Guide de transition à PostgreSQL : aide à la décision

DINSIC, Socle Interministériel du Logiciel Libre, PGGTIE

PostgreSQL.fr

10/12/2019

Table des matières

1 Versions **5**

2 Contributeurs **6**

3 Introduction **7**

4 Satisfaire les besoins des maîtres d'ouvrage **8**
4.1 Les types d'usage . . . 8
4.2 Accès aux données . . . 8
4.3 Les besoins de sécurité . . . 8
4.4 Scénarios de migration des SI vers PostgreSQL . . . 9

5 Intégration de PostgreSQL sur les plateformes techniques **10**
5.1 Plateformes techniques . . . 10
5.1.1 Architecture processeur . . . 10
5.1.2 Système d'exploitation . . . 10
5.1.3 Compatibilité avec la virtualisation/conteneurisation . . . 10
5.1.4 Compatibilité avec les technologies de stockage . . . 10
5.1.5 Compatibilité avec les technologies de sauvegarde . . . 11
5.2 Sécurité . . . 11
5.2.1 Gestion de l'identification des utilisateurs . . . 11
5.2.2 Garantir la confidentialité et utilisation du chiffrement . . . 11
5.2.3 Assurer la traçabilité et la journalisation . . . 11
5.2.4 Audit . . . 11

6 Scalabilité et résilience **13**
6.1 Mécanismes de clustering et de réplication . . . 13
6.2 Résilience portée par le moteur PostgreSQL . . . 14
6.3 Résilience portée par les infrastructures de stockage . . . 14
6.4 Plans de continuité et de reprise informatique . . . 14

7 Développement sous PostgreSQL **16**
7.1 Typologie des données . . . 16
7.2 Schémas et structures des bases . . . 16
7.3 Spécificités de développement . . . 17
7.4 API et modes d'accès . . . 17
7.4.1 Interfaces client . . . 17
7.4.2 Couche d'abstraction . . . 18

7.4.3 Pool de connexions . . . 18
7.5 Procédures stockées, fonctions et déclencheurs (triggers) . . . 18
7.6 Foreign Data Wrapper . . . 19

8 Administration **20**
8.1 Outils d'administration et d'exploitation de PostgreSQL . . . 20
8.1.1 PgAdmin . . . 20
8.1.2 psql . . . 20
8.2 Supervision de PostgreSQL et d'analyse des logs . . . 20
8.3 Sauvegarde . . . 21
8.4 Haute-disponibilité . . . 21
8.5 Outils de migration des données vers PostgreSQL . . . 21

9 Aide à la décision **23**

10 Limitations ou fonctionnalités non supportées **24**

11 Conduite du changement **26**
11.1 Formation des acteurs . . . 26
11.2 Support . . . 26
11.3 Plan de migration . . . 26

12 Retour sur investissement **27**
12.1 Coût de migration de la base . . . 27
12.2 Coût de possession . . . 27
12.3 Maîtrise des trajectoires . . . 27

13 Annexes **28**
13.1 Migration depuis Oracle . . . 29
13.1.1 Environnement technique . . . 29
13.1.2 Les piles logicielles . . . 29
13.1.3 Espace de stockage . . . 29
13.1.4 Migration de la base de données . . . 30
13.1.5 Criticité et sauvegardes . . . 31
13.1.6 Suivi, amélioration des performances ,et supervision . . . 31
13.2 Migration depuis Db2 . . . 33
13.2.1 Quelques spécificités du DDL pg/Db2 . . . 33
13.2.2 DCL . . . 36
13.2.3 Autres considérations . . . 37

13.3 Migration depuis Informix 39
13.3.1 Structure 39
13.3.2 Plan à suivre lors de la migration des bases de données 39
13.3.3 Intégration du framework Hibernate 40
13.3.4 Risques 40
13.4 Migration depuis MS SQL 41
13.5 Quelques références 42
13.6 Extensions et plugins pour PostgreSQL 43
13.7 Outils tiers pour PostgreSQL 44

14 Documents de référence **45**

1 Versions

Historique des versions du document:

Version	Date	Commentaire
0.1	18/12/2013	Document de travail
0.2	07/02/2014	Première séance de travail
0.3	19/03/2014	Deuxième séance de travail
0.4	01/07/2014	Préparation pour diffusion
0.5	04/08/2014	Consolidation des contributions
1.0	19/03/2015	Validation et derniers correctifs
1.0.1	26/06/2015	Précision sur la licence
1.0.1	30/09/2015	Traduction en russe par I. Panchenko et O. Bartunov
2.0	10/12/2019	Mise à jour (PGGTIE)

2 Contributeurs

— Bruce BARDOU - DGFiP
— Barek BOUTGAYOUT - MASS
— Amina CHITOUR - MENMESR
— Alain DELIGNY - MEDDE
— Yohann MARTIN - Maif
— Alain MERLE - MEDDE
— Anthony NOWOCIEN – Société Générale
— Loïc PESSONNIER - MINDEF
— Marie-Claude QUIDOZ - CNRS
— Laurent RAYMONDEAU - MENESR

3 Introduction

Ce guide a été réalisé dans le cadre de la mise en œuvre de la circulaire «Ayrault» du 19 septembre 2012 relative à l'usage du logiciel libre dans l'administration, qui a induit la constitution d'un Socle Interministériel du Logiciel Libre (SILL). Comparativement aux solutions commerciales, ce guide présente la mise en œuvre de PostgreSQL. Il a pour objectif de répondre aux interrogations des maîtres d'ouvrage et maîtres d'œuvre pour la mise en œuvre de PostgreSQL en remplacement d'une solution commerciale. Ce guide a pour objectif, sans entrer dans le détail de l'implémentation technique, de démontrer l'intérêt de PostgreSQL en décrivant les mécanismes d'intégration, de sécurité et de robustesse.

Ce document collaboratif est désormais écrit par Mimprod [1] et le PGGTIE [2].

1. Mimprod: https://pcll.ac-dijon.fr/mim/mimprod-outils-de-production/
2. PGGTIE: https://www.postgresql.fr/entreprises/accueil

4 Satisfaire les besoins des maîtres d'ouvrage

4.1 Les types d'usage

PostgreSQL est un système de gestion de bases de données au standard SQL utilisé classiquement dans les applications transactionnelles pour assurer la persistance des données comme tout produit commercial du même type (Oracle, Db2, Informix, MS SQL, Sybase,…).

PostgreSQL propose une extension pour la géomatique (PostGIS [3]) conforme aux standards de l'Open Geospatial Consortium (OGC). Sa prise en charge du format JSON, du clé-valeur lui permet de répondre à de nombreux cas d'utilisation de solutions de type NoSQL. PostgreSQL est également utilisable dans le domaine de l'informatique décisionnelle comme entrepôt de données en liaison avec les outils de reporting (BusinessObjects, Pentaho,…). Il possède également des fonctionnalités de recherche plein texte [4].

De nombreux progiciels libres s'appuient nativement sur PostgreSQL (Gestion Électronique de Documents (GED), Moteurs de règles, GroupWare, Supervision,…). Pour les logiciels ne supportant pas PostgreSQL, le PGGTIE a écrit une lettre ouverte [5] à ce sujet aux éditeurs.

PostgreSQL supporte également les traitements d'arrière plan comme les traitements par lots ou différés (batch,…).

4.2 Accès aux données

PostgreSQL est conforme [6] à la norme SQL 2011, le langage de requête commun à de nombreux SGBDs. La migration est ainsi facilitée entre SGBDs respectant les standards.

