Module N1

Plan de migration

22.09

Dalibo SCOP

https://dalibo.com/formations

Plan de migration

Module N1

TITRE : Plan de migration
SOUS-TITRE : Module N1

REVISION: 22.09
DATE: 02 septembre 2022
COPYRIGHT: © 2005-2022 DALIBO SARL SCOP
LICENCE: Creative Commons BY-NC-SA

Postgres®, PostgreSQL® and the Slonik Logo are trademarks or registered trademarks of the PostgreSQL Community Association of Canada, and used with their permission.
(Les noms PostgreSQL® et Postgres®, et le logo Slonik sont des marques déposées par PostgreSQL Community Association of Canada.
Voir https://www.postgresql.org/about/policies/trademarks/)

Remerciements : Ce manuel de formation est une aventure collective qui se transmet au sein de notre société depuis des années. Nous remercions chaleureusement ici toutes les personnes qui ont contribué directement ou indirectement à cet ouvrage, notamment : Jean-Paul Argudo, Alexandre Anriot, Carole Arnaud, Alexandre Baron, David Bidoc, Sharon Bonan, Franck Boudehen, Arnaud Bruniquel, Damien Clochard, Christophe Courtois, Marc Cousin, Gilles Darold, Jehan-Guillaume de Rorthais, Ronan Dunklau, Vik Fearing, Stefan Fercot, Pierre Giraud, Nicolas Gollet, Dimitri Fontaine, Florent Jardin, Virginie Jourdan, Luc Lamarle, Denis Laxalde, Guillaume Lelarge, Benoit Lobréau, Jean-Louis Louër, Thibaut Madelaine, Adrien Nayrat, Alexandre Pereira, Flavie Perette, Robin Portigliatti, Thomas Reiss, Maël Rimbault, Julien Rouhaud, Stéphane Schildknecht, Julien Tachoires, Nicolas Thauvin, Be Hai Tran, Christophe Truffier, Cédric Villemain, Thibaud Walkowiak, Frédéric Yhuel.

À propos de DALIBO : DALIBO est le spécialiste français de PostgreSQL. Nous proposons du support, de la formation et du conseil depuis 2005. Retrouvez toutes nos formations sur https://dalibo.com/formations

LICENCE CREATIVE COMMONS BY-NC-SA 2.0 FR

Attribution - Pas d'Utilisation Commerciale - Partage dans les Mêmes Conditions

Vous êtes autorisé à :

- Partager, copier, distribuer et communiquer le matériel par tous moyens et sous tous formats
- Adapter, remixer, transformer et créer à partir du matériel

Dalibo ne peut retirer les autorisations concédées par la licence tant que vous appliquez les termes de cette licence selon les conditions suivantes :

Attribution : Vous devez créditer l'œuvre, intégrer un lien vers la licence et indiquer si des modifications ont été effectuées à l'œuvre. Vous devez indiquer ces informations par tous les moyens raisonnables, sans toutefois suggérer que Dalibo vous soutient ou soutient la façon dont vous avez utilisé ce document.

Pas d'Utilisation Commerciale : Vous n'êtes pas autorisé à faire un usage commercial de ce document, tout ou partie du matériel le composant.

Partage dans les Mêmes Conditions : Dans le cas où vous effectuez un remix, que vous transformez, ou créez à partir du matériel composant le document original, vous devez diffuser le document modifié dans les même conditions, c'est à dire avec la même licence avec laquelle le document original a été diffusé.

Pas de restrictions complémentaires : Vous n'êtes pas autorisé à appliquer des conditions légales ou des mesures techniques qui restreindraient légalement autrui à utiliser le document dans les conditions décrites par la licence.

Note : Ceci est un résumé de la licence. Le texte complet est disponible ici :

https://creativecommons.org/licenses/by-nc-sa/2.0/fr/legalcode

Pour toute demande au sujet des conditions d'utilisation de ce document, envoyez vos questions à contact@dalibo.com[1] !

[1] mailto:contact@dalibo.com

Chers lectrices & lecteurs,

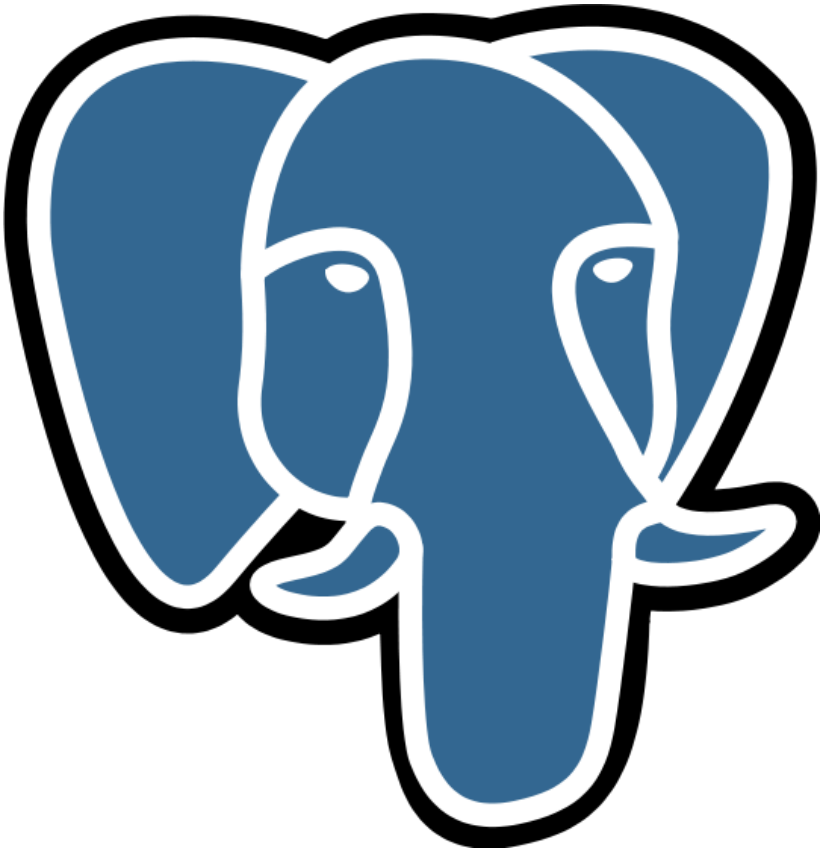
Nos formations PostgreSQL sont issues de nombreuses années d'études, d'expérience de terrain et de passion pour les logiciels libres. Pour Dalibo, l'utilisation de PostgreSQL n'est pas une marque d'opportunisme commercial, mais l'expression d'un engagement de longue date. Le choix de l'Open Source est aussi le choix de l'implication dans la communauté du logiciel.

Au-delà du contenu technique en lui-même, notre intention est de transmettre les valeurs qui animent et unissent les développeurs de PostgreSQL depuis toujours : partage, ouverture, transparence, créativité, dynamisme... Le but premier de nos formations est de vous aider à mieux exploiter toute la puissance de PostgreSQL mais nous espérons également qu'elles vous inciteront à devenir un membre actif de la communauté en partageant à votre tour le savoir-faire que vous aurez acquis avec nous.

Nous mettons un point d'honneur à maintenir nos manuels à jour, avec des informations précises et des exemples détaillés. Toutefois malgré nos efforts et nos multiples relectures, il est probable que ce document contienne des oublis, des coquilles, des imprécisions ou des erreurs. Si vous constatez un souci, n'hésitez pas à le signaler via l'adresse formation@dalibo.com !

Table des Matières

1 PLAN DE MIGRATION



1.1 INTRODUCTION

Ce module est organisé en trois parties :

- Méthodologie de la migration
- Choix de l'outil de migration
- Installation d'Ora2Pg

Ce module est une introduction aux migrations de Oracle vers PostgreSQL. Nous y abordons comment gérer sa première migration (quel que soit le SGBD source et destination),

puis nous réfléchirons sur le contenu de la migration et sur le choix de l'outil idoine.

À l'issue de ce module, l'outil Ora2Pg sera installé sur l'environnement Linux de formation pour poursuivre les exercices de migration de bout en bout.

1.2 AVERTISSEMENT

- Ceci est écrit par des spécialistes de PostgreSQL
  - pas d'Oracle

Notre expertise est sur PostgreSQL et nous nous reposons sur notre expérience de plusieurs années de migrations Oracle vers PostgreSQL pour écrire cette formation.

Malgré tous nos efforts, certaines informations sur Oracle pourraient être erronées, ou ne plus être vraies avec les dernières versions. Nos clients n'utilisent pas forcément les mêmes fonctionnalités, avec le même niveau d'expertise, et nous n'avons pas forcément correctement restitué leur retour.

Si vous constatez des erreurs ou des manques, n'hésitez pas à nous en faire part[2] pour que nous puissions améliorer ce support de formation.

