

E-Maj

Et vos données PostgreSQL voyagent dans le temps !

L'acronyme de
« *E*nregistrement des *M*ises *A* *J*our »

E-Maj, ça sert à quoi ?

- E-Maj permet de **déplacer dans le temps** des contenus de données, avec une granularité de niveau table
- En **enregistrant les mises à jours** sur des ensembles de tables applicatives, on peut
  - les **dénombrer** (fonction statistique),
  - les **consulter** facilement (fonction d'audit),
  - les **annuler** (fonction de « rollback »),
  - les **rejouer** (génération de script, ou annulation d'une annulation…)
- Utilisable avec
  - des applications en test ou en production
  - des bases de données de toute taille

La plus-value

- En environnement de **test**
  - Aide l'organisation des tests applicatifs en fournissant un moyen rapide
    - d'examiner les mises à jour générées par l'application
    - d'annuler les mises à jour issues d'une exécution de programmes et de pouvoir ainsi répéter facilement des tests
- En environnement de **production**
  - Permet d'annuler des traitements
    - sans devoir sauver et restaurer l'instance par `pg_dump`/`pg_restore` ou copie physique
    - avec une granularité plus fine
  - Évite de perdre des nuits complètes de traitement batch, en facilitant les reprises sur incident
  - D'autant plus intéressant que les tables sont volumineuses et les mises à jour peu nombreuses

Les composants



- **E-Maj,** le cœur
  - Une extension PostgreSQL
  - Open Source, sous licence GPL
  - Téléchargeable depuis pgxn.org - https://pgxn.org/dist/e-maj/
  - Sources disponibles sur github.com - https://github.com/dalibo/emaj
- 1 **clients web**
  - Emaj_web - https://github.com/dalibo/emaj_web
- Une **documentation** en ligne
  - En français (ou anglais) - https://emaj.readthedocs.io/fr/latest/

Les caractéristiques qui ont guidé le design

- **Fiabilité**
  - Intégrité absolue des données après annulation de mises à jour
  - Gestion de tous les objets usuels (tables, séquences, contraintes,...)
- **Facilité** d'utilisation pour les DBAs, exploitants, développeurs et testeurs d'applications,...
  - Facilement compréhensible et utilisable
  - Facile à automatiser (donc scriptable)
- **Performance**
  - Surcoût du log limité
  - Durée de « rollback » acceptable
- **Sécurité**
- **Maintenabilité**

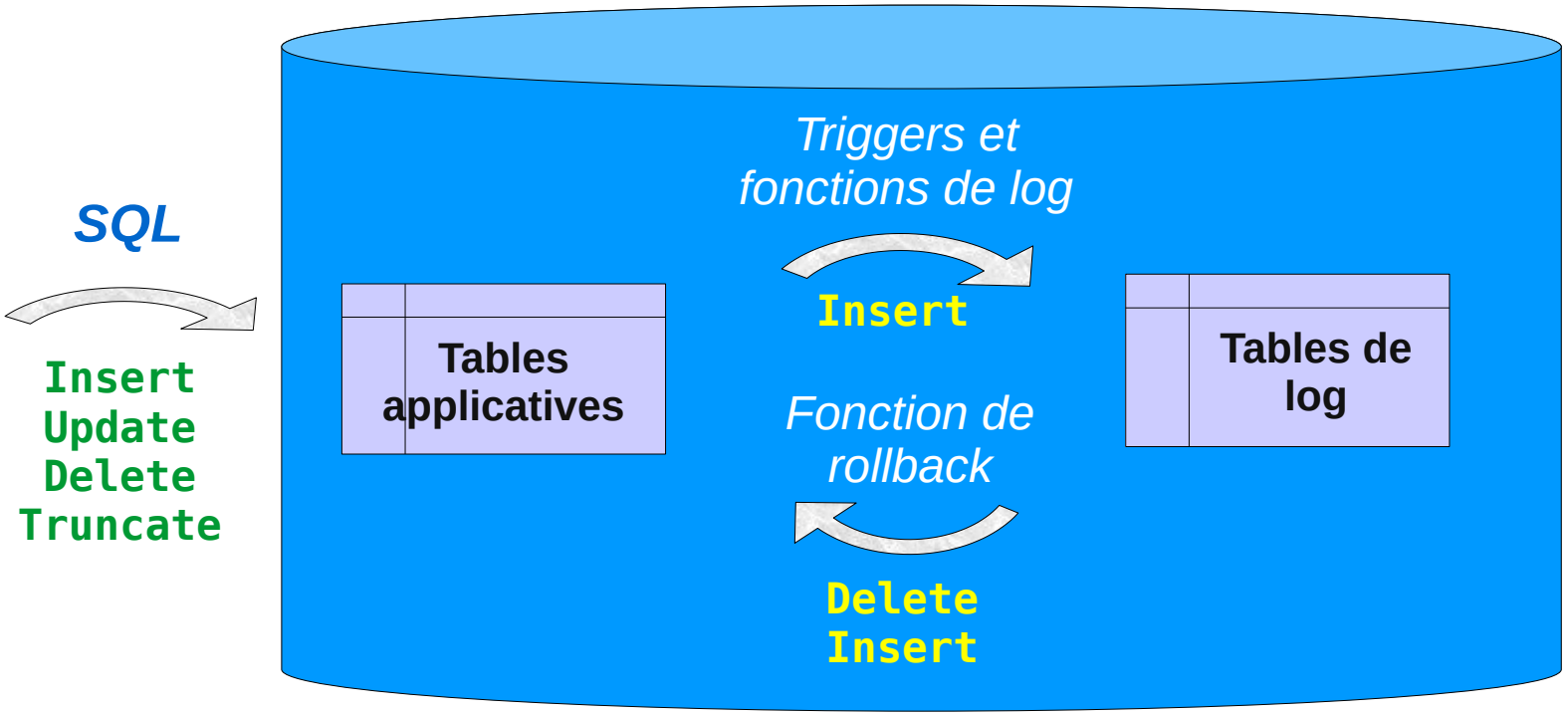
Concepts

- **Groupe de tables** = ensemble de tables et/ou séquences d'une base de données, appartenant à un ou plusieurs schémas, et ayant le même rythme de vie ; c'est le principal objet manipulé par l'utilisateur
- **Marque** = point stable dans la vie d'un « groupe de tables », et dont on peut retrouver l'état ; elle est identifiée par un nom
- **Rollback E-Maj** = positionnement d'un « groupe de tables » à l'état dans lequel il se trouvait lors de la prise d'une « marque »
  - NB : ce concept est différent du rollback des transactions effectué par le SGBD
    - le « rollback du SGBD » annule la transaction courante
    - le « rollback E-Maj » annule les mises à jour de multiples transactions « commitées »

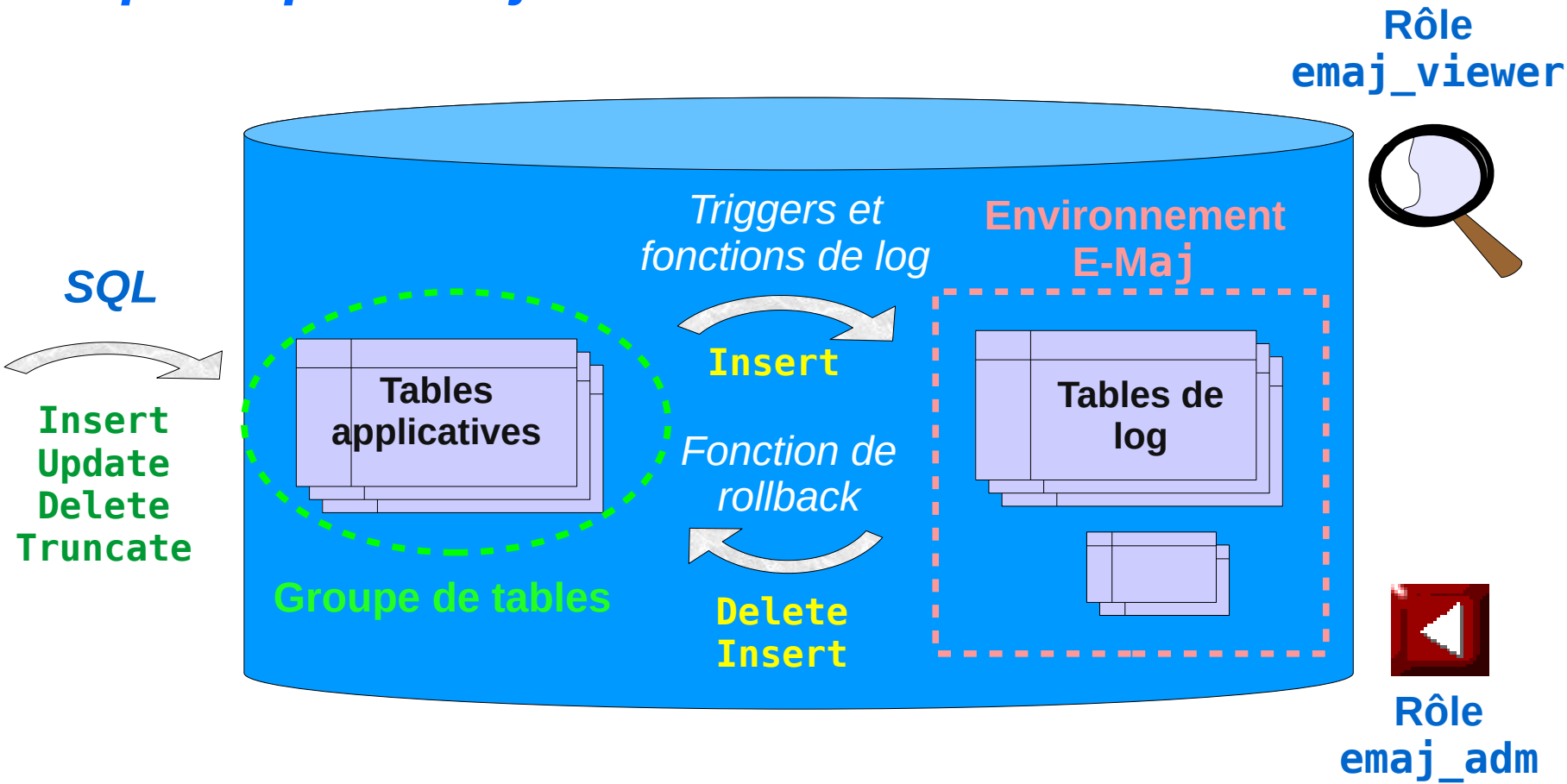
Concepts (2)