4.3 Les besoins de sécurité

PostgreSQL répond aux besoins de sécurité exprimé en terme de disponibilité, d'intégrité, de confidentialité et de traçabilité (DICT [7]).

Le chiffrement des flux est en standard et plusieurs méthodes d'authentification fortes (comme SCRAM, Kerberos) sont possibles. Les besoins de cryptographie sont couverts par des modules complémentaires, en particulier *pg_crypto* [8]. A l'heure actuelle, il n'est pas possible de chiffrer l'ensemble d'une instance, même si des travaux sont en cours (cf wiki sur Transparent Data Encryption).

3. PostGIS: https://postgis.net/
4. Recherche full-text: https://www.postgresql.org/docs/current/static/textsearch.html
5. Lettre ouverte: https://www.postgresql.fr/entreprises/20171206_lettre_ouverte_aux_editeurs_de_logiciels
6. Conformité SQL: https://www.postgresql.org/docs/current/static/features.html
7. DICT : https://fr.wikipedia.org/wiki/Sécurité_des_systèmes_d%27information
8. pgcrypto: https://www.postgresql.org/docs/current/static/pgcrypto.html

PostgreSQL dispose d'une communauté très réactive qui fournit les patchs correctifs de sécurité et assure la gestion de l'obsolescence des composants. Il est recommandé d'appliquer les patchs mineurs le plus rapidement possible. Une version majeure est publiée chaque année et supportée [9] pendant 5 ans.

PostgreSQL garantit l'intégrité des données manipulées même en cas d'incident par son respect des propriétés d'atomicité, de cohérence, d'isolation et de durabilité (ACID [10]).

PostgreSQL offre nativement les mécanismes permettant de répondre aux besoins de confidentialité, de gestion des droits par le biais des rôles [11]. Il est possible d'obtenir une granularité très fine, y compris au niveau des lignes renvoyées (voir les politiques de sécurité pour l'accès aux lignes [12]).

4.4 Scénarios de migration des SI vers PostgreSQL

La migration d'un SI représente un coût non négligeable constitué principalement :

— de la migration des données, même s'il existe de nombreux outils permettant de réaliser cette migration en fonction des SGBD source ;
— des corrections à faire dans le code source de l'application. L'importance des adaptations est liée à l'usage des fonctionnalités spécifiques au produit existant (procédures stockées, triggers, fonctions non standards) et l'utilisation d'une couche d'abstraction (Hibernate) ;
— de la recette permettant de garantir à iso-fonctionnalités, qu'aucune erreur n'a été induite lors de la migration (tests de non régression).

De ce fait, les migrations vers PostgreSQL s'effectuent généralement lors d'une évolution fonctionnelle de l'application. Le coût de migration est intégré au projet dans son ensemble notamment en terme de recette technique et fonctionnelle.

Cependant un plan ambitieux de migration globale des SI peut être programmé. PostgreSQL peut ainsi devenir le SGBD à privilégier pour toute nouvelle application. C'est par exemple le cas pour de nombreuses entreprises du PGGTIE, chez qui l'utilisation d'un autre SGBD relationnel doit être justifié (demande de dérogation).

9. Support et EOL: https://www.postgresql.org/support/versioning/
10. Propriétés ACID : https://fr.wikipedia.org/wiki/Propri%C3%A9t%C3%A9s_ACID
11. Rôles : https://www.postgresql.org/docs/current/static/user-manag.html
12. https://www.postgresql.org/docs/current/static/ddl-rowsecurity.html

5 Intégration de PostgreSQL sur les plateformes techniques

5.1 Plateformes techniques

5.1.1 Architecture processeur

PostgreSQL fonctionne sur les architectures processeur suivantes [13]: x86, x86_64, IA64, PowerPC, PowerPC 64, S/390, S/390x, Sparc, Sparc 64, ARM, MIPS, MIPSEL et PA-RISC.

5.1.2 Système d'exploitation

PostgreSQL fonctionne sur la plupart des systèmes d'exploitation. Il existe des versions binaires pour la famille Red Hat (incluant CentOS/Fedora/Scientific), la famille Debian GNU/Linux et ses dérivées, la famille Ubuntu Linux et ses dérivées, SuSE, OpenSuSE, Solaris, Windows, MacOSX, FreeBSD, OpenBSD…

Pour les distributions utilisant des fichiers .rpm, il est possible d'utiliser le dépôt communautaire https://yum.postgresql.org/. Pour celles utilisant les .deb, on pourra se référer à https://apt.postgresql.org. Les codes sources sont disponibles en ligne [14].

5.1.3 Compatibilité avec la virtualisation/conteneurisation

PostgreSQL est compatible avec notamment VMWare et KVM. La virtualisation apporte des avantages en matière de résilience. Il est également possible de faire fonctionner PostgreSQL avec des conteneurs comme Docker.

5.1.4 Compatibilité avec les technologies de stockage

PostgreSQL fonctionne sur les baies de stockage (SAN, NAS,…) mais comme pour tout SGBD, l'attachement doit être permanent en mode blocs. La virtualisation du stockage est totalement transparente pour le SGBD.

Les communautés fournissent des recommandations sur les types de système de fichiers à utiliser (ext4, …). On déconseillera l'utilisation de NFS.

13. Plateformes supportées: https://docs.postgresql.fr/current/supported-platforms.html
14. Code source: https://www.postgresql.org/ftp/source/

5.1.5 Compatibilité avec les technologies de sauvegarde

PostgreSQL supporte les modes de sauvegarde habituels :

— sauvegarde à froid : sauvegarde de niveau système de fichier – la base doit être arrêtée ;
— sauvegarde à chaud : sauvegarde de niveau système de fichiers base démarrée. Pas de possibilité de réaliser une sauvegarde incrémentale de manière native. Les outils barman et pgbackrest le permettent toutefois ;
— sauvegarde continue: sauvegarde de niveau système de fichiers, base démarrée et permettant le *Point In Time Recovery* (PITR). Il est possible d'utiliser les outils précédents, tout comme *pitrery* ;
— export SQL : la base peut être exportée sous un format SQL ou compressée propre à PostgreSQL.

PostgreSQL est compatible avec les outils de sauvegarde (TINA, TSM, Veeam, Netbackup …).

5.2 Sécurité

5.2.1 Gestion de l'identification des utilisateurs

PostgreSQL fournit les mécanismes d'authentification et gère l'attribution des privilèges (GRANT,…). Il permet également la séparation des schémas de base de données.

5.2.2 Garantir la confidentialité et utilisation du chiffrement

Le chiffrement est possible via le module complémentaire *pgcrypto* permettant le chiffrement de colonnes par clé publique et clé privée.

5.2.3 Assurer la traçabilité et la journalisation

La traçabilité des actions dans PostgreSQL est assurée par des journaux de transactions (appelés *wals*, Write-Ahead Logs):

— journalisation du fonctionnement du moteur (démarrage, arrêt,…) ;
— journalisation d'accès à PostgreSQL (requêtes, accès utilisateurs, erreurs,…).

5.2.4 Audit

Si un nombre conséquent d'informations peuvent être enregistrées dans les logs [15], il peut être nécessaire dans certains cas de mettre en place une politique d'audit plus poussée. L'outil le plus abouti pour

15. Configuration des logs: https://www.postgresql.org/docs/current/static/runtime-config-logging.html

se faire est *PGAudit* [16]. Il permettra en particulier de comprendre le contexte dans lequel une opération a été effectuée. Un exemple d'implémentation par la DISA (Defense Information Systems Agency) est disponible [17].