1.3 MÉTHODOLOGIE DE MIGRATION

La première migration est importante :

- Les méthodes employées seront réutilisées, améliorées…
- Un nouveau SGBD doit être supporté pendant de nombreuses années
- Elle influence la vision des utilisateurs vis-à-vis du SGBD
- Une migration ratée ou peu représentative est un argument pour les détracteurs du projet

La façon dont la première migration va se dérouler est essentielle. C'est sur cette expérience que les autres migrations seront abordées. Si l'expérience a été mauvaise, il est même probable que les migrations prévues après soient repoussées fortement, voire annulées.

Il est donc essentiel de réussir sa première migration. Réussir veut aussi dire bien la documenter, car elle servira de base pour les prochaines migrations : les méthodes employées seront réutilisées, et certainement améliorées. Réussir veut aussi dire la publiciser, au

[2] mailto:formation@dalibo.com

moins en interne, pour que tout le monde sache que ce type de migration est réalisable et qu'une expérience est disponible en interne.

De plus, cette migration va influencer fortement la vision des développeurs et des utilisateurs vis-à-vis de ce SGBD. Réussir la migration veut donc aussi dire réussir à faire apprécier et accepter ce moteur de bases de données. Sans cela, il y a de fortes chances que les prochaines migrations ne soient pas demandées volontairement, ce qui rendra les migrations plus difficiles.

1.3.1 PROJET DE MIGRATION

Le projet doit être choisi avec soin :

- Ni trop gros (trop de risque)
- Ni trop petit (sans valeur)
- Transversal :
  - Implication maximale
  - Projet de groupe, pas individuel

Le premier projet de migration doit être sélectionné avec soin.

S'il est trop simple, il n'aura pas réellement de valeur. Par exemple, dans le cas d'une migration d'une base de 100 Mo, sans routines stockées, sans fonctionnalités avancées, cela ne constituera pas une base qui permettra d'aborder tranquillement une migration d'une base de plusieurs centaines de Go et utilisant des fonctionnalités avancées.

L'inverse est aussi vrai. Un projet trop gros risque d'être un souci. Prenez une base critique de plusieurs To, dotée d'un très grand nombre de routines stockées. C'est un véritable challenge, y compris pour une personne expérimentée. Il y a de fortes chances que la migration soit longue, dure, mal vécue… et possiblement annulée à cause de sa complexité. Ceci aura un retentissement fort sur les prochaines migrations.

Il est préférable de choisir un projet un peu entre les deux : une base conséquente (plusieurs dizaines de Go), avec quelques routines stockées, de la réplication, etc. Cela aura une vraie valeur, tout en étant à portée de main pour une première migration.

Une fois une telle migration réussie, il sera plus simple d'aborder correctement et sans crainte la migration de bases plus volumineuses ou plus complexes.

Il faut aussi ne pas oublier que la migration doit impliquer un groupe entier, pas seulement une personne. Les développeurs, les administrateurs, les équipes de support doivent tous être impliqués dans ce projet, pour qu'ils puissent intégrer les changements quant à l'utilisation de ce nouveau SGBD.

1.3.2 ÉQUIPE DU PROJET DE MIGRATION

- Chef de projet
- Équipe hétérogène (pas que des profils techniques)
- Recetteurs et utilisateurs nombreux (validation du projet la plus continue possible)

L'équipe du projet de migration doit être interne, même si une aide externe peut être sollicitée. Un chef de projet doit être nommé au sein d'une équipe hétérogène, composée de développeurs, d'administrateurs, de testeurs et d'utilisateurs. Il est à noter que les testeurs sont une partie essentielle de l'équipe.

1.3.3 EXPERTISE EXTÉRIEURE

- Société de service
- Contrat de support
- Expert PostgreSQL

Même si l'essentiel du projet est porté en interne, il est toujours possible de faire appel à une société externe spécialisée dans ce genre de migrations. Cela permet de gagner du temps sur certaines étapes de la migration pour éviter certains pièges, ou mettre en place l'outil de migration.

1.3.4 GESTION DE PROJET

- Réunions de lancement, de suivi
- Reporting
- Serveurs de projet
- ...
- Pas un projet au rabais, ou un travail de stagiaire

Cette migration doit être gérée comme tout autre projet :

- une réunion de lancement ;
- des réunions de suivi ;
- des rapports d'avancements.

De même, ce projet a besoin de ressources, et notamment des serveurs de tests : par exemple un serveur Oracle contenant la base à migrer (mais qui ne soit pas le serveur

de production), et un serveur PostgreSQL contenant la base à migrer. Ces deux serveurs doivent avoir la volumétrie réelle de la base de production, sinon les tests de performance n'auront pas vraiment de valeur.

En fait, il faut vraiment que cette migration soit considérée comme un vrai projet, et pas comme un projet au rabais, ce qui arrive malheureusement assez fréquemment. C'est une opération essentielle, et des ressources compétentes et suffisantes doivent être offertes pour la mener à bien.

1.3.5 PASSER À POSTGRESQL

- Ce n'est pas une révolution
- Le but est de faire des économies
  - … sans chamboulement

En soi, passer à PostgreSQL n'est pas une révolution. C'est un moteur de base de données comme les autres, avec un support du SQL (et quelques extensions) et ses fonctionnalités propres. Ce qui change est plutôt l'implémentation, mais, comme nous le verrons dans cette formation, si une fonctionnalité identique n'existe pas, une solution de contournement est généralement disponible.

La majorité des utilisateurs de PostgreSQL vient à PostgreSQL pour faire des économies (sur les coûts de licence). Si jamais une telle migration demandait énormément de changements, ils ne viendraient pas à PostgreSQL. Or la majorité des migrations se passe bien, et les utilisateurs restent ensuite sur PostgreSQL. Les migrations qui échouent sont généralement celles qui n'ont pas été correctement gérées dès le départ (pas de ressources pour le projet, un projet trop gros dès le départ, etc.).

1.3.6 MOTIVER

- Formations indispensables
- Divers cursus
  - du chef de projet au développeur
- Adoption grandissante de PostgreSQL
  - pérennité

Le passage à un nouveau SGBD est un peu un saut dans l'inconnu pour la majorité des personnes impliquées. Elles connaissent bien un moteur de bases de données et souvent ne comprennent pas pourquoi on veut les faire passer à un autre moteur. C'est pour cela

qu'il est nécessaire de les impliquer dès le début du projet, et, le cas échéant, de les former. Il est possible d'avoir de nombreuses formations autour de PostgreSQL pour les différents acteurs : chefs de projet, administrateurs de bases de données, développeurs, etc.

1.3.7 VALORISER

- Concepts PostgreSQL très proches des SGBD propriétaires
  - Adapter les compétences
  - Ne pas tout reprendre à zéro

De toute façon, les concepts utilisés par PostgreSQL sont très proches des concepts des moteurs SGBD propriétaires. La majorité du temps, il suffit d'adapter les compétences. Il n'est jamais nécessaire de reprendre tout à zéro. La connaissance d'un autre moteur de bases de données permet de passer très facilement à PostgreSQL, ce qui valorise l'équipe.

1.3.8 GESTION DES DÉLAIS

Souvent moins important :

- Le service existe déjà
- Donner du temps aux acteurs

Contrairement à d'autres projets, le service existe déjà. Les délais sont donc généralement moins importants, ce qui permet de donner du temps aux personnes impliquées dans le projet pour fournir une migration de qualité (et surtout documenter cette opération).

1.3.9 COÛTS

- Budget ?
- Open source <> gratuit
  - coûts humains
  - coûts matériels

Une migration aura un coût important. Ce n'est pas parce que PostgreSQL est un logiciel libre que tout est gratuit. La mise à disposition de ressources humaines et matérielles aura un coût. La formation du personnel aura un coût. Mais ce coût sera amoindri par le fait que, une fois cette migration réalisée, les prochaines migrations n'auront un coût qu'au niveau matériel principalement.

1.3.10 QUALITÉ

- Cruciale
  - la réussite est obligatoire
- Le travail effectué doit être réutilisable
- Ou tout du moins l'expérience et les méthodologies

La qualité de la première migration est cruciale. Si le but est de migrer les autres bases de données de l'entreprise, il est essentiel que la première migration soit une réussite totale. Il est essentiel qu'elle soit documentée, discutée, pour que le travail effectué soit réutilisable (soit complètement, soit uniquement l'expérience et les méthodes) afin que la prochaine migration soit moins coûteuse.

1.3.11 BUT DE LA PREMIÈRE MIGRATION

- Privilégier la qualité
- Contrôler les coûts
- N'est souvent pas contrainte par des délais stricts

Pour résumer, la première migration doit être suffisamment simple pour ne pas être un échec et suffisamment complexe pour être en confiance pour les prochaines migrations. Il est donc essentiel de bien choisir la base de sa première migration.