- Par défaut, un groupe de tables est créé **rollbackable**
- Un groupe de tables peut être créé **audit-only**
  - Pas de rollback E-Maj possible
  - Utile pour tracer les mises à jour de tables sans PRIMARY KEY ou de type UNLOGGED
- **Session de log** = intervalle de temps pendant lequel un « groupe de tables » enregistre les mises à jour ; elle est bornée par les actions de **démarrage** et d'**arrêt** du « groupe de tables »

Un log des mises à jour basé sur des triggers



Les principaux objets



Gestion des séquences applicatives

- Les incréments des séquences ne sont pas enregistrées individuellement
- Pose d'une marque sur un groupe de tables
  - L'état de chaque séquence du groupe est enregistré dans une table interne
- Rollback E-Maj
  - Chaque séquence est remise dans l'état enregistré à la pose de la marque ciblée

Installation

- Installation standard
  - `pgxn install E-Maj --sudo`
  - Se connecter à la base ciblée en tant que super-utilisateur puis
    - `CREATE EXTENSION emaj CASCADE;`
- Installation possible sans droit superuser (avec quelques limitations)
  - Télécharger et décompresser la dernière version d'E-Maj depuis pgxn.org
  - `psql … -f sql/emaj-<version>.sql`
- L'installation ajoute à la database
  - les extensions `dblink` et `btree_gist` si besoin
  - 2 rôles (`emaj_adm`, `emaj_viewer`)
  - 1 schema '`emaj`' avec : 21 tables techniques, 12 types, 2 vues, 1 séquence, 3 event triggers et plus de 200 fonctions

Initialisation

- Pour chaque groupe de tables :
  - 1) Création du groupe vide
    `SELECT emaj_create_group (groupe, est_rollbackable [,commentaire]);`
  - 2) Ajout de tables et séquences
    `SELECT emaj_assign_tables (schéma, regexp inclusion, regexp exclusion, groupe);`
    `SELECT emaj_assign_sequences (schéma, regexp inclusion, regexp exclusion, groupe);`
    - Ex : toutes les tables d'une schéma sauf celles suffixées par sav :
      `'.*', 'sav$'`
    - crée pour chaque table applicative : 1 table de log + 1 séquence de log + 1 trigger de log et sa fonction
- NB : `SELECT emaj_drop_group (groupe)`
  - … supprime un groupe existant

Les 3 fonctions principales de gestion des groupes

- « Démarrage » d'un groupe
  - `emaj_start_group (`*`groupe [,marque] [,reset.logs] [,groupe.actif.admis]`*`)`
    active les triggers de log et pose une marque initiale
- Pose une marque intermédiaire
  - `emaj_set_mark_group (`*`groupe [,marque] [,commentaire]`*`)`
    pose une marque intermédiaire
- « Arrêt » d'un groupe
  - `emaj_stop_group (`*`groupe [,marque] [,reset.logs] [,groupe.inactif.admis]`*`)`
    désactive les triggers de log => le rollback n'est plus possible
- Un % dans un nom de marque = heure courante (ex : M1_23.59.59.1234)

Examiner les logs

- L'examen des tables de log peut grandement aider le debuging des applications
- Chaque table applicative a sa table de log
  - `emaj_<schéma>.<table>_log`
- Une table de log contient
  - les mêmes colonnes que la table applicative associée
  - et quelques colonnes techniques
- Une ligne mise à jour dans une table applicative génère
  - 1 ligne de log pour un `INSERT` (nouvelle ligne)
  - 1 ligne de log pour un `DELETE` ou `TRUNCATE` (ancienne ligne)
  - 2 lignes de log pour un `UPDATE` (ancienne et nouvelle lignes)
- Un `TRUNCATE` génère aussi une ligne de log

Les colonnes techniques des tables de log

- 6 colonnes techniques en fin de chaque ligne de log
  - `emaj_verb` : type de mise à jour - INS/UPD/DEL/TRU
  - `emaj_tuple` : type de ligne - OLD/NEW
  - `emaj_gid` : numéro de séquence interne
  - `emaj_changed` : heure de la mise à jour - `clock_timestamp()`
  - `emaj_txid` : numéro de la transaction - `txid_current()`
  - `emaj_user` : rôle de connexion du client - `session_user`
- … et on peut en ajouter d'autres
- On peut ainsi identifier les clients, le découpage des transactions et analyser le timing d'exécution du traitement

Les colonnes générées des tables applicatives
(GENERATED ALWAYS AS expression)

Type	Physique (STORED)	Virtuelle
Version minimum Postgres	12	18
Structure en table de log	Une colonne standard	
Contenu en table de log	Résultat de l'expression	NULL
Visualisation en SQL de l'impact des mises à jour	Examen direct de la colonne de la table de log	Utilisation de l'expression
Changement de l'expression (*ALTER TABLE … ALTER COLUMN … SET EXPRESSION …*)	Bloqué par E-Maj (*)	Possible
Suppression de l'expression (*ALTER TABLE … ALTER COLUMN … DROP EXPRESSION*)	Possible mais risque sur l'intégrité des données aux rollbacks E-Maj suivants	Sans objet (interdit par PostgreSQL)
Transformation d'une colonne : générée ⇔ standard (*ALTER TABLE … DROP COLUMN …, ADD COLUMN …*)	Détecté par les contrôles E-Maj lors des tentatives de poses de marques et de rollback (*)	

(*) Il faut temporairement sortir la table de son groupe

Compter les mises à jour enregistrées

- 3 fonctions statistiques, de niveau **groupe de tables**, et pour une tranche de marques donnée
  - `emaj_log_stat_group (`*`groupe`*`, `*`marque_début`*`, `*`marque_fin`*`)` retourne rapidement des estimations du nombre de mises à jour enregistrées, par table
  - `emaj_detailed_log_stat_group (`*`groupe`*`, `*`marque_début`*`, `*`marque_fin`*`)` parcourt les tables de log et retourne des statistiques précises sur leur contenu, par table, par type de requête (INSERT / UPDATE / DELETE / TRUNCATE), et par ROLE à l'origine des mises à jour
  - `emaj_sequence_stat_group (`*`groupe`*`, `*`marque_début`*`, `*`marque_fin`*`)` retourne le nombre d'incréments par séquence

Compter les mises à jour enregistrées (2)

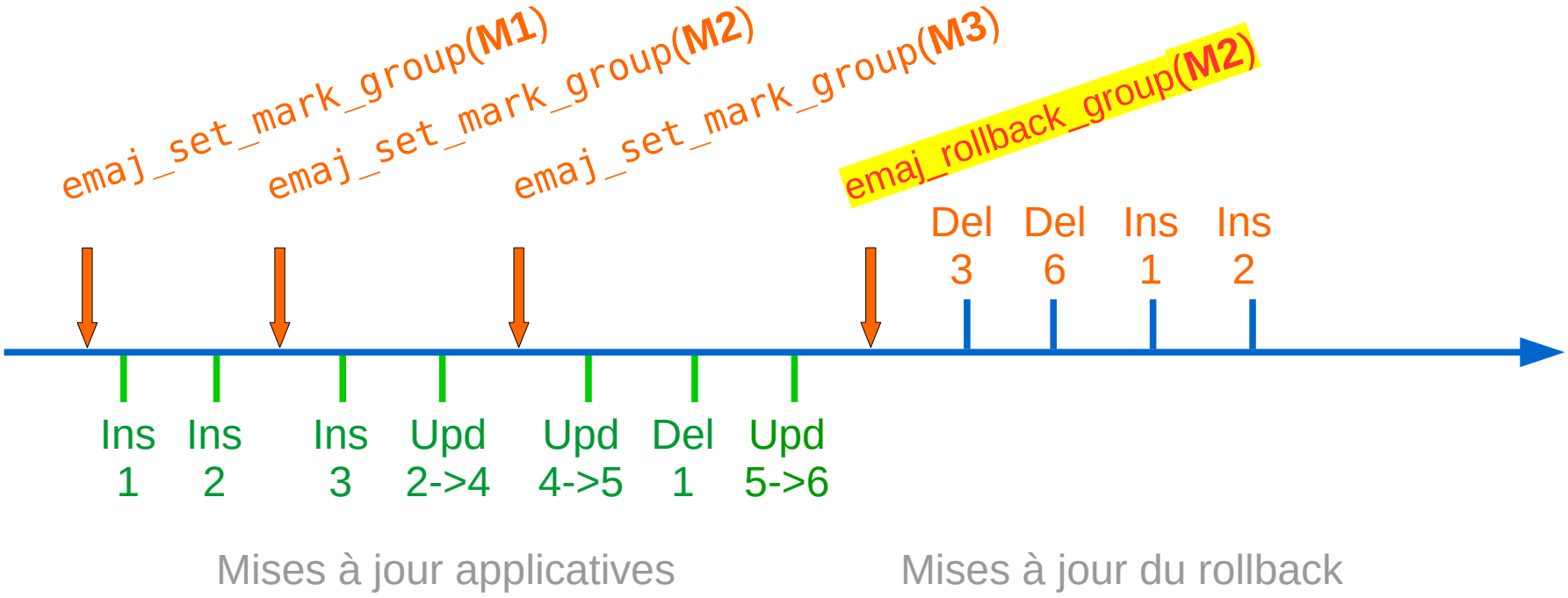
- 2 fonctions retournent rapidement des estimations du nombre de mises à jour enregistrées pour **une table** pour chaque tranche élémentaire de marques sur un intervalle de temps donné
  - `emaj_log_stat_table (schéma, table, date_heure_début, date_heure_fin)`
  - `emaj_log_stat_table (schéma, table, groupe_début, marque_début, groupe_fin, marque_fin)`
- 2 fonctions retournent le nombre d'incréments pour **une séquence** pour chaque tranche élémentaire de marques sur un intervalle de temps donné
  - `emaj_log_stat_sequence (schéma, séquence, date_heure_début, date_heure_fin)`
  - `emaj_log_stat_sequence (schéma, séquence, groupe_début, marque_début, groupe_fin, marque_fin)`

