Quand une table ou une séquence est ajoutée à un groupe de tables actif, celle-ci est ensuite traitée par les éventuelles opérations de rollback. Mais si l'opération de rollback cible une marque posée avant l'ajout de la table ou de la séquence dans le groupe, la table ou la séquence sera remise dans l'état qu'elle avait au moment où elle a été ajoutée au groupe, et un message d'avertissement est généré. En revanche une telle table ou séquence ne sera pas traitée par une fonction de génération de script SQL si la marque de début souhaitée est antérieure à l'ajout de la table dans le groupe.

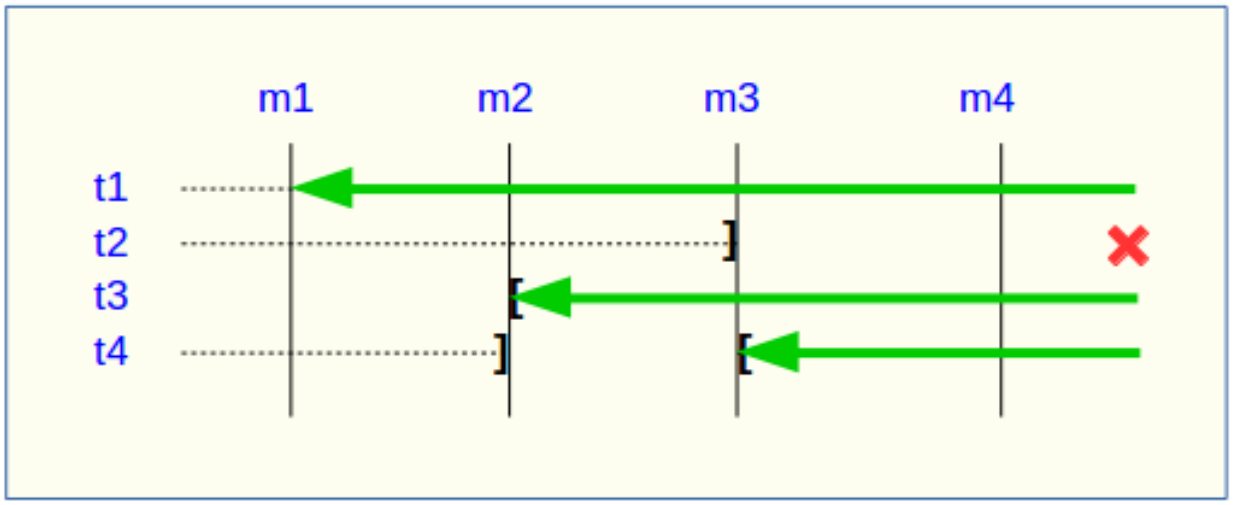
Quelques graphiques permettent de visualiser plus facilement les conséquences de l'ajout ou la suppression d'une table ou d'une séquence dans un groupe de tables actif.

Prenons 4 tables affectées à un groupe (t1 à t4) et 4 marques posées au fil du temps (m1 à m4). En m2, t3 a été ajoutée au groupe et t4 en a été retirée. En m3, t2 a été retirée du groupe alors que t4 y a été remis.



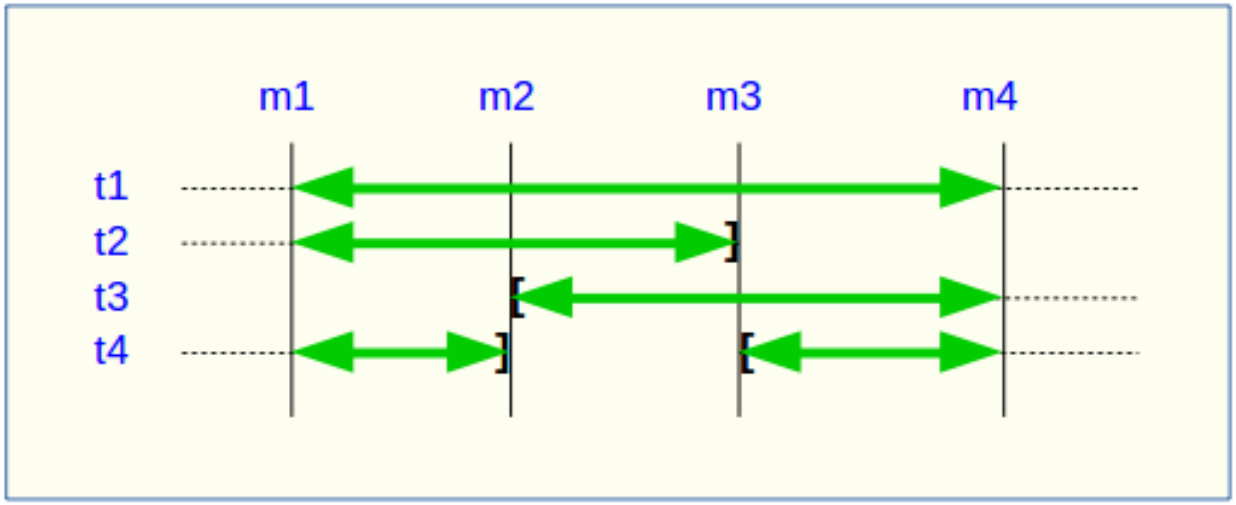
Un rollback à la marque m1 :

- traiterait la table t1,
- **NE** traiterait **PAS** la table t2, faute de log après m3,
- traiterait la table t3, mais en ne remontant que jusqu'à m2,
- traiterait la table t4, mais en ne remontant que jusqu'à m3, faute de log entre m2 et m3.



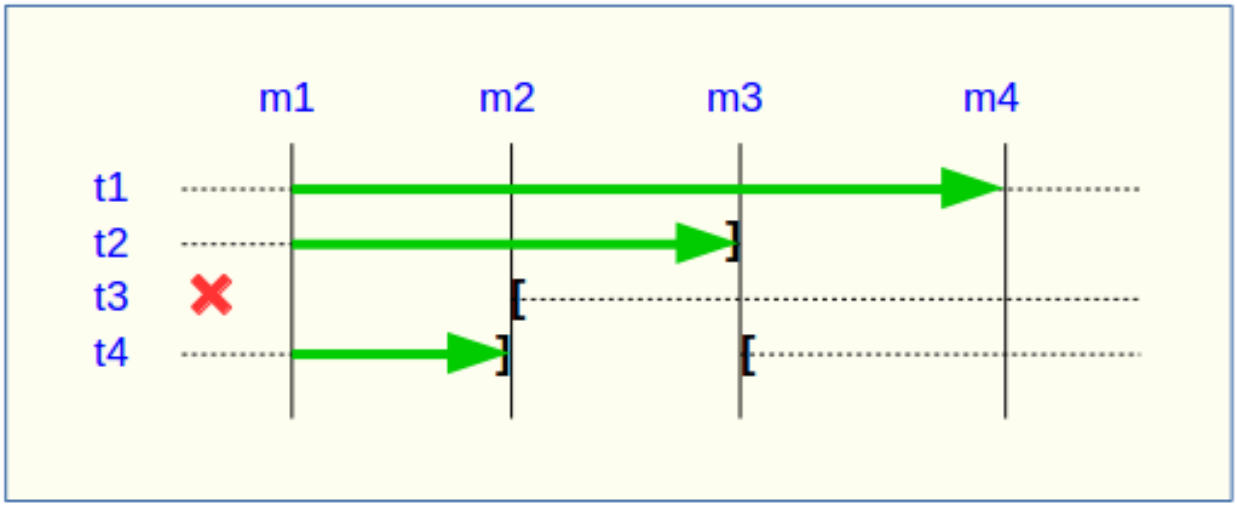
Une restitution de statistiques entre les marques m1 et m4 produirait :

- 1 ligne pour t1 (m1,m4),
- 1 ligne pour t2 (m1,m3),
- 1 ligne pour t3 (m2,m4),
- 2 lignes pour t4 (m1,m2) et (m3,m4).



La génération d'un script SQL pour l'intervalle m1 à m4 :

- traiterait la table t1,
- traiterait la table t2, mais en n'allant pas au-delà de m3,
- **NE** traiterait **PAS** la table t3, faute de log avant m2,
- traiterait la table t4, mais en n'allant pas au-delà de m2, faute de log entre m2 et m3.



Si la structure d'une table applicative a été modifiée par mégarde alors qu'elle se trouvait dans un groupe de tables actif, les opérations de pose de marque et de rollback seront bloquées par les contrôles internes d'E-Maj. On peut éviter de devoir arrêter, modifier puis relancer le groupe de tables en retirant la table concernée de son groupe puis en la rajoutant.

Quand une table change de groupe d'affectation, l'incidence sur la capacité de générer un script SQL ou de procéder à un rollback des groupes de tables source et destination est similaire à ce que serait la suppression de la table du groupe source puis son ajout dans le groupe destination.

13.5 Réparation de groupe de tables

Même si les triggers sur événements mis en place avec E-Maj limitent les risques, il peut arriver que des composants E-Maj supportant une table applicative (table, fonction ou trigger de log) soient supprimés. Le groupe de tables contenant cette table ne peut alors plus fonctionner correctement.

Pour résoudre le problème sans arrêter le groupe de tables (et ainsi perdre le bénéfice des logs enregistrés), il est possible de sortir puis réintégrer la table de son groupe de tables en le laissant actif. Pour ce faire, il suffit d'enchaîner les 2 commandes :

```
SELECT emaj.emaj_remove_table('<schéma>', '<table>' [,'<marque>']);

SELECT emaj.emaj_assign_table('<schéma>', '<table>', '<groupe>' [,'propriétés' [,'
↪<marque>']] );
```

Naturellement, une fois la table sortie de son groupe, le contenu des logs associés n'est plus exploitable pour un éventuel rollback ou une éventuelle génération de script.

Néanmoins, si la séquence de log est absente (cas de figure hautement improbable) et que le groupe de tables est en état *LOGGING*, la réparation nécessite de *forcer l'arrêt du groupe* avant de sortir puis réassigner la table.

Il peut arriver également qu'une table ou séquence applicative soit supprimée accidentellement avant d'avoir été sortie de son groupe de tables. Dans ce cas, on pourra sortir à posteriori cette table ou cette séquence de son groupe de tables, même si celui-ci est actif

- soit en exécutant uniquement la fonction *emaj_remove_table()* ou *emaj_remove_sequence()* appropriée,
- soit en enchaînant les 2 étapes de suppression de la ligne correspondant à la table/séquence dans la table *emaj_group_def*, et d'appel de la fonction *emaj_alter_group()* pour le groupe de tables concerné.

13.6 Combiner gestion dynamique des groupes de tables et configuration par emaj_group_def

En principe, il faut choisir une fois pour toute la méthode de gestion du contenu des groupes de tables : gestion dynamique ou par la table de configuration *emaj_group_def*. Il est néanmoins possible de mixer les 2 méthodes.

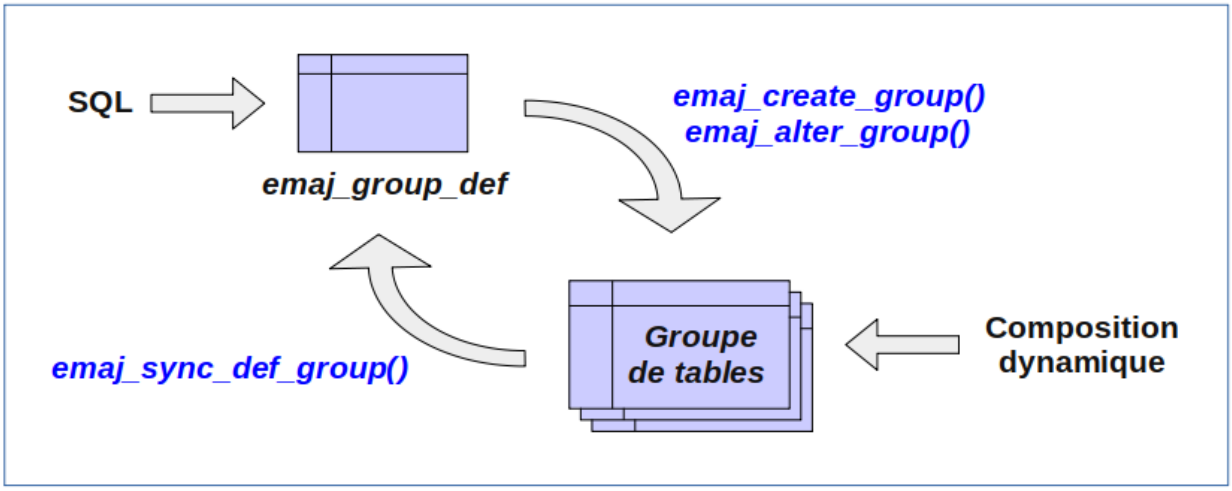
La difficulté réside alors dans le fait qu'une fois effectuées des modifications en dynamique du contenu des groupes de tables, la table *emaj_group_def* ne reflète plus la configuration courante des groupes.

Pour éviter de reporter manuellement les modifications apportées en dynamique dans la table *emaj_group_def*, sans garantie de fiabilité, l'administrateur E-Maj peut, pour un groupe de tables donné, synchroniser le contenu de la table *emaj_group_def* à partir de la situation courante. Pour cela, il peut exécuter :

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

La fonction retourne le nombre de tables et séquences contenues dans le groupe de tables traité.

Le schéma suivant représente les flux de modifications possibles :



CHAPITRE 14

Autres fonctions de gestion des groupes

14.1 Réinitialisation des tables de log d'un groupe

En standard, et sauf indication contraire, les tables de log sont vidées lors du démarrage du groupe de tables auquel elles appartiennent. En cas de besoin, il est néanmoins possible de réinitialiser ces tables de log avec la commande SQL suivante

```
SELECT emaj.emaj_reset_group('<nom.du.groupe>');
```

La fonction retourne le nombre de tables et de séquences contenues dans le groupe.

Pour réinitialiser les tables de log d'un groupe, ce dernier doit bien sûr être à l'état inactif (« *IDLE* »).

14.2 Commentaires sur les groupes

Il est possible de positionner un commentaire sur un groupe quelconque. Pour se faire, il suffit d'exécuter la requête suivante

```
SELECT emaj.emaj_comment_group('<nom.du.groupe>', '<commentaire>');
```

La fonction ne retourne aucune donnée.

Pour modifier un commentaire, il suffit d'exécuter à nouveau la fonction pour le même groupe de tables, avec le nouveau commentaire.

Pour supprimer un commentaire, il suffit d'exécuter la fonction avec une valeur *NULL* pour le paramètre commentaire.

Les commentaires sont stockés dans la colonne *group_comment* de la table *emaj.emaj_group* qui décrit les groupes.

14.3 Protection d'un groupe de tables contre les rollbacks

Il peut être utile à certains moments de se protéger contre des rollbacks intempestifs de groupes de tables, en particulier sur des bases de données de production. Deux fonctions répondent à ce besoin.

La fonction *emaj_protect_group()* pose une protection sur un groupe de tables.

```
SELECT emaj.emaj_protect_group('<nom.du.groupe>');
```

La fonction retourne l'entier 1 si le groupe de tables n'était pas déjà protégé, ou 0 s'il était déjà protégé.

Une fois le groupe de tables protégé, toute tentative de rollback, tracé ou non, sera refusée.

Un groupe de tables de type « *audit-seul* » ou dans un état « inactif » ne peut être protégé.

Au démarrage d'un groupe de tables, ce dernier n'est pas protégé. Lorsqu'il est arrêté, un groupe de tables protégé contre les rollbacks perd automatiquement sa protection.

La fonction *emaj_unprotect_group()* ôte une protection existante sur un groupe de tables.

```
SELECT emaj.emaj_unprotect_group('<nom.du.groupe>');
```

La fonction retourne l'entier 1 si le groupe de table était protégé au préalable, ou 0 s'il n'était pas déjà protégé.

Un groupe de tables de type « *audit-seul* » ne peut être déprotégé.

Une fois la protection d'un groupe de tables ôtée, il devient à nouveau possible d'effectuer tous types de rollback sur le groupe.

Un mécanisme de *protection au niveau des marques* complète ce dispositif.

14.4 Arrêt forcé d'un groupe de tables

Il peut arriver qu'un groupe de tables endommagé ne puisse pas être arrêté. C'est par exemple le cas si une table applicative du groupe de tables a été supprimée par inadvertance alors que ce dernier était actif. Si les fonctions usuelles *emaj_stop_group()* ou *emaj_stop_groups()* retournent une erreur, il est possible de forcer l'arrêt d'une groupe de tables à l'aide de la fonction *emaj_force_stop_group()*.

```
SELECT emaj.emaj_force_stop_group('<nom.du.groupe>');
```

La fonction retourne le nombre de tables et de séquences contenues dans le groupe.

Cette fonction *emaj_force_stop_group()* effectue le même traitement que la fonction *emaj_stop_group()*, Elle présente néanmoins les différences suivantes :

- elle gère les éventuelles absences des tables et triggers E-Maj à désactiver, des messages de type « *Warning* » étant générés dans ces cas,
- elle ne pose PAS de marque d'arrêt.

Une fois la fonction exécutée, le groupe de tables est en état « *IDLE* ». Il peut alors être supprimé ou modifié avec les fonctions *emaj_alter_group()* ou *emaj_drop_group()*.

Il est recommandé de n'utiliser cette fonction qu'en cas de réel besoin.

14.5 Suppression forcée d'un groupe de tables

Il peut arriver qu'un groupe de tables endommagé ne puisse pas être arrêté. Mais n'étant pas arrêté, il est impossible de le supprimer. Pour néanmoins pouvoir supprimer un groupe de tables en état actif, une fonction spéciale est disponible.

```
SELECT emaj.emaj_force_drop_group('<nom.du.groupe>');
```

La fonction retourne le nombre de tables et de séquences contenues dans le groupe.

Cette fonction *emaj_force_drop_group()* effectue le même traitement que la fonction *emaj_drop_group()*, mais sans contrôler l'état du groupe au préalable. Il est recommandé de n'utiliser cette fonction qu'en cas de réel besoin.

Note : Depuis la création de la fonction *emaj_force_stop_group()*, cette fonction *emaj_force_drop_group()* devient en principe inutile. Elle est susceptible de disparaître dans une future version d'E-Maj.

14.6 « Consolidation » d'un rollback tracé

Suite à l'exécution d'un « *rollback tracé* », et une fois que l'enregistrement de l'opération de rollback devient inutile, il est possible de « consolider » ce rollback, c'est à dire, en quelque sorte, de le transformer en « *rollback non tracé* ». A l'issue de l'opération de consolidation, les logs entre la marque cible du rollback et la marque de fin de rollback sont supprimés. La fonction *emaj_consolidate_rollback_group()* répond à ce besoin. :

```
SELECT emaj.emaj_consolidate_rollback_group('<nom.du.groupe>', <marque.de.fin.de.
↪rollback>);
```

L'opération de rollback tracé concernée est identifiée par le nom de la marque de fin qui a été générée par le rollback. Cette marque doit toujours exister, mais elle peut avoir été renommée.

Le mot clé "*EMAJ_LAST_MARK*" peut être utilisé comme nom de marque pour indiquer la dernière marque posée.

La fonction *emaj_get_consolidable_rollbacks()* peut aider à identifier les rollbacks susceptibles d'être consolidés.

A l'image des fonctions effectuant des rollbacks, cette fonction retourne le nombre de tables et de séquence effectivement concernées par la consolidation.

Le groupe de table peut être en état « actif » ou non.

La marque cible du rollback doit également toujours exister mais elle peut avoir été renommée. Néanmoins, des marques intermédiaires peuvent avoir été supprimées.

A l'issue de la consolidation, ne sont conservées que la marque cible du rollback et la marque de fin du rollback. Les marques intermédiaires sont supprimées.

La place occupée par les lignes supprimées redeviendra réutilisable une fois que ces tables de log auront été traitées par le *VACUUM*.

Bien évidemment, une fois consolidé, un « *rollback tracé* » ne peut plus être annulé, la marque de début de rollback et les logs couvrant ce rollback étant supprimés.

L'opération de consolidation est insensible aux éventuelles protections posées sur les groupes ou les marques.

Si une base n'a pas de contraintes d'espace disque trop fortes, il peut être intéressant de remplacer un « *rollback simple* » (non tracé) par un « *rollback tracé* » suivi d'une « *consolidation* » pour que les tables applicatives soient accessibles en lecture durant l'opération de rollback, en tirant profit du plus faible niveau de verrou posé lors des rollbacks tracés.

14.7 Liste des « rollbacks consolidables »

La fonction *emaj_get_consolidable_rollbacks()* permet d'identifier les rollbacks susceptibles d'être consolidés

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

La fonction retourne un ensemble de lignes comprenant les colonnes :

Colonne	Type	Description
cons_group	TEXT	groupe de tables rollbackés
cons_target_rlbk_mark_name	TEXT	nom de la marque cible du rollback
cons_target_rlbk_mark_time_id	BIGINT	référence temporelle de la marque cible (*)
cons_end_rlbk_mark_name	TEXT	nom de la marque de fin de rollback
cons_end_rlbk_mark_time_id	BIGINT	référence temporelle de la marque de fin (*)
cons_rows	BIGINT	nombre de mises à jour intermédiaires
cons_marks	INT	nombre de marques intermédiaires

(*) identifiants de la table emaj_time_stamp contenant les dates heures des moments importants de la vie des groupes.

A l'aide de cette fonction, il est ainsi facile de consolider tous les rollbacks possibles de tous les groupes de tables d'une base de données pour récupérer le maximum d'espace disque possible :

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

La fonction *emaj_get_consolidable_rollbacks()* est utilisable par les rôles *emaj_adm* et *emaj_viewer*.

14.8 Exporter et importer des configurations de groupes de tables

Un jeu de fonctions permet d'exporter et d'importer des configurations de groupes de tables. Elles peuvent être utiles pour déployer un jeu standardisé de configuration de groupes de tables sur plusieurs bases de données ou lors de changements de version E-Maj par *désinstallation et réinstallation complète de l'extension*.

14.8.1 Export d'une configuration de groupes de tables

Deux versions de la fonction *emaj_export_groups_configuration()* exportent sous forme de structure JSON une description d'un ou plusieurs groupes de tables.

On peut écrire dans un fichier une configuration de groupes de tables par :

```
SELECT emaj_export_groups_configuration('<chemin.fichier>', <tableau.noms.groupes>);
```

Le chemin du fichier doit être accessible en écriture par l'instance PostgreSQL.

Le seconde paramètre, optionnel, liste sous forme d'un tableau les groupes de tables dont on souhaite exporter la configuration. Si le paramètre est absent ou valorisé à NULL, tous les groupes de tables existants sont exportés.

La fonction retourne le nombre de groupes de tables exportés.

Si le chemin du fichier n'est pas renseigné (i.e. est valorisé à NULL), la fonction retourne directement la structure JSON contenant la configuration des groupes de tables. Cette structure ressemble à ceci :

```
{
    "_comment": "Generated on database <db> with E-Maj version <version> at <date_
↪heure>",
    "tables_groups": [
            {
```

(suite sur la page suivante)

(suite de la page précédente)

```
            "group": "...",
            "is_rollbackable": ...,
            "comment": "...",
            "tables": [
                    {
                    "schema": "...",
                    "table": "...",
                    "priority": ...,
                    "log_data_tablespace": "...",
                    "log_index_tablespace": "...",
                    "ignored_triggers": [
                            {
                            "trigger": "...",
                            ...
                             },
                    ]
                    },
                    {
                    ...
                    }
            ],
            "sequences": [
                    {
                    "schema": "myschema1",
                    "sequence": "mytbl1",
                    },
                    {
                    ...
                    }
            ],
            },
            ...
    ]
}
```

14.8.2 Import d'une configuration de groupes de tables

Deux versions de la fonction *emaj_import_groups_configuration()* importent des groupes de tables décrits sous la forme de structure JSON.