16. PGAudit: https://www.pgaudit.org/
17. STIG: https://www.crunchydata.com/postgres-stig/PGSQL-STIG-9.5+.pdf

6 Scalabilité et résilience

6.1 Mécanismes de clustering et de réplication

Définitions

Scalabilité et Elasticité

> La scalabilité désigne la capacité de PostgreSQL à s'adapter à un changement d'ordre de grandeur de la demande (montée en charge ou diminution).

> L'élasticité d'un système est la faculté d'un système à étendre ou réduire automatiquement ses ressources en fonction des besoins.

Résilience et Robustesse (ou stabilité)

> La résilience est la capacité d'un système ou d'une architecture réseau à continuer de fonctionner en cas de panne.

> La robustesse est la qualité d'un système qui ne plante pas, qui fonctionne bien même dans un environnement hostile (pénétration, DoS…) ou anormal (entrées incorrectes…)

Cluster

> Un cluster est une grappe de serveurs (ou « ferme de calcul ») partageant un stockage commun. Un cluster fournit des fonctions de haute disponibilité et de répartition de charge.

Réplication

> La réplication est un processus de partage d'informations pour assurer la cohérence de données entre plusieurs sources de données redondantes, pour améliorer la fiabilité, la tolérance aux pannes, ou la disponibilité.

Comme pour la grande majorité des SGBD relationnels, il est aisé d'augmenter les ressources du serveur (CPU, RAM, disque) et de réaliser une scalabilité verticale pour le service Postgres. Certaines solutions permettent désormais de réaliser du sharding (répartir les données sur plusieurs instances, comme *Citus*, *PostgreSQL-XL*, …).

Plusieurs modes de clustering et de réplication sont supportés par PostgreSQL avec les avantages et les inconvénients propres à chacun. Ces considérations sont indépendantes du choix du SGBD ; tous les produits sont concernés :

— mode « Actif-Passif » : une base principale est en lecture/écriture, l'autre base, dite « miroir » est synchronisée en arrière plan. La réplication peut être asynchrone (meilleures performances) ou synchrone (meilleure sécurité).

— mode « Actif-Actif partiel » ou «Read/Write - Read »: une base principale est en lecture/écriture, les bases « miroir » sont accessibles en lecture seule.

Tout comme d'autres SGBD, PostgreSQL ne permet pas d'assurer la continuité des transactions sur plusieurs serveurs en mode « Actif-Actif » ou « Read/Write – Read/Write ». Il faut passer par des contributions tierses, comme le projet BDR [18] (Bidirectionnal Replication) de 2ndQuadrant propose cette fonctionnalité. Il s'agit toutefois d'un développement propriétaire sur une version modifiée de PostgreSQL.

6.2 Résilience portée par le moteur PostgreSQL

PostgreSQL fournit les mécanismes de réplication pour la mise en place d'un cluster « actif-passif » ou « actif-actif partiel ».

Il existe deux grands types de réplication, à savoir la réplication physique (par exemple la *streaming replication* [19] basée sur la modification des blocs de données) et la réplication logique (basée sur la modification des objets de la base). Pour des besoins de haute-disponibilité, on se concentrera sur la réplication physique. La réplication logique se destine à d'autres cas d'usages, comme par exemple l'alimentation sélective d'une autre base (ex: datawarehouse), un upgrade avec pour objectif de minimiser l'indisponibilité, …

6.3 Résilience portée par les infrastructures de stockage

PostgreSQL est compatible avec le clustering, porté par la plateforme indépendamment du SGBD. L'écriture sur un nœud est réalisée par PostgreSQL et la réplication sur un second nœud est assurée par la baie de stockage.

Pour relancer le moteur sur le second nœud il y a obligatoirement une interruption de service.

6.4 Plans de continuité et de reprise informatique

Les projets de plans de continuité et de reprise informatique doivent être portés par l'application. Des recommandations de choix de scénarios sont faites suivant les besoins PRI/PCI indépendamment du SGDB.

18. BDR: http://bdr-project.org/docs/stable/index.html
19. https://www.postgresql.org/docs/current/static/warm-standby.html#STREAMING-REPLICATION

PostgreSQL fournit les outils permettant une intégration dans les PRI/PCI.

7 Développement sous PostgreSQL

7.1 Typologie des données

PostgreSQL offre les types de données standards (alphanumérique, date, data, index, blob,…) ainsi que des types de données complexes (géospatiales, objets, hash, XML,…). On se référera à la documentation [20].

Il est recommandé d'utiliser la norme ISO 8601 [21] pour les dates (YYYY-MM-DD). Pour les types comprenant la date et l'heure, on utilisera le type *timestamp with time zone*.

Le type de donnée *Binary Large OBject* (BLOB) permet le stockage en base de données de contenus divers sous forme binaire (fichiers bureautiques, pdf, photos, audios, vidéos, multimédias,…).

PostgreSQL fournit deux modes de stockage des données binaires :

— directement dans la table en utilisant le type *bytea* ;
— dans une table séparée avec un format spécial qui stocke les données binaires et s'y réfère par une valeur de type *oid* dans la table principale en utilisant la fonction Large Object.

Le type de données *bytea* qui peut contenir jusqu'à 1 Go, n'est pas adapté au stockage de très grandes quantités de données binaires.

La fonction Large Object est mieux adaptée au stockage de très grands volume. Elle a cependant ses propres limites :

— la suppression d'un enregistrement ne supprime pas les données binaires associées. Une action de maintenance doit être effectuée à cette fin sur la base ;
— tout utilisateur connecté peut accéder aux données binaires même s'il n'a pas de droits sur la base principale.

L'utilisation des champs BLOB est déconseillée s'il n'y a pas de besoin de recherche dans les informations.

7.2 Schémas et structures des bases

Le partitionnement de table est implémenté en standard [22]. On notera une amélioration significative avec la V10, qui permet un partitionnement déclaratif alors qu'il était précédemment réalisé grâce à la notion d'héritage et l'écriture de triggers. La version 11 continue les améliorations, en particulier sur la

20. Types de données: https://docs.postgresql.fr/current/datatype.html
21. Norme ISO: http://fr.wikipedia.org/wiki/ISO_8601
22. Partitionnement: https://docs.postgresql.fr/current/ddl-partitioning.html

possibilité d'avoir des clés primaires/étrangères sur les partitions, le partition-pruning, la possibilité de définir une partition par défaut, …

En pratique, on évitera toutefois de créer plus de 500/1000 partitions pour une même table. La version 12 apporte toutefois des gains de performance très appréciables de ce côté.

7.3 Spécificités de développement

PostgreSQL est conforme au standard SQL 2011.

La documentation [23] propose des liens permettant de voir les fonctions effectivement couvertes et celles qui ne le sont pas encore : globalement, la couverture de la norme est très bonne même s'il est surprenant de voir que l'instruction `MERGE` n'est pas supportée. PostgreSQL dispose toutefois d'une instruction `INSERT...ON CONFLICT` [24] répondant à un besoin proche.

PostgreSQL ne nécessite donc pas un apprentissage lourd pour la syntaxe des requêtes. Toutefois, les habitudes des développeurs quant à l'utilisation des fonctions spécifiques et des syntaxes particulières pour les jointures de leur SGBD habituel nécessiteront un accompagnement. Par exemple, les fonctions sur les dates sont souvent propres à chaque SGBD (cf Annexes par SGBD).

Il est possible de créer des bibliothèques de fonctions personnalisées pour simuler les fonctions spécifiques des autres SGBD et faciliter la migration et la prise en main.

Le paramétrage des jeux de caractères se fait en fonction des applicatifs. L'UTF-8 est recommandé pour toute la chaîne. L'ISO peut être également utilisé si nécessaire.