Annuler des mises à jour : le rollback « simple »

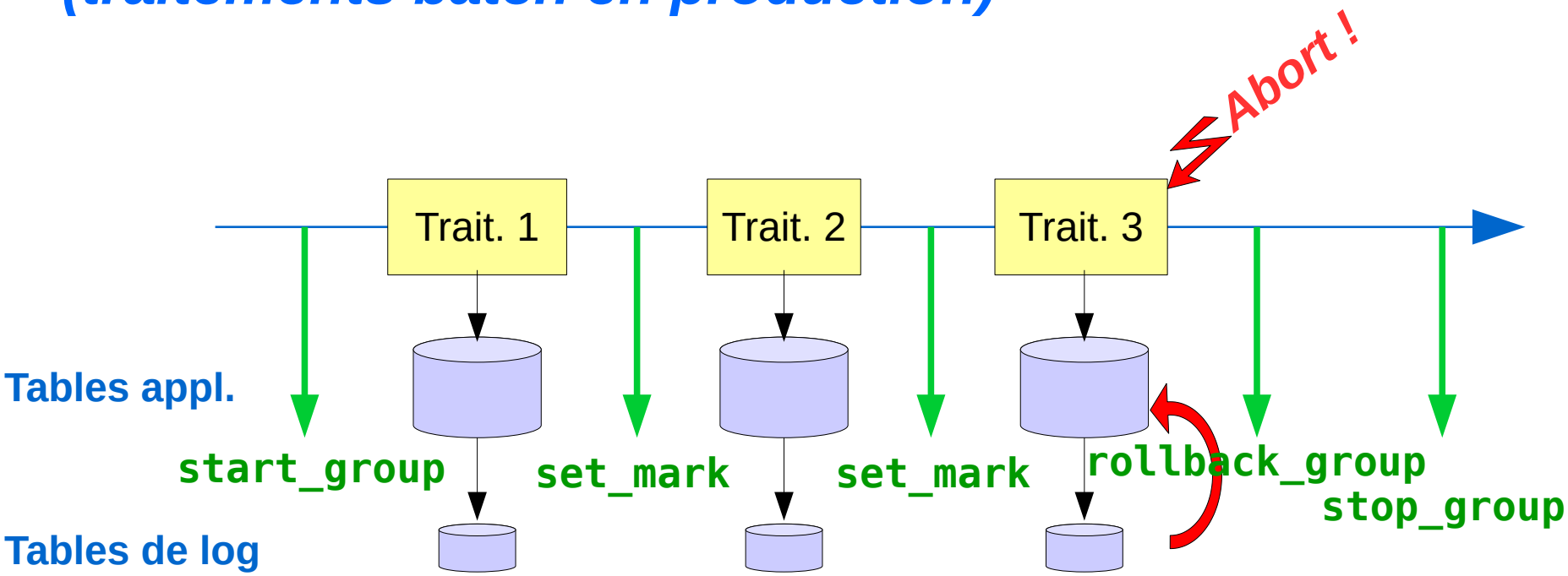
- Une fonction de « rollback » permet de remettre un groupe de tables dans l'état dans lequel il se trouvait à une marque donnée
  - `emaj_rollback_group (groupe, marque [, false [,commentaire]])`
- Fonctionnement
  - Les triggers de log sont désactivés le temps de l'opération
  - Chaque table est remise à l'état correspondant à la marque par un algorithme optimisé
  - Les séquences applicatives sont remises à l'état correspondant à la marque
  - Prend en compte les éventuelles clés étrangères
  - Les logs et les marques annulés sont supprimés
    => tout ce qui est postérieur à la marque de rollback est « oublié »

Un algorithme de rollback optimisé

- Ne traite qu'une seule fois chaque valeur de clé primaire



Un enchaînement E-Maj typique (traitements batch en production)

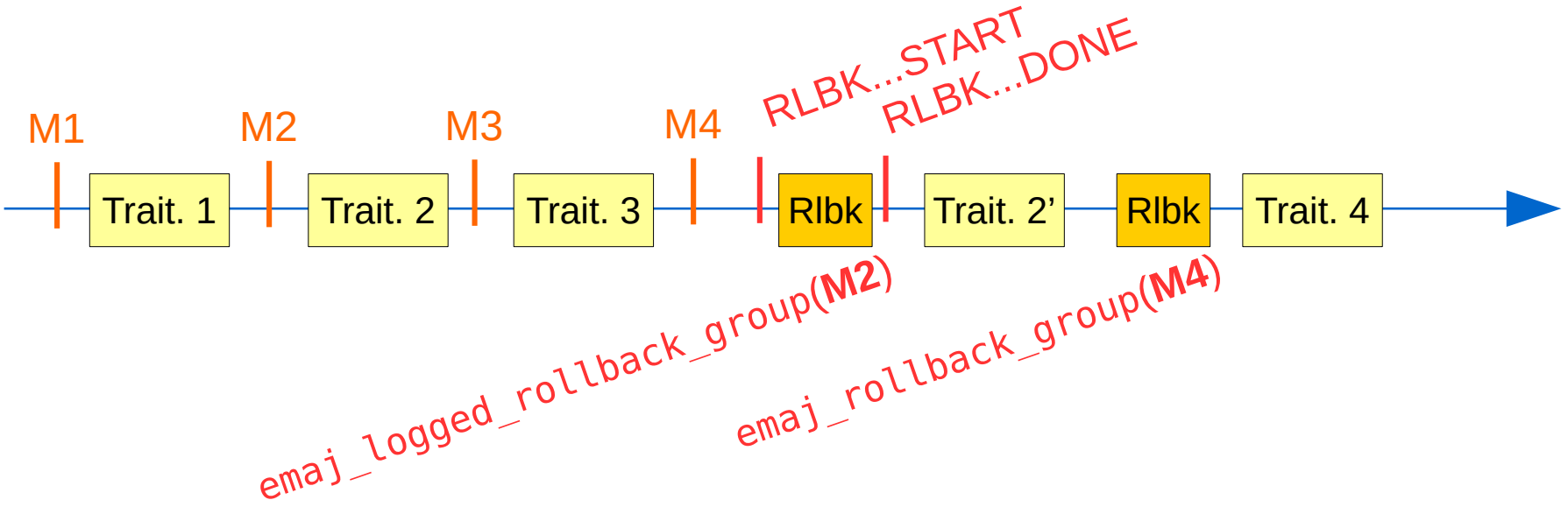


Annuler des mises à jour : le rollback « tracé »

- `emaj_logged_rollback_group (`*`groupe, marque [, false [, commentaire]]`*`)`
- Il se distingue du rollback « simple » par le fait que
  - les triggers de logs ne sont PAS désactivés lors de l'opération
    => les mises à jour générées par le rollback sont enregistrées
  - les logs et marques annulées ne sont PAS supprimés
- On pourra donc annuler un rollback E-Maj ! Et plus généralement faire voyager un groupe de tables dans le temps !
- 2 marques sont automatiquement posées avant et après le rollback
  - `RLBK_<id rollback>_START` et `RLBK_<id rollback>_DONE`
- Pendant le rollback les tables restent accessibles en lecture

Un enchaînement E-Maj typique en environnement de test

- Un enchaînement de 4 traitements à tester
- Après le test 3,une nouvelle version du traitement 2 à retester
- Puis poursuite des tests restants



Estimer la durée d'un rollback E-Maj

- Pour savoir si on a le temps de réaliser l'opération ou si un autre moyen de remise en état ne serait pas plus rapide
- Une fonction estime la durée nécessaire pour rollbacker un groupe à une marque donnée
  - `emaj_estimate_rollback_group (`*`groupe`*`, `*`marque`*`)`

Paralléliser un rollback E-Maj

- Un client perl effectue des rollbacks avec parallélisme
  - `emajParallelRollback.pl -d <database> -h <host> -p <port> -U <user> -W <password> -g <nom_groupe ou listes_groupes> -m <marque> -s <nb_sessions> [-l] [-c commentaire]`
- Répartit automatiquement les tables à traiter dans un nombre donné de sessions
- Toutes les sessions appartiennent à une seule transaction (2PC)

  => max_prepared_transactions >= nb sessions
- Nécessite perl avec son extension PostgreSQL

Suivre les rollbacks E-Maj en exécution

- Une fonction
  - `SELECT * FROM emaj.emaj_rollback_activity ();`
  - restitue
    - les caractéristiques des rollbacks (groupe, marque,...)
    - leur état
    - leur durée écoulée
    - une estimation de la durée restante et du % réalisé
- Nécessite la valorisation du paramètre « `dblink_user_password` »