Il est aussi essentiel d'avoir des ressources humaines et matérielles suffisantes, tout en contrôlant les coûts.

Enfin, il est important de ne pas stresser les acteurs de cette migration avec des délais difficiles à tenir. Le service est déjà présent et fonctionnel, la première migration doit être un succès si l'on veut continuer, autant donner du temps aux équipes responsables de la migration.

1.4 CHOIX DE L'OUTIL DE MIGRATION

Avant de pouvoir porter l'application et le PL :

- Migrer le schéma
- Migrer les données

Avant de pouvoir traiter le code, qu'il soit applicatif ou issu des routines stockées, il faut procéder à la migration du schéma et des données. C'est donc ce dont nous allons parler dans cette partie.

1.4.1 BESOINS DE LA MIGRATION : SCHÉMA

- Veut-on migrer le schéma tel quel ?
- Utiliser les fonctionnalités de PostgreSQL ?
  - n'est plus vraiment à isofonctionnalité
- Créer un nouveau schéma :
  - d'un coup
  - les tables d'abord, les index et contraintes ensuite ?

La première question à se poser concerne le schéma : veut-on le migrer tel quel ? Le changer peut permettre d'utiliser des fonctionnalités plus avancées de PostgreSQL. Cela peut être intéressant pour des raisons de performances, mais a comme inconvénient de ne plus être une migration isofonctionnelle.

Généralement, il faudra créer un nouveau schéma, et intégrer les objets par étapes : tables, index, puis contraintes.

1.4.2 BESOINS DE LA MIGRATION : TYPES

- On rencontre souvent les types suivants sous Oracle :
  - `number(18,0)`, `number(4,0)` …
  - `int` : `-2147483648` à `+2147483647` (4 octets, `number(9,0)`)
  - `bigint` : `-9223372036854775808` à `9223372036854775807` (8 octets, `number(18,0)`)
- Type natifs bien plus performants (gérés par le processeur, taille fixe)
- Certains outils migrent en `numeric(x,0)`, d'autres en `int`/`bigint`
  - peut être un critère de choix

Oracle utilise généralement `number` pour les types entiers. L'équivalent strict au niveau PostgreSQL est `numeric` mais il est préférable de passer à d'autres types de données comme `int` (un entier sur quatre octets) ou `bigint` (un entier sur huit octets) qui sont bien plus performants.

L'outil pour la migration devra être sélectionné suivant ses possibilités au niveau de la transformation de certains types en d'autres types, si jamais il est décidé de procéder ainsi.

1.4.3 BESOINS DE LA MIGRATION : AUTRES TYPES

- Types plein texte ?
- Blob ?
- GIS ?
- ...
- Un développement peut être nécessaire pour des types spéciaux

L'outil de migration doit pouvoir aussi gérer des types particuliers, comme les types spécifiques à la recherche plein texte, ceux spécifiques aux objets binaires, ceux spécifiques à la couche spatiale, etc. Il est possible qu'un développement soit nécessaire pour faciliter la migration. Un outil libre est préférable dans ce cas.

1.4.4 BESOINS DE LA MIGRATION

- Déclarer les tables
- Les remplir
- Puis seulement déclarer les index, PK, FK, contraintes...
- Performances...

Pour des raisons de performances, il est toujours préférable de ne déclarer les index et les contraintes qu'une fois les tables remplies. L'outil de migration doit aussi prendre cela en compte : création des tables, remplissage des tables et enfin création des index et contraintes.

1.4.5 MIGRATION DES DONNÉES

Veut-on :

- Migrer en une seule fois les données ? (« Big Bang »)
- Pouvoir réaliser des incréments ?
- Paralléliser sur plusieurs sessions/threads ?
- Modifier des données « au passage » ?

Toujours dans les décisions à prendre avant la migration, il est important de savoir si l'on veut tout migrer d'un coup ou le faire en plusieurs étapes. Les deux possibilités ont leurs avantages et inconvénients.

De même, souhaite-t-on paralléliser l'import et l'export ? De ce choix dépend principalement l'outil que l'on va sélectionner pour la migration.

Enfin, souhaite-t-on modifier des données lors de l'opération de migration ? Là aussi, cela peut se concevoir, notamment si certains types de données sont modifiés. Mais c'est une décision à prendre lors des premières étapes du projet.

1.4.6 FONCTIONNALITÉS PROBLÉMATIQUES

Lors de la migration, certaines fonctionnalités d'Oracle auront peu ou pas d'équivalent :

- Vues matérialisées (mise à jour)
- Partitionnement (différences)
- Synonymes
- Conversion de type implicite
- Absence de *hint* (tag de requête)
- Accès par ROWID

Certaines fonctionnalités Oracle n'ont pas d'équivalents natifs dans PostgreSQL. Il faut avoir conscience que l'outil de migration ne pourra pas convertir intégralement les objets avancés disponibles sur Oracle.

Vues matérialisées

Concernant les vues matérialisées, elles existent sous PostgreSQL depuis la version 9.3. Cependant, elles ne disposent pas de toutes les fonctionnalités accessibles sous Oracle. Il est possible de les implémenter de toutes pièces en utilisant des fonctions et triggers. En attendant leur implémentation complète au cœur du code de PostgreSQL, voici des documents expliquant de manière détaillée comment implémenter des vues matérialisées :

- PostgreSQL/Materialized Views[3]
- Materialized Views that Really Work[4]

Partitionnement

Les partitions telles que gérées sous Oracle existent sous PostgreSQL depuis la version 10. Avec une version inférieure, il faut utiliser l'héritage et définir des triggers et contraintes CHECK. Pour plus de détails, consultez le document 5.9. Partitionnement de tables[5] ainsi que le document Partitionnement (module DBA2)[6] .

Les types de partitions supportées sont `LIST` et `RANGE`. Les partitions de type `HASH` sont supportées par PostgreSQL 11. Le partitionnement par référence n'est pas supporté.

[3] http://tech.jonathangardner.net/wiki/PostgreSQL/Materialized_Views
[4] https://www.pgcon.org/2008/schedule/events/69.en.html
[5] https://docs.postgresql.fr/current/ddl-partitioning.html
[6] https://dali.bo/v1_html

Synonymes

Les synonymes d'Oracle n'ont pas d'équivalent sous PostgreSQL. Il doit être possible d'utiliser des vues pour tenter d'émuler cette fonctionnalité dans la mesure où il s'agit d'accéder à des objets d'autres schémas.

Absence de *hint*

L'optimiseur Oracle supporte des *hints*, qui permettent au DBA de tromper l'optimiseur pour lui faire prendre des chemins que l'optimiseur a jugé trop coûteux. Ces *hints* sont exprimés sous la forme de commentaires et ne seront donc pas pris en compte par PostgreSQL, qui ne gère pas ces hints.

Néanmoins, une requête comportant un *hint* pour contrôler l'optimiseur Oracle doit faire l'objet d'une attention particulière, et l'analyse de son plan d'exécution devra être faite minutieusement, pour s'assurer que, sous PostgreSQL, la requête n'a pas de problème particulier, et agir en conséquence le cas échéant. C'est notamment vrai lorsque l'une des tables mises en œuvre est particulièrement volumineuse. Mais, de manière générale, l'ensemble des requêtes portées devront voir leur plan d'exécution vérifié.

Le plan d'exécution de la requête sera vérifié avec l'ordre `EXPLAIN ANALYZE` qui fournit non seulement le plan d'exécution en précisant les estimations de sélectivité réalisées par l'optimiseur, mais va également exécuter la requête et fournir la sélectivité réelle de chaque nœud du plan d'exécution. Une forte divergence entre la sélectivité estimée et réelle permet de détecter un problème. Souvent, il s'agit d'un problème de précision des statistiques. Il est possible d'agir sur cette précision de plusieurs manières.

Tout d'abord, il est possible d'augmenter le nombre d'échantillons collectés, pour construire notamment les histogrammes. Le paramètre `default_statistics_target` contrôle la précision de cet échantillon. Pour une base de forte volumétrie, ce paramètre sera augmenté systématiquement dans une proportion raisonnable. Pour une base de volumétrie normale, ce paramètre sera plutôt augmenté en ciblant une colonne particulière avec l'ordre SQL `ALTER TABLE … ALTER COLUMN … SET STATISTICS …;`. De plus, il est possible de forcer artificiellement le nombre de valeurs distinctes d'une colonne avec l'ordre SQL `ALTER TABLE … SET COLUMN … SET n_distinct = …;`. Il est aussi souvent utile d'envisager une réécriture de la requête : si l'optimiseur, sous Oracle comme sous PostgreSQL, n'arrive pas à trouver un bon plan, c'est probablement qu'elle est écrite d'une façon qui empêche ce dernier de travailler correctement.