On peut charger une configuration de groupes de tables à partir d'un fichier par :

```
SELECT emaj_import_groups_configuration('<chemin.fichier>', <tableau.noms.groupes>,
↪<modifier.groupes.démarrés>);
```

Le fichier doit être accessible par l'instance PostgreSQL.

Le fichier doit contenir une structure JSON ayant un attribut nommé « tables-groups » de type tableau, et contenant des sous-structures décrivant chaque groupe de tables, tels que décrits ci-dessus pour l'exportation de configurations de groupes de tables.

La fonction peut directement charger un fichier généré par la fonction *emaj_export_groups_configuration()*.

Le second paramètre est de type tableau et est optionnel. Il indique la liste des groupes de tables que l'on veut importer. Par défaut, tous les groupes de tables décrits dans le fichier sont importés.

Si un groupe de tables à importer n'existe pas, il est créé et ses tables et séquences lui sont assignées.

Si un groupe de tables à importer existe déjà, sa configuration est ajustée pour refléter la configuration cible. Des tables et séquences peuvent être ajoutées ou retirées, et des attributs peuvent être modifiés. Dans le cas où le groupe de tables est démarré, l'ajustement de sa configuration n'est possible que si le troisième paramètre, de type booléen, est explicitement positionné à TRUE.

La fonction retourne le nombre de groupes de tables importés.

Notons que l'importation d'un groupe de tables écrase l'éventuelle configuration précédente du groupe dans la table technique *emaj_group_def*.

Dans une variante de la fonction, le premier paramètre en entrée contient directement la structure JSON des groupes de tables à charger

```
SELECT emaj_import_groups_configuration('<structure.JSON>', <tableau.noms.groupes>,
↪<modification.groupes.démarrés>);
```

CHAPITRE 15

Fonctions de gestion des marques

15.1 Commentaires sur les marques

Il est possible de positionner un commentaire sur une marque quelconque. Pour se faire, il suffit d'exécuter la requête suivante

```
SELECT emaj.emaj_comment_mark_group('<nom.du.groupe>', '<nom.de.marque>', '
↪<commentaire>');
```

Le mot clé "*EMAJ_LAST_MARK*" peut être utilisé comme nom de marque à commenter pour indiquer la dernière marque posée.

La fonction ne retourne aucune donnée.

Pour modifier un commentaire, il suffit d'exécuter à nouveau la fonction pour le même groupe de tables et la même marque, avec le nouveau commentaire.

Pour supprimer un commentaire, il suffit d'exécuter la fonction avec une valeur *NULL* pour le paramètre commentaire.

Les commentaires sont stockés dans la colonne *mark_comment* de la table *emaj.emaj_mark* qui décrit les marques.

Les commentaires sont surtout intéressants avec l'utilisation des *clients web*. En effet, ces derniers les affichent systématiquement dans le tableau des marques d'un groupe.

15.2 Recherche de marque

La fonction *emaj_get_previous_mark_group()* permet de connaître, pour un groupe de tables, le nom de la dernière marque qui précède soit une date et une heure donnée, soit une autre marque.

```
SELECT emaj.emaj_get_previous_mark_group('<nom.du.groupe>', '<date.et.heure>');
```

ou

```
SELECT emaj.emaj_get_previous_mark_group('<nom.du.groupe>', '<marque>');
```

Dans la première forme, la date et l'heure doivent être exprimées sous la forme d'un *TIMESTAMPTZ*, par exemple le littéral *"2011/06/30 12 :00 :00 +02"*. Si l'heure fournie est strictement égale à l'heure d'une marque existante, la marque retournée sera la marque précédente.

Dans la seconde forme, le mot clé "*EMAJ_LAST_MARK*" peut être utilisé comme nom de marque pour indiquer la dernière marque posée.

15.3 Renommage d'une marque

Une marque précédemment posée par l'une des fonctions *emaj_create_group()* ou *emaj_set_mark_group()* peut être renommée avec la commande SQL

```
SELECT emaj.emaj_rename_mark_group('<nom.du.groupe>', '<nom.de.marque>', '<nouveau.
↪nom.de.marque>');
```

Le mot clé *"EMAJ_LAST_MARK"* peut être utilisé comme nom de marque à renommer pour indiquer la dernière marque posée.

La fonction ne retourne aucune donnée.

Une marque portant le nouveau nom souhaité ne doit pas déjà exister pour le groupe de tables.

15.4 Effacement d'une marque

Une marque peut également être effacée par l'intermédiaire de la commande SQL

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

Le mot clé *"EMAJ_LAST_MARK"* peut être utilisé comme nom de marque pour indiquer la dernière marque posée.

La fonction retourne la valeur 1, c'est à dire le nombre de marques effectivement effacées.

Pour qu'il reste au moins une marque après l'exécution de la fonction, l'effacement d'une marque n'est possible que s'il y a au moins 2 marques pour le groupe de tables concerné.

Si la marque effacée est la première marque pour le groupe, les lignes devenues inutiles dans les tables de log sont supprimées.

Si une table a été *détachée d'un groupe de tables* et que la marque effacée correspond à la dernière marque connue pour cette table, les logs couvrant l'intervalle de temps entre cette marque et la marque précédente sont supprimés.

15.5 Effacement des marques les plus anciennes

Pour facilement effacer en une seule opération toutes les marques d'un groupe de tables antérieures à une marque donnée, on peut exécuter la requête

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

Le mot clé *"EMAJ_LAST_MARK"* peut être utilisé comme nom de marque pour indiquer la dernière marque posée.

La fonction efface les marques antérieures à la marque spécifiée, cette dernière devenant la nouvelle première marque. Elle supprime également des tables de log toutes les données concernant les mises à jour de tables applicative antérieures à cette marque.

La fonction retourne le nombre de marques effacées.

La fonction procède également à la purge des événements les plus anciens de la table technique *emaj_hist*.

Cette fonction permet ainsi d'utiliser E-Maj sur de longues périodes sans avoir à arrêter et redémarrer les groupes, tout en limitant l'espace disque utilisé pour le log.

Néanmoins, comme cette suppression de lignes dans les tables de log ne peut utiliser de verbe SQL *TRUNCATE*, la durée d'exécution de la fonction *emaj_delete_before_mark_group()* peut être plus longue qu'un simple arrêt et relance de groupe. En contrepartie, elle ne nécessite pas de pose de verrou sur les tables du groupe concerné. Son exécution peut donc se poursuivre alors que d'autres traitements mettent à jour les tables applicatives. Seules d'autres actions E-Maj sur le même groupe de tables, comme la pose d'une nouvelle marque, devront attendre la fin de l'exécution d'une fonction *emaj_delete_before_mark_group()*.

Associées, les fonctions *emaj_delete_before_mark_group()*, et *emaj_get_previous_mark_group()* permettent d'effacer les marques antérieures à un délai de rétention. Ainsi par exemple, pour effacer toutes les marques (et supprimer les logs associés) posées depuis plus de 24 heures, on peut exécuter la requête

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

15.6 Protection d'une marque contre les rollbacks

Pour compléter le mécanisme de *protection des groupes de tables* contre les rollbacks intempestifs, il est possible de positionner des protections au niveau des marques. Deux fonctions répondent à ce besoin.

La fonction *emaj_protect_mark_group()* pose une protection sur une marque d'un groupe de tables. :

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

La fonction retourne l'entier 1 si la marque n'était pas déjà protégée, ou 0 si elle était déjà protégée.

Une fois une marque protégée, toute tentative de rollback, tracé ou non, sera refusée si elle repositionne le groupe de tables à un état antérieur à cette marque protégée.

Une marque d'un groupe de tables de type « *audit-seul* » ou en état « *inactif* » ne peut être protégée.

Lorsqu'une marque est posée, elle n'est pas protégée. Les marques protégées d'un groupe de tables perdent automatiquement leur protection lorsque ce groupe de tables est arrêté. Attention, la suppression d'une marque protégée supprime de facto la protection. Elle ne reporte pas la protection sur une marque adjacente.

La fonction *emaj_unprotect_mark_group()* ôte une protection existante sur une marque d'un groupe de tables. :

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

La fonction retourne l'entier 1 si la marque était bien protégée au préalable, ou 0 si elle n'était déjà déjà protégée.

Une marque d'un groupe de tables de type « *audit-seul* » ne peut être déprotégée.

Une fois la protection d'une marque ôtée, il devient à nouveau possible d'effectuer tous types de rollback sur une marque antérieure.

CHAPITRE 16

Fonctions statistiques

Deux fonctions permettent d'obtenir des statistiques sur le contenu des tables de log :

- *emaj_log_stat_group()* permet d'avoir rapidement une vision du nombre de mises à jour enregistrées entre deux marques, ou depuis une marque, pour chaque table d'un groupe,
- *emaj_detailed_log_stat_group()* permet d'avoir, pour un groupe de tables, une vision détaillée du nombre de mises à jour enregistrées entre deux marques, ou depuis une marque, par table, type de verbe (*INSERT/UPDATE/DELETE*) et rôle de connexion.

En complément, E-Maj fournit 2 fonctions, *emaj_estimate_rollback_group()* et *emaj_estimate_rollback_groups()*, qui permettent d'estimer la durée que prendrait un éventuel rollback d'un ou plusieurs groupes à une marque donnée.

Toutes ces fonctions statistiques sont utilisables par tous les rôles E-Maj : *emaj_adm* et *emaj_viewer*.

16.1 Statistiques générales sur les logs

On peut obtenir les statistiques globales complètes à l'aide de la requête SQL

```
SELECT * FROM emaj.emaj_log_stat_group('<nom.du.groupe>', '<marque.début>', '<marque.
↪fin>');
```

La fonction retourne un ensemble de lignes, de type *emaj.emaj_log_stat_type* et comportant les colonnes suivantes :

Column	Type	Description
stat_group	TEXT	nom du groupe de tables
stat_schema	TEXT	nom du schéma
stat_table	TEXT	nom de la table
stat_first_mark	TEXT	nom de la marque de début de période
stat_first_mark_datetime	TIMESTAMPTZ	date et heure de la marque de début de période
stat_last_mark	TEXT	nom de la marque de fin de période
stat_last_mark_datetime	TIMESTAMPTZ	date et heure de la marque de fin de période
stat_rows	BIGINT	nombre de modifications de lignes enregistrées dans la table de log associée à la table

Une valeur *NULL* ou une chaîne vide (“”), fournie comme marque de début, représente la plus ancienne marque accessible.

Une valeur *NULL* fournie comme marque de fin représente la situation courante.

Le mot clé *“EMAJ_LAST_MARK”* peut être utilisé comme nom de marque. Il représente alors la dernière marque posée.

La fonction retourne une ligne par table, même si aucune mise à jour n’est enregistrée pour la table entre les deux marques. Dans ce cas, la colonne *stat_rows* contient la valeur 0.

La plupart du temps, les colonnes *stat_first_mark*, *stat_first_mark_datetime*, *stat_last_mark* et *stat_last_mark_datetime* référencent les marques de début et de fin de période demandée. Mais elles peuvent contenir des valeurs différentes si une table a été ajoutée ou supprimée du groupe de tables pendant l’intervalle de temps demandé.

Il est possible aisément d’exécuter des requêtes plus précises sur ces statistiques. Ainsi par exemple, on peut obtenir le nombre de mises à jour par schéma applicatif avec une requête du type :

```
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)
```

L’obtention de ces statistiques ne nécessite pas le parcours des tables de log. Elles sont donc restituées rapidement.

Mais, les valeurs retournées peuvent être approximatives (en fait surestimées). C’est en particulier le cas si, entre les deux marques citées, des transactions ont mis à jour des tables avant d’être annulées.

Des statistiques peuvent être obtenues sur plusieurs groupes de tables en même temps, en utilisant la fonction *emaj_log_stat_groups()* :

```
SELECT emaj.emaj_log_stat_groups('<tableau.des.groupes>', '<marque.début>', '<marque.
↪fin>');
```

Plus d’information sur les *fonctions multi-groupes*.

16.2 Statistiques détaillées sur les logs

Le parcours des tables de log permet d'obtenir des informations plus détaillées, au prix d'un temps de réponse plus long. Ainsi, on peut obtenir les statistiques détaillées complètes à l'aide de la requête SQL

```
SELECT * FROM emaj.emaj_detailed_log_stat_group('<nom.du.groupe>', '<marque.début>', '
↪<marque.fin>');
```

La fonction retourne un ensemble de lignes, de type *emaj.emaj_detailed_log_stat_type* et comportant les colonnes suivantes :

Column	Type	Description
stat_group	TEXT	nom du groupe de tables
stat_schema	TEXT	nom du schéma
stat_table	TEXT	nom de la table
stat_first_mark	TEXT	nom de la marque de début de période
stat_first_mark_datetime	TIMESTAMPTZ	date et heure de la marque de début de période
stat_last_mark	TEXT	nom de la marque de fin de période
stat_last_mark_datetime	TIMESTAMPTZ	date et heure de la marque de fin de période
stat_role	VARCHAR(32)	rôle de connexion
stat_verb	VARCHAR(6)	verbe SQL à l'origine de la mise à jour (avec les valeurs *INSERT* / *UPDATE* / *DELETE*)
stat_rows	BIGINT	nombre de modifications de lignes enregistrées dans la table de log associée à la table

Une valeur *NULL* ou une chaîne vide (“”), fournie comme marque de début représente la plus ancienne marque accessible.

Une valeur *NULL* fournie comme marque de fin représente la situation courante.

Le mot clé *“EMAJ_LAST_MARK”* peut être utilisé comme nom de marque. Il représente alors la dernière marque posée.

Contrairement à la fonction *emaj_log_stat_group()*, *emaj_detailed_log_stat_group()* ne retourne aucune ligne pour les tables sans mise à jour enregistrée sur l'intervalle de marques demandées. La colonne *stat_rows* ne contient donc jamais de valeur 0.

La plupart du temps, les colonnes *stat_first_mark*, *stat_first_mark_datetime*, *stat_last_mark* et *stat_last_mark_datetime* référencent les marques de début et de fin de période demandée. Mais elles peuvent contenir des valeurs différentes si une table a été ajoutée ou supprimée du groupe de tables pendant l'intervalle de temps demandé.

Des statistiques détaillées peuvent être obtenues sur plusieurs groupes de tables en même temps, en utilisant la fonction *emaj_detailed_log_stat_groups()* :

```
SELECT emaj.emaj_detailed_log_stat_groups('<tableau.des.groupes>', '<marque.début>', '
↪<marque.fin>');
```

Plus d'information sur les *fonctions multi-groupes*.

16.3 Estimation de la durée d'un rollback

La fonction *emaj_estimate_rollback_group()* permet d'obtenir une estimation de la durée que prendrait le rollback d'un groupe de tables à une marque donnée. Elle peut être appelée de la façon suivante

```
SELECT emaj.emaj_estimate_rollback_group('<nom.du.groupe>', '<nom.de.marque>', <est␣
↪tracé>);
```

Le mot clé *"EMAJ_LAST_MARK"* peut être utilisé comme nom de marque. Il représente alors la dernière marque posée.

Le troisième paramètre, de type booléen, indique si le rollback à simuler est tracé ou non.

La fonction retourne un donnée de type *INTERVAL*.

Le groupe de tables doit être en état démarré (*LOGGING*) et la marque indiquée doit être utilisable pour un rollback, c'est à dire qu'elle ne doit pas être marquée comme logiquement supprimée (*DELETED*).

L'estimation de cette durée n'est qu'approximative. Elle s'appuie sur :

- le nombre de lignes à traiter dans les tables de logs, tel que le retourne la fonction *emaj_log_stat_group()*,
- des relevés de temps issus d'opérations de rollback précédentes pour les mêmes tables
- 6 *paramètres* génériques qui sont utilisés comme valeurs par défaut, lorsqu'aucune statistique n'a été enregistrée pour les tables à traiter.

Compte tenu de la répartition très variable entre les verbes *INSERT*, *UPDATE* et *DELETE* enregistrés dans les logs, et des conditions non moins variables de charge des serveurs lors des opérations de rollback, la précision du résultat restitué est faible. L'ordre de grandeur obtenu peut néanmoins donner une indication utile sur la capacité de traiter un rollback lorsque le temps imparti est contraint.

Sans statistique sur les rollbacks précédents, si les résultats obtenus sont de qualité médiocre, il est possible d'ajuster les *paramètres* génériques. Il est également possible de modifier manuellement le contenu de la table *emaj.emaj_rlbk_stat* qui conserve la durée des rollbacks précédents, en supprimant par exemple les lignes correspondant à des rollbacks effectués dans des conditions de charge inhabituelles.

La fonction *emaj_estimate_rollback_groups()* permet d'estimer la durée d'un rollback portant sur plusieurs groupes de tables

```
SELECT emaj.emaj_estimate_rollback_groups('<tableau.des.groupes>', '<nom.de.marque>',
↪<est tracé>);
```

Plus d'information sur les *fonctions multi-groupes*.

CHAPITRE 17

Fonctions d'extraction de données

Trois fonctions permettent d'extraire des données de l'infrastructure E-Maj et de les stocker sur des fichiers externes.

17.1 Vidage des tables d'un groupe

Il peut s'avérer utile de prendre des images de toutes les tables et séquences appartenant à un groupe, afin de pouvoir en observer le contenu ou les comparer. Une fonction permet d'obtenir le vidage sur fichiers des tables d'un groupe

```
SELECT emaj.emaj_snap_group('<nom.du.groupe>', '<répertoire.de.stockage>', '<options.
↪COPY>');
```

Le nom du répertoire fourni doit être un chemin absolu. Ce répertoire doit exister au préalable et avoir les permissions adéquates pour que l'instance PostgreSQL puisse y écrire.

Le troisième paramètre précise le format souhaité pour les fichiers générés. Il prend la forme d'une chaîne de caractères reprenant la syntaxe précise des options disponibles pour la commande SQL *COPY TO*.

La fonction retourne le nombre de tables et de séquences contenues dans le groupe.

Cette fonction *emaj_snap_group()* génère un fichier par table et par séquence appartenant au groupe de tables cité. Ces fichiers sont stockés dans le répertoire ou dossier correspondant au second paramètre de la fonction. D'éventuels fichiers de même nom se trouveront écrasés.

Le nom des fichiers créés est du type : <*nom.du.schema*>_<*nom.de.table/séquence*>.*snap*

D'éventuels caractères peu pratiques dans un nom de fichier, les espaces, « / », « \ », « $ », « > », « < », et « * » sont remplacés par des « _ ».

Les fichiers correspondant aux séquences ne comportent qu'une seule ligne, qui contient les caractéristiques de la séquence.

Les fichiers correspondant aux tables contiennent un enregistrement par ligne de la table, dans le format spécifié en paramètre. Ces enregistrements sont triés dans l'ordre croissant de la clé primaire.

En fin d'opération, un fichier *_INFO* est créé dans ce même répertoire. Il contient un message incluant le nom du groupe de tables et la date et l'heure de l'opération.

Il n'est pas nécessaire que le groupe de tables soit dans un état inactif, c'est-à-dire qu'il ait été arrêté au préalable.

Comme la fonction peut générer de gros ou très gros fichiers (dépendant bien sûr de la taille des tables), il est de la responsabilité de l'utilisateur de prévoir un espace disque suffisant.

Avec cette fonction, un test simple de fonctionnement d'E-Maj peut enchaîner :
- *emaj_create_group()*,
- *emaj_start_group()*,
- emaj_snap_group(<répertoire_1>),
- mises à jour des tables applicatives,
- *emaj_rollback_group()*,
- emaj_snap_group(<répertoire_2>),
- comparaison du contenu des deux répertoires par une commande *diff* par exemple.

17.2 Vidage des tables de log d'un groupe

Il est également possible d'obtenir le vidage total ou partiel sur fichiers des tables de log d'un groupe de tables. Ceci peut permettre de conserver une trace des mises à jour effectuées par un ou plusieurs traitements, à des fins d'archivage ou de comparaison entre plusieurs traitements. Pour ce faire, il suffit d'exécuter une requête

```
SELECT emaj.emaj_snap_log_group('<nom.du.groupe>', '<marque.début>', '<marque.fin>', '
↪<répertoire.de.stockage>', '<options.COPY>');
```

Un *NULL* ou une chaîne vide peuvent être utilisés comme marque de début. Ils représentent alors la première marque connue. Un *NULL* ou une chaîne vide peuvent être utilisés comme marque de fin. Ils représentent alors la situation courante.

Le mot clé *"EMAJ_LAST_MARK"* peut être utilisé comme marque de fin. Il représente alors la dernière marque posée.

Le nom du répertoire fourni doit être un chemin absolu. Ce répertoire doit exister au préalable et avoir les permissions adéquates pour que l'instance PostgreSQL puisse y écrire.

Le cinquième paramètre précise le format souhaité pour les fichiers générés. Il prend la forme d'une chaîne de caractères reprenant la syntaxe précise des options disponibles pour la commande SQL *COPY TO*.

La fonction retourne le nombre de fichiers générés.

Cette fonction *emaj_snap_log_group()* génère un fichier par table de log, contenant la partie de cette table correspond aux mises à jour effectuées entre les deux marques citées ou la marque de début et la situation courante. Le nom des fichiers créés pour chaque table est du type : <*nom.de.la.table.de.log*>.*snap*. Le plus souvent, ce nom ressemblera donc à : <*nom.du.schema*>_<*nom.de.table*>_*log.snap*

La fonction génère également deux fichiers, contenant l'état des séquences applicatives lors de la pose respective des deux marques citées, et nommés <*nom.du.groupe*>_*sequences_at*_<*nom.de.marque*>.

Si la borne de fin représente la situation courante, le nom du fichier devient <*nom.du.groupe*>_*sequences_at*_<*heure*>, l'heure étant exprimée avec un format *HH.MM.SS.mmm*.

Tous ces fichiers sont stockés dans le répertoire ou dossier correspondant au quatrième paramètre de la fonction. D'éventuels fichiers de même nom se trouveront écrasés.

D'éventuels caractères peu pratiques dans un nom de fichier, les espaces, « / », « \ », « $ », « > », « < », et « * » sont remplacés par des « _ ».

En fin d'opération, un fichier *_INFO* est créé dans ce même répertoire. Il contient un message incluant le nom du groupe de tables, les marques qui ont servi de bornes et la date et l'heure de l'opération.

Il n'est pas nécessaire que le groupe de tables soit dans un état inactif, c'est-à-dire qu'il ait été arrêté au préalable. Si aucune marque de fin n'a été spécifiée, le vidage des tables de logs est bornée par une pseudo marque posée en début

d'exécution de la fonction. Ceci garantit que, si le groupe est actif, les fichiers ne contiendront pas de mises à jour postérieures au début d'exécution de la fonction.

Comme la fonction peut générer de gros, voire très gros, fichiers (en fonction du volume des tables), il est de la responsabilité de l'utilisateur de prévoir un espace disque suffisant.

Les tables de log ont une structure qui découlent directement des tables applicatives dont elles enregistrent les mises à jour. Elles contiennent les mêmes colonnes avec les mêmes types. Mais elles possèdent aussi quelques colonnes techniques complémentaires :

La structure des tables de log est décrite *ici*.

17.3 Génération de scripts SQL rejouant les mises à jour tracées

Les tables de log contiennent toutes les informations permettant de rejouer les mises à jour. Il est dès lors possible de générer des requêtes SQL correspondant à toutes les mises à jour intervenues entre 2 marques particulières ou à partir d'une marque. C'est l'objectif de la fonction *emaj_gen_sql_group()*.

Ceci peut permettre de ré-appliquer des mises à jour après avoir restauré les tables du groupe dans l'état correspondant à la marque initiale, sans avoir à ré-exécuter aucun traitement applicatif.

Pour générer ce script SQL, il suffit d'exécuter une requête

```
SELECT emaj.emaj_gen_sql_group('<nom.du.groupe>', '<marque.début>', '<marque.fin>', '
↪<fichier>'[,<liste.tables.séquences>]);
```

Un *NULL* ou une chaîne vide peuvent être utilisés comme marque de début. Ils représentent alors la première marque connue. Un *NULL* ou une chaîne vide peuvent être utilisés comme marque de fin. Ils représentent alors la situation courante.