Suivre les rollbacks E-Maj

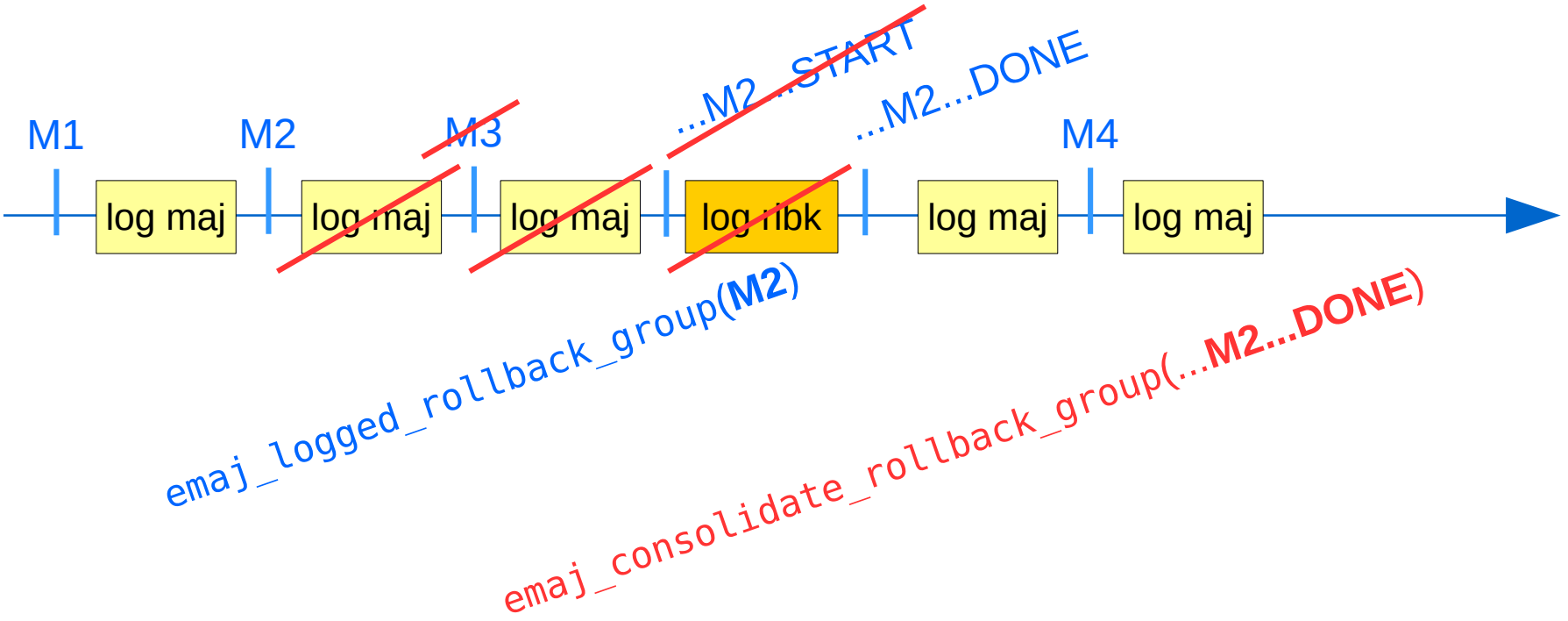
- Un client perl pour suivre les rollbacks en cours / terminés
  - `emajRollbackMonitor.pl -d <database> -h <host> -p <port> -U <user> -W <password> -n <nb_itérations> -i <rafraichissement_en_secondes> -l <nb_rollbacks_terminés> -a <historique_rollbacks_terminés_en_heures>`

```
 E-Maj (version 4.2.0) - Monitoring rollbacks activity
------------------------------------------------------------
21/03/2023 - 08:31:23
** rollback 34 started at 2023-03-21 08:31:16.777887+01 for groups {myGroup1}
   status: COMMITTED ; ended at 2023-03-21 08:31:16.9553+01
** rollback 35 started at 2023-03-21 08:31:17.180421+01 for groups {myGroup1}
   status: COMMITTED ; ended at 2023-03-21 08:31:17.480194+01
-> rollback 36 started at 2023-03-21 08:29:26.003502+01 for groups {group20101}
   status: EXECUTING ; completion 85 %; 00:00:20 remaining
```

Consolider un rollback « tracé »

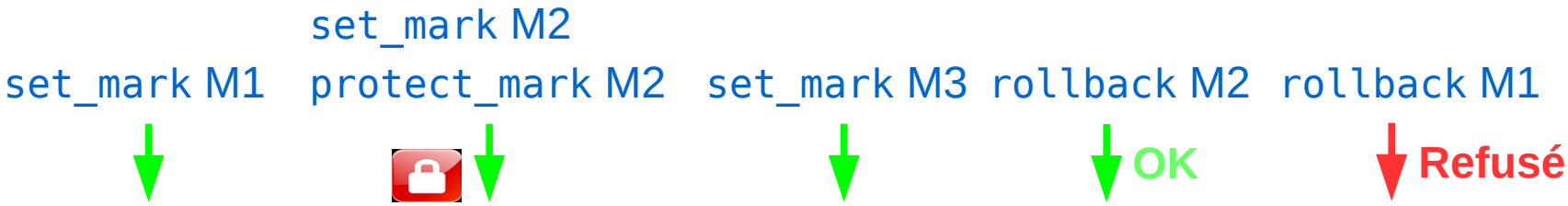
- « Consolider » un rollback, c'est transformer un « rollback tracé » en « rollback simple »
- Les logs et marques intermédiaires sont supprimés, permettant de récupérer de la place dans les logs
  - `emaj_consolidate_rollback_group (`*`groupe`*`, `*`marque_fin_de_rollback`*`)`
- Les tables peuvent être mises à jour pendant la consolidation
- Une fonction restitue la liste des rollbacks consolidables
  - `emaj_get_consolidable_rollbacks ()`

Exemple de consolidation de rollback E-Maj



Se protéger contre des rollbacks E-Maj accidentels

- 2 fonctions pour gérer la protection d'un groupe de tables
  - `emaj_protect_group (`*`groupe`*`)`
  - `emaj_unprotect_group (`*`groupe`*`)`
- 2 fonctions pour gérer la protection d'une marque
  - `emaj_protect_mark_group (`*`groupe`*`, `*`marque`*`)` bloque toute tentative de rollback à une marque antérieure à la marque protégée
  - `emaj_unprotect_mark_group (`*`groupe`*`, `*`marque`*`)`