Accès par ROWID

Dans de très rares cas, des requêtes SQL utilisent la colonne `ROWID` d'Oracle, par exemple pour dédoublonner des enregistrements. Le `ROWID` est la localisation physique d'une ligne

dans une table. L'équivalent dans PostgreSQL est le `ctid`.

Plus précisément, le `ROWID` Oracle représente une *adresse* logique d'une ligne, encodée sous la forme `OOOOOO.FFF.BBBBBB.RRR` où O représente le numéro d'objet, F le fichier, B le numéro de bloc et R la ligne dans le bloc. Le format est différent dans le cas d'une table stockée dans un BIG FILE TABLESPACE, mais le principe reste identique.

Quant au `ctid` de PostgreSQL, il ne représente qu'un couple *(numéro du bloc, numéro de l'enregistrement)*, aucune autre information de localisation physique n'est disponible. Le `ctid` n'est donc unique qu'au sein d'une table. De part ce fait, une requête ramenant le `ctid` des lignes d'une table partitionnée peut présenter des `ctid` en doublons. On peut dans ce cas utiliser le champ caché `tableoid` (l'identifiant unique de la table dans le catalogue) de chaque table pour différencier les doublons par partition.

Cette méthode d'accès est donc à proscrire, sauf opération particulière et cadrée.

1.4.7 CHOIX DE L'OUTIL

Suivant les réponses aux questions précédentes, vous choisirez :

- Ora2Pg
- Un ETL :
  - Kettle (Pentaho Data Integrator)
  - Talend
- De développer votre propre programme
- De mixer les solutions

Après avoir répondu aux questions précédentes et évalué la complexité de la migration, il sera possible de sélectionner le bon outil de migration. Il en existe différents, qui répondront différemment aux besoins.

Ora2Pg est un outil libre développé par Gilles Darold. Le rythme de développement est rapide. De nouvelles fonctionnalités sont proposées rapidement, suivant les demandes des utilisateurs, les nouveautés dans PostgreSQL et les découvertes réalisées par son auteur.

Les ETL sont intéressants pour les possibilités plus importantes. Ora2Pg ne fait que de la conversion Oracle vers PostgreSQL et MySQL, alors que les ETL autorisent un plus grand nombre de sources de données et de destinations, si bien que l'expérience acquise pour la migration d'Oracle vers PostgreSQL peut être réutilisée pour réaliser la migration d'un autre moteur vers un autre moteur ou pour l'import ou l'export de données.

Il est aussi possible de développer sa propre solution si les besoins sont vraiment spécifiques au métier, voire de mixer différentes solutions. Par exemple, il était intéressant d'utiliser Ora2Pg pour la transformation du schéma et un ETL pour un export et import des données parallélisés (ce n'est plus le cas maintenant qu'Ora2Pg est lui aussi parallélisé).

1.4.8 ORA2PG - INTRODUCTION

- En Perl
- Se connecte à Oracle
- Génère un fichier SQL compatible avec PostgreSQL, en optimisant les types
- Conversion automatique d'une partie du code PL/SQL en PL/pgSQL
- Simple de mise en œuvre
- Rapide au chargement (utilise `COPY`)

Ora2Pg est un outil écrit en Perl. Il se connecte à Oracle via le connecteur Perl pour Oracle. Après analyse des catalogues système Oracle et lecture de son fichier de configuration, il est capable de générer un fichier SQL compatible avec PostgreSQL ou de se connecter à une base PostgreSQL pour y exécuter ce script. Dans les dernières versions, il est même capable de convertir automatiquement une partie du code PL/SQL d'Oracle vers du PL/pgSQL sur PostgreSQL.

L'outil est plutôt simple de mise en œuvre et de prise en main. Il est rapide au chargement, notamment grâce à sa gestion de la commande `COPY`.

1.4.9 ORA2PG - DÉFAUTS

- Big-Bang
  - pas d'incrémental

Pour aborder immédiatement les inconvénients de Ora2Pg, il ne propose pas de solution incrémentale : c'est tout ou partie d'une table ou rien.

1.4.10 ORA2PG - FONCTIONNALITÉS

- Exporte tout le schéma Oracle :
  - tables, vues, séquences, contraintes d'intégrité, trigger, etc.
  - utilisateurs et droits
- Gère la conversion des types
  - `blob` et `clob` -> `bytea` et `text`
  - `number` -> `int`, `bigint`, `real`, `double`, `decimal`
- Réécrit les entêtes de fonction correspondant aux fonctions Oracle
- Aide à :
  - la conversion PL/SQL -> PL/pgSQL
  - au partitionnement (par héritage, ou déclaratif)

Ora2Pg dispose néanmoins de nombreuses fonctionnalités. Il est capable d'exporter tout le schéma de données Oracle. Il est capable de convertir utilisateurs et droits sur les objets. Il convertit aussi automatiquement la conversion des types de données. Enfin, il s'occupe de la déclaration et du code des routines stockées (uniquement PL/SQL vers PL/pgSQL). Il propose aussi une aide au partitionnement, dont l'implémentation est vraiment différente entre Oracle et PostgreSQL.

1.4.11 LES ETL - AVANTAGES

- Spécialisés dans la transformation et le chargement de données
- Rapides (cœur de métier)
- Parallélisables
- Très souples sur la transformation
- Migration incrémentale possible (fusion, *slow changing dimensions*, etc.)

Les ETL sont spécialisées dans la transformation et le chargement des données. Ils permettent la parallélisation pour leur traitement, ils sont très souples au niveau de la transformation de données. Tout cela leur permet d'être très rapide, quelques fois plus qu'Ora2Pg.

De plus, ils permettent de faire de la migration incrémentale.

1.4.12 LES ETL - INCONVÉNIENTS

- Migration sommaire du schéma
  - quand c'est supporté
- Beaucoup de travail de paramétrage
  - peut-être 200 jobs à créer si 200 tables…
- Apprentissage long
  - outil complexe et riche fonctionnellement

La migration du schéma est au mieux sommaire, voire inexistante. Ce n'est clairement pas la fonctionnalité visée par les ETL.

Le paramétrage d'un ETL est souvent très long. Si vous devez migrer les données de 200 tables, vous aurez 200 jobs à créer. Dans ce cas, Ora2Pg est bien plus intéressant, vu que la migration de la totalité des tables est l'option par défaut.

Ce sont des outils riches et donc complexes. Cela demandera un apprentissage bien plus long que pour Ora2Pg. Cependant, ils sont utilisables dans bien plus de cas qu'Ora2Pg.

1.5 INSTALLATION D'ORA2PG

Étapes :

- Téléchargement
- Pré-requis
- Compilation
- Installation
- Utilisation

Nous allons aborder dans cette première partie les différentes étapes à réaliser pour installer Ora2Pg à partir des sources :

- Où trouver les fichiers sources ?
- Quelle version choisir ?
- Comment préparer le serveur pour accueillir PostgreSQL ?
- Quelle procédure de compilation suivre ?
- Comment installer les fichiers compilés ?

1.5.1 TÉLÉCHARGEMENT

- Disponible via :
  - HTTP : https://ora2pg.darold.net/
  - Git : https://github.com/darold/ora2pg/releases
- Télécharger le fichier `ora2pg-X.Y.tar.bz2`
- Version au 12 octobre 2020 : 21.0

Les fichiers sources et les instructions de compilation sont accessibles depuis le site officiel du projet[7] .

Il est très important de toujours télécharger la dernière version car l'ajout de fonctionnalités et les corrections de bogues sont permanentes. En effet, ce projet bénéficie d'améliorations et de corrections au fur et à mesure des retours d'expérience des utilisateurs. Il est constamment mis à jour.