Le mot clé *"EMAJ_LAST_MARK"* peut être utilisé comme marque de fin. Il représente alors la dernière marque posée.

S'il est fourni, le nom du fichier de sortie doit être exprimé sous forme de chemin absolu. Le fichier doit disposer des permissions adéquates pour que l'instance postgreSQL puisse y écrire. Si le fichier existe déjà, son contenu est écrasé.

Le nom du fichier de sortie peut prendre une valeur NULL. Dans ce cas, le script SQL est préparé dans une table temporaire, accessible ensuite au travers d'une vue temporaire *emaj_sql_script*. A partir du client *psql*, on peut donc enchaîner dans une même session :

```
SELECT emaj.emaj_gen_sql_group('<nom.du.groupe>', '<marque.début>', '<marque.fin>',␣
↪NULL [,<liste.tables.séquences>]);
\copy (SELECT * FROM emaj_sql_script) TO 'fichier'
```

Cette méthode permet de générer un fichier en dehors des systèmes de fichiers accessibles par l'instance PostgreSQL.

Le dernier paramètre de la fonction *emaj_gen_sql_group()* est optionnel. Il permet de filtrer la liste des tables et séquences à traiter. Si le paramètre est omis ou a la valeur *NULL*, toutes les tables et séquences du groupe de tables sont traitées. S'il est spécifié, le paramètre doit être exprimé sous la forme d'un tableau non vide d'éléments texte, chacun d'eux représentant le nom d'une table ou d'une séquence préfixé par le nom de schéma. On peut utiliser indifféremment les syntaxes

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

ou

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

La fonction retourne le nombre de requêtes générées (hors commentaire et gestion de transaction).

Le groupe de tables peut être dans un état actif ou inactif.

Pour que le script puisse être généré, toutes les tables doivent avoir une clé primaire explicite (*PRIMARY KEY*).

Prudence : Si une liste de tables et séquences est spécifiée pour restreindre le champ d'application de la fonction *emaj_gen_sql_group()*, il est de la responsabilité de l'utilisateur de prendre en compte l'existence éventuelle de clés étrangères (*foreign keys*) pour la validité du script SQL généré par la fonction.

Les requêtes sont générées dans l'ordre d'exécution initial.

Elles sont insérées dans une transaction. Elles sont entourées d'une requête *BEGIN TRANSACTION;* et d'une requête *COMMIT;*. Un commentaire initial rappelle les caractéristiques de la génération du script : la date et l'heure de génération, le groupe de tables concerné et les marques utilisées.

Enfin, les séquences appartenant au groupe de tables sont repositionnées à leurs caractéristiques finales en fin de script.

Le fichier généré peut ensuite être exécuté tel quel par l'outil psql, pour peu que le rôle de connexion choisi dispose des autorisations d'accès adéquates sur les tables et séquences accédées.

La technique mise en œuvre aboutit à avoir des caractères antislash doublés dans le fichier de sortie. Il faut alors supprimer ces doublons avant d'exécuter le script, par exemple dans les environnement Unix/Linux par une commande du type

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

Comme la fonction peut générer un gros, voire très gros, fichier (en fonction du volume des logs), il est de la responsabilité de l'utilisateur de prévoir un espace disque suffisant.

Il est aussi de la responsabilité de l'utilisateur de désactiver d'éventuels triggers applicatifs avant d'exécuter le script généré.

La fonction *emaj_gen_sql_groups()* permet de générer des scripts SQL portant sur plusieurs groupes de tables

```
SELECT emaj.emaj_gen_sql_groups('<tableau.des.groupes>', '<marque.début>', '<marque.
↪fin>', '<fichier>'[,<liste.tables.séquences>]);
```

Plus d'information sur les *fonctions multi-groupes*.

CHAPITRE 18

Autres fonctions

18.1 Vérification de la consistance de l'environnement E-Maj

Une fonction permet de vérifier la consistance de l'environnement E-Maj. Cela consiste à vérifier l'intégrité de chaque schéma d'E-Maj et de chaque groupe de tables créé. Cette fonction s'exécute par la requête SQL suivante

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

Pour chaque schéma E-Maj (*emaj* et les schémas de log), la fonction vérifie :

- que toutes les tables, fonctions et séquences et tous les types soit sont des objets de l'extension elle-même, soit sont bien liés aux groupes de tables créés,
- qu'il ne contient ni vue, ni « *foreign table* », ni domaine, ni conversion, ni opérateur et ni classe d'opérateur.

Ensuite, pour chaque groupe de tables créé, la fonction procède aux mêmes contrôles que ceux effectués lors des opérations de démarrage de groupe, de pose de marque et de rollback (*plus de détails*).

La fonction retourne un ensemble de lignes qui décrivent les éventuelles anomalies rencontrées. Si aucune anomalie n'est détectée, la fonction retourne une unique ligne contenant le message

```
'No error detected'
```

La fonction retourne également des avertissements quand :

- une séquence associée à une colonne est assignée à un groupe de tables mais la table associée ne fait pas partie de ce groupe de tables,
- une table d'un groupe est liée à une autre table par une clé étrangère, mais la table associée ne font pas partie du même groupe de tables.

La fonction *emaj_verify_all()* peut être exécutée par les rôles membres de *emaj_adm* et *emaj_viewer*.

Si des anomalies sont détectées, par exemple suite à la suppression d'une table applicative référencée dans un groupe, les mesures appropriées doivent être prises. Typiquement, les éventuelles tables de log ou fonctions orphelines doivent être supprimées manuellement.

18.2 Exporter et importer des configurations de paramètres

Deux jeux de fonctions permettent de respectivement exporter et importer des jeux de paramètres. Elles peuvent être utiles pour déployer un jeu standardisé de paramètres sur plusieurs bases de données ou lors de *changements de version E-Maj* par désinstallation et réinstallation complète de l'extension.

18.2.1 Export d'une configuration de paramètres

Deux versions de la fonction *emaj_export_parameters_configuration()* exportent sous forme de structure JSON l'ensemble des paramètres de la configuration présents dans la table *emaj_param*, à l'exception du paramètre de clé *« emaj_version »*, lié à l'extension *emaj* elle-même et qui n'est pas à proprement parler un paramètre de configuration.

On peut écrire dans un fichier les données de paramétrage par :

```
SELECT emaj_export_parameters_configuration('<chemin.fichier>');
```

Le chemin du fichier doit être accessible en écriture par l'instance PostgreSQL.

La fonction retourne le nombre de paramètres exportés.

Si le chemin du fichier n'est pas renseigné, la fonction retourne directement la structure JSON contenant les valeurs de paramètres. Cette structure ressemble à ceci :

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

18.2.2 Import d'une configuration de paramètres

Deux versions de la fonction *emaj_import_parameters_configuration()* importent des paramètres sous forme de structure JSON dans la table *emaj_param*.

On peut lire dans un fichier des paramètres à charger par :

```
SELECT emaj_import_parameters_configuration('<chemin.fichier>', <suppression.
↪configuration.courante>);
```

Le chemin du fichier doit être accessible par l'instance PostgreSQL.

Le fichier doit contenir une structure JSON ayant un attribut nommé *« parameters »* de type tableau, et contenant des sous-structures avec les attributs *« key »* et *« value »*

```
{"parameters": [
  {
    "key": "...",
```

(suite sur la page suivante)

(suite de la page précédente)

```
    "value": "..."
  },
  {
        ...
  }
]}
```

Si un paramètre n'a pas d'attribut *« value »* ou si cet attribut est valorisé à *NULL*, le paramètre n'est pas inséré dans la table *emaj_param*, et est supprimé s'il existait déjà dans la table. En conséquence, la valeur par défaut du paramètre sera utilisée par l'extension *emaj*.

La fonction peut directement charger un fichier généré par la fonction *emaj_export_parameters_configuration()*.

S'il est présent, le paramètre de clé *« emaj_version »* n'est pas traité.

Le second paramètre, de type booléen, est optionnel. Il indique si l'ensemble de la configuration présente doit être supprimée avant le chargement. Par défaut, sa valeur *FALSE* indique que les clés présentes dans la table *emaj_param* mais absentes de la structure JSON sont conservées (chargement en mode différentiel). Si la valeur du second paramètre est positionnée à *TRUE*, la fonction effectue un remplacement complet de la configuration de paramétrage (chargement en mode complet).

La fonction retourne le nombre de paramètres importés.

Dans une variante de la fonction, le premier paramètre en entrée contient directement la structure JSON des valeurs à charger

```
SELECT emaj_import_parameters_configuration('<structure.JSON>', <suppression.
↪configuration.courante>);
```

18.3 Identité de la table de log courante associée à une table applicative

La fonction *emaj_get_current_log_table()* permet d'obtenir le schéma et le nom de la table de log courante associée à une table applicative.

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

La fonction retourne toujours 1 ligne. Si la table applicative n'appartient pas actuellement à un groupe de tables, les colonnes *log_schema* et *log_table* ont une valeur NULL.

La fonction *emaj_get_current_log_table()* peut être exécutée par les rôles membres de *emaj_adm* et *emaj_viewer*.

Il est ainsi possible de construire une requête accédant à une table de log. Par exemple :

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

18.4 Non désactivation de triggers applicatifs lors des Rollbacks E-Maj

Les triggers applicatifs sont automatiquement désactivés pendant les opérations de rollback E-Maj. Dans certains cas, il peut être souhaitable de les conserver activés (plus de détails *ici*). Pour ce faire, on peut utiliser la fonction *emaj_ignore_app_trigger()*. Elle permet de gérer l'ajout ou la suppression de triggers dans une liste de triggers ne devant pas être désactivés lors des opérations de rollback.

```
SELECT emaj.emaj_ignore_app_trigger(<action>, <nom.schéma>, <nom.table>, <trigger>);
```

Le paramètre <action> peut prendre les valeurs *'ADD'* pour ajouter un trigger à la liste ou *'REMOVE'* pour supprimer un trigger de la liste.

L'identité du trigger est définie par les 3 composantes nom de schéma, nom de table et nom du trigger.

Le nom du trigger peut contenir des caractères génériques '%' et '_', dont la signification est identique à ceux présents dans les clauses *LIKE* du langage SQL. On peut ainsi traiter plusieurs triggers d'une même table en un seul appel de la fonction.

La fonction retourne le nombre de triggers effectivement ajoutés ou supprimés.

La fonction ne traite pas les triggers E-Maj (trigger de log ou de traitement des *TRUNCATEs*).

Les triggers ainsi référencés comme « ne devant pas être automatiquement désactivés lors des rollbacks E-Maj » sont enregistrés dans la table *emaj.emaj_ignored_app_trigger*. Cette table comprend 3 colonnes :

- trg_schema : schéma de la table portant le trigger
- trg_table : table portant le trigger
- trg_name : nom du trigger

Pour connaître la liste des triggers enregistrés, il suffit de lister le contenu de la table.

18.5 Suivi des opérations de rollback en cours

Lorsque le volume de mises à jour à annuler rend un rollback long, il peut être intéressant de suivre l'opération afin d'en apprécier l'avancement. Une fonction, *emaj_rollback_activity()*, et un client *emajRollbackMonitor.php* répondent à ce besoin.

18.5.1 Pré-requis

Pour permettre aux administrateurs E-Maj de suivre la progression d'une opération de rollback, les fonctions activées dans l'opération mettent à jour plusieurs tables techniques au fur et à mesure de son avancement. Pour que ces mises à jour soient visibles alors que la transaction dans laquelle le rollback s'effectue est encore en cours, ces mises à jour sont effectuées au travers d'une connexion *dblink*.

Le suivi des rollbacks nécessite donc d'une part l'*installation de l'extension dblink*, et d'autre part l'enregistrement dans la table des paramètres, *emaj_param*, d'un identifiant de connexion utilisable par *dblink*.

L'enregistrement de l'identifiant de connexion peut s'effectuer au travers d'une requête du type

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

Le rôle de connexion déclaré doit disposer des droits *emaj_adm* (ou être super-utilisateur).

Enfin, la transaction principale effectuant l'opération de rollback doit avoir un mode de concurrence « *read committed* » (la valeur par défaut).

18.5.2 Fonction de suivi

La fonction *emaj_rollback_activity()* permet de visualiser les opérations de rollback en cours.

Il suffit d'exécuter la requête

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

La fonction ne requiert aucun paramètre en entrée.

Elle retourne un ensemble de lignes de type *emaj.emaj_rollback_activity_type*. Chaque ligne représente une opération de rollback en cours, comprenant les colonnes suivantes :

Column	Type	Description
rlbk_id	INT	identifiant de rollback
rlbk_groups	TEXT[]	tableau des groupes de tables associés au rollback
rlbk_mark	TEXT	marque de rollback
rlbk_mark_datetime	TIMESTAMPTZ	date et heure de pose de la marque de rollback
rlbk_is_logged	BOOLEAN	booléen prenant la valeur « vrai » pour les rollbacks annulables
rlbk_nb_session	INT	nombre de sessions en parallèle
rlbk_nb_table	INT	nombre de tables contenues dans les groupes de tables traités
rlbk_nb_sequence	INT	nombre de séquences contenues dans les groupes de tables traités
rlbk_eff_nb_table	INT	nombre de tables ayant eu des mises à jour à annuler
rlbk_status	ENUM	état de l'opération de rollback
rlbk_start_datetime	TIMESTAMPTZ	date et heure de début de l'opération de rollback
rlbk_elapse	INTERVAL	durée écoulée depuis le début de l'opération de rollback
rlbk_remaining	INTERVAL	durée restante estimée
rlbk_completion_pct	SMALLINT	estimation du pourcentage effectué

Une opération de rollback en cours est dans l'un des états suivants :
- PLANNING : l'opération est dans sa phase initiale de planification,
- LOCKING : l'opération est dans sa phase de pose de verrou,
- EXECUTING : l'opération est dans sa phase d'exécution des différentes étapes planifiées

Si les fonctions impliquées dans les opérations de rollback ne peuvent utiliser de connexion *dblink*, (extension *dblink* non installée, paramétrage de la connexion absente ou incorrect,...), la fonction *emaj_rollback_activity()* ne retourne aucune ligne.

L'estimation de la durée restante est approximative. Son degré de précision est similaire à celui de la fonction *emaj_estimate_rollback_group()*.

18.6 Mise à jour de l'état des rollbacks

La table technique *emaj_rlbk*, et ses tables dérivées, contient l'historique des opérations de rollback E-Maj.

Lorsque les fonctions de rollback ne peuvent pas utiliser une connexion *dblink*, toutes les mises à jour de ces tables techniques s'effectuent dans le cadre d'une unique transaction. Dès lors :
- toute transaction de rollback E-Maj qui n'a pu aller à son terme est invisible dans les tables techniques,
- toute transaction de rollback E-Maj qui a été validé est visible dans les tables techniques avec un état « *COMMITTED* » (validé).

Lorsque les fonctions de rollback peuvent utiliser une connexion *dblink*, toutes les mises à jour de la table technique *emaj_rlbk* et de ses tables dérivées s'effectuent dans le cadre de transactions indépendantes. Dans ce mode de fonctionnement, les fonctions de rollback E-Maj positionnent l'opération de rollback dans un état « *COMPLETED* » (terminé) en fin de traitement. Une fonction interne est chargée de transformer les opérations en état « *COMPLETED* », soit en état « *COMMITTED* » (validé), soit en état « *ABORTED* » (annulé), selon que la transaction principale ayant effectuée

l'opération a ou non été validée. Cette fonction est automatiquement appelée lors de la pose d'une marque ou du suivi des rollbacks en cours,

Si l'administrateur E-Maj souhaite de lui-même procéder à la mise à jour de l'état d'opérations de rollback récemment exécutées, il peut à tout moment utiliser la fonction *emaj_cleanup_rollback_state()*

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

La fonction retourne le nombre d'opérations de rollback dont l'état a été modifié.

18.7 Purge des historiques

E-Maj historise certaines données : traces globales de fonctionnement, détail des rollbacks E-Maj, évolutions de structures de groupes de tables (*plus de détails...*), Les traces les plus anciennes sont automatiquement purgées par l'extension. Mais une fonction permet également de déclencher la purge de manière manuelle :

```
SELECT emaj.emaj_purge_histories('<délai.rétention>');
```

La paramètre <délai.rétention> est de type *INTERVAL*. Il surcharge le paramètre *"history_retention"* de la table *emaj_param*.

18.8 Désactivation/réactivation des triggers sur événements

L'installation de l'extension E-Maj créé et active des *triggers sur événements* pour la protéger. En principe, ces triggers doivent rester en l'état. Mais si l'administrateur E-Maj a besoin de les désactiver puis les réactiver, il dispose de deux fonctions.

Pour désactiver les triggers sur événement existants :

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

La fonction retourne le nombre de triggers désactivés (cette valeur dépend de la version de PostgreSQL installée).

Pour réactiver les triggers sur événement existants :

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

La fonction retourne le nombre de triggers réactivés.

CHAPITRE 19

Fonctions multi-groupes

19.1 Généralités

Pour pouvoir synchroniser les opérations courantes de démarrage, arrêt, pose de marque et rollback entre plusieurs groupes de tables, les fonctions usuelles associées disposent de fonctions jumelles permettant de traiter plusieurs groupes de tables en un seul appel.

Les avantages qui en résultent sont :

- de pouvoir traiter tous les groupes de tables dans une seule transaction,
- d'assurer un verrouillage de toutes les tables à traiter en début d'opération, et ainsi minimiser les risques d'étreintes fatales.

19.2 Liste des fonctions multi-groupes

Le tableau suivant liste les fonctions multi-groupes existantes et leur fonction mono-groupe jumelle. Certaines des fonctions mono-groupes sont présentées plus loin.

Fonctions multi-groups	Fonctions mono-groupe jumelles
emaj.emaj_alter_groups()	*emaj.emaj_alter_group()*
emaj.emaj_start_groups()	*emaj.emaj_start_group()*
emaj.emaj_stop_groups()	*emaj.emaj_stop_group()*
emaj.emaj_set_mark_groups()	*emaj.emaj_set_mark_group()*
emaj.emaj_rollback_groups()	*emaj.emaj_rollback_group()*
emaj.emaj_logged_rollback_groups()	*emaj.emaj_logged_rollback_group*
emaj.emaj_estimate_rollback_groups()	*emaj.emaj_estimate_rollback_group()*
emaj.emaj_log_stat_groups()	*emaj.emaj_log_stat_group()*
emaj.emaj_detailed_log_stat_groups()	*emaj.emaj_detailed_log_stat_group()*
emaj.emaj_gen_sql_groups()	*emaj.emaj_gen_sql_group()*

Les paramètres des fonctions multi-groupes sont les mêmes que ceux de leurs fonctions mono-groupe associées, à

l'exception du premier. Le paramètre groupe de tables de type *TEXT* est remplacé par une paramètre de type *tableau de TEXT* représentant la liste des groupes de tables.

19.3 Syntaxes pour exprimer un tableau de groupes

Le paramètre <tableau de groupes> passé aux fonctions multi-groupes est de type SQL *TEXT[ ]*, c'est à dire un tableau de données de type *TEXT*.

Conformément au langage SQL, il existe deux syntaxes possibles pour saisir un tableau de groupes, utilisant soit les accolades { }, soit la fonction *ARRAY*.

Lorsqu'on utilise les caractères {}, la liste complète est entre simples guillemets, puis les accolades encadrent la liste des éléments séparés par une virgule, chaque élément étant délimité par des doubles guillemets. Par exemple dans notre cas, nous pouvons écrire

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

La fonction SQL *ARRAY* permet de construire un tableau de données. La liste des valeurs est entre crochets et les littéraux sont séparés par une virgule. Par exemple dans notre cas, nous pouvons écrire

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

Ces deux syntaxes sont équivalentes, et le choix de l'une ou de l'autre est à l'appréciation de chacun.

19.4 Autres considérations

L'ordre dans lequel les groupes sont listés n'a pas d'importance. L'ordre de traitement des tables dans les opérations E-Maj dépend du niveau de priorité associé à chaque table, et pour les tables de même priorité de l'ordre alphabétique de nom de schéma et nom de table, tous groupes confondus.

Il est possible d'appeler une fonction multi-groupes pour traiter une liste … d'un seul groupe, voire une liste vide. Ceci peut permettre une construction ensembliste de la liste, en utilisant par exemple la fonction *array_agg()*.

Les listes de groupes de tables peuvent contenir des doublons, des valeurs *NULL* ou des chaînes vides. Ces valeurs *NULL* et ces chaînes vides sont simplement ignorées. Si un nom de groupe de tables est présent plusieurs fois, une seule occurrence du nom est retenue.

Le formalisme et l'usage des autres paramètres éventuels des fonctions est strictement le même que pour les fonctions jumelles mono-groupes.

Néanmoins, une condition supplémentaire existe pour les fonctions de rollbacks, La marque indiquée doit strictement correspondre à un même moment dans le temps pour chacun des groupes. En d'autres termes, cette marque doit avoir été posée par l'appel d'une même fonction *emaj_set_mark_group()*.

CHAPITRE 20

Client de rollback avec parallélisme

Sur les serveurs équipés de plusieurs processeurs ou cœurs de processeurs, il peut être intéressant de réduire la durée des rollbacks en parallélisant l'opération sur plusieurs couloirs. A cette fin, E-Maj fournit un client spécifique qui se lance en ligne de commande. Celui-ci active les fonctions de rollback d'E-Maj au travers de plusieurs connexions à la base de données en parallèle.

20.1 Sessions

Pour paralléliser un rollback, E-Maj affecte les tables et séquences à traiter pour un ou plusieurs groupes de tables à un certain nombre de « **sessions** ». Chaque *session* est ensuite traitée dans un couloir propre.

Néanmoins, pour garantir l'intégrité de l'opération, le rollback de toutes les sessions s'exécute au sein d'une unique transaction.

Pour obtenir des sessions les plus équilibrées possibles, E-Maj tient compte :

- du nombre de sessions spécifiés par l'utilisateur dans sa commande,
- des statistiques des lignes à annuler, telles que la fonction *emaj_log_stat_group()* les restitue,
- des contraintes de clés étrangères qui relient plusieurs tables entre-elles, 2 tables mises à jour et reliées entre-elles par une clé étrangère étant affectées à une même *session*.

20.2 Préalables

Deux outils équivalents sont en fait proposés, l'un codé en *php*, l'autre en *perl*. L'un ou l'autre nécessite que certains composants logiciel soient installés sur le serveur qui exécute cette commande (qui n'est pas nécessairement le même que celui qui héberge l'instance PostgreSQL) :

- pour le client *php*, le logiciel **php** et son interface PostgreSQL
- pour le client *perl*, le logiciel **perl** avec les modules *DBI* et *DBD : :Pg*

Le rollback de chaque session au sein d'une unique transaction implique l'utilisation de commit à deux phases. En conséquence, le paramètre **max_prepared_transaction** du fichier *postgresql.conf* doit être ajusté. La valeur par défaut du paramètre est 0. Il faut donc la modifier en spécifiant une valeur au moins égale au nombre maximum de *sessions* qui seront utilisées.

20.3 Syntaxe

Les deux commandes php et perl partagent la même syntaxe

```
emajParallelRollback.php -g <nom.du.ou.des.groupes> -m <marque> -s <nombre.de.
↪sessions> [OPTIONS]...
```

et :

```
emajParallelRollback.pl -g <nom.du.ou.des.groupes> -m <marque> -s <nombre.de.sessions>
↪ [OPTIONS]...
```

Options générales :

- -l spécifie que le rollback demandé est de type *logged rollback*
- -a spécifie que le rollback demandé est *autorisé à remonter à une marque antérieure à une modification de groupe de tables*
- -v affiche davantage d'information sur le déroulement du traitement
- –help affiche uniquement une aide sur la commande
- –version affiche uniquement la version du logiciel

Options de connexion :

- -d <base de données à atteindre>
- -h <hôte à atteindre>
- -p <port ip à utiliser>
- -U <rôle de connexion>
- -W <mot de passe associé à l'utilisateur> si nécessaire

Pour remplacer tout ou partie des paramètres de connexion, les variables habituelles *PGDATABASE*, *PGPORT*, *PGHOST* et/ou *PGUSER* peuvent être également utilisées.