Attention aux options par défaut à l'installation de PostgreSQL : si aucun paramétrage n'est fait, *initdb* et `CREATE DATABASE` utilisent la configuration de langue du serveur soit la page de codes LATIN9 (ISO 8859-15) ou à défaut de l'ASCII (https://www.postgresql.org/docs/current/static/app-initdb.html). Il faudra préciser UTF-8 lors de la création des bases de données.

PostgreSQL permet l'utilisation de vues matérialisées. Si elles peuvent être rafraîchies à la demande (et avec un blocage minimum via la clause `CONCURRENTLY`), il n'est actuellement pas possible de bénéficier nativement d'une mise à jour en continu.

7.4 API et modes d'accès

7.4.1 Interfaces client

PostgreSQL fournit plusieurs interfaces client pour l'accès aux données.

23. Couverture: https://www.postgresql.org/docs/current/static/features.html
24. INSERT ON CONFLICT : https://www.postgresql.org/docs/current/static/sql-insert.html#SQL-ON-CONFLICT

Nom	Langage	Commentaires	Adresse
DBD::Pg	Perl	Driver Perl DBI	https://metacpan.org/release/DBD-Pg
JDBC	Java	Type 4 JDBC driver	http://jdbc.postgresql.org/
libpqxx	C++	New-style C++ interface	http://pqxx.org/development/libpqxx/
Npgsql	.NET	.NET data provider	http://www.npgsql.org/
pgtclng	Tcl	Driver Tcl	http://sourceforge.net/projects/pgtclng/
psqlODBC	ODBC	Driver ODBC	https://odbc.postgresql.org/
psycopg	Python	DB API 2.0-compliant	http://initd.org/psycopg/

7.4.2 Couche d'abstraction

L'utilisation d'une couche d'abstraction (mapping objets/tables) de type « ORM » est compatible.

7.4.3 Pool de connexions

Un gestionnaire de connexion est fortement recommandé à partir de plusieurs centaines de connexions simultanées. PostgreSQL utilisant un processus à chaque nouvelle session, l'établissement et la fermeture d'une connexion peuvent devenir important par rapport au temps d'exécution de la requête. *Pgbouncer* [25] est stable et performant. Nous le recommandons. *PgPool* [26], d'une prise en main plus complexe, est pour sa part plutôt à conseiller pour réaliser du load-balancing.

7.5 Procédures stockées, fonctions et déclencheurs (triggers)

Un déclencheur ou « trigger » est une action (procédure stockée ou requête SQL) exécutée sur un événement. Parmi ses nombreux usages, on peut noter la mise en place de traces liées à l'application (gestion des erreurs, activité…).

Une procédure stockée est un programme stocké dans la base de données. Il s'agit généralement d'une fonction, et la version 11 permet de véritablement définir une procédure, appelée via la méthode CALL.

25. PgBouncer: https://pgbouncer.github.io/
26. Pgpool-II: http://www.pgpool.net/mediawiki/index.php/Main_Page

L'utilisation des procédures stockées et des déclencheurs est déconseillée pour éviter les adhérences et faciliter la portabilité. Par ailleurs, la logique métier ne doit pas être embarquée dans la base de données.

Les fonctions pourront être utilisées pour étendre l'usage des types de données pré-existants.

PostgreSQL permet l'écriture de fonctions et de procédures dans des langages différents du SQL et du C. Ces autres langages sont appelés génériquement des langages de procédures. Il existe à ce jour quatre langages de procédures dans la distribution standard de PostgreSQL:

— PL/pgSQL: langage de procédures SQL ;
— PL/Tcl: langage de procédures Tcl ;
— PL/Perl: langage de procédures Perl ;
— PL/Python: langage de procédures Python.

Il existe d'autres langages de procédures qui ne sont pas inclus dans la distribution principale (PL/Java, PL/PHP, PL/Py, PL/R, PL/Ruby, PL/Scheme, PL/sh). D'autres langages peuvent être définis par les utilisateurs, mais la procédure peut être un peu complexe.

7.6 Foreign Data Wrapper

Il s'agit d'extension qui permettent à PostgreSQL de communiquer avec d'autres sources de données. Les sources de données peuvent être des bases de données relationnelles (PostgreSQL, MySQL, Oracle, …), des bases de type NoSQL (CouchDB, MongoDB, ..), voire des fichiers CSV, des annuaires LDAP. Le fait que les données proviennent d'autres sources est transparent pour l'utilisateur final.

Certains FDW ont des possibilités de lecture/écriture comme Oracle, MySQL ; d'autres uniquement de lecture.

Pour obtenir la liste complète, se référer au wiki [27].

27. FDW: https://wiki.postgresql.org/wiki/Foreign_data_wrappers

8 Administration

8.1 Outils d'administration et d'exploitation de PostgreSQL

8.1.1 PgAdmin

PgAdmin est une application d'administration en mode client-serveur. Il s'agit d'un logiciel libre diffusé sous license PostgreSQL. Cet outil est disponible sur toutes les plateformes. Il est installé de base lors de l'installation de PostgreSQL sous Windows. Mais il est disponible aussi à l'url suivante : https://www.pgadmin.org/download/.

Il possède une interface graphique, qui le rend utilisable par tous. Parmi les points forts, le mode graphique pour réaliser une requête, la personnalisation de l'affichage, l'ajout de plugins (par exemple PostGIS Shapefile and DBF loader) et la présence d'un langage de script. Il permet également d'afficher graphiquement le résultat du plan d' exécution d'une requête (commande `EXPLAIN`).

Il peut aussi s'installer en mode serveur et éviter une installation sur un poste utilisateur.

Ce logiciel est mis à jour régulièrement.

8.1.2 psql

L'outil psql permet une administration en ligne de commande, il permet la saisie de requêtes SQL, l'affichage du schéma de base, ainsi que l'import et l'export. Il est présent dans toute distribution PostgreSQL et on recommandera son usage en tant que client par défaut.

Pour obtenir de l'aide sur les commandes SQL, il est possible d'utiliser `\help`. A partir de la version 12, cette commande renverra également le lien vers la documentation de l'instruction. Pour de l'aide concernant les meta-commandes psql, utiliser `\?`.

8.2 Supervision de PostgreSQL et d'analyse des logs

Les outils de supervision (Nagios, Munin,…) proposent des modules complémentaires (plugins) pour PostgreSQL, permettant la surveillance des logs et tables systèmes. Le script Perl *check_postgres* de Bucardo est le principal plugin pour ces outils. Il existe également un certain nombre d'outils dédiés, on notera par exemple *temboard* et *pgwatch2*.

D'autres modules permettent de suivre des statistiques d'évolution de la base (*pg_stat_statements* [28] dont on recommandera l'activation, *pgtop*,…) ou d'analyser les logs (*pgbadger* [29]).

28. pg_stat_statements: https://www.postgresql.org/docs/current/static/pgstatstatements.html
29. pgbadger: http://dalibo.github.io/pgbadger/

A noter que de nombreux modules (*pgfouine*, *pgstatpack*, …) ne sont pas maintenus sur la durée. Avant d'utiliser de tels modules généralement packagés sous forme d'extension, on en vérifiera la santé, les mises à jour…

8.3 Sauvegarde

Si PostgreSQL permet de réaliser en natif des sauvegardes à chaud, de faire du Point In Time Recovery, … il va souvent être nécessaire d'industrialiser le processus de sauvegarde, soit par un développement interne, soit en utilisant un outil dédié à la sauvegarde de bases PostgreSQL.