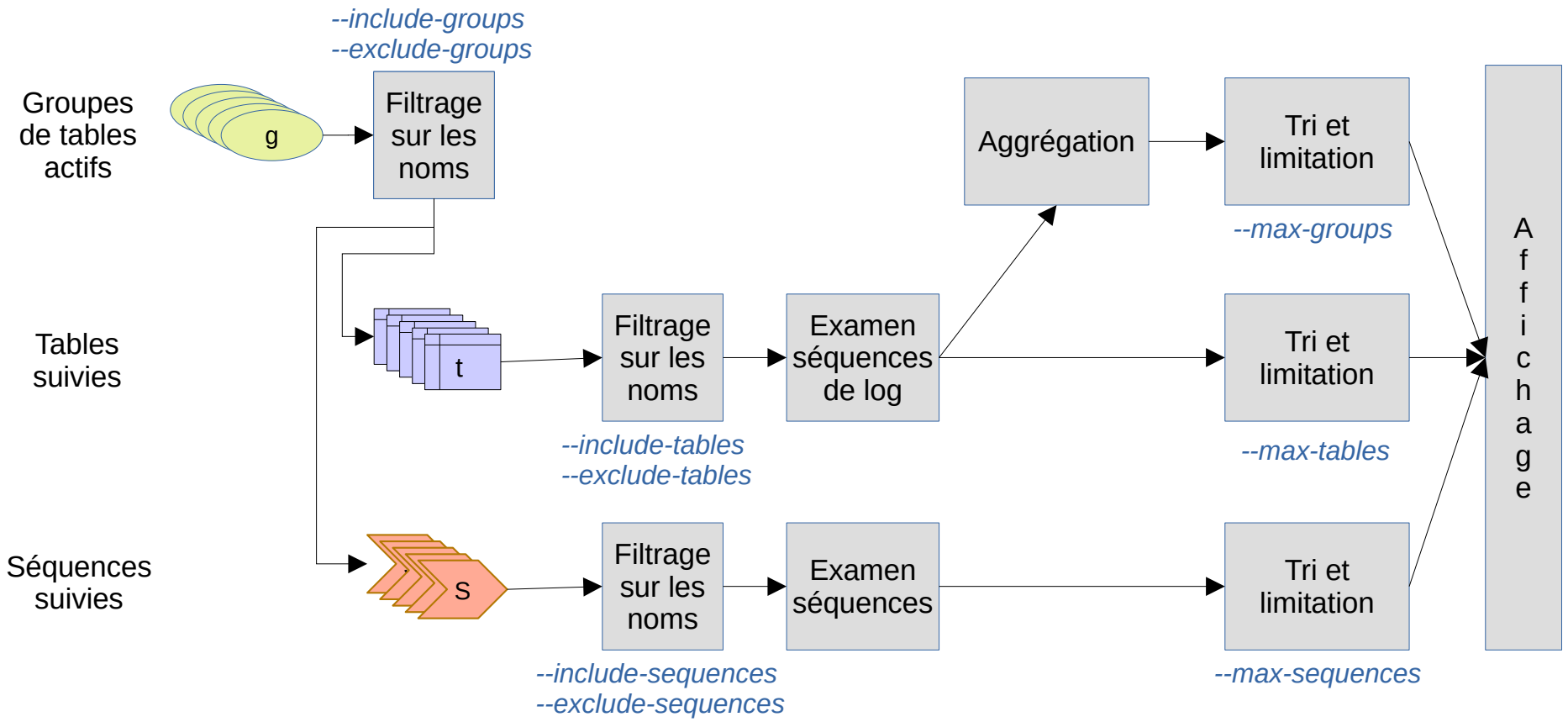


Client emajStat pour suivre l'activité de mise à jour

- Une commande perl pour compter le nombre de mises à jour des tables et séquences, depuis le dernier affichage et depuis la dernière marque du groupe, en valeur absolue et en mises à jour par secondes
- Nombreuses options pour filtrer les groupes, tables et séquences, définir le rafraichissement, …
  - Pour le détail : `emajStat.pl --help`

```
 E-Maj (version 4.5.0) - Monitoring logged changes on database regression (@127.0.0.1:5412)
-------------------------------------------------------------------------------
2024/08/15 08:12:59 - Logging: groups=2/3 tables=11/11 sequences=4/4 - Changes since 1.004 sec: 0 (0.000
c/s)
  Group name + Latest mark                    + Changes since mark + Changes since prev.
    myGroup1 | Multi-1 (2024/08/15 08:12:38) |   359 (17.045 c/s) |      0 (0.000 c/s)
  Table name           + Group    + Changes since mark + Changes since prev.
    myschema1.mytbl1   | myGroup1 |   211 (10.018 c/s) |      0 (0.000 c/s)
    myschema1.myTbl3   | myGroup1 |    60 ( 2.849 c/s) |      0 (0.000 c/s)
    myschema1.mytbl2b  | myGroup1 |    52 ( 2.469 c/s) |      0 (0.000 c/s)
    myschema1.mytbl2   | myGroup1 |    27 ( 1.282 c/s) |      0 (0.000 c/s)
    myschema1.mytbl4   | myGroup1 |     9 ( 0.427 c/s) |      0 (0.000 c/s)
  Sequence name                  + Group    + Changes since mark + Changes since prev.
    myschema1.mytbl2b_col20_seq | myGroup1 |    -5 (-0.237 c/s) |      0 (0.000 c/s)
    myschema1.myTbl3_col31_seq  | myGroup1 |   -20 (-0.950 c/s) |      0 (0.000 c/s)
```

EmajStat : fonctionnement et paramétrage



Analyser les mises à jour enregistrées

- Vider sur fichiers dans un répertoire, par COPY, une partie des tables de log et des séquences d'un groupe
  - `emaj_dump_changes_group (groupe, marque_début, marque_fin, liste_options, liste_tables/seq, répertoire)`
- Générer le SQL d'extraction des mises à jour enregistrées, entre 2 marques, pour tout ou partie des tables et séquences d'un groupe
  - Dans l'espace disque de l'instance :
    `emaj_gen_sql_dump_changes_group (groupe, marque_début, marque_fin, liste_options, liste_tables/seq, fichier)`
  - Dans une table temporaire `emaj_temp_sql`, pour utilisation quelconque par un client :
    `emaj_gen_sql_dump_changes_group (groupe, marque_début, marque_fin, liste_options, liste_tables/seq)`

Analyser les mises à jour : les options

- Communes à emaj_dump_changes_group() et emaj_gen_sql_dump_changes_group()
  - **CONSOLIDATION** = NONE (défaut) | PARTIAL | FULL
  - **EMAJ_COLUMNS** = ALL | MIN | (liste) : sélection des colonnes techniques E-Maj
  - **COLS_ORDER** = LOG_TABLE | PK : ordre des colonnes restituées
  - **ORDER_BY** = PK | TIME : tri des lignes sur clé primaire ou emaj_gid
  - **SEQUENCES_ONLY** : exclusion des tables
  - **TABLES_ONLY** : exclusion des séquences
- Pour emaj_dump_changes_group()
  - **COPY_OPTIONS** = (liste des options) : pour la génération des COPY TO
  - **NO_EMPTY_FILES** : pour supprimer les fichiers vides (les tables sans changements)
- Pour emaj_gen_sql_dump_changes_group()
  - **PSQL_COPY_DIR** = répertoire : pour générer des \copy pour chaque requête
  - **PSQL_COPY_OPTIONS** = (liste options) : options des \copy
  - **SQL_FORMAT** = RAW | PRETTY : formatage de chaque requête SQL sur 1 ou plusieurs lignes

Analyser les mises à jour : la vision consolidée des mises à jour

- La vision consolidée des mises à jour fournit, pour une tranche de temps donnée, et pour chaque clé primaire, un bilan net des changements enregistrés
  - Au maximum : 1 ligne « OLD » (état initial) et 1 ligne « NEW » (état final)
  - Ex : si UPDATE ‘A’→’B’ puis UPDATE ‘B’→’C’ , ligne OLD = ‘A’ et ligne NEW = ‘C’
- Nécessite donc … une clé primaire pour chaque table examinée
- 2 types de consolidation
  - « Consolidation partielle » : sans prise en compte des contenus des colonnes
  - « Consolidation totale » : avec examen des données réellement modifiées
    - Si, pour une PK, toutes les colonnes de « OLD » et « NEW » sont identiques, aucune restitution
    - Ex : aucune ligne restituée pour une PK donnée si UPDATE ‘A’→’B’ puis UPDATE ‘B’→’A’, ou si INSERT puis DELETE
- Séquences
  - 1 ligne « OLD » et 1 ligne « NEW » pour les caractéristiques initiales et finales de la séquence
  - Aucune ligne en « Consolidation totale » si la séquence n’a pas été modifiée

Analyser les mises à jour : structure de la table temporaire emaj_temp_sql

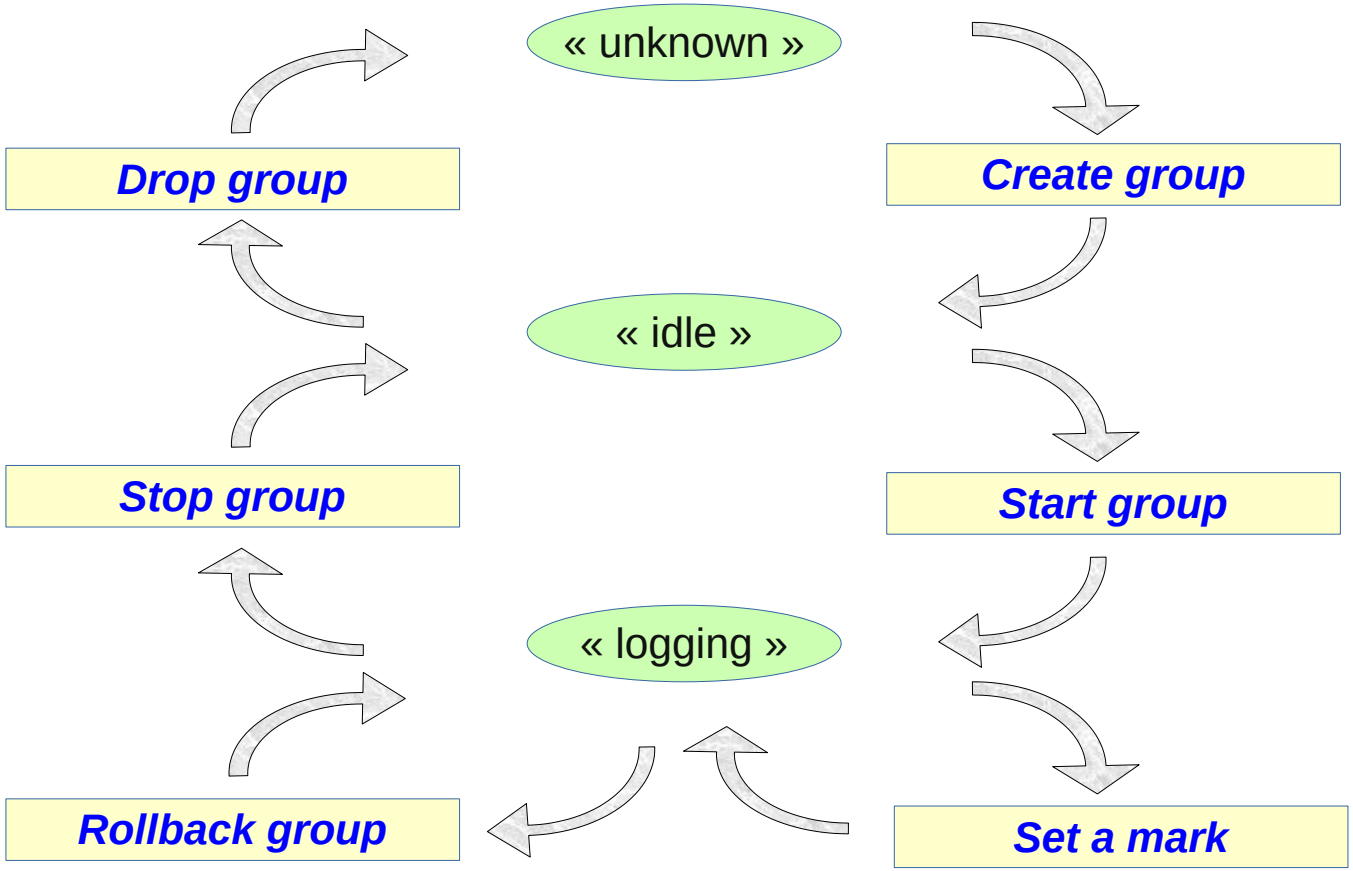
```
CREATE TEMP TABLE emaj_temp_sql (
  sql_stmt_number         INT,          -- N° de requête
                                        --    (0 pour le commentaire initial)
  sql_line_number         INT,          -- N° de ligne pour la requête
                                        --    (0 pour le commentaire initial de la requête)
  sql_rel_kind            TEXT,         -- Type de relation : "table" ou "sequence"
  sql_schema              TEXT,         -- Nom du schéma
  sql_tblseq              TEXT,         -- Nom de la table ou séquence
  sql_first_mark          TEXT,         -- Marque de début (pour la table/séquence)
  sql_last_mark           TEXT,         -- Marque de fin (pour la table/séquence)
  sql_group               TEXT,         -- Rappel du groupe de tables
  sql_nb_changes          BIGINT,       -- Nb estimé de MàJ à traiter pour les tables
  sql_file_name_suffix    TEXT,         -- Suffixe de nom de fichier
  sql_text                TEXT,         -- Texte de la requête SQL (sur 1 ou plusieurs lignes)
  sql_result              BIGINT        -- Colonne destinée à l'appelant pour ses opérations
                                        --    (on peut en ajouter d'autres par ALTER TABLE)
);
```