Pour spécifier une liste de groupes de tables dans le paramètre -g, séparer le nom de chaque groupe par une virgule.

Le rôle de connexion fourni doit être soit un super-utilisateur, soit un rôle ayant les droits *emaj_adm*.

Pour des raisons de sécurité, il n'est pas recommandé d'utiliser l'option -W pour fournir un mot de passe. Il est préférable d'utiliser le fichier *.pgpass* (voir la documentation de PostgreSQL).

Pour que l'opération de rollback puisse être exécutée, le ou les groupes de tables doivent être actifs. Si le rollback concerne plusieurs groupes, la marque demandée comme point de rollback doit correspondre à un même moment dans le temps, c'est à dire qu'elle doit avoir été créée par une unique commande *emaj_set_mark_group()*.

Le mot clé *"EMAJ_LAST_MARK"* peut être utilisé pour référencer la dernière marque du ou des groupes de tables.

Il est possible de suivre l'avancement des opérations de rollback multi-sessions de la même manière que celui des opérations de rollbacks mono-session.

Pour tester les commandes **emajParallelRollback**, E-Maj fournit un script, *emaj_prepare_parallel_rollback_test.sql*. Il prépare un environnement avec deux groupes de tables contenant quelques tables et séquences, sur lesquelles des mises à jour ont été effectuées, entrecoupées de marques. Suite à l'exécution de ce script sous *psql*, on peut lancer la commande telle qu'indiquée dans le message de fin d'exécution du script.

20.4 Exemples

La commande

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

se connecte à la base de données *mydb* et exécute un rollback du groupe *myGroup1* à la marque *Mark1*, avec 3 sessions en parallèle.

La commande :

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

se connecte à la base de données *mydb* et exécute un rollback annulable (« *logged rollback* ») des 2 groupes *myGroup1* et *myGroup2* à la marque *Mark1*, avec 3 sessions en parallèle.

CHAPITRE 21

Client de suivi des rollbacks

E-Maj fournit un client externe qui se lance en ligne de commande et qui permet de suivre l'avancement des opérations de rollback en cours.

21.1 Préalables

Deux outils équivalents sont en fait proposés, l'un codé en *php*, l'autre en *perl*. L'un ou l'autre nécessite que certains composants logiciel soient installés sur le serveur qui exécute cette commande (qui n'est pas nécessairement le même que celui qui héberge l'instance PostgreSQL) :

- pour le client *php*, le logiciel **php** et son interface PostgreSQL
- pour le client *perl*, le logiciel **perl** avec les modules *DBI* et *DBD : :Pg*

21.2 Syntaxe

Les deux commandes php et perl partagent la même syntaxe

```
emajRollbackMonitor.php [OPTIONS]...
```

et :

```
emajRollbackMonitor.pl [OPTIONS]...
```

Options générales :

- -i <intervalle de temps entre 2 affichages> (en secondes, défaut = 5s)
- -n <nombre d'affichages> (défaut = 1)
- -a <intervalle de temps maximum pour les opérations de rollback terminés à afficher> (en heures, défaut = 24h)
- -l <nombre maximum d'opérations de rollback terminés à afficher> (défaut = 3)
- –help affiche uniquement une aide sur la commande
- –version affiche uniquement la version du logiciel

Options de connexion :

- -d <base de données à atteindre>
- -h <hôte à atteindre>
- -p <port ip à utiliser>
- -U <rôle de connexion>
- -W <mot de passe associé à l'utilisateur>

Pour remplacer tout ou partie des paramètres de connexion, les variables habituelles *PGDATABASE*, *PGPORT*, *PGHOST* et/ou *PGUSER* peuvent être également utilisées.

Le rôle de connexion fourni doit être soit un super-utilisateur, soit un rôle ayant les droits *emaj_adm* ou *emaj_viewer*.

Pour des raisons de sécurité, il n'est pas recommandé d'utiliser l'option -W pour fournir un mot de passe. Il est préférable d'utiliser le fichier *.pgpass* (voir la documentation de PostgreSQL).

21.3 Exemples

La commande

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

affiche 10 fois la liste des opérations de rollback en cours et celles des au plus 3 dernières opérations terminés depuis 24 heures, avec 3 secondes entre chaque affichage.

La commande

```
./client/emajRollbackMonitor.pl -a 12 -l 10
```

affichera une seule fois la liste des opérations de rollback en cours et celle des au plus 10 opérations terminées dans les 12 dernières heures.

Exemple d'affichage de l'outil

```
 E-Maj (version 3.3.0) - Monitoring rollbacks activity
------------------------------------------------------------
04/02/2020 - 12:07:17
** rollback 34 started at 2020-02-04 12:06:20.350962+02 for groups {myGroup1,myGroup2}
   status: COMMITTED ; ended at 2020-02-04 12:06:21.149111+02
** rollback 35 started at 2020-02-04 12:06:21.474217+02 for groups {myGroup1}
   status: COMMITTED ; ended at 2020-02-04 12:06:21.787615+02
-> rollback 36 started at 2020-02-04 12:04:31.769992+02 for groups {group1232}
   status: EXECUTING ; completion 89 % ; 00:00:20 remaining
-> rollback 37 started at 2020-02-04 12:04:21.894546+02 for groups {group1233}
   status: LOCKING ; completion 0 % ; 00:22:20 remaining
-> rollback 38 started at 2020-02-04 12:05:21.900311+02 for groups {group1234}
   status: PLANNING ; completion 0 %
```

CHAPITRE 22

Paramétrage

L'extension E-Maj fonctionne avec quelques paramètres. Ceux-ci sont stockés dans la table interne *emaj_param*.

La structure de la table **emaj_param** est la suivante :

Colonne	Type	Description
param_key	TEXT	mot-clé identifiant le paramètre
param_value_text	TEXT	valeur du paramètre, s'il est de type texte (sinon NULL)
param_value_numeric	NUMERIC	valeur du paramètre, s'il est de type numérique (sinon NULL)
param_value_boolean	BOOLEAN	valeur du paramètre, s'il est de type booléen (sinon NULL)
param_value_interval	INTERVAL	valeur du paramètre, s'il est de type intervalle (sinon NULL)

La procédure d'installation de l'extension E-Maj ne crée qu'une seule ligne dans la table *emaj_param*. Cette ligne, qui ne doit pas être modifiée, décrit le paramètre :

— **version** : (texte) version courante d'E-Maj

Mais l'administrateur d'E-Maj peut insérer d'autres lignes dans *emaj_param* pour modifier la valeur par défaut de certains paramètres.

Les valeurs de clé des paramètres sont, par ordre alphabétique :

— **alter_log_table** : (texte) directive d'*ALTER TABLE* exécuté à la création des tables de log ; aucun *ALTER TABLE* exécuté par défaut (pour *ajouter une ou plusieurs colonnes techniques*).

— **avg_fkey_check_duration** : (intervalle) valeur par défaut = 20 µs ; définit la durée moyenne du contrôle d'une clé étrangère ; peut être modifiée pour mieux représenter la performance du serveur qui héberge la base de données à l'exécution d'une fonction *emaj_estimate_rollback_group()*.

— **avg_row_delete_log_duration** : (intervalle) valeur par défaut = 10 µs ; définit la durée moyenne de suppression d'une ligne du log ; peut être modifiée pour mieux représenter la performance du serveur qui héberge la base de données à l'exécution d'une fonction *emaj_estimate_rollback_group()*.

— **avg_row_rollback_duration** : (intervalle) valeur par défaut = 100 µs ; définit la durée moyenne de rollback d'une ligne ; peut être modifiée pour mieux représenter la performance du serveur qui héberge la base de données à l'exécution d'une fonction *emaj_estimate_rollback_group()*.

— **fixed_dblink_rollback_duration** : (intervalle) valeur par défaut = 4 ms ; définit un coût additionnel pour chaque étape de rollback quand une connexion dblink est utilisée ; peut être modifiée pour mieux représenter la performance du serveur qui héberge la base de données à l'exécution d'une fonction *emaj_estimate_rollback_group()*.

- **fixed_table_rollback_duration** : (intervalle) valeur par défaut = 1 ms ; définit un coût fixe de rollback de toute table ou séquence appartenant à un groupe ; peut être modifiée pour mieux représenter la performance du serveur qui héberge la base de données à l'exécution d'une fonction *emaj_estimate_rollback_group()*.
- **fixed_step_rollback_duration** : (intervalle) valeur par défaut = 2,5 ms ; définit un coût fixe pour chaque étape de rollback ; peut être modifiée pour mieux représenter la performance du serveur qui héberge la base de données à l'exécution d'une fonction *emaj_estimate_rollback_group()*.
- **history_retention** (intervalle) valeur par défaut = 1 an ; elle peut être ajustée pour changer la durée de rétention des lignes dans la table historique d'E-Maj, *emaj_hist* et dans quelques autres tables techniques ; une valeur supérieure ou égale à 100 ans équivaut à un délai infini.

Exemple de requête SQL permettant de spécifier une durée de rétention des lignes dans l'historique de 3 mois :

```
INSERT INTO emaj.emaj_param (param_key, param_value_interval) VALUES ('history_
↪retention','3 months'::interval);
```

Toute modification de la table *emaj_param* est tracée dans la table *emaj_hist*.

Seuls les super-utilisateurs et les utilisateurs ayant acquis les droits *emaj_adm* ont accès à la table *emaj_param*.

Les utilisateurs ayant acquis les droits *emaj_viewer* n'ont accès qu'à une partie de la table *emaj_param*. au travers de la vue *emaj.emaj_visible_param*. Cette vue masque simplement le contenu réel de la colonne *param_value_text* pour la clé *"dblink_user_password"*.

Des fonctions *emaj_export_parameters_configuration()* et *emaj_import_parameters_configuration()* permettent de sauver les valeurs de paramètres et de les restaurer.

CHAPITRE 23

Structure des tables de log

23.1 Structure standard

Les tables de log ont une structure qui découle directement des tables applicatives dont elles enregistrent les mises à jour. Elles contiennent les mêmes colonnes avec les mêmes types. Mais elles possèdent aussi quelques colonnes techniques complémentaires :

- emaj_verb : type de verbe SQL ayant généré la mise à jour (*INS*, *UPD*, *DEL*, *TRU*)
- emaj_tuple : version des lignes (*OLD* pour les *DEL*, *UPD* et *TRU* ; *NEW* pour *INS* et *UPD* ; NULL pour les événements *TRUNCATE*)
- emaj_gid : identifiant de la ligne de log
- emaj_changed : date et heure de l'insertion de la ligne dans la table de log
- emaj_txid : identifiant de la transaction à l'origine de la mise à jour (*txid* PostgreSQL)
- emaj_user : rôle de connexion à l'origine de la mise à jour

Lorsqu'une requête SQL *TRUNCATE* est exécutée sur une table, chaque ligne présente dans la table est enregistrée (avec *emaj_verb = TRU* et *emaj_tuple = OLD*). Une ligne est ajoutée avec *emaj_verb = TRU*, les colonnes de la table source et *emaj_tuple* étant positionnées à NULL. Cette ligne est utilisée pour la génération de scripts SQL.

23.2 Ajouter des colonnes techniques

Il est possible d'ajouter une ou plusieurs colonnes techniques pour enrichir les traces. Ces colonnes doivent doivent être valorisées avec une valeur par défaut (clause *DEFAULT*) associée à une fonction (pour que les triggers de logs ne soient pas impactés).

Pour ajouter une ou plusieurs colonnes techniques, il faut ajouter le paramètre de clé « alter_log_table » dans la table *emaj_param*. La valeur texte associée doit contenir une clause d'*ALTER TABLE*. Lors de la création d'une table de log, si le paramètre existe, une requête *ALTER TABLE* avec ce paramètre est exécutée.

Par exemple, on peut ajouter dans les tables de log une colonne pour enregistrer la valeur du champ de connexion *application_name* de la manière suivante :

```
INSERT INTO emaj.emaj_param (param_key, param_value_text) VALUES ('alter_log_table',
  'ADD COLUMN extra_col_appname TEXT DEFAULT current_setting(''application_name'')');
```

Plusieurs directives *ADD COLUMN* peuvent être concaténées, séparées par une virgule. Par exemple pour créer des colonnes enregistrant l'adresse ip et le port du client connecté :

```
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()');
```

Pour changer la structure de tables de log existantes après valorisation ou modification du paramètre alter_log_table, les groupes de tables doivent être supprimés puis recréés.

CHAPITRE 24

Fiabilisation du fonctionnement

Deux éléments complémentaires concourent à la fiabilité de fonctionnement d'E-Maj : des contrôles internes effectués à certains moments clé de la vie des groupes de tables, et l'activation de triggers sur événement bloquant certaines opérations risquées.

24.1 Contrôles internes

Lors de l'exécution des fonctions de démarrage de groupe, de pose de marque et de rollback, E-Maj effectue un certain nombre de contrôles afin de vérifier l'intégrité des groupes de tables sur lesquels porte l'action.

Ces **contrôles d'intégrité du groupe de tables** vérifient que :

- la version de PostgreSQL avec laquelle le groupe a été créé est bien compatible avec la version actuelle,
- chaque séquence ou chaque table applicative du groupe existe toujours bien,
- chacune des tables d'un groupe a toujours sa table de log associée, sa fonction de log ainsi que ses triggers,
- la structure des tables de log est toujours en phase avec celle des tables applicatives associées, et comprend toujours les colonnes techniques nécessaires,
- pour les groupes de tables « rollbackables », aucune table n'a été transformée en table *UNLOGGED* ou *WITH OIDS*,
- pour les groupes de tables « rollbackables », les tables applicatives ont toujours leur clé primaire et que leur structure n'a pas changé.

En utilisant la fonction *emaj_verify_all()*, l'administration peut effectuer à la demande ces mêmes contrôles sur l'ensemble des groupes de tables.

24.2 Triggers sur événements

L'installation d'E-Maj inclut la création de 2 triggers sur événements de type « *sql_drop* » :

- *emaj_sql_drop_trg* bloque la suppression :
  - de tout objet E-Maj (schéma de log, table de logs, séquence de log, fonction de log et trigger de log),
  - de toute table ou séquence applicatives appartenant à un groupe de tables en état « *LOGGING* »,

— de tout schéma contenant au moins une table ou séquence appartenant à un groupe de tables en état « *LOGGING* ».

— *emaj_protection_trg* bloque la suppression de l'extension *emaj* elle-même et du schéma principal *emaj*.

L'installation d'E-Maj inclut aussi la création d'un 3ème trigger sur événements, de type « *table_rewrite* » :

— *emaj_table_rewrite_trg* bloque tout changement de structure de table applicative ou de table de log.

Il est possible de désactiver/réactiver ces triggers grâce aux deux fonctions : *emaj_disable_protection_by_event_triggers()* et *emaj_enable_protection_by_event_triggers()*.

Les protections mises en place ne couvrent néanmoins pas tous les risques. En particulier, le renommage de tables ou de séquences ou leur changement de schéma d'appartenance ne sont pas couverts ; et certaines requêtes changeant la structure d'une table ne déclenchent aucun trigger.

CHAPITRE 25

Traçabilité des opérations

25.1 La table emaj_hist

Toutes les opérations réalisées par E-Maj et qui modifient d'une manière ou d'une autre un groupe de tables sont tracées dans une table nommée *emaj_hist*.

La structure de la table **emaj_hist** est la suivante.

Colonne	Type	Description
hist_id	BIGSERIAL	numéro de série identifiant une ligne dans cette table historique
hist_datetime	TIMESTAMPTZ	date et heure d'enregistrement de la ligne
hist_function	TEXT	fonction associée à l'événement
hist_event	TEXT	type d'événement
hist_object	TEXT	nom de l'objet sur lequel porte l'événement (groupe, table, séquence,. . .)
hist_wording	TEXT	commentaires complémentaires
hist_user	TEXT	rôle à l'origine de l'événement
hist_txid	BIGINT	numéro de la transaction à l'origine de l'événement

La colonne *hist_function* peut prendre les valeurs suivantes.

Valeur	Signification
ADJUST_GROUP_PROPERTIES	ajustement du contenu de la colonne group_has_waiting_changes de la table emaj_group
ALTER_GROUP	modification d'un groupe de tables
ALTER_GROUPS	modification de plusieurs groupes de tables
ASSIGN_SEQUENCE	affectation d'une séquence à un groupe de tables
ASSIGN_SEQUENCES	affectation de séquences à un groupe de tables
ASSIGN_TABLE	affectation d'une table à un groupe de tables
ASSIGN_TABLES	affectation de tables à un groupe de tables
CLEANUP_RLBK_STATE	nettoyage du code état des opérations de rollback récemment terminées
COMMENT_GROUP	positionnement d'un commentaire sur un groupe

Suite sur la page suivante

Tableau 1 – suite de la page précédente

Valeur	Signification
COMMENT_MARK_GROUP	positionnement d'un commentaire sur une marque
CONSOLIDATE_RLBK_GROUP	consolide une opération de rollback tracé
CREATE_GROUP	création d'un groupe de tables
DBLINK_OPEN_CNX	ouverture d'une connexion dblink pour un rollback
DBLINK_CLOSE_CNX	fermeture d'une connexion dblink pour un rollback
DELETE_MARK_GROUP	suppression d'une marque pour un groupe de tables
DISABLE_EVENT_TRIGGERS	désactivation des triggers sur événements
DROP_GROUP	suppression d'un groupe de tables
EMAJ_INSTALL	installation ou mise à jour de la version d'E-Maj
ENABLE_EVENT_TRIGGERS	activation des triggers sur événements
EXPORT_GROUPS	export d'une configuration de groupes de tables
EXPORT_PARAMETERS	export d'une configuration de paramètres E-Maj
FORCE_DROP_GROUP	suppression forcée d'un groupe de tables
FORCE_STOP_GROUP	arrêt forcé d'un groupe de tables
GEN_SQL_GROUP	génération d'un script psql pour un groupe de tables
GEN_SQL_GROUPS	génération d'un script psql pour plusieurs groupes de tables
IMPORT_GROUPS	import d'une configuration de groupes de tables
IMPORT_PARAMETERS	import d'une configuration de paramètres E-Maj
LOCK_GROUP	pose d'un verrou sur les tables d'un groupe
LOCK_GROUPS	pose d'un verrou sur les tables de plusieurs groupes
LOCK_SESSION	pose d'un verrou sur les tables d'une session de rollback
MODIFY_TABLE	modification des propriétés d'une table
MODIFY_TABLES	modification des propriétés de tables
MOVE_SEQUENCE	déplacement d'une séquence vers un autre groupe de tables
MOVE_SEQUENCES	déplacement de séquences vers un autre groupe de tables
MOVE_TABLE	déplacement d'une table vers un autre groupe de tables
MOVE_TABLES	déplacement de tables vers un autre groupe de tables
PROTECT_GROUP	pose d'une protection contre les rollbacks sur un groupe
PROTECT_MARK_GROUP	pose d'une protection contre les rollbacks sur une marque d'un groupe
PURGE_HISTORIES	suppression des tables historisées des événements antérieurs au délai de rétention
REMOVE_SEQUENCE	suppression d'une séquence de son groupe de tables
REMOVE_SEQUENCES	suppression de séquences de leur groupe de tables
REMOVE_TABLE	suppression d'une table de son groupe de tables
REMOVE_TABLES	suppression de tables de leur groupe de tables
RENAME_MARK_GROUP	renommage d'une marque pour un groupe de tables
RESET_GROUP	réinitialisation du contenu des tables de log d'un groupe
ROLLBACK_GROUP	rollback des mises à jour pour un groupe de tables
ROLLBACK_GROUPS	rollback des mises à jour pour plusieurs groupes de tables
ROLLBACK_SEQUENCE	rollback d'une séquence
ROLLBACK_TABLE	rollback des mises à jour d'une table
SET_MARK_GROUP	pose d'une marque pour un groupe de tables
SET_MARK_GROUPS	pose d'une marque pour plusieurs groupes de tables
SNAP_GROUP	vidage des tables et séquences d'un groupe
SNAP_LOG_GROUP	vidage des tables de log d'un groupe
START_GROUP	démarrage d'un groupe de tables
START_GROUPS	démarrage de plusieurs groupes de tables
STOP_GROUP	arrêt d'un groupe de tables
STOP_GROUPS	arrêt de plusieurs groupes de tables
SYNC_DEF_GROUP	synchronisation de la table *emaj_group_def*

Suite sur la page suivante

Tableau 1 – suite de la page précédente

Valeur	Signification
UNPROTECT_GROUP	suppression d'une protection contre les rollbacks sur un groupe
UNPROTECT_MARK_GROUP	suppression d'une protection contre les rollbacks sur une marque d'un groupe

La colonne *hist_event* peut prendre les valeurs suivantes.

Valeur	Signification
BEGIN	début
DELETED PARAMETER	paramètre supprimé dans *emaj_param*
EMAJ_GROUP_DEF SYNCHRONIZED	groupe de tables synchronisé dans *emaj_group_def*
END	fin
EVENT TRIGGERS DISABLED	triggers sur événements désactivés
EVENT TRIGGERS ENABLED	triggers sur événements activés
GROUP_CREATED	nouveau groupe de tables créé
INSERTED PARAMETER	paramètre inséré dans *emaj_param*
LOG DATA TABLESPACE CHANGED	tablespace pour la table de log modifié
LOG INDEX TABLESPACE CHANGED	tablespace pour l'index de log modifié
LOG_SCHEMA CREATED	schéma secondaire créé
LOG_SCHEMA DROPPED	schéma secondaire supprimé
MARK DELETED	marque supprimée
NOTICE	message d'information issu d'un rollback
PRIORITY CHANGED	priorité modifiée
SEQUENCE ADDED	séquence ajoutée à un groupe de tables actif
SEQUENCE MOVED	séquence déplacée d'un groupe à un autre
SEQUENCE REMOVED	séquence supprimée d'un groupe de tables actif
TABLE ADDED	table ajoutée à un groupe de tables actif
TABLE MOVED	table déplacée d'un groupe à un autre
TABLE REMOVED	table supprimée d'un groupe de tables actif
UPDATED PARAMETER	paramètre modifié dans *emaj_param*
WARNING	message d'avertissement issu d'un rollback

Le contenu de la table *emaj_hist* peut être visualisé par quiconque dispose des autorisations suffisantes (rôles super-utilisateur, *emaj_adm* ou *emaj_viewer*)

25.2 Autres données de trace

Deux autres tables internes conservent également des traces des opérations effectuées :

— *emaj_alter_plan* liste les actions élémentaires réalisées lors de l'exécution d'opérations de *modification de groupes de tables*,

— *emaj_rlbk_plan* liste les actions élémentaires réalisées lors de l'exécution d'opérations de *rollback E-Maj*.

25.3 Purge des traces obsolètes

A chaque démarrage de groupe (fonction *emaj_start_group()*) et suppression des marques les plus anciennes (fonction *emaj_delete_before_mark_group()*), les événements les plus anciens de la table *emaj_hist* sont supprimés. Les événements conservés sont ceux à la fois postérieurs à un délai de rétention paramétrable, postérieurs à la pose de la plus ancienne marque active et postérieurs à la plus ancienne opération de rollback non terminée. Par défaut, la durée de rétention des événements est de 1 an. Mais cette valeur peut être modifiée à tout moment en insérant par une requête

SQL le paramètre *history_retention* dans la table *emaj_param*. La même rétention s'applique aux contenus des tables qui historisent les actions élémentaires des opérations de modification ou de rollback de groupes de tables.