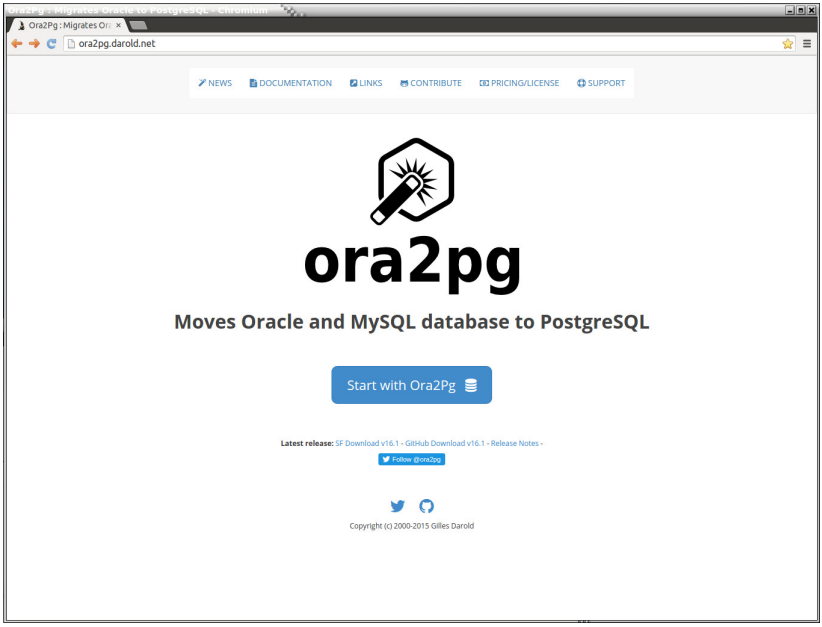
Les distributions officielles peuvent être téléchargées directement depuis SourceForge.net[8] , par exemple :

```
wget http://downloads.sourceforge.net/project/ora2pg/21.0/ora2pg-21.0.tar.bz2
```

Voici comment récupérer la version 21.0 des sources de Ora2Pg :

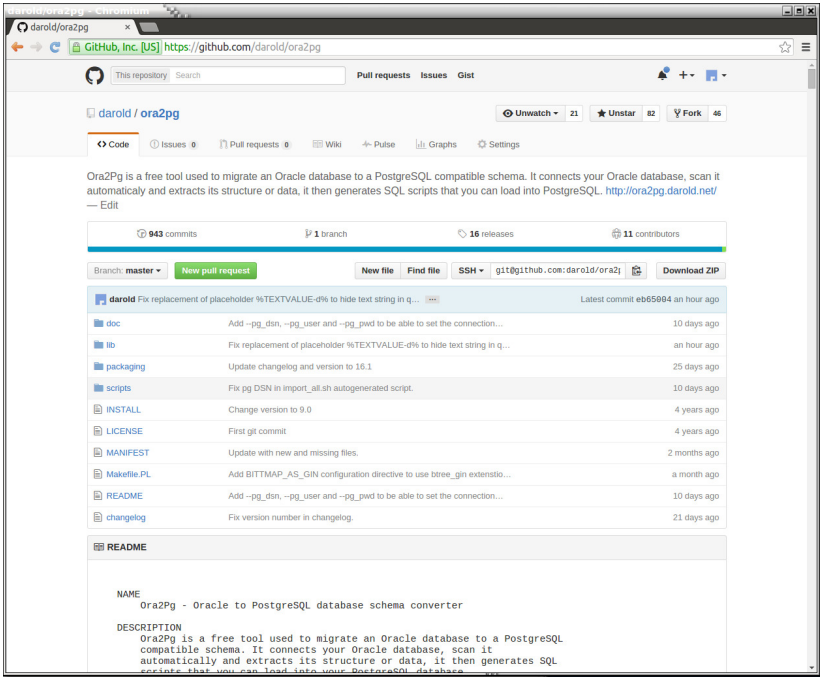
- Aller sur la page d'accueil du projet Ora2Pg et dans la section *Latest release* cliquer sur le lien *SF Download v21.0*.

[7] https://ora2pg.darold.net/
[8] https://sourceforge.net/projects/ora2pg/



- Vous arriverez sur la page Sourceforge du projet, cliquez sur le bouton *Download*.

Pour obtenir le dernier code en développement, il faut aller sur le dépôt GitHub (https://github.com/darold/ora2pg) du projet et cliquer sur le bouton *Download ZIP*.



Le fichier téléchargé sera nommé `ora2pg-master.zip`.

1.5.2 DÉPENDANCES REQUISES

- Oracle >= 8i client ou serveur
- Environnement Oracle correct (`ORACLE_HOME` et `PATH` comprenant `sqlplus`)
- `libaio1`
- Unix : Perl 5.10+
- Windows : Strawberry Perl 5.10+ ou ActiveStep Perl 5.10+
- Modules Perl
  - `Time::HiRes`
  - `Perl DBI > v1.614` et `DBD::Oracle`

Ora2Pg est entièrement codé en `Perl`.

Ora2Pg se connecte à Oracle grâce à l'interface de bases de données pour `Perl`, appelée `DBI`.

Dans cette interface, Ora2Pg va utiliser le connecteur Oracle, appelé `DBD::Oracle` (DBD, acronyme de *DataBase Driver*). Tous les modules Perl, s'ils ne sont pas disponibles en paquet pour votre distribution, peuvent toujours être téléchargés à partir du site CPAN (http://search.cpan.org/). Il suffit de saisir le nom du module (ex : `DBD::Oracle`) dans la case de recherche et la page de téléchargement du module vous sera proposée.

Le client lourd d'Oracle est nécessaire pour utiliser la couche `OCI`. Cependant, les nouvelles versions *Instant Client* (à partir de la version 10g) suffisent amplement à Ora2Pg. Attention toutefois, s'il est possible d'utiliser un client Oracle 12c pour se connecter à des bases Oracle de versions inférieures, l'inverse n'est pas vrai.

Ainsi il convient d'installer au minimum un client Oracle, comme `instantclient-basic`, `instantclient-sdk` ou `instantclient-sqlplus`. Pour que `sqlplus` puisse fonctionner correctement il faut au préalable installer la librairie `libaio1`. Cette bibliothèque permet aux applications en espace utilisateur d'utiliser les appels système asynchrones d'E/S du noyau Linux.

`DBD::Oracle` va s'appuyer sur les variables d'environnement pour déterminer où se trouvent les bibliothèques d'Oracle.

Dans le monde Oracle, ces variables d'environnement sont très connues (`ORACLE_BASE`, `ORACLE_HOME`, `NLS_LANG`, etc.). Pour `DBD::Oracle`, le positionnement de la variable `ORACLE_HOME` suffit.

```
export ORACLE_HOME=/usr/lib/oracle/x.x/client64
```

Pour une installation sous Windows, l'utilisation de Strawberry Perl[9] nécessitera les outils de compilation pour l'installation de `DBD::Oracle` alors que l'utilisation de la distribution libre d'ActiveState[10] permet d'installer directement une version binaire de la bibliothèque.

Pour les anciennes versions de Perl ou certaines distributions il peut être nécessaire d'installer le module Perl, `Time::HiRes`.

---

[9] http://strawberryperl.com/
[10] https://www.activestate.com/activeperl/downloads

1.5.3 DÉPENDANCES OPTIONNELLES

En option :

- PostgreSQL >= 8.4 client ou serveur
- `DBD::Pg` pour l'import direct dans PostgreSQL
- `Compress::Zlib` : compression des fichiers en sortie
- `DBD::MySQL` pour migrer les bases MySQL

Le connecteur PostgreSQL pour `DBI`, `DBD::Pg` est nécessaire uniquement si l'on veut migrer directement les données depuis Oracle vers PostgreSQL sans avoir à passer par des fichiers intermédiaires. `DBD::Pg` nécessite au minimum les bibliothèques du client PostgreSQL.

On peut se passer de ce module dans la mesure où, par défaut, Ora2Pg va écrire les objets et données à migrer dans des fichiers. Ces fichiers peuvent alors être chargés à l'aide de la commande `psql` ou être transférés sur une autre machine disposant de cet outil.

La bibliothèque `Perl Compress::Zlib` est nécessaire si vous souhaitez que les fichiers de sortie soient compressés avec `gzip`. C'est notamment très utile pour les fichiers de données volumineux par exemple.

Fort heureusement, on peut aussi utiliser le binaire `bzip2` pour compresser le fichier de sortie. Dans ce cas, il suffit d'indiquer, dans le fichier de configuration d'Ora2Pg, l'emplacement du binaire `bzip2` sur le système si celui-ci n'est pas dans le `PATH`.

Ora2Pg, depuis la version 16.0, permet de migrer les bases de données MySQL. Tout comme pour Oracle, Ora2Pg a besoin de se connecter à l'instance MySQL au travers d'un driver Perl. C'est le rôle du module Perl `DBD::MySQL`.

1.5.4 COMPILATION ET INSTALLATION

- Décompresser l'archive téléchargée
- Générer les fichiers de compilation
- Compiler et installer
- Ora2Pg est prêt à être configuré !

Voici les lignes de commande à saisir pour compiler et installer Ora2Pg :

```
tar xjf ora2pg-21.0.tar.bz2
cd ora2pg-21.0/
perl Makefile.PL
make && sudo make install
```

Sous Windows, il faut remplacer la dernière ligne par la ligne suivante :

```
dmake && dmake install
```

`dmake` est l'équivalent de `make` pour Windows, il peut être téléchargé depuis cette URL : http://search.cpan.org/dist/dmake/. Téléchargez-le et installez `dmake` quelque part dans le `PATH` Windows.

Le fichier de configuration d'Ora2Pg par défaut est `/etc/ora2pg/ora2pg.conf` sous Unix et `C:\ora2pg\ora2pg.conf` sous Windows.