Parmi les plus avancés, on notera *Barman* et *pgBackRest*. Il permettront:

— de faciliter le PITR ;
— de réaliser des sauvegardes incrémentales ;
— d'avoir un catalogue centralisé de sauvegardes, pouvant provenir de plusieurs serveurs ;
— de gérer les durées de rétention ;
— …

8.4 Haute-disponibilité

Plusieurs outils existent pour améliorer la résilience ou faciliter la gestion d'un ensemble d'instances. On notera par exemple:

— *repmgr*: facilite la mise en place d'architectures complexes ainsi que les opérations administratives comme le failover/switchover/l'ajout d'une instance …
— *patroni*: un template de mise en place d'une solution de haute-disponibilité. Il est généralement utilisé avec des conteneurs et un outil de configuration distribué comme *etcd* ou *Consul*.
— *PAF*: PostgreSQL Automatic Failover se base sur Pacemaker/Corosync pour assurer la haute disponibilité et la possibilité de basculer de manière automatique.

8.5 Outils de migration des données vers PostgreSQL

L'outil *ora2pg* [30] permet d'analyser une base de données Oracle. L'extension ora2pg de la DGFiP permet une évaluation des coûts de migration. L'outil permet également de réaliser la migration.

Pour le transfert des données, l'utilisation d'un outil ETL (Talend Open Studio, *pgloader* [31], …) permet de réaliser des transformations sur les données au moment du transfert (adaptation des types de

30. ora2pg: https://ora2pg.darold.net/
31. pgLoader: https://pgloader.io/

données,…). Pour un chargement initial de données, on pourra se tourner dans le cas de volumes importants vers un outil comme *pg_bulkload*.

Le code applicatif devra souvent être adapté et nécessiter la traduction d'instructions SQL compatibles avec PostgreSQL. L'outil *code2pg* [32] pourra aider en ce sens en fournissant une estimation et une aide au développeur.

Il est fortement recommandé d'expérimenter la migration dans des conditions les plus proches de la production (base de production, logs des sessions applicatives,…).

32. code2pg: https://github.com/societe-generale/code2pg

9 Aide à la décision

Si PostgreSQL peut répondre à un grand nombre de besoins, certains projets pourront être mieux servis en utilisant d'autres technologies. Ce chapitre va s'attacher à décrire un processus de prise de décision tel qu'il est utilisé dans un certain nombre d'entreprises.

PostgreSQL n'est pas un produit soumis à licence dans sa version communautaire. Il n'a pas de surcoût logiciel à sa mise en place. On peut par volonté de garantie souscrire soit à du support sur la version communautaire soit à une version payante qui inclut du support et souvent des fonctionnalités avancées. Une stratégie souvent utilisée en entreprise est de prôner « PostgreSQL first » pour tout nouveau projet sauf en cas de fonctionnalité manquante SQL ou préconisation éditeur.

Concernant les migrations, leur ROI peut être difficile à justifier et peut même se révéler inexistant dans certains cas. Cela dépendra de la complexité du projet, des économies réalisées sur les licences, … Il est plus simple de faire cohabiter une montée de version logiciel avec une migration de moteur de bases de données. Il est souvent annoncé que PostgreSQL est efficace sur de petites volumétries mais certains membres du PGGTIE possèdent des bases (OLTP, Batch, données scientifiques, géo…) de plusieurs dizaines de To sans problème de performance ou de stabilité.

Seront listées dans le tableau plus bas:

— Fonctionnalité
— Le support natif (à partir de quelle version) ou via extension (à partir de quelle version)
— D'éventuels commentaires - limitations

Fonctionnalités	Disponibilité	Commentaires - Limitations
Pooler de connexion	via modules pgpool/pgbouncer …	voir § 7.4.3
Réplication sur cluster de serveurs maître-esclave sans répartition de charge	9.0 +	voir § 6.1
Haute-Disponibilité failover automatique & routage de connexion	via modules Patroni/repmgr …	voir § 8.4
Cryptage des données	avec extension Pgcrypto (8.3 +)	voir § 4.3

Fonctionnalités	Disponibilité	Commentaires - Limitations
Audit de l'activité	avec extension pgaudit (9.5 +)	voir § 5.2.4
NoSQL	json (9.2+)	voir § 4.1
Vues matérialisées	9.3 +	voir § 7.3
Partitionnement des tables	10 +	voir § 7.2/12.1.4
Gestions des LOBs	7.2+	
Données spatiales	avec module Postgis	voir § 4.1/7.1/12.5
Monitoring et supervision	avec module temboard/pgwatch2	
UPSERT	9.5 +	
transactions autonomes	via extension pg_background	
Indexation		
Utilitaires à disposition (maintenance, chargement des données…)	7.2+ (Vacuum) 7.1+ (Copy)	
auto-explain	via extension auto_explain	
Recherche plein texte	8.3 +	
Stockage (local ou SAN), externalisation des sauvegardes	Pg_basebackup ou via module Pgbarman/Pgbackrest	voir § 5.1.5/6.3/8.3

10 Limitations ou fonctionnalités non supportées

Ce paragraphe liste des fonctionnalités non supportées par la version communautaire de PostgreSQL. Cette liste est non exhaustive.

— Le chiffrement complet d'une base ;

— Les clusters actifs-actifs ;
— La compression ;

11 Conduite du changement

11.1 Formation des acteurs

La formation ciblera trois publics :

— les développeurs initiés : prise en main et migration vers PostgreSQL (2 à 3 jours) ;
— les exploitants et architectes : installation, maintenance, sauvegarde, supervision,… (3 jours) ;
— les administrateurs (DBA) avancés: (formation exploitants + 3 jours).

11.2 Support

Le support est assuré de plusieurs manières :

— Au niveau ministériel : par le marché interministériel porté par le ministère de l'intérieur ou en s'appuyant sur le réseau interministériel (MimPROD,…) ;
— pour les entreprises/administrations membres, par des échanges via le PGGTIE ;
— par la communauté PostgreSQL (mailing lists, forums, …).

Un support professionnel français est aussi possible, on se référera à la page support [33].

11.3 Plan de migration

La migration nécessite des adaptations plus ou moins importantes des applications pour lesquelles une recette visant à garantir la non régression doit être menée. Son coût est significatif.

Ce coût pourra être minimisé si la migration s'effectue à l'occasion d'une évolution fonctionnelle significative, qui nécessitera des recettes fonctionnelles et techniques indépendamment du changement de SGDB. A mettre toutefois en balance avec les risques de la réalisation de deux migrations simultanées, technique et fonctionnelle.

L'accompagnement des MOA et MOE voire de la production, pour une montée en compétences dans les meilleures conditions, devra également être planifié.

33. Support: https://www.postgresql.org/support/professional_support/europe/

12 Retour sur investissement

12.1 Coût de migration de la base

La migration vers un nouveau SGDB doit tenir compte :

— du coût d'entrée (formation des acteurs, montée en compétences,…) ;
— des adaptations des applicatifs incluant les coûts de recettes (technique, fonctionnelle et tests de non régression) ;
— de l'éventuel renouvellement du marché de support ;
— de la mise à niveau des procédures d'exploitation.

12.2 Coût de possession

PostgreSQL est sous une licence [34] open source similaire aux licences BSD ou MIT. Il n'y a donc pas de politique tarifaire subie de la part d'un éditeur.

12.3 Maîtrise des trajectoires

PostgreSQL est un produit dont la feuille de route (roadmap) est bien connue et maîtrisée.