Un index sur les 2 premières colonnes

Rejouer des mises à jour

- Générer un script sql rejouant les mises à jour élémentaires enregistrées entre 2 marques, pour tout ou partie des tables et séquences d'un groupe
  - Dans l'espace disque de l'instance :
    ```
    emaj_gen_sql_group (groupe, marque_début, marque_fin,
    fichier [,liste_tables/seq])
    ```
  - N'importe où, avec psql :
    ```
    SELECT emaj_gen_sql_group (groupe, marque_début,
    marque_fin, NULL [,liste_tables/seq])
    \copy (SELECT * FROM emaj_sql_script) TO 'fichier'
    ```
- Utile en test pour « répliquer » les mises à jour d'un traitement

Le cycle de vie d'un groupe de tables



Ajustement dynamique des groupes de tables

- Pour ajouter une ou plusieurs tables
  - `emaj_assign_table(schéma, table, groupe, propriétés [, marque])`
  - `emaj_assign_tables(schéma, liste de tables, groupe, propriétés [, marque])`
  - `emaj_assign_tables(schéma, filtre de sélection, filtre d'exclusion, groupe, propriétés [, marque])`
- Propriétés :
  - Format JSON
  - Pour définir la priorité et les tablespaces de data et index de log
- Filtres de sélection et d'exclusion : des RegExp

Ajustement dynamique des groupes de tables (suite)

- Exemple
  - `emaj_assign_tables('monschéma', 'tbl.*','_sav$', 'monGroupe', '{"priority":1}'::json)`
    affecte au groupe 'monGroupe' et avec une priorité 1 toutes les tables du schéma 'monschema' dont le nom commence par 'tbl' et ne se termine pas par '_sav'

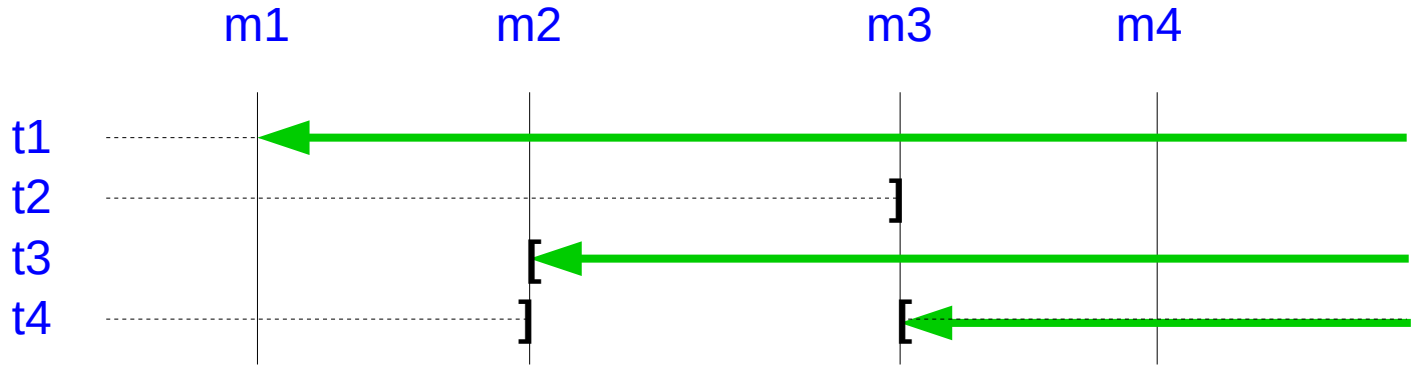
Ajustement dynamique des groupes de tables (suite)

- Sur le même principe :
  - `emaj_assign_sequence()` et `emaj_assign_sequences()`
  - `emaj_modify_table()` et `emaj_modify_tables()`
  - `emaj_move_table()` et `emaj_move_tables()`
  - `emaj_move_sequence()` et `emaj_move_sequences()`
  - `emaj_remove_table()` et `emaj_remove_tables()`
  - `emaj_remove_sequence()` et `emaj_remove_sequences()`
- Pour connaître le groupe assigné pour une table ou une séquence
  - `emaj_get_assigned_group_table()`
  - `emaj_get_assigned_group_sequence()`

Impact des changements de structure de groupes actifs sur les rollbacks

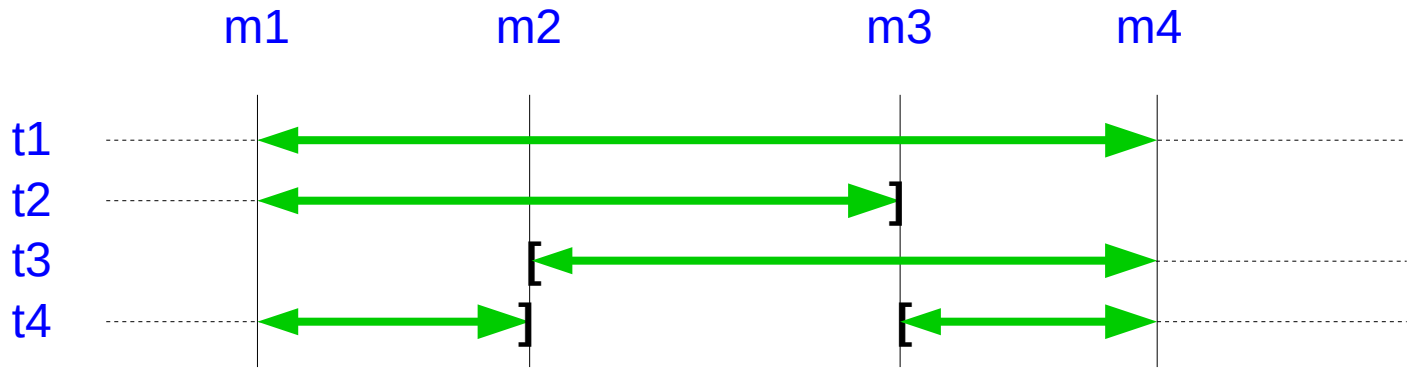
Table t2 sortie à la marque m3, t3 entrée à m2, t4 sortie à m2 et entrée à m3

emaj_rollback_group(<groupe>,'m1', true) traite



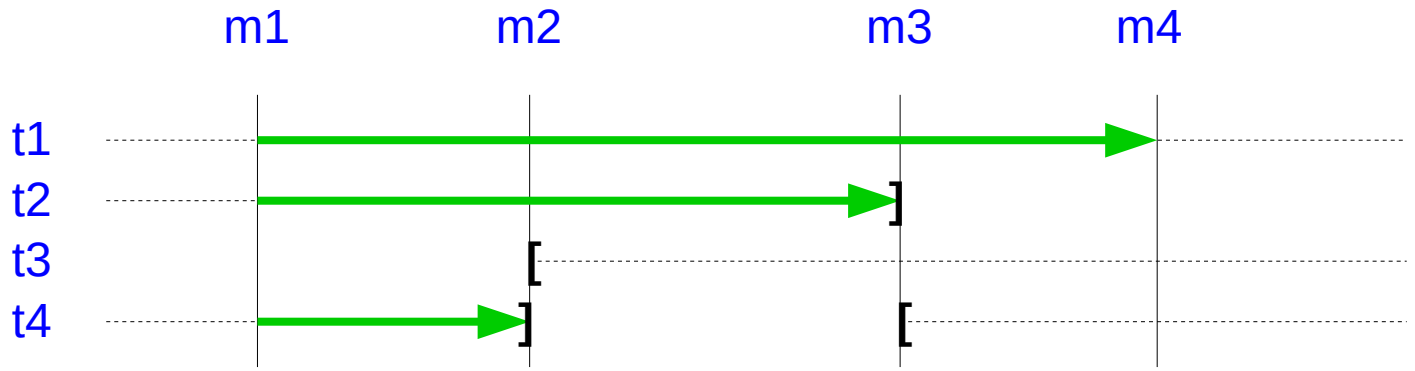
Impact des changements de structure de groupes actifs sur les statistiques et les extractions des mises à jour

emaj_log_stat_group(<groupe>,'m1','m4') et
emaj_dump_changes_group(<groupe>,'m1','m4',...) produiraient :



Impact des changements de structure de groupes actifs sur la génération de script de rejeu

emaj_gen_sql_group(<groupe>,'m1','m4') traiterait :



Modifier la structure d'une table d'un groupe actif

- Pour des actions comme : renommer la table, changer son schéma, ajouter/supprimer/renommer une colonne, changer le type d'une colonne
- La table de log doit aussi changer de structure
- 3 étapes
  - Sortie de la table de son groupe de tables
  - ALTER TABLE
  - Entrée de la table dans son groupe de tables
- Contrainte : un rollback E-Maj à une marque antérieure ne pourra pas remonter au-delà du changement de structure
- Idem pour renommer une séquence ou changer son schéma ou pour les SPLIT/MERGE de partitions

Traiter plusieurs groupes en une seule opération

- Quelques variantes « multi-groupes » de fonctions
  - `emaj_start_groups (`*`tableau_de_groupes`*`, … )`
  - `emaj_stop_groups (`*`tableau_de_groupes`*`, … )`
  - `emaj_set_mark_groups (`*`tableau_de_groupes`*`, … )`
  - `emaj_rollback_groups (`*`tableau_de_groupes`*`, … )`
  - `emaj_logged_rollback_groups (`*`tableau_de_groupes`*`, … )`
  - `emaj_log_stat_groups (`*`tableau_de_groupes`*`, … )`
  - `emaj_gen_sql_groups (`*`tableau_de_groupes`*`, … )`
- Permettent d'avoir des marques communes à plusieurs groupes
- Les 2 syntaxes PostgreSQL pour valoriser un tableau de groupes
  - `ARRAY['`*`groupe 1`*`', '`*`groupe 2`*`', … ]`
  - `'{"`*`groupe 1`*`", "`*`groupe 2`*`", … }'`

Exporter et importer une configuration de groupes de tables

- Format JSON, sur fichier ou en direct
- Lire la configuration
  - `emaj_export_groups_configuration (NULL, tableau_de_groupes)`
- Exporter la configuration dans un fichier
  - `emaj_export_groups_configuration (fichier, tableau_de_groupes)`
- Importer un configuration
  - `emaj_import_groups_configuration (description_JSON, tableau_de_groupes, ...)`
- Importer un configuration depuis un fichier
  - `emaj_import_groups_configuration (fichier, tableau_de_groupes, ...)`

Gérer les marques

- Commenter une marque d'un groupe (ajout/modification/suppression)
  - `emaj_comment_mark_group (groupe, marque, commentaire)`
- Renommer une marque
  - `emaj_rename_mark_group (groupe, ancien_nom, nouveau_nom)`
- Supprimer une marque
  - `emaj_delete_mark_group (groupe, marque)`
  - Si la marque supprimée est la 1ère, les logs antérieurs à la 2ème sont effacés
- Supprimer toutes les marques antérieures à une marque donnée
  - `emaj_delete_before_mark_group (groupe, marque)`
  - Efface les logs antérieurs à la marque (çà peut être long !)