La purge des données périmées peut également être initiée par l'appel explicite de la fonction *emaj_purge_histories()* . La paramètre en entrée de cette fonction définit un délai de rétention qui surcharge le paramètre *history_retention* de la table *emaj_param*.

Si on souhaite planifier des purges régulières, il est donc possible de :

— positionner une valeur de paramètre *history_retention* très élevée (par exemple *"100 YEARS"*), afin que les démarrages de groupe de tables ou les suppressions des plus anciennes marques ne déclenchent pas de purge, et

— planifier par un ordonnanceur quelconque (crontab, pgAgent, pgTimeTable ou tout autre outil) les purges.

CHAPITRE 26

Impacts sur l'administration de l'instance et de la base de données

26.1 Arrêt/relance de l'instance

L'utilisation d'E-Maj n'apporte aucune contrainte particulière sur l'arrêt et la relance des instances PostgreSQL.

26.1.1 Règle générale

Au redémarrage de l'instance, tous les objets d'E-Maj se retrouvent dans le même état que lors de l'arrêt de l'instance : les triggers de logs des groupes de tables actifs restent activés et les tables de logs sont alimentées avec les mises à jours annulables déjà enregistrées.

Si une transaction avait des mises à jour en cours non validées lors de l'arrêt de l'instance, celle-ci est annulée lors du redémarrage, les écritures dans les tables de logs se trouvant ainsi annulées en même temps que les modifications de tables.

Cette règle s'applique bien sûr aux transactions effectuant des opérations E-Maj telles que le démarrage ou l'arrêt d'un groupe, un rollback, une suppression de marque, etc.

26.1.2 Rollback des séquences

Lié à une contrainte de PostgreSQL, seul le rollback des séquences applicatives n'est pas protégé par les transactions. C'est la raison pour laquelle les séquences sont rollbackées en toute fin d'"*opération de rollback*. (Pour la même raison, lors de la pose d'une marque, les séquences applicatives sont traitées en début d'opération.)

Au cas où un rollback serait en cours au moment de l'arrêt de l'instance, il est recommandé de procéder à nouveau à ce même rollback juste après le redémarrage de l'instance, afin de s'assurer que les séquences et tables applicatives restent bien en phase.

26.2 Sauvegarde et restauration

Prudence : E-Maj peut permettre de diminuer la fréquence avec laquelle les sauvegardes sont nécessaires. Mais E-Maj ne peut se substituer totalement aux sauvegardes habituelles, qui restent nécessaires pour conserver sur un support externe des images complètes des bases de données !

26.2.1 Sauvegarde et restauration au niveau fichier

Lors des sauvegardes ou des restauration des instances au niveau fichier, il est essentiel de sauver ou restaurer **TOUS** les fichiers e l'instance, y compris ceux stockés sur des tablespaces dédiés.

Après restauration des fichiers, les groupes de tables se retrouveront dans l'état dans lequel ils se trouvaient lors de la sauvegarde, et l'activité de la base de données peut reprendre sans opération E-Maj particulière.

26.2.2 Sauvegarde et restauration logique de base de données complète

Pour les sauvegardes et restaurations logiques de base de données avec E-Maj, utilisant *pg_dump*, et *psql* ou *pg_restore*, il est essentiel que la base d'origine et la base restaurée utilisent la **même version d'E-Maj**. Dans le cas contraire, le contenu de certaines tables techniques peut ne pas correspondre à leur structure. La lecture de la ligne de clé *"emaj_version"* de la table *emaj.emaj_param* peut permettre de connaître la version d'une extension E-Maj créée dans une base de données.

Pour les groupes de tables arrêtés (en état *IDLE*), comme les triggers de logs sont inactifs et que le contenu des tables de log n'a pas d'importance, il n'y a aucune précaution particulière à prendre pour les retrouver dans le même état après une restauration.

Pour les groupes de tables en état *LOGGING* au moment de la sauvegarde, il faut s'assurer que les triggers de logs ne sont pas activés au moment de la reconstitution (restauration) des tables applicatives. Dans le cas contraire, pendant la reconstruction des tables, toutes les insertions de lignes seraient aussi enregistrées dans les tables de logs !

Lorsqu'on utilise les commandes *pg_dump* pour la sauvegarde et *psql* ou *pg_restore* pour la restauration et que l'on traite des bases complètes (schéma et données), ces outils font en sorte que les triggers, dont les triggers de log E-Maj, ne soient activés qu'en fin de restauration. Il n'y a donc pas de précautions particulières à prendre.

En revanche, dans le cas de sauvegarde et restauration des données seulement (sans schéma, avec les options -a ou *–data-only*), alors il faut spécifier l'option *–disable-triggers* :

- à la commande *pg_dump* (ou *pg_dumpall*) pour les sauvegardes au format plain (*psql* utilisé pour le rechargement),
- à la commande *pg_restore* pour les sauvegardes au format *tar* ou *custom*.

La restauration de la structure de la base de données génère 2 messages d'erreur indiquant que la fonction *_emaj_protection_event_trigger_fnct()* et que le trigger sur événement *emaj_protection_trg* existent déjà :

```
...
ERROR:  function "_emaj_protection_event_trigger_fnct" already exists with same␣
↪argument types
...
ERROR:  event trigger "emaj_protection_trg" already exists
...
```

L'affichage de ces messages est normal et n'est pas le signe d'une restauration défectueuse. En effet, ces 2 objets sont créés avec l'extension mais en sont détachés ensuite, de sorte que le trigger puisse être capable de bloquer la suppression éventuelle de l'extension. L'outil *pg_dump* les sauvegarde donc comme des objets indépendants. Lors de la restauration, ces objets sont donc créés 2 fois, une première fois avec l'extension emaj et une seconde fois en tant qu'objet indépendant. C'est cette seconde tentative de création qui provoque les 2 messages d'erreur.

26.2.3 Sauvegarde et restauration logique de base de données partielle

Les outils *pg_dump* et *pg_restore* permettent de ne traiter qu'un sous-ensemble des schémas et/ou des tables d'une base de données.

Restaurer un sous-ensemble des tables applicatives et/ou des tables de log comporte un risque très élevé de corruption des données en cas de rollback E-Maj ultérieur sur le groupe de tables concerné. En effet, dans ce cas, il est impossible de garantir la cohérence entre les tables applicatives, les tables de log et les tables internes d'E-Maj, qui contiennent des données essentielles aux opérations de rollback.

S'il s'avère nécessaire de procéder à une restauration partielle de tables applicatives, il faut faire suivre cette restauration de la suppression puis recréation du ou des groupes de tables touchées par l'opération.

De la même manière il est fortement déconseillé de procéder à une restauration partielle des tables du schéma *emaj*.

Le seul cas de restauration partielle sans risque concerne la restauration du contenu complet du schéma *emaj*, ainsi que de toutes les tables et séquences appartenant à tous les groupes de tables créés dans la base de données.

26.3 Chargement de données

Au delà de l'utilisation de *pg_restore* ou de *psql* avec un fichier issu de *pg_dump* évoquée plus haut, il est possible de procéder à des chargements massifs de tables par la commande SQL *COPY* ou la méta-commande *psql copy*. Dans les deux cas, le chargement des données provoque le déclenchement des triggers sur *INSERT*, dont bien sûr celui utilisé pour le log d'E-Maj. Il n'y a donc aucune contrainte à l'utilisation de *COPY* et *copy* avec E-Maj.

Pour l'utilisation d'autres outils de chargement, il convient de vérifier que les triggers sont bien activés à chaque insertion de ligne.

26.4 Réorganisation des tables de la base de données

26.4.1 Réorganisation des tables applicatives

Les tables applicatives protégées par E-Maj peuvent être réorganisées par une commande SQL *CLUSTER*. Que les triggers de logs soient actifs ou non, le processus de réorganisation n'a pas d'impact pas le contenu des tables de log.

26.4.2 Réorganisation des tables E-Maj

L'index correspondant à la clé primaire de chaque table des schémas d'E-Maj est déclaré « *cluster* », que ce soit les tables de log ou les quelques tables internes.

> **Prudence :** Aussi, l'installation d'E-Maj peut avoir un impact opérationnel sur l'exécution des commandes SQL *CLUSTER* au niveau de la base de données.

Dans le cas d'une utilisation en mode continu d'E-Maj, c'est à dire sans arrêt et relance réguliers des groupes de tables, mais avec suppression des marquess les plus anciennes, il est recommandé de procéder régulièrement à des réorganisations des tables de log E-Maj. Ceci permet ainsi de récupérer de l'espace disque inutilisé suite aux suppressions des marques.

26.5 Utilisation d'E-Maj avec de la réplication

26.5.1 Réplication physique intégrée

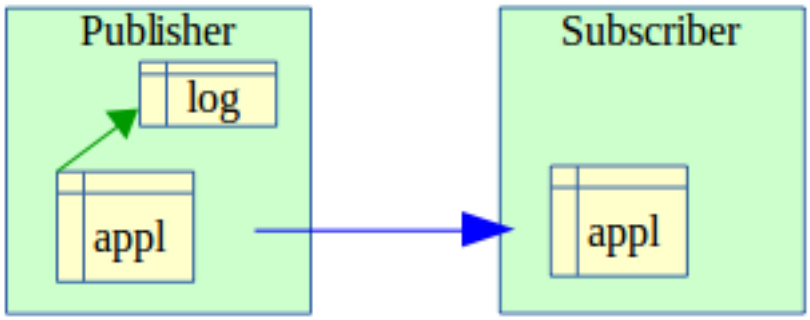
E-Maj est parfaitement compatible avec le fonctionnement des différents modes de réplication physique intégrée de PostgreSQL (archivage des *WAL* et *PITR*, *Streaming Replication* asynchrone ou synchrone). Tous les objets E-Maj des bases hébergées sur l'instance sont en effet répliqués comme toutes les autres objets de l'instance.

Néanmoins, compte tenu de la façon dont PostgreSQL gère les séquences, la valeur courante des séquences peut être un peu en avance sur les instances secondaires par rapport à l'instance maître. Pour E-Maj, ceci induit des statistiques générales indiquant parfois un nombre de lignes de log un peu supérieur à la réalité. Mais il n'y a pas de conséquence sur l'intégrité des données.

26.5.2 Réplication logique intégrée

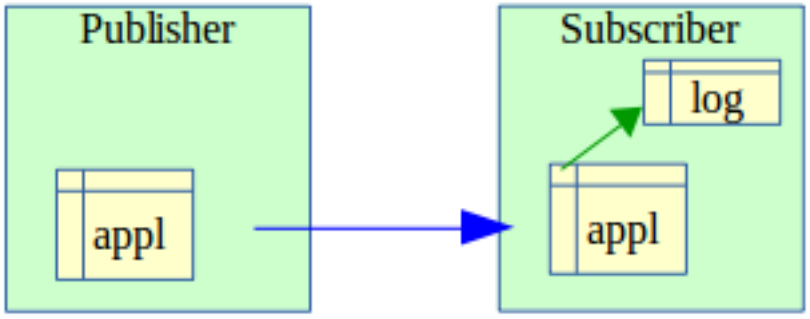
Les versions 10 et suivantes de PostgreSQL intègrent des mécanismes de réplication logique, La granularité de réplication est ici la table. L'objet de publication utilisé dans la réplication logique est assez proche du concept de groupes de tables E-Maj, à ceci près qu'une publication ne peut contenir de séquences.

Réplication de tables applicatives gérées par E-Maj



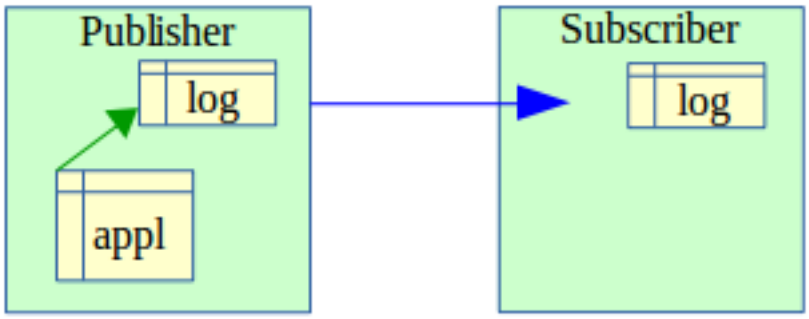
Une table applicative appartenant à un groupe de tables E-Maj peut être mise en réplication. Les éventuels rollbacks E-Maj se répliqueront naturellement côté *subscriber*, à condition qu'aucun filtre ne soit appliqué sur les types de verbes SQL répliqués.

Réplication de tables applicatives avec gestion par E-Maj côté subscriber



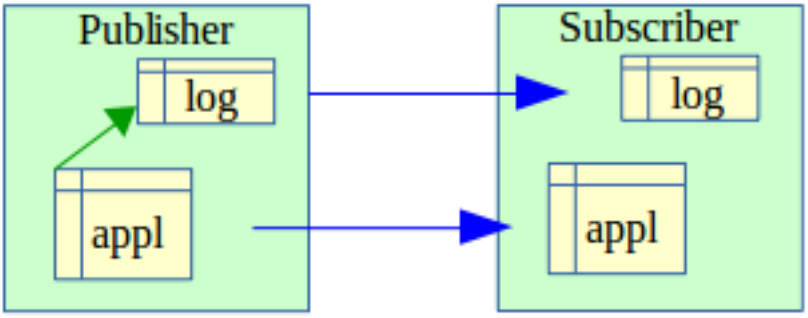
Il est possible d'insérer une table applicative dans un groupe de tables E-Maj avec des mises à jour en provenance d'un flux de réplication. Mais toutes les opérations E-Maj sont bien sûr exécutées côté *subscriber* (démarrage/arrêt du groupe, pose de marque,...). On peut effectuer un rollback E-Maj de ce groupe de tables, une fois stoppée la réplication (pour éviter des conflits dans les mises à jour). Mais à l'issue du rollback, les tables du *publisher* et du *subscriber* ne seront plus en cohérence.

Réplication de tables de log E-Maj



Il est techniquement possible de mettre une table de log E-Maj en réplication (en trouvant un moyen de construire le DDL de création – par *pg_dump* par exemple). Ceci peut permettre de dupliquer ou concentrer les données de log sur un autre serveur. Mais la table de log répliquée ne peut être utilisée qu'en **consultation**. En effet, les séquences de log et les verbes de *TRUNCATE* n'étant pas répliqués, ces logs ne peuvent pas être utilisés à d'autres fins.

Réplication de tables applicatives et de tables de log E-Maj



Tables applicatives et tables de log peuvent être répliquées simultanément. Mais comme dans le cas précédent, ces logs ne sont utilisables qu'à des fins de **consultation**. Les éventuelles opérations de rollback E-Maj ne peuvent s'effectuer que côté *publisher*.

26.5.3 Autres solutions de réplication

L'utilisation d'E-Maj avec des solutions de réplication externe basées sur des triggers, tels que *Slony* ou *Londiste*, nécessite réflexion… On évitera probablement de mettre sous réplication les tables de log et les tables techniques d'E-Maj.

CHAPITRE 27

Sensibilité aux changements de date et heure système

Pour garantir l'intégrité du contenu des tables gérées par E-Maj, il est important que le mécanisme de rollback soit insensible aux éventuels changements de date et heure du système qui héberge l'instance PostgreSQL.

Même si les date et heure de chaque mise à jour ou de chaque pose de marque sont enregistrées, ce sont les valeurs de séquences enregistrées lors des poses de marques qui servent à borner les opérations dans le temps. Ainsi, **les rollbacks comme les suppressions de marques sont insensibles aux changements éventuels de date et heure du système**.

Seules deux actions mineures peuvent être influencées par un changement de date et heure système :

- la suppression des événements les plus anciens dans la table *emaj_hist* (le délai de rétention est un intervalle de temps)
- la recherche de la marque immédiatement antérieure à une date et une heure données, telle que restituée par la fonction *emaj_get_previous_mark_group()*.

CHAPITRE 28

Performances

28.1 Surcoût de l'enregistrement des mises à jour

Enregistrer toutes les mises à jour de tables dans les tables de log E-Maj a nécessairement un impact sur la durée d'exécution de ces mises à jour. L'impact global du log sur un traitement donné dépend de nombreux facteurs. Citons en particulier :

- la part que représente l'activité de mise à jour dans ce traitement,
- les performances intrinsèques du périphérique de stockage qui supporte les tables de log.

Néanmoins, le plus souvent, le surcoût du log E-Maj sur le temps global d'un traitement se limite à quelques pourcents. Mais ce surcoût est à mettre en relation avec la durée des éventuelles sauvegardes intermédiaires de base de données évitées.

28.2 Durée d'un rollback E-Maj

La durée d'exécution d'une fonction de rollback E-Maj dépend elle aussi de nombreux facteurs, tels que :

- le nombre de mises à jour à annuler,
- les caractéristiques intrinsèques du serveur et de sa périphérie disque et la charge liée aux autres activités supportées par le serveur,
- la présence de trigger ou de clés étrangères sur les tables traitées par le rollback,
- les contentions sur les tables lors de la pose des verrous.

Pour avoir un ordre de grandeur du temps que prendrait un rollback E-Maj, on peut utiliser les fonctions *emaj_estimate_rollback_group()* et *emaj_estimate_rollback_groups()*.

28.3 Optimiser le fonctionnement d'E-Maj

Voici quelques conseils pour optimiser les performances d'E-Maj.

28.3.1 Utiliser des tablespaces

Positionner des tables sur des tablespaces permet de mieux maîtriser leur implantation sur les disques et ainsi de mieux répartir la charge d'accès à ces tables, pour peu que ces tablespaces soient physiquement implantés sur des disques ou systèmes de fichiers dédiés. Pour minimiser les perturbations que les accès aux tables de log peuvent causer aux accès aux tables applicatives, l'administrateur E-Maj dispose de deux moyens d'utiliser des tablespaces pour stocker les tables et index de log.

En positionnant un tablespace par défaut pour sa session courante avant la création des groupes de tables, les tables de log seront créées par défaut dans ce tablespace, sans autre paramétrage.

Mais, au travers de paramètres positionnés dans la table *emaj_group_def*, il est également possible de spécifier, pour chaque table et index de log, un tablespace à utiliser.

28.3.2 Déclarer les clés étrangères DEFERRABLE

Au moment de leur création, les clés étrangères (*foreign key*) peuvent être déclarées *DEFERRABLE*. Si une clé étrangère est déclarée *DEFERRABLE* et qu'aucune clause *ON DELETE* ou *ON UPDATE* n'est utilisée, elle ne sera pas supprimée en début et recréées en fin de rollback E-Maj. Les contrôles des clés étrangères pour les lignes modifiées seront simplement différés en fin de rollback, une fois toutes les tables de log traitées. En règle générale cela accélère sensiblement l'opération de rollback.

28.3.3 Modifier les paramètres sur la mémoire

Il peut être bénéfique pour les performances d'augmenter la valeur du paramètre *work_mem* avant d'effectuer un rollback E-Maj.

Si des clés étrangères doivent être recréées par une opération de rollback E-Maj, il peut être également bénéfique d'augmenter le paramètre *maintenance_work_mem*.

Si les fonctions de rollback E-Maj sont directement appelées en SQL, ces paramètres peuvent être positionnés au préalable au niveau de la session, par des requêtes du type :

```
SET work_mem = <valeur>;
SET maintenance_work_mem = <valeur>;
```

Si les opérations de rollback E-Maj sont exécutées depuis un client web, il est également possible de valoriser ces paramètres au niveau des fonctions, en tant que *superuser* :

```
ALTER FUNCTION emaj._rlbk_tbl(emaj.emaj_relation, BIGINT, BIGINT, INT, BOOLEAN) SET␣
↪work_mem = <valeur>;
ALTER FUNCTION emaj._rlbk_session_exec(INT, INT) SET maintenance_work_mem = <valeur>;
```

CHAPITRE 29

Limites d'utilisation

L'utilisation de l'extension E-Maj présente quelques limitations.

- La **version PostgreSQL** minimum requise est la version 9.5.
- Toutes les tables appartenant à un groupe de tables de type « *rollbackable* » doivent avoir une **clé primaire** explicite (*PRIMARY KEY*).
- Les tables *UNLOGGED* ou *WITH OIDS* ne peuvent pas appartenir à un groupe de tables de type « *rollbackable* ».
- Les tables temporaires (*TEMPORARY*) ne sont pas gérées par E-Maj
- L'utilisation d'une séquence globale pour une base de données induit une limite dans le nombre de mises à jour qu'E-Maj est capable de tracer tout au long de sa vie. Cette limite est égale à 2^63, soit environ 10^19 (mais seulement d'environ 10^10 sur de vieilles plate-formes). Cela permet tout de même d'enregistrer 10 millions de mises à jour par seconde (soit 100 fois les meilleurs performances des benchmarks en 2012) pendant ... 30.000 ans (et dans le pire des cas, 100 mises à jour par seconde pendant 5 ans). S'il s'avérait nécessaire de réinitialiser cette séquence, il faudrait simplement désinstaller puis réinstaller l'extension E-Maj.
- Si une **opération de DDL** est exécutée sur une table applicative appartenant à un groupe de tables, il n'est pas possible pour E-Maj de remettre la table dans un état antérieur. (plus de détails *ici*)

CHAPITRE 30

Responsabilités de l'utilisateur

30.1 Constitution des groupes de tables

La constitution des groupes de tables est fondamentale pour garantir l'intégrité des bases de données. Il est de la responsabilité de l'administrateur d'E-Maj de s'assurer que toutes les tables qui sont mises à jour par un même traitement sont bien incluses dans le même groupe de tables.

30.2 Exécution appropriée des fonctions principales

Les fonctions de *démarrage* et d'*arrêt* de groupe, de *pose de marque* et de *rollback* positionnent des verrous sur les tables du groupe pour s'assurer que des transactions de mises à jour ne sont pas en cours lors de ces opérations. Mais il est de la responsabilité de l'utilisateur d'effectuer ces opérations au « bon moment », c'est à dire à des moments qui correspondent à des points vraiment stables dans la vie de la base. Il doit également apporter une attention particulière aux éventuelles messages d'avertissement rapportés par les fonctions de rollback.

30.3 Gestion des triggers applicatifs

Des triggers peuvent avoir été créés sur des tables applicatives. Il n'est pas rare que ces triggers génèrent une ou des mises à jour sur d'autres tables. Il est alors de la responsabilité de l'administrateur E-Maj de comprendre l'impact des opérations de rollback E-Maj sur les tables concernées par des triggers et de prendre le cas échéant les mesures appropriées.

Si le trigger ajuste simplement le contenu de la ligne à insérer ou modifier, c'est la valeur finale des colonnes qui sera enregistrée dans la table de log. Ainsi en cas de rollback E-Maj, la table de log contiendra les bonnes valeurs de colonne. Et comme le trigger sera par défaut désactivé automatiquement, il ne pourra pas perturber lui même le traitement de rollback.

Si le trigger met à jour une autre table, deux cas sont à considérer :

— si la table modifiée par le trigger fait partie du même groupe de tables, la désactivation automatique du trigger et le traitement des deux tables par le rollback vont repositionner ces deux tables dans l'état attendu,

— si la table modifiée par le trigger ne fait pas partie du même groupe de tables, il est essentiel d'analyser les conséquences du rollback de la table possédant le trigger sur la table modifiée par ce trigger, afin d'éviter que le rollback ne provoque un déphasage entre les 2 tables. Le cas échéant, il peut être nécessaire de faire en sorte de ne pas désactiver le trigger. On utilisera alors la fonction *emaj_ignore_app_trigger()*. Mais la désactivation du trigger pendant l'opération de rollback peut ne pas être suffisante et d'autres actions peuvent donc être requises.

Pour des triggers plus complexes, il est indispensable de bien comprendre les impacts d'un rollback et de prendre éventuellement les mesures complémentaires appropriées lors des rollbacks E-Maj.