L'écosystème de PostgreSQL s'enrichit constamment de nouveaux modules complémentaires et d'outils. Les progiciels commerciaux proposent de plus en plus d'interfaces nécessaires pour utiliser PostgreSQL.

La communauté PostgreSQL francophone est importante et très active : voir la liste de diffusion [35] et le forum [36]. Le groupe de travail entreprises (PGGTIE) a d'ailleurs été initié en 2016 pour répondre à ce besoin d'échange.

34. Licence: https://www.postgresql.org/about/licence/
35. Liste email: https://www.postgresql.org/list/pgsql-fr-generale/
36. Forum: https://forum.postgresql.fr/

13 Annexes

13.1 Migration depuis Oracle

Contribution du ministère de l'intérieur

Il faut prendre connaissance des éléments des serveurs sources pour pouvoir définir les serveurs cibles adéquats en matière d'environnement technique, de piles logicielles, de système de stockage, de données, de criticité de l'application et du suivi de ses performances.

13.1.1 Environnement technique

Les caractéristiques des serveurs Oracle :

Vérifier le type de serveurs : dédiés , mutualisés ou virtualisés ?

S'assurer des éléments techniques suivants :

- Types de processeurs ;
- Nombre de processeurs ;
- Fréquences horloge des processeurs ;
- Taille des registres ;
- Taille de la RAM.

13.1.2 Les piles logicielles

Relever les différentes piles logicielles composant l'application.

L'OS actuel pour quelle distribution de LINUX et quelle version ?

La version Oracle vers quelle version Postgres ?

Quels sont les serveurs d'application source ?

Le type de JVM.

Le système de virtualisation.

13.1.3 Espace de stockage

Quel est le système de stockage ? En général les SGBD sont hébergés sur du SAN (Storage Area Network) ; des solutions alternatives ne manquent pas.

Quel est le mode d'accès ? Le plus souvent VMDK (Virtual Machine Disk).

13.1.4 Migration de la base de données

Les outils de migration.

Ils sont divers: *Ora2Pg* , les ETL ou ELT, le développement de programmes spécifiques, etc…

Traiter d'abord les métadonnées avant le chargement des données.

MIGRATION DES STRUCTURES ORACLE vers POSTGRESQL : sont regroupés sous ce vocable les structures des tables et des vues avec leurs annexes, les différentes procédures, les fonctions, les triggers.

Procéder à la vérification de la présence éventuelle du partitionnement et des vues matérialisées.

Le partitionnement était basique sous PostgreSQL 9.1 et fut avant tout une solution de contournement. Depuis la 9.2 il est une fonctionnalité à part entière. Comme évoqué dans l'introduction, il est désormais déclaratif depuis la version 10.

Les vues matérialisées

De la 9.0 à 9.2 , elles passaient par des artifices et depuis la 9.3 c'est une fonctionnalité intégrée.

Le langage des procédures SQL

Oracle utilise le langage PL/SQL et PostgreSQL le PL/PGSQL ; ils sont assez proches mais une adaptation est requise.

Modèle de correspondance des types de données

Les chaînes de caractères peuvent être remplacées par VARCHAR et TEXT.

Les dates Oracle sur 7 octets par TIMESTAMP en 8 octets.

Les longues chaînes de caractères CLOB sont remplacés par TEXT.

Les valeurs numériques entières NUMBER selon la précision ascendante par SMALLINT, INTEGER, BIGINT, NUMERIC.

Les valeurs numériques décimales par NUMERIC.

Les données binaires BLOB par BYTEA (attention au caractère d'échappement avec Ora2Pg).

Migration des procédures normales et stockées

Recenser toutes les procédures et fonctions PL/SQL et les transformer en PL/PgSQL.

Génération des scripts de migration de la base de données

Le script de migration des structures doit être séparé des données ; il servira à initialiser la base PostgreSQL.

Création du script d'initialisation de la base PostgreSQL

Il est créé à partir des structures des données, des procédures et fonctions réadaptées pour PostgreSQL.

La migration des données

Elle peut se faire par un script SQL de données à insérer dans la nouvelle base PostgreSQL.

Il est aussi possible d'opter pour un chargement direct de la base Oracle vers la base PostgreSQL.

Le code applicatif

La modification du code est nécessaire pour intégrer le driver PostgreSQL pour la gestion des transactions, de l'ouverture et fermeture des connexions, des mots clés Oracle à remplacer par les mots clés PostgreSQL, des jointures externes, de la pagination, etc… Suivant où se trouve la logique de l'application, cette étape peut représenter une part importante de la migration.

13.1.5 Criticité et sauvegardes

Prise en compte des notions de PRA/PCA, RTO, RPO, et réplication.

Quelle solution technique est utilisée pour assurer la haute disponibilité : Oracle RAC, Oracle DATAGUARD ou autre ?

Il n'y a pas d'équivalent Oracle RAC sous PostgreSQL ; par contre la fonction DATAGUARD est assurée sous PostgreSQL par la réplication.

Le PRA (Plan de Reprise d'Activité) nous impose un délai de reprise d'activité de l'application. Il prend en compte le RTO (*Recovery Time Objective* ou la durée maximale d'interruption admissible) et le RPO (*Recovery Point Objective* ou la durée maximum d'enregistrement des données qu'il serait tolérable de perdre lors d'une panne). Cela conditionnera aussi en fonction de la volumétrie la technologie , la fréquence et le type de sauvegarde à mettre en place.

RMAN sous Oracle gère les sauvegardes et restaurations ; il n'y a pas d'équivalent de base sous PostgreSQL. Plusieurs outils tiers existent toutefois, notamment *pgBackRest* et *Barman*. Le mode Archivelog est bien pourvu dans les deux SGBD de façon quasiment identique.

Les restaurations physiques n'ont pas une granularité plus fine sous PostgreSQL (on restaure toute l'instance).

13.1.6 Suivi, amélioration des performances ,et supervision

Il n'existe pas de monitoring global comme le Grid Control sous Oracle. Cependant il est possible d'utiliser Nagios avec le plugin *check_postgres* [37]. D'autres outils sont également disponibles (*PoWA*,

37. Sonde check_postgres: https://bucardo.org/check_postgres/

PGObserver, …).

13.2 Migration depuis Db2

La migration amène à revoir le DDL issu de Db2 : il est recommandé de prévoir un outil de transformation (script) du ddl prenant en compte les éléments décrits ci-dessous.

Concernant le DCL, il est préférable de ne pas essayer de transposer de Db2 à pg les grants définis dans Db2, les niveaux d'autorisation étant trop différents.

Ne pas oublier de modifier les procédures de maintenance : sauvegardes, historisation, défragmentation de la base, collecte de statistiques, nettoyage des fichiers obsolètes, etc.

13.2.1 Quelques spécificités du DDL pg/Db2

— Chaînes de caractères :
PG: une chaîne de caractères est définie en nombre de caractères.
Db2: les chaînes de caractères fixes ou variables sont exprimées en octets avec pour conséquence un nombre de caractères possibles différent suivant l'encodage de la base : en utf-8 certains caractères sont codés sur plusieurs octets, en iso8859-15 chaque caractère vaut un octet. Une chaîne varchar(5) peut donc contenir moins de 5 caractère en Db2 si la base est en utf-8 et la zone alimentée par des caractères accentués.
En tenir compte si on doit migrer une base Db2 utf-8 vers une base PostgreSQL.