L'installation provoquera la création d'un modèle de fichier de configuration : `ora2pg.conf.dist`, avec toutes les variables définies par défaut. Il suffira de le renommer en `ora2pg.conf` et de le modifier pour obtenir le comportement souhaité.

```
cp /etc/ora2pg/ora2pg.conf.dist /etc/ora2pg/ora2pg.conf
```

Les paramètres de ce fichier sont explicités de manière exhaustive plus loin dans la formation.

1.5.5 USAGE DE LA COMMANDE ORA2PG

Le script `ora2pg` s'utilise de la façon suivante :

```
ora2pg [-dhpqv --estimate_cost --dump_as_html] [--option value]
```

Utilisation basique :

```
ora2pg -t ACTION [-c fichier_de_configuration]
```

Certaines options ne nécessitent pas de valeurs et ne sont introduites dans la ligne de commande que pour activer certains comportements. C'est le cas notamment de l'option `-d` ou `--debug` permettant d'activer le mode trace.

Toutes les options courtes ont une version longue, par exemple `-q` et `--quiet`.

Voici l'intégralité des options disponibles en ligne de commande pour le script Perl `ora2pg` et leur explication :

```
    -a | --allow str  : Comma separated list of objects to allow from export.
            Can be used with SHOW_COLUMN too.
    -b | --basedir dir: Set the default output directory, where files
            resulting from exports will be stored.
    -c | --conf file  : Set an alternate configuration file other than the
            default /etc/ora2pg/ora2pg.conf.
    -d | --debug      : Enable verbose output.
    -D | --data_type STR : Allow custom type replacement at command line.
    -e | --exclude str: Comma separated list of objects to exclude from export.
```

```
        Can be used with SHOW_COLUMN too.
 -h | --help       : Print this short help.
 -g | --grant_object type : Extract privilege from the given object type.
        See possible values with GRANT_OBJECT configuration.
 -i | --input file : File containing Oracle PL/SQL code to convert with
        no Oracle database connection initiated.
 -j | --jobs num   : Number of parallel process to send data to PostgreSQL.
 -J | --copies num : Number of parallel connections to extract data from Oracle.
 -l | --log file   : Set a log file. Default is stdout.
 -L | --limit num  : Number of tuples extracted from Oracle and stored in
        memory before writing, default: 10000.
 -m | --mysql      : Export a MySQL database instead of an Oracle schema.
 -n | --namespace schema : Set the Oracle schema to extract from.
 -N | --pg_schema schema : Set PostgreSQL's search_path.
 -o | --out file   : Set the path to the output file where SQL will
        be written. Default: output.sql in running directory.
 -p | --plsql      : Enable PLSQL to PLPGSQL code conversion.
 -P | --parallel num: Number of parallel tables to extract at the same time.
 -q | --quiet      : Disable progress bar.
 -r | --relative   : use \ir instead of \i in the psql scripts generated.
 -s | --source DSN : Allow to set the Oracle DBI datasource.
 -t | --type export: Set the export type. It will override the one
        given in the configuration file (TYPE).
 -T | --temp_dir DIR: Set a distinct temporary directory when two
                      or more ora2pg are run in parallel.
 -u | --user name  : Set the Oracle database connection user.
            ORA2PG_USER environment variable can be used instead.
 -v | --version    : Show Ora2Pg Version and exit.
 -w | --password pwd : Set the password of the Oracle database user.
            ORA2PG_PASSWD environment variable can be used instead.
 --forceowner      : Force ora2pg to set tables and sequences owner like in
       Oracle database. If the value is set to a username this one
       will be used as the objects owner. By default it's the user
       used to connect to the Pg database that will be the owner.
 --nls_lang code: Set the Oracle NLS_LANG client encoding.
 --client_encoding code: Set the PostgreSQL client encoding.
 --view_as_table str: Comma separated list of views to export as table.
 --estimate_cost   : Activate the migration cost evaluation with SHOW_REPORT
 --cost_unit_value minutes: Number of minutes for a cost evaluation unit.
       default: 5 minutes, corresponds to a migration conducted by a
       PostgreSQL expert. Set it to 10 if this is your first migration.
--dump_as_html     : Force ora2pg to dump report in HTML, used only with
                     SHOW_REPORT. Default is to dump report as simple text.
--dump_as_csv      : As above but force ora2pg to dump report in CSV.
--dump_as_sheet    : Report migration assessment with one CSV line per database.
--init_project NAME: Initialise a typical ora2pg project tree. Top directory
```

```
                    will be created under project base dir.
--project_base DIR : Define the base dir for ora2pg project trees. Default
                    is current directory.
--print_header     : Used with --dump_as_sheet to print the CSV header
                    especially for the first run of ora2pg.
--human_days_limit num : Set the number of human-days limit where the migration
                    assessment level switch from B to C. Default is set to
                    5 human-days.
--audit_user LIST  : Comma separated list of usernames to filter queries in
                    the DBA_AUDIT_TRAIL table. Used only with SHOW_REPORT
                    and QUERY export type.
--pg_dsn DSN       : Set the datasource to PostgreSQL for direct import.
--pg_user name     : Set the PostgreSQL user to use.
--pg_pwd password  : Set the PostgreSQL password to use.
--count_rows       : Force ora2pg to perform a real row count in TEST action.
--no_header        : Do not append Ora2Pg header to output file
--oracle_speed     : Use to know at which speed Oracle is able to send
                    data. No data will be processed or written.
--ora2pg_speed     : Use to know at which speed Ora2Pg is able to send
                    transformed data. Nothing will be written.
```

1.6 CONCLUSION

Points essentiels :

- Grande importance de la première migration
- Même si Oracle et PostgreSQL sont assez similaires, il y a de nombreuses différences
- Étude de la migration, ce qui doit ou pas être migré et comment
- Choix des outils de migration
- La majorité du temps de migration est imputable à la conversion du PL/SQL

1.6.1 QUESTIONS

- N'hésitez pas, c'est le moment !

1.7 QUIZ

- **https://dali.bo/n1_quiz**

1.8 INSTALLATION DE POSTGRESQL DEPUIS LES PAQUETS COMMUNAUTAIRES

L'installation est détaillée ici pour Rocky Linux 8 (similaire à Red Hat 8), Red Hat/CentOS 7, et Debian/Ubuntu.

Elle ne dure que quelques minutes.

1.8.1 SUR ROCKY LINUX 8

Installation du dépôt communautaire :

Sauf précision, tout est à effectuer en tant qu'utilisateur **root**.

Les dépôts de la communauté sont sur https://yum.postgresql.org/. Les commandes qui suivent peuvent être générées par l'assistant sur https://www.postgresql.org/download/linux/redhat/, en précisant :

- la version majeure de PostgreSQL (ici la 14) ;
- la distribution (ici Rocky Linux 8) ;
- l'architecture (ici x86_64, la plus courante).

Il faut installer le dépôt et désactiver le module PostgreSQL par défaut :

```
# dnf install -y https://download.postgresql.org/pub/repos/yum/reporpms\
/EL-8-x86_64/pgdg-redhat-repo-latest.noarch.rpm

# dnf -qy module disable postgresql
```

Installation de PostgreSQL 14 :

```
# dnf install -y postgresql14-server postgresql14-contrib
```

Les outils clients et les librairies nécessaires seront automatiquement installés.

Tout à fait optionnellement, une fonctionnalité avancée, le JIT (*Just In Time compilation*), nécessite un paquet séparé, qui lui-même nécessite des paquets du dépôt EPEL :

```
# dnf install postgresql14-llvmjit
```

Création d'une première instance :

Il est conseillé de déclarer `PG_SETUP_INITDB_OPTIONS`, notamment pour mettre en place les sommes de contrôle et forcer les traces en anglais :

```
# export PGSETUP_INITDB_OPTIONS='--data-checksums --lc-messages=C'
# /usr/pgsql-14/bin/postgresql-14-setup initdb
# cat /var/lib/pgsql/14/initdb.log
```

Ce dernier fichier permet de vérifier que tout s'est bien passé.

Chemins :

Objet	Chemin
Binaires	`/usr/pgsql-14/bin`
Répertoire de l'utilisateur **postgres**	`/var/lib/pgsql`
`PGDATA` par défaut	`/var/lib/pgsql/14/data`
Fichiers de configuration	dans `PGDATA/`
Traces	dans `PGDATA/log`

Configuration :

Modifier `postgresql.conf` est facultatif pour un premier essai.