30.4 Modification des tables et séquences internes d'E-Maj

De par les droits qui leurs sont attribués, les super-utilisateurs et les rôles détenant les droits *emaj_adm* peuvent mettre à jour toutes les tables internes d'E-Maj.

Prudence : Mais en pratique, seules les tables *emaj_group_def* et *emaj_param* ne doivent être modifiées par ces utilisateurs. Toute modification du contenu des autres tables ou des séquences internes peut induire des corruptions de données lors d'éventuelles opérations de rollback.

CHAPITRE 31

Présentation générale d'Emaj_web

Une application web, **Emaj_web**, facilite grandement l'utilisation d'E-Maj.

Pour mémoire, un plugin pour *phpPgAdmin* existait également pour les versions d'E-Maj antérieures à la version 3.0. Mais il n'est plus maintenu depuis cette version.

Emaj_web a empreinté à *phpPgAdmin* son infrastructure (browser, barre d'icones, connexion aux bases de données,. . .) et quelques fonctions utiles telles que la consultation du contenu de tables ou la saisie de requêtes SQL.

Pour les bases de données dans lesquelles l'extension E-Maj a été installée, et si l'utilisateur est connecté avec un rôle qui dispose des autorisations nécessaires, tous les objets E-Maj sont visibles et manipulables.

Il est ainsi possible de :

- définir ou modifier la composition des groupes,
- voir la liste des groupes de tables et effectuer toutes les actions possibles, en fonction de l'état du groupe (création, suppression, démarrage, arrêt, pose de marque, rollback, ajout ou modification de commentaire),
- voir la liste des marques posées pour un groupe de tables et effectuer toutes les actions possibles les concernant (suppression, renommage, rollback, ajout ou modification de commentaire),
- obtenir toutes les statistiques sur le contenu des tables de log et en visualiser le contenu,
- suivre les opérations de rollbacks en cours d'exécution.

CHAPITRE 32

Installation du client Emaj_web

32.1 Pré-requis

Emaj_web nécessite un serveur web avec un interpréteur php.

32.2 Téléchargement du plug-in

L'application *Emaj_web* peut être téléchargée depuis le dépôt git suivant :

https://github.com/dalibo/emaj_web

32.3 Configuration de l'application

La configuration est centralisée dans un unique fichier : *emaj_web/conf/config.inc.php*. Il contient les paramètres généraux de l'application, ainsi que la description des connexions aux instances PostgreSQL.

Quand le nombre d'instances est important, il est possible de les répartir dans des *groupes d'instances*. Un groupe peut contenir des instances ou d'autres groupes d'instances.

Pour pouvoir soumettre des rollbacks en tâche de fonds (c'est à dire sans mobiliser le navigateur durant le déroulement des rollbacks), il est nécessaire de valoriser deux paramètres de configuration :

- *$conf["psql_path"]* définit le chemin de l'exécutable *psql*,
- *$conf["temp_dir"]* définit un répertoire temporaire utilisable lors des rollbacks en tâche de fonds.

Le fichier *emaj_web/conf/config.inc.php-dist* peut servir de base pour la configuration.

CHAPITRE 33

Utilisation d'Emaj_web

33.1 Accès à Emaj_web et aux bases de données

L'accès à Emaj_web depuis un navigateur affiche la page d'accueil.

Pour se connecter à une base de données, sélectionnez l'instance souhaitée dans l'arborescence de gauche ou dans l'onglet « serveurs », et remplissez les identifiants et mots de passe de connexion. Plusieurs connexions peuvent rester ouvertes simultanément.

Une fois connecté à une base de données dans laquelle l'extension emaj a été installée, l'utilisateur interagit avec l'extension en fonction des droits dont il dispose (super-utilisateur, *emaj_adm* ou *emaj_viewer*).

Située à gauche, l'arborescence de navigation offre la visibilité de toutes les instances configurées, réparties éventuellement dans des groupes d'instances, et des bases de données qu'elles contiennent. En dépliant l'objet base de données, on accède aux groupes de tables E-Maj et aux schémas existants.

Les deux icônes en bas et à droite permettent d'ajuster la largeur du l'arborescence de navigation.

33.2 Liste des groupes de tables

En sélectionnant une base de données, l'utilisateur accède à une page qui liste les groupes de tables créés dans cette base de données.

Deux listes distinctes sont affichées :
- les groupes de tables en état « *démarrés* »,
- les groupes de tables en état « *arrêtés* ».

Pour chaque groupe de tables créé, sont affichés les attributs suivants :
- sa date et son heure de création,
- le nombre de tables et de séquences applicatives qu'il contient,
- son type (« *ROLLBACKABLE* » ou « *AUDIT-SEUL* », protégé contre les rollbacks ou non),
- le nombre de marques qu'il possède,
- son éventuel commentaire associé.

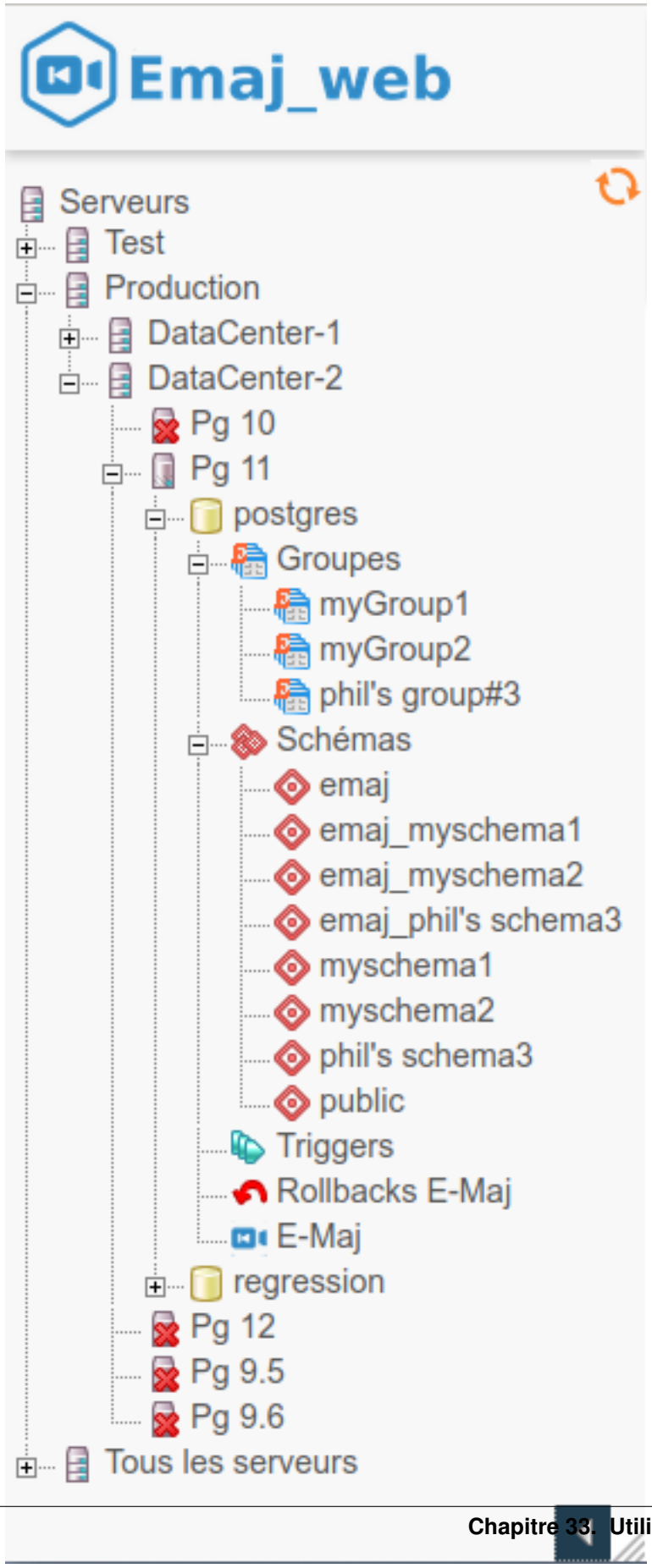


Fig. 1 – Figure 1 – Arborescence de navigation.

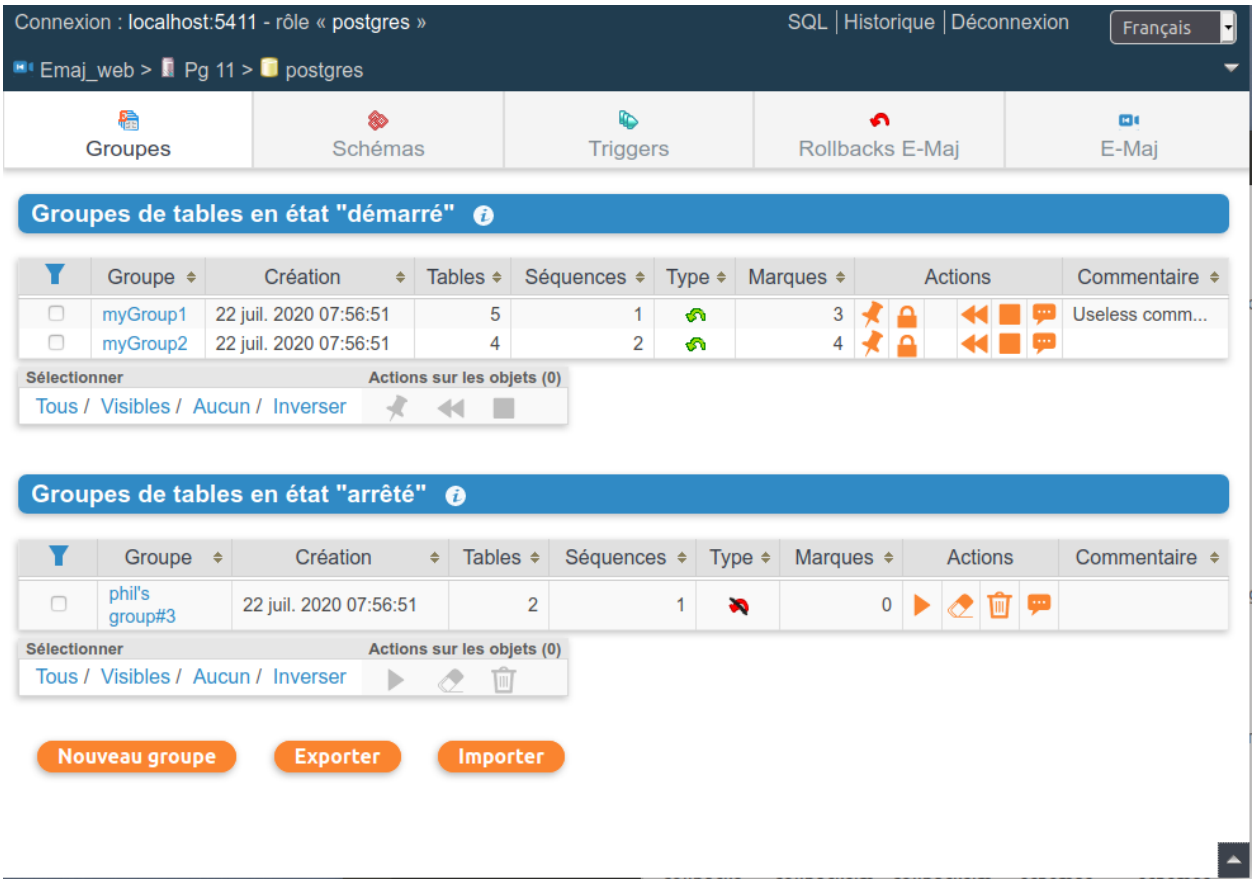


Fig. 2 – Figure 2 – Liste des groupes de tables.

Pour chaque groupe également, plusieurs boutons sont proposés afin de pouvoir effectuer les actions que son état autorise.

Trois boutons en bas de page permettent de créer un nouveau groupe de tables, d'exporter ou d'importer une configuration de groupes de tables vers ou à partir d'un fichier local.

33.3 Quelques détails de l'interface utilisateur

Les entêtes de page contiennent :
- des informations sur la connexion courante,
- 3 liens pour accéder à l'éditeur de requête SQL, à l'historique des requêtes exécutées et pour se déconnecter de l'instance courante,
- une liste déroulante pour choisir la langue utilisée dans l'interface utilisateur,
- un fil d'ariane permettant de se repérer dans l'arborescence,
- et un bouton pour aller directement en bas de page.

Deux barres d'icônes permettent de naviguer dans les différentes fonctions d'Emaj_web : l'une regroupe les fonctions globales de l'interface, et l'autre les fonctions associées à un groupe de tables particulier.

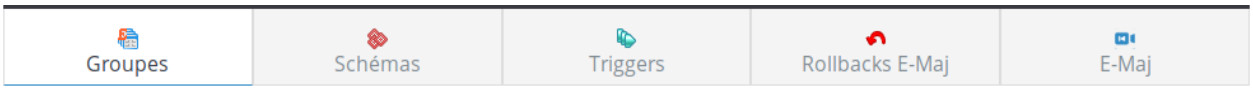


Fig. 3 – Figure 3 – Barre d'icônes principale.

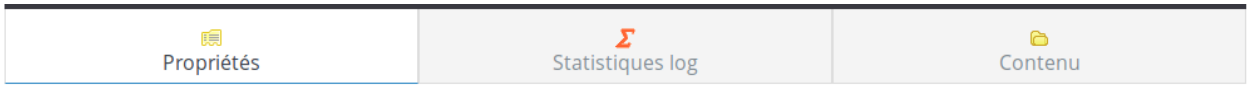


Fig. 4 – Figure 4 – Barre d'icônes des groupes de tables.

Pour les rôles de type *emaj_viewer*, certaines icônes ne sont pas visibles.

Sur certains tableaux, il est possible de trier en dynamique les lignes affichées à l'aide de petites flèches verticales situées à droite des titres de colonnes.

Sur certains tableaux également, une icône à gauche de la ligne de titre fait apparaître ou disparaître des champs de saisie permettant le filtrage des lignes affichées.

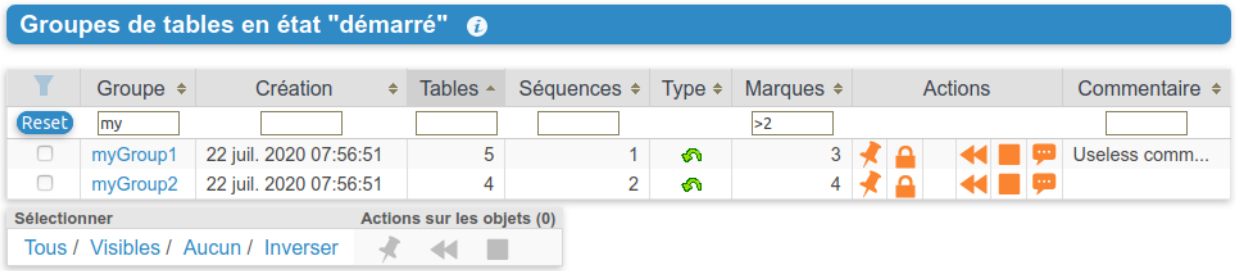


Fig. 5 – Figure 5 – Filtrage des groupes de tables démarrés. Ne sont affichés ici que les groupes de tables dont le nom comprend « *my* » et contenant plus de 2 marques, cette liste étant triée par ordre décroissant du nombre de tables.

Certains tableaux permettent d'exécuter des actions pour plusieurs objets simultanément. Dans ce cas, l'utilisateur sélectionne les objets à l'aide des cases à cocher dans la première colonne du tableau et choisit l'action à exécuter parmi les boutons accessibles sous le tableau.

Les colonnes contenant des commentaires ont une taille limitée. Mais le contenu complet des commentaires est visible en infobulle lorsque la souris passe au dessus de la cellule.

Les cellules contenant des horodatages d'événement ou des durées affichent en infobulle la valeur complète de la donnée.

33.4 Détail d'un groupe de tables

Depuis la page listant les groupes de tables, il est possible d'en savoir davantage sur un groupe de tables particulier en cliquant sur son nom. Cette page est aussi accessible par l'icône « *Propriétés* » de la barre des groupes ou par l'arborescence de gauche.

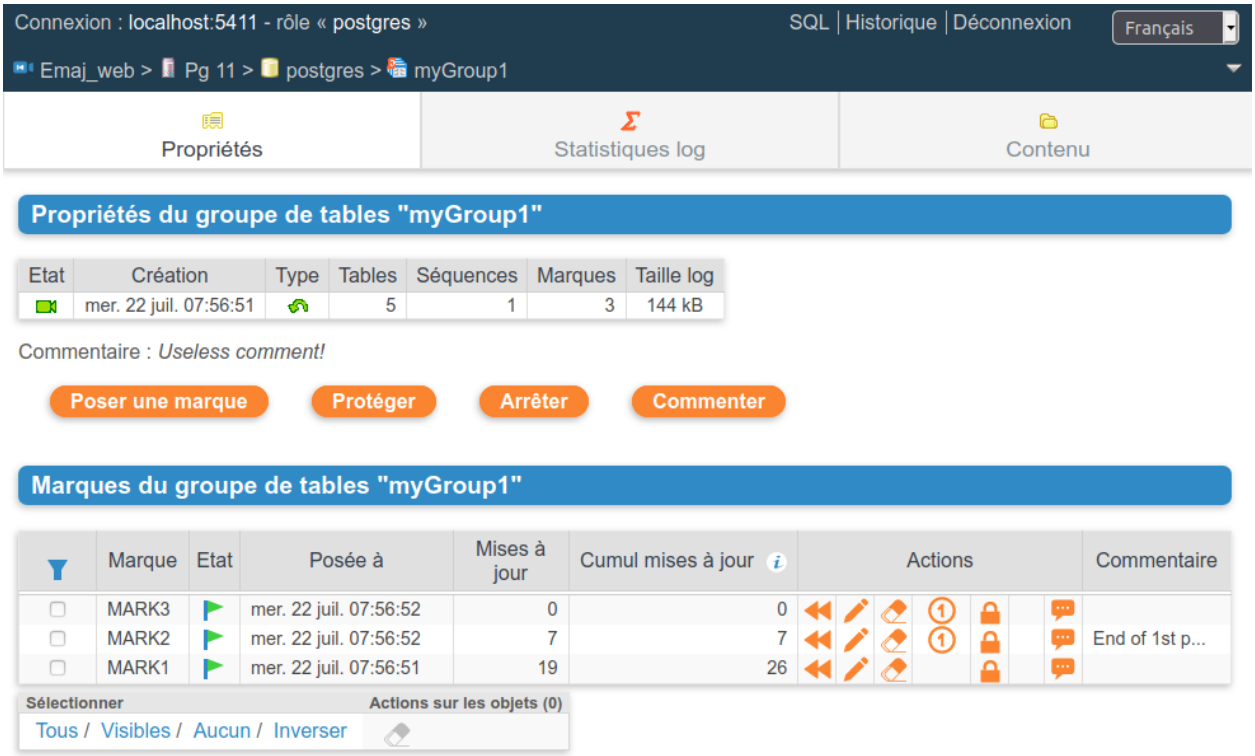


Fig. 6 – Figure 6 – Détail d'un groupe de tables

Une première ligne reprend des informations déjà affichées sur le tableau des groupes (nombre de tables et de séquences, type et nombre de marques), complété par l'espace disque utilisé par les tables de log du groupe.

Cette ligne est suivie par l'éventuel commentaire associé au groupe.

Puis une série de boutons permet de réaliser les actions que l'état du groupe permet.

L'utilisateur trouve ensuite un tableau des marques positionnées pour le groupe. Pour chacune d'elles, on trouve :

- son nom,
- sa date et son heure de pose,
- son état (actif ou non, protégé contre les rollbacks ou non),
- le nombre de lignes de log enregistrées entre cette marque et la suivante (ou la situation courante s'il s'agit de la dernière marque),
- le nombre total de lignes de log enregistrées depuis que la marque a été posée,
- l'éventuel commentaire associé à la marque.

Pour chaque marque, plusieurs boutons permettent d'exécuter toute action que son état permet.

33.5 Statistiques

L'onglet « *Statistiques log* » de la barre des groupes permet d'obtenir des statistiques sur le contenu des mises à jour enregistrées dans les tables de log pour le groupe de tables.

Deux types de statistiques peuvent être obtenues :

— des estimations du nombre de mises à jour par table, enregistrées entre 2 marques ou entre une marque et la situation présente,

— un dénombrement précis du nombre de mises à jour par table, type de requête (*INSERT/UPDATE/DELETE/TRUNCATE*) et rôle.

La figure suivante montre un exemple de statistiques détaillées.

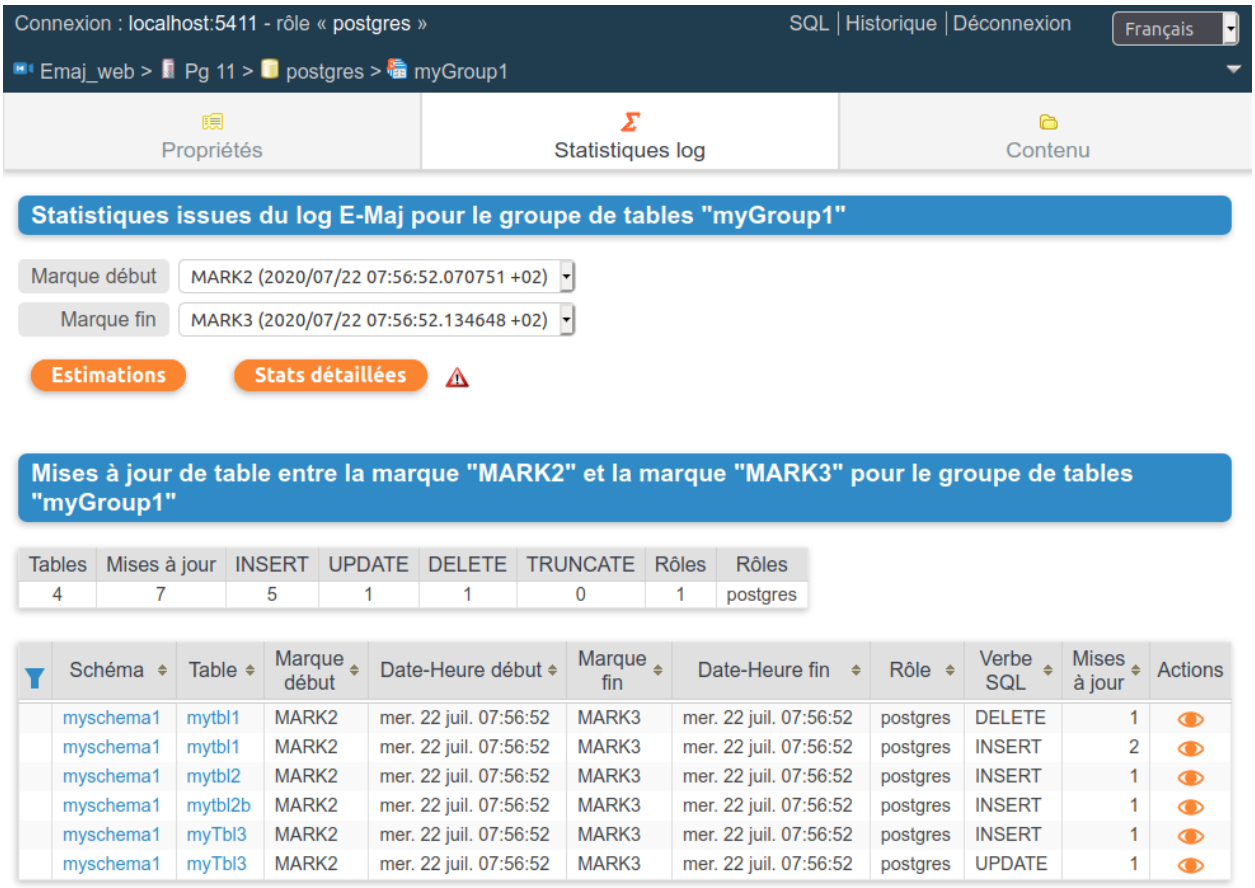


Fig. 7 – Figure 7 – Statistiques détaillées des mises à jour enregistrées entre 2 marques

La page restituée contient une première ligne contenant des compteurs globaux.