— Création d'une table dans un tablespace :

pg	Db2
create table… tablespace <nom_ts>	create table…IN <nom_ts>

— Création d'une table via le mot clé LIKE :dans pg, l'utilisation des ( ) est obligatoire autour de la clause LIKE, dans Db2 il n'en faut pas.

pg	Db2
create table eqos.statssov (LIKE eqos.stats INCLUDING ALL)	create table eqos.statssov LIKE eqos.stats

— Valeur par défaut d'une colonne de table :

pg	Db2
create table…DEFAULT <valeur>	create table …WITH DEFAULT <valeur>

— Création d'une table :
Certains mots-clés ne sont pas reconnus par pg donc à supprimer du ddl Db2 avant migration. Exemple : *append*, *with restrict on drop*, *long in*, toute la partie sur le table-partionning qui se défini différemment sous PostgreSQL.
— Incrément automatique d'un compteur colonne sur création de table: L'autoincrément de Db2 (*[generated always | generated by default] as identity*) est supporté depuis postgreSQL v12. Sur les versions antérieures il faut utiliser les séquences exclusivement.
— L'utilisation du type serial de pg permet de créer une séquence facilement mais il ne faut pas mettre le type integer (déjà porté par serial) ni « not null » sous peine d'erreur de syntaxe.
— Dans Db2, la création de séquence est possible en passant obligatoirement par un « create sequence ».

Tous les autoincréments Db2 sont donc à convertir si la version de PostgreSQL est inférieure à 12. Si la version est au moins égale à la v12, les mots-clés utilisés sont à contrôler car Db2 permet davantage d'options.

— L'utilisation du type serial de pg permet de créer une séquence facilement mais il ne faut pas mettre le type integer (déjà porté par serial) ni « not null » sous peine d'erreur de syntaxe.

Dans Db2, la création de séquence est possible en passant obligatoirement par un « create sequence ». **Tous les autoincréments Db2 sont donc à convertir**.

— Table non logguée : Une table Db2 déclarée en «*not logged initially*» devra être «*unlogged*» sous pg.
— Tables temporaires : La syntaxe et les fonctionnalités des tables temporaires diffèrent.

Exemple :

— `Db2 : declare global temporary table tabtemp like eqos.stats not logged`
— `pg : create local temporary table tabtemp (like eqos.stats) unlogged (?)`

Les tables temporaires Db2 avec du partage de données inter-sessions se font via un « create global temporary table », elles n'ont pas d'équivalence dans pg car dans pg bien que syntaxiquement les mots-clés « local » et « global » sont acceptés, le comportement de la table temporaire reste local dans les 2 cas (pas de partage des données temporaires).

— Colonne d'horodatage sur mise à jour de ligne :

Ces colonnes fréquemment utilisées dans Db2 n'existent pas dans pg.

Exemple dans Db2 : une colonne de nom TS_UPDATE est mise à jour automatiquement dès que la ligne est modifiée par UPDATE/INSERT sql:

```
CREATE TABLE...
TS_UPDATE TIMESTAMP NOT NULL GENERATED ALWAYS FOR EACH ROW ON
UPDATE
AS ROW CHANGE TIMESTAMP
```

Lors d'une insertion de ligne ou un update sur la ligne, Db2 renseigne automatiquement la colonne ts_update. Ces colonnes sont systématiquement déployées dans certaines bases Db2, **sous pg il faudra faire cette action applicativement donc modifier les programmes en conséquence**.

— Format timestamp ISO :

Non seulement la précision du timestamp est différente mais le séparateur entre date et time diffère également. **Attention lors de la migration des données de Db2 vers pg**!

pg	Db2
2014-01-31 00:00:00	2014-01-31__-__00.00.00.**000000**

A noter que le format « NOT NULL DEFAULT CURRENT TIMESTAMP » toléré dans Db2 n'est pas admis dans pg, il faut utiliser CURRENT_TIMESTAMP (reconnu par Db2 également).

— Vue matérialisée :
pg: création via un « CREATE MATERIALIZED VIEW » .
Db2 : création via un « create table » suivi d'options spécifiques à une vue matérialisée (appelée MQT dans Db2). **Le ddl issu de Db2 est donc à corriger**.
— Drop table et contraintes d'intégrité (CI):
pg : un « drop table… CASCADE » supprime les CI.
Db2 : un « drop table » suffit à supprimer les CI.
— Création d'un index, schéma :
pg : un nom d'index ne doit pas être précédé d'un nom de schéma sinon on a une erreur de syntaxe.
Db2 : il est recommandé de mettre le nom de schéma et l'outil Db2 de génération de ddl génère ce nom de schéma devant le nom d'ndex. S'il est omis dans Db2, ce sera un schéma de même nom que le schéma courant ou que l'autorité de connexion qui sera utilisé.
— Création d'un index, options :
Certains mots-clés ne sont pas reconnus par pg donc à supprimer du ddl Db2 avant migration.

Exemple : cluster, allow reverse scan, pctfree…

— Affectation des index à un tablespace dédié :
pg : sur la création de la table, l'affectation se fait dans la clause de création de la clé primaire ou de l'unicité. On peut mettre l'index dans un tablespace dédié lors de la création de l'index.
Db2 : on peut directement diriger tous les index liés à une table dans un tablespace avec la clause INDEX IN, exemple :

```
CREATE TABLE.. IN \<nom\_ts\_table\> INDEX IN \<nom\_ts\_index\>.
```

Ou lors de la création de l'index comme sous pg.

— Taille des pages :
pg : seules les pages de 8ko sont permises.
Db2 : on travaille avec des tailles de pages de 4, 8, 16 ou 32ko. Modifier le ddl Db2 qui préciserait explicitement des tailles de pages.

— Bufferpool :
La notion Db2 de bufferpool n'existe pas dans pg, on ne crée pas ces ressources donc le ddl Db2 doit être adapté en conséquence en retirant toutes les instructions de création mais aussi les références aux bufferpools dans les tablespaces.

— Tablespaces

Les options de création des ts diffèrent entre Db2 et pg, à corriger.

13.2.2 DCL

— Rôles vs groupes Unix :
pg : il faut impérativement passer par des grants donnés à des rôles.
Db2 : les grants sont donnés soit à des rôles soit directement aux groupes Unix ce qui est le cas le plus fréquent au MEN.

— Droits PUBLIC :
La notion Db2 de base RESTRICTIVE qui supprime les grants dits PUBLIC n'existe pas sous pg.

— GRANT ALTER TABLE :
Utilisé sous Db2, il n'existe pas tel quel sous pg. Il faut sous pg faire appartenir le compte souhaitant modifier la table à un rôle qui est OWNER de la table.

— TRUNCATE TABLE:
pg : il faut faire un « grant truncate » pour être autorisé à faire un truncate.
Db2 : un « grant delete » (ou control) donne ce privilège.

— GRANT CONNECT :
Pour être autorisé à se connecter à une database, il faut faire :
pg : grant connect on database <nom_base>… Db2 : grant connect on database il ne faut pas

préciser le nom de la base car le grant est fait connexion déjà prise sur la base, donc c'est la base courante. Si on précise le nom de la base on a une erreur de syntaxe.
D'autres différences de syntaxe peuvent apparaître.

— GRANTS sans équivalence Db2/pg :
Une partie des grants Db2, ceux liés à des fonctionnalités non présentes ou différentes sous pg, ne sont pas reportés sous pg. C'est le cas des grant load, grant use of tablespace, grant usage on workflow, etc.

— Catalogue système

Les informations sont stockées en majuscules dans Db2, en minuscules dans pg. En tenir compte dans la recherche d'information et notamment dans les scripts qui utilisent le catalogue pour y collecter des informations.