Gérer les marques (2)

- Rechercher des marques
  - `emaj_find_previous_mark_group (`*`groupe, date-heure`*`)` retourne la marque qui précède immédiatement la date et heure donnée
  - `emaj_find_previous_mark_group (`*`groupe, marque`*`)` retourne la marque qui précède immédiatement une marque donnée
- « `EMAJ_LAST_MARK` » représente la dernière marque posée pour un groupe
  - Utilisable pour tous les paramètres qui définissent une marque existante

Autres actions sur les groupes

- Commenter un groupe (ajout/modification/suppression)
  - `emaj_comment_group (groupe, commentaire)`
- Purger les tables de log d'un groupe arrêté (avant son prochain démarrage)
  - `emaj_reset_group (groupe)`
- Forcer l'arrêt d'un groupe (en cas de problème avec la fonction d'arrêt normale)
  - `emaj_force_stop_group (groupe)`
- Vider sur fichiers dans un répertoire, par COPY, toutes les tables et séquences d'un groupe
  - `emaj_snap_group (groupe, répertoire, options_copy)`
- Effacer les historiques d'un groupe de table supprimé
  - `emaj_forget_group (groupe)`

Paramètres E-Maj

- 3 paramètres généraux
  - ‘history_retention’ (def = 1 an) : durée de rétention des traces
  - ‘dblink_user_password’ : pour les rollbacks E-Maj
  - ‘alter_log_table’ : pour ajouter des colonnes aux tables de log
- 6 paramètres pour le modèle d’estimation des durées des rollbacks E-Maj
  - ‘fixed_step_rollback_duration’ (def = 2,5ms)
  - ‘fixed_dblink_rollback_duration’ (def = 4ms)
  - ‘fixed_table_rollback_duration’ (def = 1ms)
  - ‘avg_row_rollback_duration’ (def = 100µs)
  - ‘avg_row_delete_log_duration’ (def = 10µs)
  - ‘avg_fkey_check_duration’ (def = 20µs)

Gérer les paramètres E-Maj

- Voir les paramètres
  - Vue `emaj_all_param` pour les administrateurs
  - Vue `emaj_visible_param` pour les rôles emaj_viewer
- Modifier un paramètre
  - `emaj_set_param (clé, valeur)`
- Exporter ou importer les configurations de paramètres
  - `emaj_export_parameters_configuration ()`
  - `emaj_import_parameters_configuration ()`

Fonctions utiles pour écrire des scripts idempotents

- Pour tester des états
  - À la création/suppression de groupe de tables
    - `emaj_does_exist_group (groupe)`
  - A la pose de marque
    - `emaj_is_logging_group (groupe)`
    - `emaj_does_exist_mark_group (groupe, marque)`
- Pour construire des tableaux de groupes
  - `emaj_get_groups(regexp inclusion, regexp exclusion)`
  - `emaj_get_logging_groups(regexp inclusion, regexp exclusion)`
  - `emaj_get_idle_groups(regexp inclusion, regexp exclusion)`

Actions diverses

- Connaitre la version courante de l'extension ou la supprimer
  - `emaj_get_version ()`
  - `emaj_drop_extension ()`
- Vérifier la bonne santé de l'installation E-Maj
  - `emaj_verify_all ()`
- Récupérer l'identité de la table de log courante d'une table
  - `emaj_get_current_log_table ()`
- Purger manuellement les traces obsolètes
  - `emaj_purge_histories ()`
- Créer/modifier/supprimer un commentaire sur un rollback
  - `emaj_comment_rollback ()`

Log temporaire ou log permanent ?

- **Log temporaire** = Enchaînement du type
  - `emaj_start_group()`
  - répéter
    - traitement
    - `emaj_set_mark()`
  - `emaj_stop_group()`
- Par défaut, les logs et marques sont supprimées automatiquement au redémarrage suivant
- Mais les arrêts et relances posent des verrous très lourds

- **Log permanent** = pas d'arrêt/relance régulier des groupes
  - Il faut régulièrement vider les logs des données obsolètes, avec la fonction `emaj_delete_before_mark()`
- La suppression peut être coûteuse si le volume de log à effacer est important

Pour garantir la fiabilité

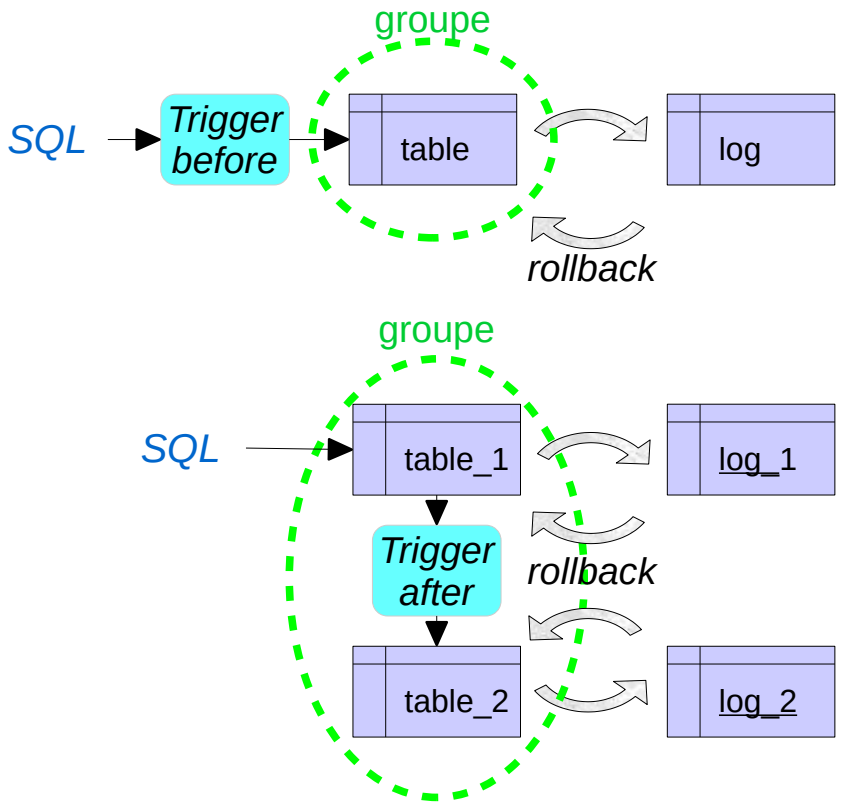
- Aucune modification du moteur PostgreSQL
- Nombreux **contrôles** systématiques, en particulier au démarrage d'un groupe, à la pose d'une marque ou à un rollback :
  - Existence de toutes les tables, séquences, fonctions et triggers ?
  - Cohérence des colonnes entre les tables applicatives et les tables de log (existence, type) ?
- **Verrous** forts sur les tables lors des `start_group`, `set_mark_group` et `rollback_group`, pour être sûr qu'aucune transaction n'est en train de mettre à jour les tables applicatives
  - On peut influencer l'ordre de pose des verrous en définissant un niveau de priorité pour chaque table dans la table `emaj_group_def`
- Rollback de toutes les tables et séquences dans une seule **transaction**

Pour garantir la fiabilité (suite)

- Utilisation de **séquences** internes au lieu de timestamp pour être insensible aux ajustements horaires du serveur
- Des « **event triggers** » bloquent la suppression intempestive ou certaines modifications de composants (tables, séquences, fonctions...)
  - 2 fonctions pour désactiver/ré-activer le blocage
  - `emaj_disable_protection_by_event_triggers ()`
  - `emaj_enable_protection_by_event_triggers ()`

Impact des triggers applicatifs sur les rollbacks E-Maj

- Trigger de type *BEFORE* sur une table appartenant à un groupe de tables
  - Les valeurs réellement écrites en base sont enregistrées dans les logs
  - => à désactiver au rollback E-Maj
- Trigger de type *AFTER* écrivant dans une autre table du même groupe de tables
  - Le rollback remettra les 2 tables dans le bon état
  - => à désactiver au rollback E-Maj



- Autres cas : Étudier les impacts

Gérer les triggers applicatifs

- Par défaut, les triggers applicatifs sont désactivés automatiquement lors des rollback E-Maj
- Un trigger peut être laissé dans son état lors du rollback s'il est enregistré comme tel
- 2 propriétés utilisables par les fonctions `emaj_assign_table()`, `emaj_assign_tables()`, `emaj_modify_table()` et `emaj_modify_tables()` pour spécifier les triggers à ne pas désactiver automatiquement
  - `"ignored_triggers": ["trg1","trg2",…]` liste les noms de triggers
  - `"ignored_triggers_profiles": ["regexp1","regexp2",…]` liste des expressions rationnelles de sélection des triggers