Sur chacune des lignes du tableau de statistiques, un bouton « *SQL* » permet à l'utilisateur de visualiser facilement le contenu des mises à jour enregistrées dans les tables de log. Un clic sur ce bouton ouvre l'éditeur de requêtes SQL et propose la requête visualisant le contenu de la table de log correspondant à la sélection (table, tranche de temps, rôle, type de requête). L'utilisateur peut la modifier à sa convenance avant de l'exécuter, afin, par exemple, de cibler davantage les lignes qui l'intéressent.

33.6 Contenu d'un groupe de tables

L'onglet « *Contenu* » de la barre des groupes permet d'obtenir une vision synthétique du contenu d'un groupe de tables.

Le tableau affiché reprend, pour chaque table et séquence du groupe, les caractéristiques configurées dans la table *emaj_group_def*, ainsi que la place prise par la table de log et son index.

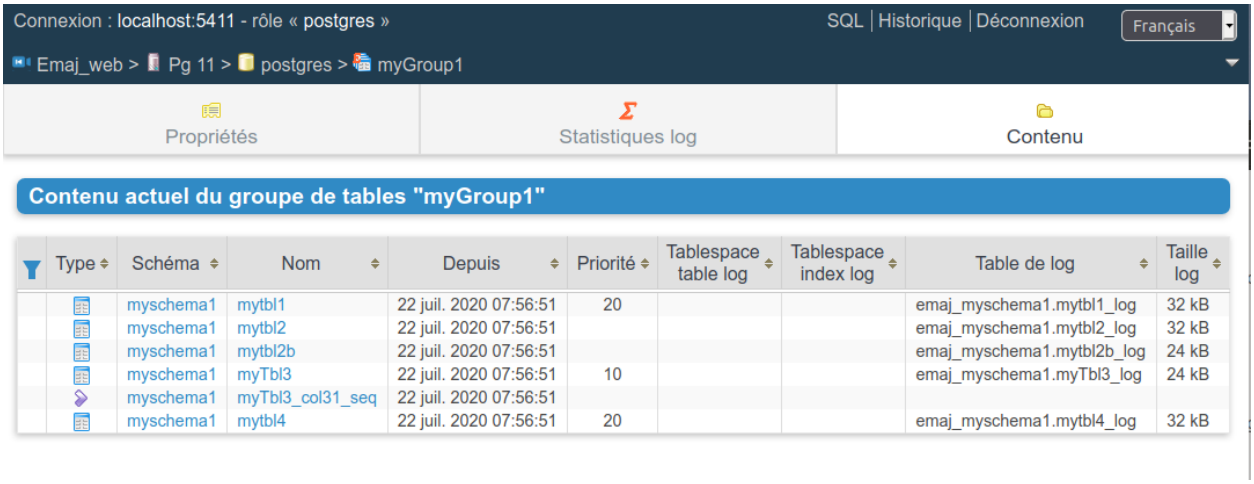


Fig. 8 – Figure 8 – Contenu d'un groupe de tables.

33.7 Schémas et configuration des groupes de tables

L'onglet « *Schémas* » permet de lister les schémas présents dans la base de données.

En en sélectionnant un, deux listes sont alors affichées : celle des tables puis celle des séquences contenues dans ce schéma.

Pour chaque liste, on trouve les propriétés E-Maj et quelques caractéristiques générales des objets. Des boutons d'action permettent d'accéder à leur structure ou contenu, et de gérer l'assignation des objets dans les groupes de tables.

33.8 Triggers

L'onglet « *Triggers* » liste les triggers applicatifs (ceux qui ne sont pas liés à E-Maj), avec leurs principales caractéristiques.

Un bouton permet de changer le mode de désactivation lors des rollbacks E-Maj.

33.9 Suivi des opérations de rollback

Une page, accessible par l'icône « *Rollbacks* » de la barre globale, permet de suivre les opérations de rollback. Trois listes distinctes sont affichées :

- les opérations de rollback en cours, avec le rappel des caractéristiques de l'opération et une estimation de la part de l'opération déjà effectuée et de la durée restante,
- les opérations de rollback terminées,

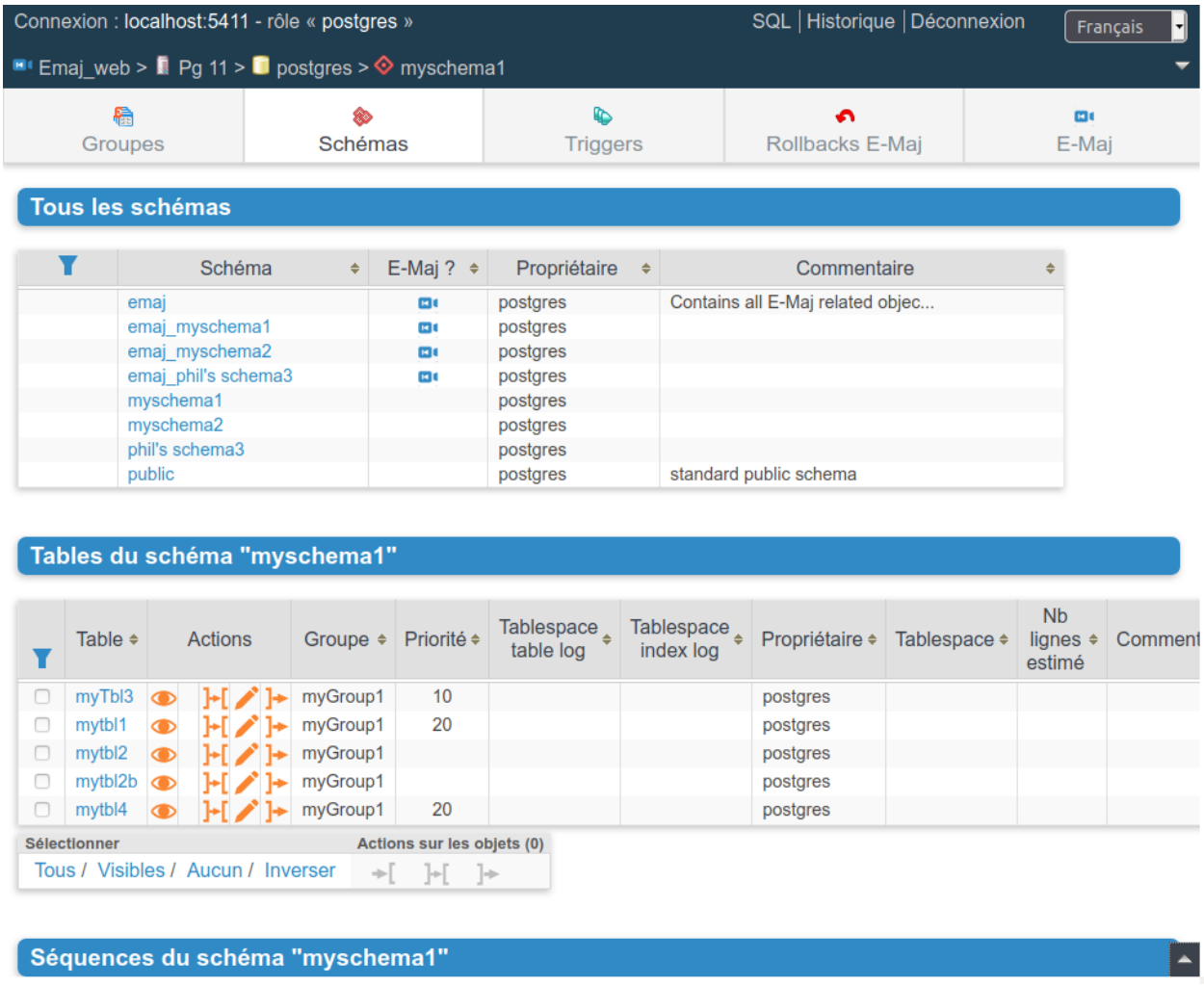


Fig. 9 – Figure 9 – Contenu des schémas et configuration des groupes de tables.

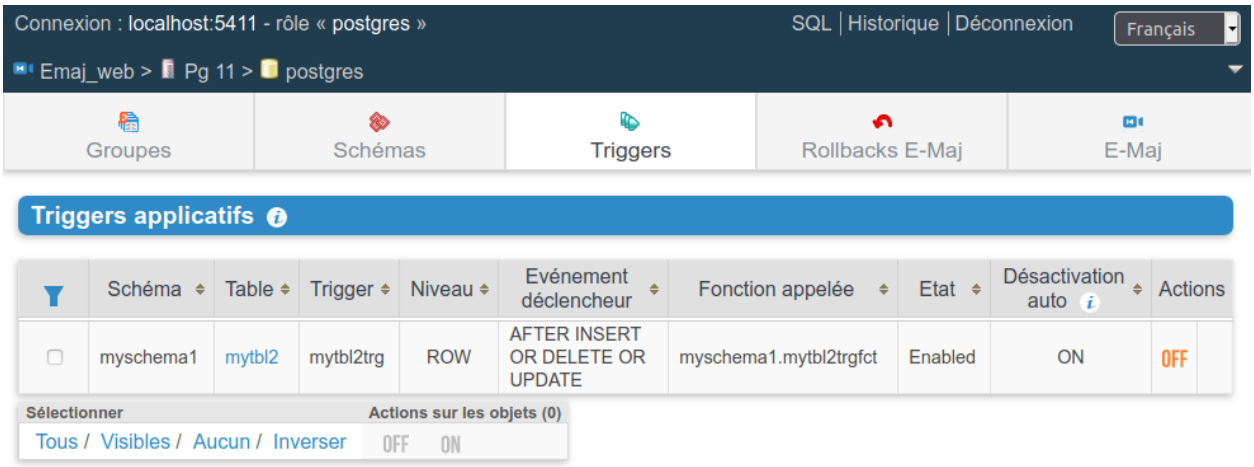


Fig. 10 – Figure 10 – Liste des triggers applicatifs

— les opérations de rollback tracés susceptibles d'être consolidées.

Pour chaque rollback tracé consolidable listé, un bouton permet d'exécuter la consolidation.

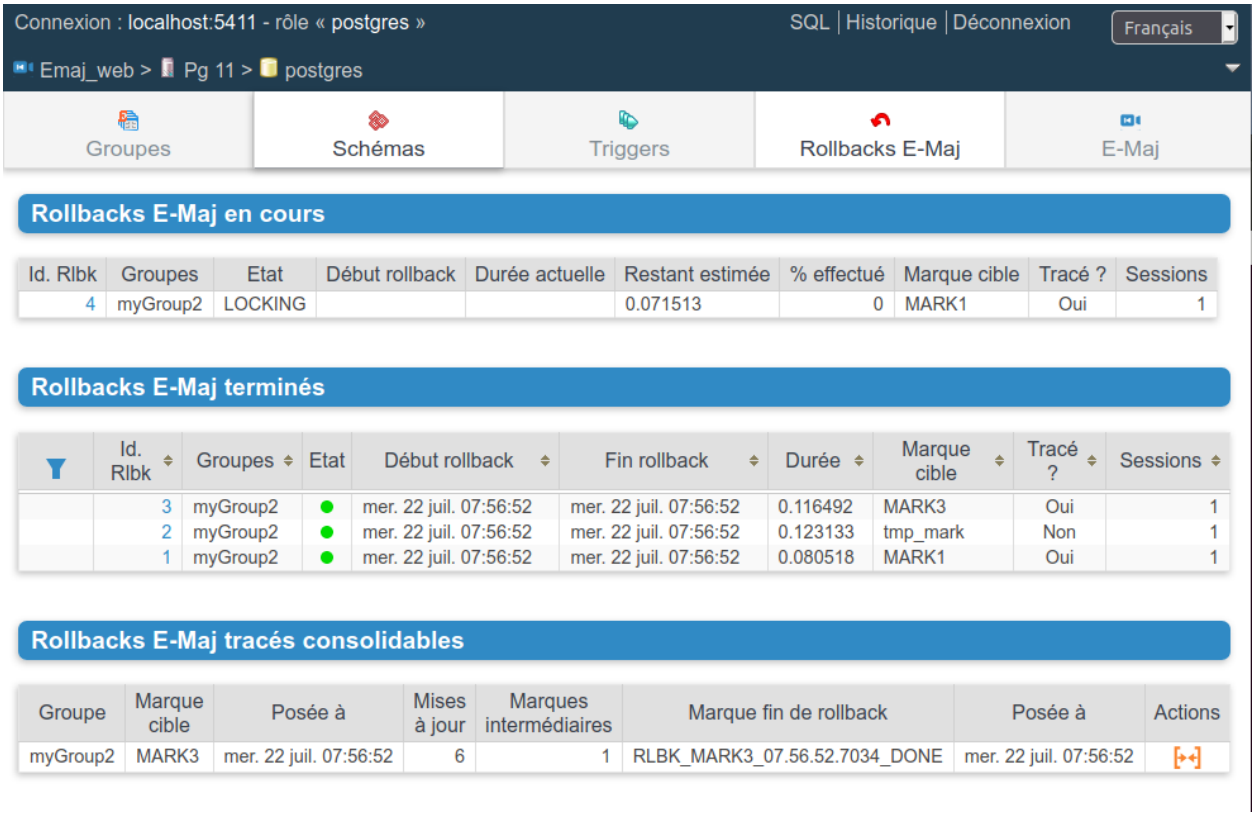


Fig. 11 – Figure 11 – Suivi des opérations de rollback.

En cliquant sur un identifiant de rollback dans l'un de ces tableaux, on accède à une page présentant des informations détaillées sur le déroulement en cours ou passé de l'opération sélectionnée.

On y trouve plus précisément des données concernant :

— l'identification du rollback,
— sa progression,
— le rapport final restitué à l'utilisateur, quand l'opération est terminée,
— ses principales caractéristiques techniques,
— la ou les sessions lancées,
— et le détail de la planification de l'opération, montrant chaque étape élémentaire, avec notamment sa durée, et optionnellement les estimations initiales calculées par E-Maj.

33.10 État de l'environnement E-Maj

En sélectionnant l'onglet « *E-Maj* » de la barre principale, l'utilisateur accède à une synthèse de l'état de l'environnement E-Maj.

Sont d'abord restitués :

— les versions de PostgreSQL et d'E-Maj installées,
— la place disque occupée par E-Maj (tables de log, tables techniques et index associés) et la part que cela représente dans la taille globale de la base de données.

Lorsque l'utilisateur est connecté avec un rôle « *superuser* », des boutons permettent, en fonction du contexte, de créer, mettre à jour ou supprimer l'extension *emaj*.

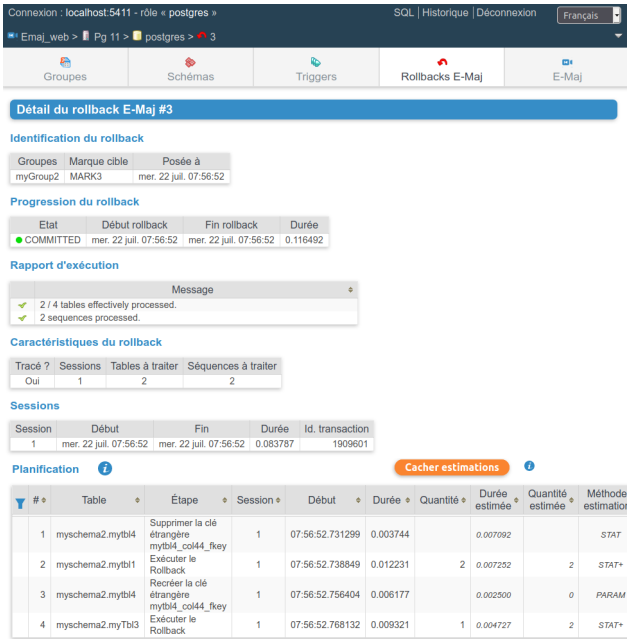


Fig. 12 – Figure 12 – Détails d'une opération de rollback.

Puis l'intégrité de l'environnement est testé ; le résultat de l'exécution de la fonction *emaj_verify_all()* est affiché.

Enfin sont listés les paramètres de fonctionnement de l'extension emaj, qu'ils soient présents dans la table *emaj_param* ou valorisés par défaut.

Deux boutons en bas de page permettent d'exporter ou d'importer une configuration de paramètres vers ou à partir d'un fichier local.

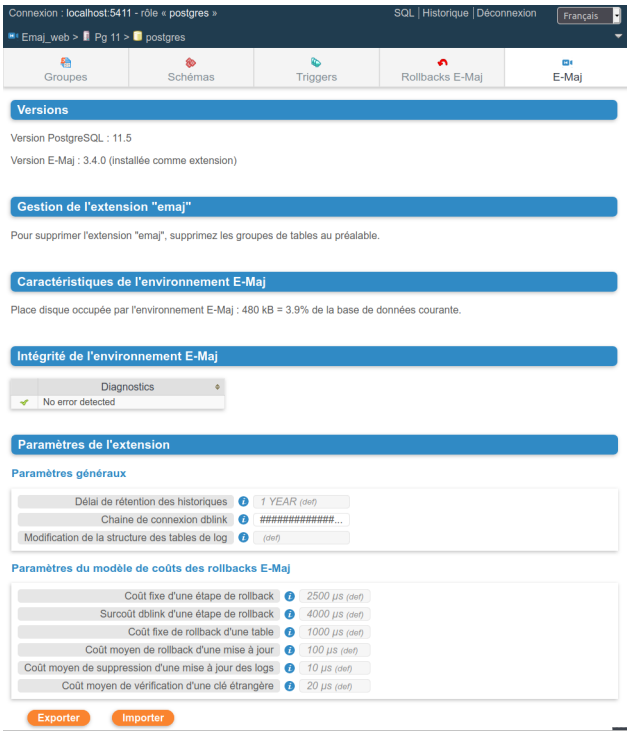


Fig. 13 – Figure 13 – État de l'environnement E-Maj

CHAPITRE 34

Contribuer au développement d'E-Maj

Toute contribution au développement et à l'amélioration de l'extension E-Maj est la bienvenue. Cette page fournit quelques informations pour faciliter ces contributions.

34.1 Bâtir l'environnement E-Maj

Le référentiel de l'extension E-Maj est hébergé sur le site *github* : https://github.com/dalibo/emaj

34.1.1 Cloner le dépôt E-Maj

La première opération à réaliser consiste donc à cloner ce dépôt en local sur son serveur/poste. Pour ce faire, utiliser les fonctionnalités de l'interface web de *github* ou taper la commande *shell* :

```
git clone https://github.com/dalibo/emaj.git
```

34.1.2 Description de l'arborescence E-Maj

On dispose alors d'une arborescence complète (hors clients web). Elle comprend tous les répertoires et fichiers décrits en *annexe*, à l'exception du contenu du répertoire *doc* maintenu séparément (voir plus bas).

L'arborescence comprend les éléments complémentaires suivants :

- le fichier *tar.index* qui permet de créer le fichier contenant la version E-Maj distribuée sur *pgxn.org*
- un répertoire *docs* avec tous les sources de la *documentation* en ligne
- dans le répertoire *sql* :
  - le fichier *emaj- -devel.sql*, source de l'extension dans sa version courante
  - le source de la version précédente *emaj- -<version_précédente>.sql*
  - un script *emaj_prepare_emaj_web_test.sql* qui prépare un environnement E-Maj pour les tests du client *Emaj_web*
- un répertoire *test* contenant tous les éléments permettant de *tester l'extension*
- un répertoire *tools* contenant un certain nombre d'outils.

34.1.3 Paramétrer les outils

Les outils présents dans le répertoire tools nécessitent d'être paramétrés en fonction de l'environnement de chacun. Un système de paramétrage couvre certains outils. Pour les autres, le fichier *tools/README* détaille les adaptations à réaliser.

Le fichier *tools/emaj_tools.profile* contient des définitions de variables et de fonctions utilisées par différents outils. En principe, ces éléments n'ont pas besoin d'être modifiés.

Création du fichier emaj_tools.env

Les paramétres susceptibles d'être modifiés sont regroupés dans le fichier *tools/emaj_tools.env*, lui même appelé par *tools/emaj_tools.profile*.

Le dépôt contient un fichier *tools/emaj_tools.env-dist* qui peut servir de squelette pour créer le fichier *emaj_tools.env*.

Le fichier *emaj_tools.env* doit contenir :

- la liste des versions de PostgreSQL supportées par la version courante d'E-Maj et qui disposent d'une instance PostgreSQL de test (variable *EMAJ_USER_PGVER*),
- pour chaque version d'instance PostgreSQL utilisée pour les tests, 6 variables décrivant la localisation des binaires et du répertoire principal de l'instance, les rôle et port ip à utiliser pour la connexion à l'instance.

34.2 Coder

34.2.1 Versionning

La version en cours de développement est nommée *devel*.

Régulièrement, et lorsque cela se justifie, une nouvelle version est créée. Elle porte un nom au format X.Y.Z.

L'outil *tools/create_version.sh* aide à la création de cette version. Il est utilisé uniquement par les mainteneurs d'E-Maj. Son utilisation n'est donc pas décrite ici.

34.2.2 Règles de codage

Le codage du script emaj- -devel.sql respecte les règles suivantes :

- structure du script : après quelques contrôles vérifiant que les conditions d'exécution du script sont respectées, les objets sont créés dans l'ordre suivant : rôles, types énumérations, séquences, tables (avec leurs index et leurs contraintes), types composites, paramètres E-Maj, fonctions de bas niveau, fonctions élémentaires de gestion des tables et séquences, fonctions de gestion des groupes de tables, fonctions d'ordre général, triggers sur événements, droits, compléments pour les extensions. Le script se termine par quelques opérations finales.
- tous les objets sont créés dans le schéma *emaj*, à l'exception de la fonction *_emaj_protection_event_trigger_fnct()*, créée dans le schéma *public*,
- les noms des tables, séquences sont préfixés par *emaj_*,
- les noms des fonctions sont préfixés par *emaj_* lorsqu'elles ont une visibilité utilisateur, ou par _ pour les fonctions internes,
- les tables internes et les fonctions appelables par les utilisateurs doivent avoir un commentaire,
- les mots-clé du langage sont mis en majuscule, les noms d'objets sont en minuscule,
- l'indentation est de 2 caractères espace,
- les lignes ne doivent pas comporter de caractère de tabulation, ne doivent pas dépasser 140 caractères et ne doivent pas se terminer par des espaces,
- dans la structure des fonctions, les délimiteurs du code doivent reprendre le nom de la fonction entouré par un caractère $ (ou *do* pour les blocs de code)

— les noms de variables sont préfixés par *v_* pour les variables simples ou *r_* pour les variables de type *RECORD*,
— le code doit être compatible avec toutes les versions de PostgreSQL supportées par la version E-Maj courante. Quand cela s'avére strictement nécessaire, le code peut être différencié en fonction de la version de PostgreSQL.

Un script perl, *tools/check_code.pl* permet d'effectuer quelques contrôles sur le formatage du script de création de l'extension. Il permet aussi de détecter les variables inutilisées. Ce script est appelé directement dans les scénarios de tests de non régression.

34.2.3 Script d'upgrade de version

E-Maj s'installe dans une database comme une extension. L'administrateur E-Maj doit pouvoir facilement *mettre à jour la version de l'extension*. Un script d'upgrade de l'extension est donc fourni pour chaque version, permettant de passer de la version précédente installée à la version suivante. Le script d'upgrade se nomme *emaj- -<version_précédente>- -devel.sql*.