Exemple : rechercher les informations des tables appartenant à un schéma de nom IDENTITE
pg: `select * from pg\_tables where schemaname = 'identite'`
Db2: `select * from syscat.tables where tabschema = 'IDENTITE'`

13.2.3 Autres considérations

— Utilitaire de chargement :
Db2 : une table Db2 peut être chargée par *import*, *load*, *ingest* ou *db2move*.
pg utilise exclusivement l'utilitaire *copy* qui est bien moins riche que les utilitaires Db2. Le chargement partiel d'une table comme dans Db2 n'est possible que depuis la V12 avec l'introduction d'une clause WHERE.

— Autres utilitaires Db2/pg:
reorg devient *vacuum*
runstats devient *analyse*
backup devient *pg_dump*, *pg_dumpall* ou *pg_basebackup*

— Double quote “:

Lors de la migration, prendre garde à la double quote " : elle est le délimiteur de chaîne par défaut dans Db2 quand on exporte les données donc elle se retrouve dans le fichier déchargé (export) entourant chaque chaîne de caractère. Au chargement vers pg, cette double quote se retrouve par défaut dans la table si elle est présente dans le fichier de données. Pour éviter ce chargement non souhaité, les fichiers exportés depuis Db2 doivent être en mode « del » (ascii) et importés dans pg comme du csv via les options WITH CSV delimiter « , » QUOTE ’“’ sur le copy.

— Journaux de transaction:
Db2 : chaque base possède ses propres journaux.
pg : ce n'est pas le cas, les journaux sont communs à plusieurs bases au sein d'une même

instance ce qui peut poser des problèmes (journaux mobilisés par une base, impact multi-bases en cas de corruption ou de disparition des journaux.…).

— LOBs:
Les BLOB Db2 sont à remplacer par des colonnes au format BYTEA.
Les CLOB Db2 sont à remplacer par le format TEXT. contrairement à Db2 qui permet d'embarquer les LOBs dans sa sauvegarde (backup), pg ne sauvegarde pas les LOBs via *pg_dumpall* (pg sauvegarde les LOBs vi pg_dump).
Les options Db2 liées aux LOBs ne s'appliquent plus sous pg (logged/not logged, compact/not compact, inline length…).

— Procédures stockées:

Ecrites dans Db2 en SQL/PL, elles devront être adaptées à pg (PL/PgSQL).

— Code source (programmes):
Modifier les méthodes d'appel du driver et l'utilisation de ses properties.
Changer le code SQL qui pourrait être spécifique à Db2/pg.

13.3 Migration depuis Informix

Contribution du ministère des affaires sociales

13.3.1 Structure

Schéma des bases de données (tables, indexes, contraintes, …).

Des scripts devront permettre la création des différentes bases de données sous PostgreSQL.

Il faudra s'assurer que l'ensemble des objets soient créés en respectant la syntaxe de PostgreSQL, ainsi que les normes préconisées :

— Tables ;
— Vues ;
— Triggers ;
— Contraintes ;
— Indexes.

Le tableau suivant décrits quelques différences pour les objets précités entre les SGBD Informix et PostgreSQL :

Objets / SGBD	Informix	PostgreSQL
Type de données	BLOB	BYTEA
Type de données	DATETIME	TIMESTAMP
Contraintes (clé étrangère)	alter table nom_table add constraint (foreign key (nom_colonne) references adr constraint nom_contrainte);	alter table nom_table add constraint nom_contrainte foreign key (nom_colonne) references adr (nom_colonne);
Index	create index nom_index on nom_table (nom_colonne) using btree ;	create index nom_index on nom_table using btree(nom_colonne) ;

13.3.2 Plan à suivre lors de la migration des bases de données

Les étapes suivantes seront suivies lors de la migration :

— Dans les bases de données sources (Informix), suppression des procédures stockées n'étant pas utilisées par les applications java, cela afin d'éviter de migrer vers PostgreSQL des traitements

qui ne seront jamais exécutés ;
— Création des bases de données ;
— Chargement des données ;
— Création des indexes et des contraintes. Les indexes et contraintes seront créées après le chargement des données afin d'optimiser la durée d'exécution du traitement.

13.3.3 Intégration du framework Hibernate

Si aucun framework n'est utilisé dans une application pour gérer la persistance des objets en base de données relationnelle, la solution serait l'utilisation du framework Hibernate.

Cette solution permettrait de ne pas écrire en dur les requêtes SQL dans le code source Java, les requêtes étant générées par Hibernate.

Les applications seraient donc moins dépendantes d'un SGBD (il resterait les procédures stockées).

Les applications qui utilisent le framework *Hibernate* gérant la persistance des objets en base de données relationnelle. Les requêtes SQL ne sont donc pas écrites dans le code Java, contrairement aux applications sans Hibernate, mais générées par *Hibernate*.

La migration d'Informix vers PostgreSQL aura donc un impact limité sur les sources de ces applications.

Des tests seront toutefois nécessaires afin de s'assurer qu'il n'y ait pas de régression.

13.3.4 Risques

Ce type de risques peut être identifié :

— Interruption de l'utilisation des applications durant la migration pendant la phase de chargement des données des bases Informix vers les bases PostgreSQL ;
— Une solution permettant de réduire la durée d'exécution consisterait à charger dans un premier temps les données statiques (comme les tables de nomenclature), pour ne charger que les données susceptibles d'évoluer (demandes, dossiers, …) ;
— le jour de la migration en production. Cette solution serait à envisager dans le cas où la durée du chargement dans l'environnement du Ministère serait trop longue et si un gain de temps est réalisé (cela dépend de la proportion des données pouvant être qualifiées comme données statiques) ;
— Ralentissement de certains traitements qui avaient été optimisés pour le SGBD Informix IDS.

13.4 Migration depuis MS SQL

Quelques éléments :

Les outils pouvant faciliter la migration des données sont:

— https://github.com/dalibo/sqlserver2pgsql ;
— http://pgloader.io/howto/quickstart.html.

Pour la migration des procédures, le code est très différent et doit être réécrit.

13.5 Quelques références

Quelques sites utilisant PostgreSQL, tirés des témoignages (parfois datés) disponibles sur postgresql.fr:

Meteo France http://www.postgresql.fr/temoignages:meteo_france Volume de données : 3.5 To;

Le Bon Coin http://www.postgresql.fr/temoignages:le_bon_coin Volume de données : > 6 To;

IGN http://www.postgresql.fr/temoignages:ign Capacité de traiter plus de 100 millions d'objets géométriques ;

Mappy http://www.oslandia.com/oslandia-et-mappy-vers-lopensource.html

13.6 Extensions et plugins pour PostgreSQL

Le tableau ci-dessous présente quelques plugins évoqués dans le document :

nom	fonction
PostGIS	Cartouche spatiale
pgstatpack	Statistiques
pgcrypto	Cryptographie

13.7 Outils tiers pour PostgreSQL

Le tableau ci-dessous présente quelques outils tiers évoqués dans le document :

nom	fonction
barman	Sauvegarde / restauration
ora2pg	Outil de migration
pgBackRest	Sauvegarde / restauration
pgbadger	Analyse des logs
pgLoader	Chargement de données
pgPool	Gestion des pools de connexions
pgtop	Statistiques
pgwatch2	Supervision
pitrery	Sauvegarde / restauration
temboard	Supervision

14 Documents de référence

Documentation PostgreSQL en français:

— Manuel: https://docs.postgresql.fr/current/
— Forums: https://forums.postgresql.fr/

Documentation en anglais:

— Manuel: https://www.postgresql.org/docs/current/index.html
— Wiki: https://wiki.postgresql.org
— Listes de diffusion: https://www.postgresql.org/list/