Démarrage/arrêt de l'instance, rechargement de configuration :

```
# systemctl start postgresql-14
# systemctl stop postgresql-14
# systemctl reload postgresql-14
```

Test rapide de bon fonctionnement

```
# systemctl --all |grep postgres
# sudo -iu postgres psql
```

Démarrage de l'instance au démarrage du système d'exploitation :

```
# systemctl enable postgresql-14
```

Consultation de l'état de l'instance :

```
# systemctl status postgresql-14
```

Ouverture du *firewall* pour le port 5432 :

Si le *firewall* est actif (dans le doute, consulter `systemctl status firewalld`) :

```
# firewall-cmd --zone=public --add-port=5432/tcp --permanent
# firewall-cmd --reload
# firewall-cmd --list-all
```

Création d'autres instances :

Si des instances de *versions majeures différentes* doivent être installées, il faudra installer les binaires pour chacune, et l'instance par défaut de chaque version vivra dans un sous-répertoire différent de `/var/lib/pgsql` automatiquement créé à l'installation. Il faudra juste modifier les ports dans les `postgresql.conf`.

Si plusieurs instances d'une même version majeure (forcément de la même version mineure) doivent cohabiter sur le même serveur, il faudra les installer dans des `PGDATA` différents.

- Ne pas utiliser de tiret dans le nom d'une instance (problèmes potentiels avec systemd).
- Respecter les normes et conventions de l'OS : placer les instances dans un sous-répertoire de `/var/lib/pgsqsl/14/` (ou l'équivalent pour d'autres versions majeures).
- Création du fichier service de la deuxième instance :

```
# cp /lib/systemd/system/postgresql-14.service \
       /etc/systemd/system/postgresql-14-secondaire.service
```

- Modification du fichier avec le nouveau chemin :

```
Environment=PGDATA=/var/lib/pgsql/14/secondaire
```

- Option 1 : création d'une nouvelle instance vierge :

```
# /usr/pgsql-14/bin/postgresql-14-setup initdb postgresql-14-secondaire
```

- Option 2 : restauration d'une sauvegarde : la procédure dépend de votre outil.
- Adaptation de `postgresql.conf` (port !), `recovery.conf`...
- Commandes de maintenance :

```
# systemctl [start|stop|reload|status] postgresql-14-secondaire
# systemctl [enable|disable] postgresql-14-secondaire
```

- Ouvrir un port dans le firewall au besoin.

1.8.2 SUR RED HAT 7 / CENT OS 7

Fondamentalement, le principe reste le même qu'en version 8. Il faudra utiliser `yum` plutôt que `dnf`.

ATTENTION : Red Hat et CentOS 6 et 7 fournissent par défaut des versions de PostgreSQL qui ne sont plus supportées. Ne jamais installer les packages `postgresql`, `postgresql-client` et `postgresql-server` !
L'utilisation des dépôts du PGDG est donc obligatoire.

Il n'y a pas besoin de désactiver de module AppStream.

Le JIT (*Just In Time compilation*), nécessite un paquet séparé, qui lui-même nécessite des paquets du dépôt EPEL :

```
# yum install epel-release
# yum install postgresql14-llvmjit
```

La création de l'instance et la suite sont identiques.

1.8.3 SUR DEBIAN / UBUNTU

Sauf précision, tout est à effectuer en tant qu'utilisateur **root**.

Installation du dépôt communautaire :

Référence : https://apt.postgresql.org/

- Import des certificats et de la clé :

```
# apt install curl ca-certificates gnupg
# curl https://www.postgresql.org/media/keys/ACCC4CF8.asc | apt-key add -
```

- Création du fichier du dépôt `/etc/apt/sources.list.d/pgdg.list` (ici pour Debian 11 « bullseye » ; adapter au nom de code de la version de Debian ou Ubuntu correspondante : **stretch**, **bionic**, **focal**...) :

```
deb http://apt.postgresql.org/pub/repos/apt bullseye-pgdg main
```

Installation de PostgreSQL 14 :

La méthode la plus propre consiste à modifier la configuration par défaut avant l'installation :

```
# apt update
# apt install postgresql-common
```

Dans `/etc/postgresql-common/createcluster.conf`, paramétrer au moins les sommes de contrôle et les traces en anglais :

```
initdb_options = '--data-checksums --lc-messages=C'
```

Puis installer les paquets serveur et clients et leurs dépendances :

```
# apt install postgresql-14 postgresql-client-14
```

(Pour les versions 9.x, installer aussi le paquet `postgresql-contrib-9.x`).

La première instance est automatiquement créée, démarrée et déclarée comme service à lancer au démarrage du système. Elle est immédiatement accessible par l'utilisateur système **postgres**.

Chemins :

Objet	Chemin

Binaires	`/usr/lib/postgresql/14/bin/`
Répertoire de l'utilisateur **postgres**	`/var/lib/postgresql`
PGDATA de l'instance par défaut	`/var/lib/postgresql/14/main`
Fichiers de configuration	dans `/etc/postgresql/14/main/`
Traces	dans `/var/log/postgresql/`

Configuration

Modifier `postgresql.conf` est facultatif pour un premier essai.

Démarrage/arrêt de l'instance, rechargement de configuration :

Debian fournit ses propres outils :

```
# pg_ctlcluster 14 main [start|stop|reload|status]
```

Démarrage de l'instance au lancement :

C'est en place par défaut, et modifiable dans `/etc/postgresql/14/main/start.conf`.

Ouverture du firewall :

Debian et Ubuntu n'installent pas de firewall par défaut.

Statut des instances :

```
# pg_lsclusters
```

Test rapide de bon fonctionnement

```
# systemctl --all |grep postgres
# sudo -iu postgres psql
```

Destruction d'une instance :

```
# pg_dropcluster 14 main
```

Création d'autres instances :

Ce qui suit est valable pour remplacer l'instance par défaut par une autre, par exemple pour mettre les *checksums* en place :

- les paramètres de création d'instance dans `/etc/postgresql-common/createcluster.conf` peuvent être modifiés, par exemple ici pour : les *checksums*, les messages en anglais, l'authentification sécurisée, le format des traces et un emplacement séparé pour les journaux :

```
initdb_options = '--data-checksums --lc-messages=C --auth-host=scram-sha-256 --auth-local=peer'
log_line_prefix = '%t [%p]: [%l-1] user=%u,db=%d,app=%a,client=%h '
waldir = '/var/lib/postgresql/wal/%v/%c/pg_wal'
```

- création de l'instance, avec possibilité là aussi de préciser certains paramètres du `postgresql.conf` voire de modifier les chemins des fichiers (déconseillé si vous pouvez l'éviter) :

```
# pg_createcluster 14 secondaire \
  --port=5433 \
  --datadir=/PGDATA/11/basedecisionnelle \
  --pgoption shared_buffers='8GB' --pgoption work_mem='50MB' \
  --  --data-checksums --waldir=/ssd/postgresql/11/basedecisionnelle/journaux
```

- démarrage :

```
# pg_ctlcluster 14 secondaire start
```

1.8.4 ACCÈS À L'INSTANCE

Par défaut, l'instance n'est accessible que par l'utilisateur système `postgres`, qui n'a pas de mot de passe. Un détour par `sudo` est nécessaire :

```
$ sudo -iu postgres psql
psql (14.0)
Saisissez « help » pour l'aide.
postgres=#
```

Ce qui suit permet la connexion directement depuis un utilisateur du système :

Pour des tests (pas en production !), il suffit de passer à `trust` le type de la connexion en local dans le `pg_hba.conf` :

```
local   all             postgres                                trust
```

La connexion en tant qu'utilisateur `postgres` (ou tout autre) n'est alors plus sécurisée :

```
dalibo:~$ psql -U postgres
psql (14.0)
Saisissez « help » pour l'aide.
postgres=#
```

Une authentification par mot de passe est plus sécurisée :

- dans `pg_hba.conf`, mise en place d'une authentification par mot de passe (`md5` par défaut) pour les accès à `localhost` :

```
# IPv4 local connections:
host    all             all             127.0.0.1/32            md5
```

```
# IPv6 local connections:
host    all             all             ::1/128                 md5
```

(une authentification `scram-sha-256` est plus conseillée mais elle impose que `password_encryption` soit à cette valeur dans `postgresql.conf` avant de définir les mots de passe).

- ajout d'un mot de passe à l'utilisateur `postgres` de l'instance ;

```
dalibo:~$ sudo -iu postgres psql
psql (14.0)
Saisissez « help » pour l'aide.
postgres=# \password
Saisissez le nouveau mot de passe :
Saisissez-le à nouveau :
postgres=# \q

dalibo:~$ psql -h localhost -U postgres
Mot de passe pour l'utilisateur postgres :
psql (14.0)
Saisissez « help » pour l'aide.
postgres=#
```

- pour se connecter sans taper le mot de passe, un fichier `.pgpass` dans le répertoire personnel doit contenir les informations sur cette connexion :

```
localhost:5432:*:postgres:motdepassetrèslong
```

- ce fichier doit être protégé des autres utilisateurs :

```
$ chmod 600 ~/.pgpass
```

- pour n'avoir à taper que `psql`, on peut définir ces variables d'environnement dans la session voire dans `~/.bashrc` :

```
export PGUSER=postgres
export PGDATABASE=postgres
export PGHOST=localhost
```

Rappels :

- en cas de problème, consulter les traces (dans `/var/lib/pgsql/14/data/log` ou `/var/log/postgresql/`) ;
- toute modification de `pg_hba.conf` implique de recharger la configuration par une de ces trois méthodes selon le système :

```
root:~# systemctl reload postgresql-14

root:~# pg_ctlcluster 14 main reload
```

```
postgres:~$ psql -c 'SELECT pg_reload_conf();'
```

1.9 TRAVAUX PRATIQUES

Environnement

> Installer les outils de compilation `gcc`, `make` et `perl-devel`.