Pour concourir à la sécurité

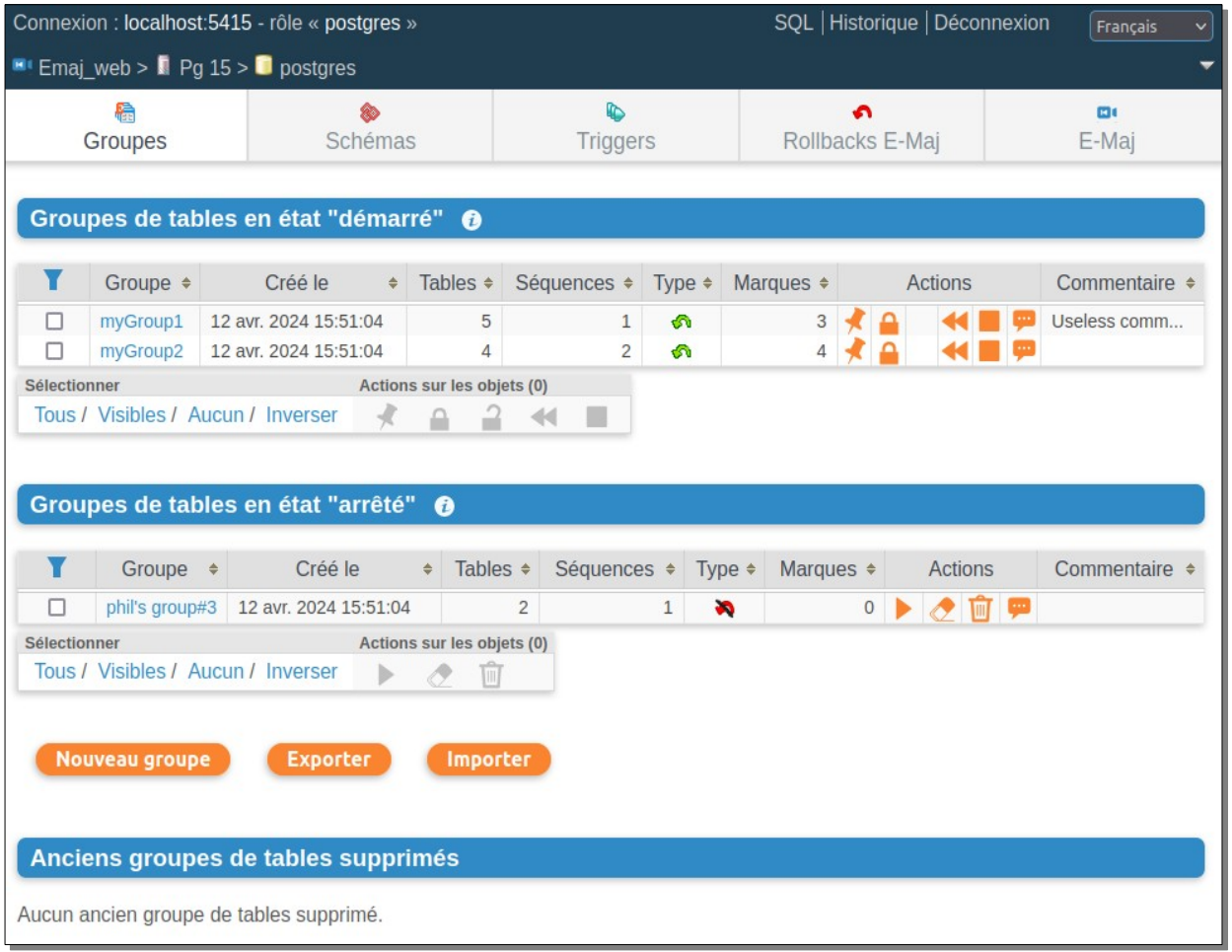
- 2 rôles `NOLOGIN` dont les droits peuvent être donnés :
  - `emaj_adm` pour l'administration E-Maj
  - `emaj_viewer` pour la simple consultation des objets E-Maj (logs, marques, statistiques)
- Les objets E-Maj ne sont créés et manipulés que par un super-utilisateur ou un membre de `emaj_adm`
- Aucun autre droit n'est donné sur les schémas, tables et fonctions d'E-Maj
- Les triggers de log sont créés en « `SECURITY DEFINER` »
- Pas besoin de donner des droits supplémentaires sur les tables ou séquences applicatives

Performances

- Surcoût du log
  - Dépend largement du matériel et de la part des mises à jour SQL dans les traitements
  - Typiquement quelques % sur les durées de traitement
  - Mais beaucoup plus sur des chargements purs de données
- Durée de rollback
  - Dépend évidemment du nombre de mises à jour à annuler
  - Dépend aussi largement
    - de la configuration du serveur,
    - de la structure des tables (taille des lignes, index, clés étrangères et autres contraintes…)
  - Mais sauf cas particulier, toujours moins long qu'une restauration logique

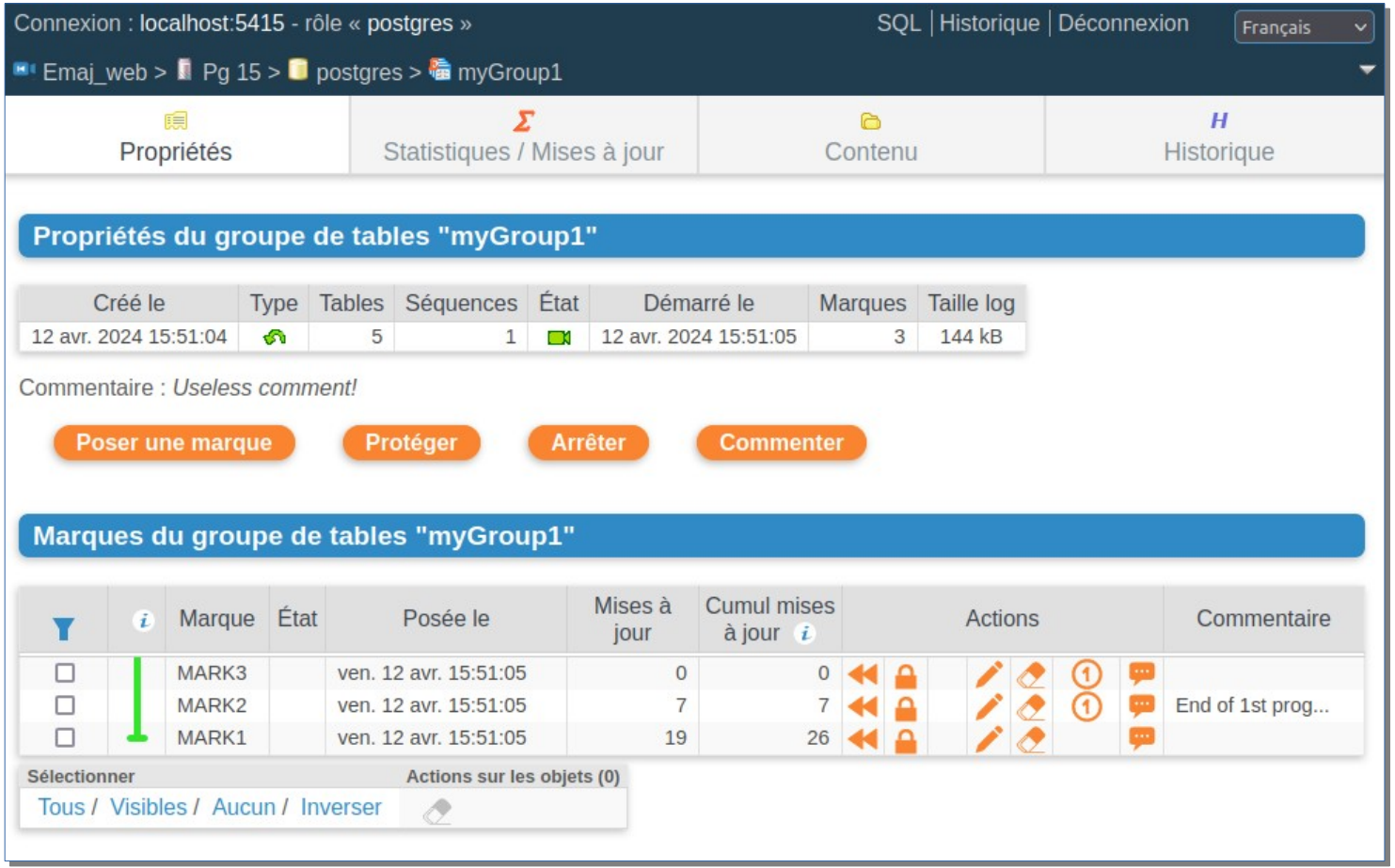
Emaj_web

- Pour les administrateurs et utilisateurs
- Tous les objets E-Maj (groupes, marques...) et leurs attributs
- (presque) toutes les actions possibles sur les objets E-Maj



Liste des groupes de tables

Emaj_web : détail d'un groupe de tables



Limitations actuelles

- Version PostgreSQL minimum = **14**
- Les tables applicatives appartenant à un groupe « rollbackable » doivent avoir une **`PRIMARY KEY`**
- Les requêtes de **DDL** ne peuvent pas être tracées ou annulées par E-Maj
  - Changer la structure d'une table nécessite une sortie temporaire du groupe de tables
- Les **`FOREIGN KEY`**s définies sur les **`tables partitionnées`** sont incompatibles avec les rollbacks E-Maj
  - => les définir sur chaque partition

Pour conclure...

- Beaucoup plus d'**informations** dans
  - la documentation : https://emaj.readthedocs.io/fr/latest/index.html
  - les fichiers `README` et `CHANGES`
- Grand **merci** à tous les contributeurs et utilisateurs fidèles
- N'hésitez pas à faire des **retours** sur github ou par email (phb.emaj@free.fr)