Quelques règles guident les développements de ce script :
— Développer/maintenir le script d'*upgrade* en même temps que le script principal *emaj- -devel.sql*., de sorte que les tests d'une évolution incluent les cas de changement de version,
— Appliquer les mêmes règles de codage que pour le script principal,
— Autant que faire ce peut, faire en sorte que l'*upgrade* puisse être réalisé sur des groupes de tables actifs (en cours d'enregistrement) sans entamer la capacité à exécuter un *rollback E-Maj* sur une marque antérieure au changement de version.

En début de version, le script d'*upgrade* est bâti à partir d'un squelette (le fichier *tools/emaj_upgrade.template*).

Au fur et à mesure des développements, un script perl permet de synchroniser la création, la modification ou la suppression des fonctions. Il compare le script *emaj- -devel.sql* et le script de création de la version précédente et met à jour le script *emaj- -<version_précédente>- -devel.sql*. Pour son bon fonctionnement, il est essentiel de conserver les 2 balises qui délimitent le début et la fin de la partie de script qui décrit les fonctions.

Après adaptation du paramétrage (voir le fichier *TOOLS/README*), il faut simplement exécuter :

```
perl tools/sync_fct_in_upgrade_script.pl
```

Les autres parties du script doivent être codées manuellement. Si la structure d'une table interne est modifiée, le contenu de la table doit être migré (les scripts pour les versions antérieures peuvent servir d'exemple).

34.3 Tester

L'extension E-Maj, par les fonctions de *rollback*, modifie le contenu des bases de données. La fiabilité du code est donc une caractéristique essentielle. L'attention à porter aux tests est donc tout aussi essentielle.

34.3.1 Créer des instances PostgreSQL

L'idéal est de pouvoir tester E-Maj avec toutes les versions PostgreSQL supportées par l'extension (actuellement de la version 9.5 à la version 11).

Le script *tools/create_cluster.sh* est une aide à la création des instances de test. On peut s'inspirer de son contenu pour voir les caractéristiques des instances à créer. On peut aussi l'exécuter (après paramétrage comme indiqué dans *tools/README*) :

```
sh tools/create_cluster.sh <version_majeure_PostgreSQL>
```

34.3.2 Installer les dépendances logicielles

Les tests des clients peut nécessiter l'installation de quelques composants logiciels supplémentaires :
- le logiciel **php** et son interface PostgreSQL,
- le logiciel **perl** avec les modules *DBI* et *DBD : :Pg*.

34.3.3 Exécuter les tests de non régression

Un solide environnement de test est fourni dans le dépôt. Il contient :
- un outil de test,
- des scénarios de tests,
- des résultats attendus.

Les scénarios de test

Le système de test comprend 4 scénarios de test :
- un scénario standard complet,
- le même scénario mais en installant l'extension à partir de la version précédente puis *upgrade* dans la version courante,
- le même scénario mais en installant l'extension à partir du script *emaj-devel.sql* fourni pour les cas où une requête « *CREATE EXTENSION emaj* » n'est pas possible,
- un scénario réduit mais avec un *upgrade* dans la version courante alors que des groupes de tables sont actifs.

Ces scénarios font appel à des scripts *psql*, tous localisés dans *test/sql*. Les scripts enchaînent dans différents contextes des séquences d'appels de fonctions E-Maj et de requêtes SQL de préparation et de contrôle des résultats obtenus.

Généralement, en fin de script, des séquences internes sont réinitialisées pour qu'un simple ajout d'un appel de fonction dans le script ne génère pas d'impact dans le résultat des scripts suivants.

Les scripts *psql* de test doivent être maintenus en même temps que le code de l'extension.

Les résultats attendus

Pour chaque script *psql*, l'outil de test génère un fichier résultat. Ces fichiers sont différenciés en fonction de la version de PostgreSQL. Ils sont localisés dans le répertoire *test/<version_postgres>/results*.

En fin d'exécution, l'outil de test compare ces fichiers avec une référence située dans *test/<version_postgres>/expected*.

Contrairement aux fichiers du répertoire *test/<version_postgres>/results*, les fichiers du répertoire *test/<version_postgres>/expected* font partie du dépôt *git*. Ils doivent être maintenus en cohérence avec le source de l'extension et les scripts *psql*.

L'outil de test

L'outil de test, *regress.sh*, regroupe l'ensemble des fonctions de test.

Avant de pouvoir l'utiliser, il faut :
- que les instances PostgreSQL de test et le fichier *tools/emaj_tools.env* aient été créés,
- créer manuellement les répertoires *test/<version_postgres>/results*

L'outil de test se lance avec la commande :

```
tools/regress.sh
```

Comme il commence par copier le fichier *emaj.control* dans les répertoires *SHAREDIR/extension* des versions de PostgreSQL configurées, il peut demander le mot de passe du compte Linux pour exécuter des commandes *sudo*. Au lancement, il génère aussi automatiquement le fichier *emaj-devel.sql*, la version *psql* du script de création de l'extension.

Il affiche ensuite la liste des fonctions de test dans un menu. Il suffit d'indiquer la lettre correspondant au test souhaité.

On trouve :

- les tests standards pour chaque version de PostgreSQL configurée,
- les tests avec installation de la version précédente puis upgrade,
- les tests avec installation de la version par le script *emaj-devel.sql*,
- les tests avec *upgrade* de version E-Maj sur des groupes actifs,
- des tests de sauvegarde de la base par *pg_dump* et restauration, avec des versions de PostgreSQL différentes,
- un test d'*upgrade* de version de PostgreSQL par *pg_upgrade* avec une base contenant l'extension E-Maj.

Il est important d'exécuter ces quatre premières séries de test pour chaque évolution E-Maj.

Valider les résultats

Après avoir exécuté un script *psql*, *regress.sh* compare le résultat obtenu avec le résultat attendu et affiche le résultat de la comparaison sous la forme *"ok"* ou *"FAILED"*.

Voici un exemple d'affichage du déroulement d'un test (ici le scénario avec installation et upgrade de version et avec une différence détectée) :

```
Run regression test
============== dropping database "regression"         ==============
DROP DATABASE
============== creating database "regression"         ==============
CREATE DATABASE
ALTER DATABASE
============== running regression test queries        ==============
test install_upgrade          ... ok
test setup                    ... ok
test create_drop              ... ok
test start_stop               ... ok
test mark                     ... ok
test rollback                 ... ok
test misc                     ... ok
test alter                    ... ok
test alter_logging            ... ok
test viewer                   ... ok
test adm1                     ... ok
test adm2                     ... ok
test adm3                     ... ok
test client                   ... ok
test check                    ... FAILED
test cleanup                  ... ok

======================
1 of 15 tests failed.
======================

The differences that caused some tests to fail can be viewed in the
file "/home/postgres/proj/emaj/test/11/regression.diffs".  A copy of the test summary␣
↪that you see
above is saved in the file "/home/postgres/proj/emaj/test/11/regression.out".
```

Dans le cas où au moins un script ressort en différence, il convient d'analyser scrupuleusement le contenu du fichier *test/<version_postgres>/regression.diffs* pour vérifier si les écarts sont bien liés aux modifications apportées dans le code source de l'extension ou dans les scripts de test.

Une fois que les écarts relevés sont tous jugés valides, il faut copier le contenu des répertoires *test/<version_postgres>/result* dans *test/<version_postgres>/expected*. Un script *shell* permet de traiter toutes les versions PostgreSQL en une seule commande :

```
sh tools/copy2Expected.sh
```

Il peut arriver que certains résultats soient en écart à cause d'une différence de fonctionnement de PostgreSQL d'une exécution à une autre. La répétition du test permet alors de détecter ces cas.

34.3.4 Couverture des tests

Couverture de test des fonctions

Les clusters PostgreSQL de test sont configurés pour compter les exécutions des fonctions. Le script de test *check.sql* affiche les compteurs d'exécution des fonctions. Il liste aussi les fonctions E-Maj qui n'ont été exécutées dans aucun script.

Couverture de test des messages d'erreur

Un script *perl* extrait les messages d'erreur et de *warning* codés dans le fichier *sql/emaj- -devel.sql*. Il extrait ensuite les messages présents dans les fichiers du répertoire *test/10/expected*. Ceci lui permet d'afficher les cas d'erreur ou de *warning* non couverts par les tests.

Le script s'exécute avec la commande :

```
perl tools/check_error_messages.pl
```

Certains messages sont connus pour ne pas être couverts (cas d'erreurs difficilement reproductibles par exemple). Ces messages, codés dans le script *perl*, sont exclus de l'affichage final.

34.3.5 Évaluer les performances

Le répertoire *tools/performance* contient quelques scripts shell permettant de réaliser des mesures de performances. Comme le résultat des mesures est totalement dépendant de la plateforme et de l'environnement utilisés, aucun résultat de référence n'est fourni.

Les scripts couvrent les domaines suivants :

- *log_overhead/pgbench.sh* évalue le surcoût du mécanisme de log, à l'aide de pgbench,
- *large_group/large_group.sh* évalue le fonctionnement de groupes contenant un grand nombre de tables,
- *rollback/rollback_perf.sh* évalue les performances des rollbacks E-Maj avec différents profils de tables.

Pour chacun de ces fichiers, des variables sont à configurer en début de scripts,

34.4 Documenter

Une documentation au format *LibreOffice* est encore gérée par les mainteneurs. Elle dispose de son propre dépôt *github* : *emaj_doc*. De ce fait, le dossier *doc* du dépôt principal reste vide.

La documentation en ligne est gérée avec *sphinx*. Elle est localiséeœ dans le répertoire *docs*.

Pour installer *sphinx*, se référer au fichier *docs/README.rst*.

La documentation existe en deux langues, l'anglais et le français. En fonction de la langue, les sources des documents sont localisés dans */docs/en* et */docs/fr*. Ces documents sont au format *ReStructured Text*.

Pour compiler la documentation dans une langue, se placer dans le répertoire *docs/<langue>* et lancer la commande :

```
make html
```

Quand il n'y a plus d'erreur de compilation, la documentation peut être visualisée en local sur un navigateur, en ouvrant le fichier *docs/<langue>/_build/html/index.html*.

La mise à jour de la documentation présente sur le site *readthedocs.org* est automatique dès que le dépôt présent sur *github* est mis à jour.

34.5 Soumettre un patch

Tout patch peut être proposé aux mainteneurs d'E-Maj au travers d'un *Pull Request* sur le site *github*.

Avant de soumettre un patch, il peut être utile d'ouvrir une « *issue* » sur *github*, afin d'engager un dialogue avec les mainteneurs et ainsi avancer au mieux dans la réalisation du patch.

CHAPITRE 35

Liste des fonctions E-Maj

Les fonctions E-Maj disponibles pour les utilisateurs peuvent être regroupées en trois catégories. Elles sont listées ci-dessous par ordre alphabétique.

Toutes ces fonctions sont appelables par les rôles disposant des privilèges *emaj_adm*. Les tableaux précisent celles qui sont également appelables par les rôles *emaj_viewer* (marque *(V)* derrière le nom de la fonction).

35.1 Fonctions de niveau tables et séquences

Fonctions	Paramètres en entrée	Données restituées
emaj_assign_sequence	schéma TEXT, séquence TEXT, groupe TEXT, [marque TEXT]	1 INT
emaj_assign_sequences	schéma TEXT, tableau.séquences TEXT[], groupe TEXT, [marque TEXT]	nb.séquences INT
emaj_assign_sequences	schéma TEXT, filtre.séquences.à.inclure TEXT, filtre.séquences.à.exclure TEXT, groupe TEXT, [marque TEXT]	nb.séquences INT
emaj_assign_table	schéma TEXT, table TEXT, groupe TEXT, [propriétés JSONB] [marque TEXT]	1 INT
emaj_assign_tables	schéma TEXT, tableau.tables TEXT[], groupe TEXT, [propriétés JSONB] [marque TEXT]	nb.tables INT
emaj_assign_tables	schéma TEXT, filtre.tables.à.inclure TEXT, filtre.tables.à.exclure TEXT, groupe TEXT, [propriétés JSONB] [marque TEXT]	nb.tables INT

emaj_get_current_log_table (V)	schéma TEXT,	(schéma.log TEXT, table.log TEXT)

35.2 Fonctions de niveau groupe de tables

Fonctions	Paramètres en entrée	Données restituées
emaj_alter_group	groupe TEXT, [marque TEXT]	nb.tables.et.seq INT
emaj_alter_groups	tableau.groupes TEXT[], [marque TEXT]	nb.tables.et.seq INT
emaj_comment_group	groupe TEXT, commentaire TEXT	
emaj_comment_mark_group	groupe TEXT, marque TEXT, commentaire TEXT	
emaj_consolidate_rollback_group	groupe TEXT, marque.fin.rollback TEXT	nb.tables.et.seq INT
emaj_create_group	groupe TEXT, [est.rollbackable BOOLEAN] [est.vide BOOLEAN]	nb.tables.et.seq INT
emaj_delete_before_mark_group	groupe TEXT, marque TEXT	nb.marques.supprimées INT
emaj_delete_mark_group	groupe TEXT, marque TEXT	1 INT
emaj_detailed_log_stat_group (V)	groupe TEXT, marque.début TEXT, marque.fin TEXT	SETOF emaj_detailed_log_stat_type
emaj_detailed_log_stat_groups (V)	tableau.groupes TEXT[], marque.début TEXT, marque.fin TEXT	SETOF emaj_detailed_log_stat_type

Suite sur la page suivante

Tableau 1 – suite de la page précédente

Fonctions	Paramètres en entrée	Données restituées
emaj_drop_group	groupe TEXT	nb.tables.et.seq INT
emaj_estimate_rollback_group (V)	groupe TEXT, marque TEXT	durée INTERVAL
emaj_estimate_rollback_groups (V)	tableau.groupes TEXT[], marque TEXT	durée INTERVAL
emaj_force_drop_group	groupe TEXT	nb.tables.et.seq INT
emaj_force_stop_group	groupe TEXT	nb.tables.et.seq INT
emaj_gen_sql_group	groupe TEXT, marque.début TEXT, marque.fin TEXT, fichier.sortie TEXT, [tableau.tables.seq TEXT[]]	nb.req.générées BIGINT
emaj_gen_sql_groups	tableau.groupes TEXT[], marque.début TEXT, marque.fin TEXT, fichier.sortie TEXT, [tableau.tables.seq TEXT[]]	nb.req.générées BIGINT
emaj_get_previous_mark_group (V)	groupe TEXT, date.heure TIMESTAMPTZ	marque TEXT
emaj_get_previous_mark_group (V)	groupe TEXT, marque TEXT	marque TEXT
emaj_log_stat_group (V)	groupe TEXT, marque.début TEXT, marque.fin TEXT	SETOF emaj_log_stat_type

Suite sur la page suivante

Tableau 1 – suite de la page précédente

Fonctions	Paramètres en entrée	Données restituées
emaj_log_stat_groups (V)	tableau.groupes TEXT[], marque.début TEXT, marque.fin TEXT	SETOF emaj_log_stat_type
emaj_logged_rollback_group	groupe TEXT, marque TEXT, est.modif.groupe.autorisée BOOLEAN	SETOF (sévérité TEXT, message TEXT)
emaj_logged_rollback_groups	tableau.groupes TEXT[], marque TEXT, est.modif.groupe.autorisée BOOLEAN	SETOF (sévérité TEXT, message TEXT)
emaj_protect_group	groupe TEXT	0/1 INT
emaj_protect_mark_group	groupe TEXT, marque TEXT	0/1 INT
emaj_rename_mark_group	groupe TEXT, marque TEXT, nouveau.nom TEXT	
emaj_reset_group	groupe TEXT	nb.tables.et.seq INT
emaj_rollback_group	groupe TEXT, marque TEXT, est_modif_groupe_autorisé BOOLEAN	SETOF (sévérité TEXT, message TEXT)
emaj_rollback_groups	tableau.groupes TEXT[], marque TEXT, est_modif_groupe_autorisé BOOLEAN	SETOF (sévérité TEXT, message TEXT)
emaj_set_mark_group	groupe TEXT, [marque TEXT]	nb.tables.et.seq INT

Suite sur la page suivante

Tableau 1 – suite de la page précédente

Fonctions	Paramètres en entrée	Données restituées
emaj_set_mark_groups	tableau.groupes TEXT[], [marque TEXT]	nb.tables.et.seq INT
emaj_snap_group	groupe TEXT, répertoire TEXT, options.copy TEXT	nb.tables.et.seq INT
emaj_snap_log_group	groupe TEXT, marque.début TEXT, marque.fin TEXT, répertoire TEXT, options.copy TEXT	nb.tables.et.seq INT
emaj_start_group	groupe TEXT, [marque TEXT], [reset.log BOOLEAN]	nb.tables.et.seq INT
emaj_start_groups	tableau.groupes TEXT[], [marque TEXT], [reset.log BOOLEAN]	nb.tables.et.seq INT
emaj_stop_group	groupe TEXT, [marque TEXT]	nb.tables.et.seq INT
emaj_stop_groups	tableau.groupes TEXT[], [marque TEXT]	nb.tables.et.seq INT
emaj_sync_def_group	groupe TEXT	nb.tables.et.seq INT
emaj_unprotect_group	groupe TEXT	0/1 INT
emaj_unprotect_mark_group	groupe TEXT, marque TEXT	0/1 INT

35.3 Fonctions de niveau général

Fonctions	Paramètres en entrée	Données restituées
emaj_cleanup_rollback_state		nb.rollback INT
emaj_disable_protection_by_event_triggers		nb.triggers INT
emaj_enable_protection_by_event_triggers		nb.triggers INT
emaj_export_groups_configuration	NULL, [tableau.groupes TEXT[]]	configuration JSON
emaj_export_groups_configuration	fichier TEXT, [tableau.groupes TEXT[]]	nb.groupes INT
emaj_export_parameters_configuration		paramètres JSON
emaj_export_parameters_configuration	fichier TEXT	nb.paramètres INT
emaj_get_consolidable_rollbacks (V)		SETOF emaj_consolidable_rollback_type
emaj_import_groups_configuration	groupes JSON, [tableau.groupes TEXT[]] [traiter.groupes.démarrés BOOLEAN]	nb.groupes INT
emaj_import_groups_configuration	fichier TEXT, [tableau.groupes TEXT[]] [traiter.groupes.démarrés BOOLEAN]	nb.groupes INT
emaj_import_parameters_configuration	paramètres JSON, [suppression.conf BOOLEAN)]	nb.paramètres INT
emaj_import_parameters_configuration	fichier TEXT, [suppression.conf BOOLEAN)]	nb.paramètres INT
emaj_purge_histories	délai.rétention INTERVAL	
emaj_rollback_activity (V)		SETOF emaj_rollback_activity_type
emaj_verify_all (V)		SETOF TEXT

CHAPITRE 36

Contenu de la distribution E-Maj

Après *installation*, une version d'E-Maj contient les fichiers suivants.

Fichiers	Usage
sql/emaj–<version>.sql	script d'installation de l'extension
sql/emaj-<version>.sql	script psql alternatif d'installation
sql/emaj–3.3.0–3.4.0.sql	script d'upgrade de l'extension de 3.3.0 vers 3.4.0
sql/emaj–3.2.0–3.3.0.sql	script d'upgrade de l'extension de 3.2.0 vers 3.3.0
sql/emaj–3.1.0–3.2.0.sql	script d'upgrade de l'extension de 3.1.0 vers 3.2.0
sql/emaj–3.0.0–3.1.0.sql	script d'upgrade de l'extension de 3.0.0 vers 3.1.0
sql/emaj–2.3.1–3.0.0.sql	script d'upgrade de l'extension de 2.3.1 vers 3.0.0
sql/emaj–2.3.0–2.3.1.sql	script d'upgrade de l'extension de 2.3.0 vers 2.3.1
sql/emaj–2.2.3–2.3.0.sql	script d'upgrade de l'extension de 2.2.3 vers 2.3.0
sql/emaj–2.2.2–2.2.3.sql	script d'upgrade de l'extension de 2.2.2 vers 2.2.3
sql/emaj–2.2.1–2.2.2.sql	script d'upgrade de l'extension de 2.2.1 vers 2.2.2
sql/emaj–2.2.0–2.2.1.sql	script d'upgrade de l'extension de 2.2.0 vers 2.2.1
sql/emaj–2.1.0–2.2.0.sql	script d'upgrade de l'extension de 2.1.0 vers 2.2.0
sql/emaj–2.0.1–2.1.0.sql	script d'upgrade de l'extension de 2.0.1 vers 2.1.0
sql/emaj–2.0.0–2.0.1.sql	script d'upgrade de l'extension de 2.0.0 vers 2.0.1
sql/emaj–1.3.1–2.0.0.sql	script d'upgrade de l'extension de 1.3.1 vers 2.0.0
sql/emaj–unpackaged–1.3.1.sql	script de transformation en extension d'une version 1.3.1 existante
sql/emaj-1.3.0-to-1.3.1.sql	script de mise à jour d'une version E-Maj 1.3.0
sql/emaj-1.2.0-to-1.3.0.sql	script de mise à jour d'une version E-Maj 1.2.0
sql/emaj-1.1.0-to-1.2.0.sql	script de mise à jour d'une version E-Maj 1.1.0
sql/emaj-1.0.2-to-1.1.0.sql	script de mise à jour d'une version E-Maj 1.0.2
sql/emaj-1.0.1-to-1.0.2.sql	script de mise à jour d'une version E-Maj 1.0.1
sql/emaj-1.0.0-to-1.0.1.sql	script de mise à jour d'une version E-Maj 1.0.0
sql/emaj-0.11.1-to-1.0.0.sql	script de mise à jour d'une version E-Maj 0.11.1
sql/emaj-0.11.0-to-0.11.1.sql	script de mise à jour d'une version E-Maj 0.11.0
sql/emaj_demo.sql	script psql de démonstration d" E-Maj
sql/emaj_prepare_parallel_rollback_test.sql	script psql de test pour les rollbacks parallélisés

Suite sur la page suivante

Tableau 1 – suite de la page précédente

Fichiers	Usage
sql/emaj_uninstall.sql	script psql de désinstallation
README.md	documentation réduite de l'extension
CHANGES.md	notes de versions
AUTHORS.md	identification des auteurs
LICENSE	information sur la licence utilisée pour E-Maj
META.json	données techniques destinées à PGXN
emaj.control	fichier de contrôle utilisé par la gestion intégrée des extensions
doc/Emaj.<version>_doc_en.pdf	documentation en anglais de l'extension E-Maj
doc/Emaj.<version>_doc_fr.pdf	documentation en français de l'extension E-Maj
doc/Emaj.<version>_pres.en.odp	présentation en anglais de l'extension E-Maj
doc/Emaj.<version>_pres.fr.odp	présentation en français de l'extension E-Maj
doc/Emaj.<version>_pres.en.pdf	présentation en anglais de l'extension E-Maj (version pdf)
doc/Emaj.<version>_pres.fr.pdf	présentation en français de l'extension E-Maj (version pdf)
client/emajParallelRollback.php	client php pour les rollbacks parallélisés
client/emajParallelRollback.pl	client perl pour les rollbacks parallélisés
client/emajRollbackMonitor.php	client php pour le suivi des rollbacks
client/emajRollbackMonitor.pl	client perl pour le suivi des rollbacks

CHAPITRE 37

Matrice de compatibilité des versions PostgreSQL et E-Maj

Versions PostgreSQL		Versions E-Maj	
Min	Max	Min	Date
8.2	8.4	0.4.0	01/2010
8.2	9.0	0.8.0	10/2010
8.2	9.1	0.10.0	11/2011
8.2	9.2	0.11.1	07/2012
8.3	9.3	1.1.0	10/2013
8.3	9.5	1.2.0	01/2016
8.3	9.6	1.3.1	09/2016
9.1	9.6	2.0.0	11/2016
9.1	10	2.1.0	08/2017
9.2	10	2.3.0	07/2018
9.2	11	2.3.1	09/2018
9.5	11	3.0.0	03/2019
9.5	12	3.1.0	06/2019
9.5	13	3.3.0	03/2020

CHAPITRE 38

Index et tables

— genindex
— modindex
— search