Installation d'Oracle client

Requis si aucune instance Oracle Database n'est présente sur la machine.

> Installer le client Oracle à l'aide des fichiers `.rpm` de la dernière version en ligne sur http://www.oracle.com[a] .
>
> [a] https://www.oracle.com/database/technologies/instant-client/downloads.html

Installation du driver DBI pour Oracle

> Installer `DBD::Oracle`.

Installation du driver DBI pour PostgreSQL

> Installer `DBD::Pg`.

Téléchargement d'Ora2Pg

> Consulter le site officiel du projet et relevez la dernière version d'Ora2Pg.
> Télécharger les fichiers sources de la dernière version et les placer dans `/opt/ora2pg/src`.

Compilation et installation d'Ora2Pg

> Compiler et installer Ora2Pg.

1.10 TRAVAUX PRATIQUES (SOLUTIONS)

Les opérations d'installation de paquets (`yum` ou `dnf`, `make install`) sont à exécuter en tant que **root**. Le reste, dont les étapes de compilation, sont à effectuer avec un utilisateur normal.

Ce qui suit fonctionne sur CentOS 7. Sur Rocky Linux 8, remplacer `yum` par `dnf`.

Environnement

> Installer les outils de compilation `gcc`, `make` et `perl-devel`.

S'ils ne sont pas présents, installer les outils de compilation :

```
# yum install -y gcc make perl perl-devel perl-DBI cpan
```

Installation d'Oracle client

Requis si aucune instance Oracle Database n'est présente sur la machine.

> Installer le client Oracle à l'aide des fichiers `.rpm` de la dernière version en ligne sur http://www.oracle.com[a] .
>
> [a] https://www.oracle.com/database/technologies/instant-client/downloads.html

La version de l'*Instant Client* doit être au moins v12 pour accéder aux bases en 12c ; au plus en v10 pour accéder aux bases Oracle 8 ou 9.

Voici les archives à télécharger pour la version 19c, en architecture 64 bits (adapter le numéro de version au besoin) :

```
export D=https://download.oracle.com/otn_software/linux/instantclient/1915000
wget $D/oracle-instantclient19.15-basic-19.15.0.0.0-1.x86_64.rpm
wget $D/oracle-instantclient19.15-devel-19.15.0.0.0-1.x86_64.rpm
wget $D/oracle-instantclient19.15-sqlplus-19.15.0.0.0-1.x86_64.rpm
```

Installer ensuite les paquets téléchargés :

```
# yum install -y oracle-instantclient19.15-basic-19.15.0.0.0-1.x86_64.rpm \
                 oracle-instantclient19.15-devel-19.15.0.0.0-1.x86_64.rpm \
                 oracle-instantclient19.15-sqlplus-19.15.0.0.0-1.x86_64.rpm
```

Installation du driver DBI pour Oracle

> Installer `DBD::Oracle`.

Le driver Oracle pour Perl DBI peut être compilé à partir du module de recherche du site CPAN : http://search.cpan.org/search?query=DBD::Oracle

L'utilitaire `cpan` permet de réaliser automatiquement les opérations de téléchargement, de contrôle des dépendances et de compilation de l'ensemble des modules Perl proposés par la communauté. Il est à privilégier pour la gestion des mises à jour de ces modules.

Voici la procédure d'installation complète de la dernière version du driver `DBD::Oracle` à suivre avec l'utilisateur **root** :

- Avec les librairies *Instant Client*(adapter éventuellement le numéro de version) :

```
# export ORACLE_HOME=/usr/lib/oracle/19.15/client64
# export LD_LIBRARY_PATH=$ORACLE_HOME/lib
```

- Avec les librairies *Oracle Database* :

```
# export ORACLE_HOME=/opt/oracle/product/21c/dbhomeXE
# export LD_LIBRARY_PATH=$ORACLE_HOME/lib
```

- Installation du driver

Pour un serveur Rocky Linux 8, il est nécessaire d'activer les `PowerTools`.

```
# dnf config-manager --set-enabled powertools
```

La compilation nécessite l'installation de paquets supplémentaires concernant les tests effectués durant la compilation :

```
# yum install -y perl-Test-NoWarnings perl-Test-Simple libaio-devel
# cpan install DBD::Oracle
```

Installation du driver DBI pour PostgreSQL

> Installer `DBD::Pg`.

Le driver PostgreSQL pour Perl DBI peut être trouvé sur le CPAN[11] , mais privilégier l'installation par paquet :

```
# yum install -y perl-DBD-Pg
```

Téléchargement d'Ora2Pg

> Consulter le site officiel du projet et relevez la dernière version d'Ora2Pg.
> Télécharger les fichiers sources de la dernière version

[11] http://search.cpan.org/search?query=DBD::Pg

> et les placer dans `/opt/ora2pg/src`.

Consulter le site officiel du projet https://ora2pg.darold.net/, la page *News* indique la dernière version d'Ora2Pg. Le téléchargement des fichiers source de la dernière version officielle se fait depuis le dépôt Github[12] :

```
# mkdir -p /opt/ora2pg/src
# cd /opt/ora2pg/src/
# wget https://github.com/darold/ora2pg/archive/refs/tags/v23.1.tar.gz
# tar xvf v23.1.tar.gz
# cd ora2pg-23.1
```

La version **master** de Github est celle en développement et peut être éventuellement nécessaire.

Compilation et installation d'Ora2Pg

> Compiler et installer Ora2Pg.

Les instructions peuvent être exécutées avec le compte **root**, notamment la commande `make install` pour déployer les binaires dans les répertoires systèmes.

```
# unset PERL_MM_OPT
# perl Makefile.PL
# make && make install
```

La variable d'environnement `PERL_MM_OPT` peut parfois surchager le répertoire d'installation `INSTALL_BASE` prévu dans le `Makefile.PL`. Pour que les binaires et les librairies soient accessibles pour tous les utilisateurs, vérifier que cette variable n'a aucune valeur.

[12] https://github.com/darold/ora2pg/releases

NOTES

NOTES

NOTES

NOTES

NOTES

NOS AUTRES PUBLICATIONS

FORMATIONS

- **DBA1 : Administration PostgreSQL**
  https://dali.bo/dba1
- **DBA2 : Administration PostgreSQL avancé**
  https://dali.bo/dba2
- **DBA3 : Sauvegarde et réplication avec PostgreSQL**
  https://dali.bo/dba3
- **DEVPG : Développer avec PostgreSQL**
  https://dali.bo/devpg
- **PERF1 : PostgreSQL Performances**
  https://dali.bo/perf1
- **PERF2 : Indexation et SQL avancés**
  https://dali.bo/perf2
- **MIGORPG : Migrer d'Oracle à PostgreSQL**
  https://dali.bo/migorpg
- **HAPAT : Haute disponibilité avec PostgreSQL**
  https://dali.bo/hapat

LIVRES BLANCS

- **Migrer d'Oracle à PostgreSQL**
- **Industrialiser PostgreSQL**
- **Bonnes pratiques de modélisation avec PostgreSQL**
- **Bonnes pratiques de développement avec PostgreSQL**

TÉLÉCHARGEMENT GRATUIT

Les versions électroniques de nos publications sont disponibles gratuitement sous licence open-source ou sous licence Creative Commons. Contactez-nous à l'adresse contact@dalibo.com pour plus d'information.

DALIBO, L'EXPERTISE POSTGRESQL

Depuis 2005, DALIBO met à la disposition de ses clients son savoir-faire dans le domaine des bases de données et propose des services de conseil, de formation et de support aux entreprises et aux institutionnels.

En parallèle de son activité commerciale, DALIBO contribue aux développements de la communauté PostgreSQL et participe activement à l'animation de la communauté francophone de PostgreSQL. La société est également à l'origine de nombreux outils libres de supervision, de migration, de sauvegarde et d'optimisation.

Le succès de PostgreSQL démontre que la transparence, l'ouverture et l'auto-gestion sont à la fois une source d'innovation et un gage de pérennité. DALIBO a intégré ces principes dans son ADN en optant pour le statut de SCOP : la société est contrôlée à 100 % par ses salariés, les décisions sont prises collectivement et les bénéfices sont partagés à parts égales.