Module N1

Plan de migration

24.04

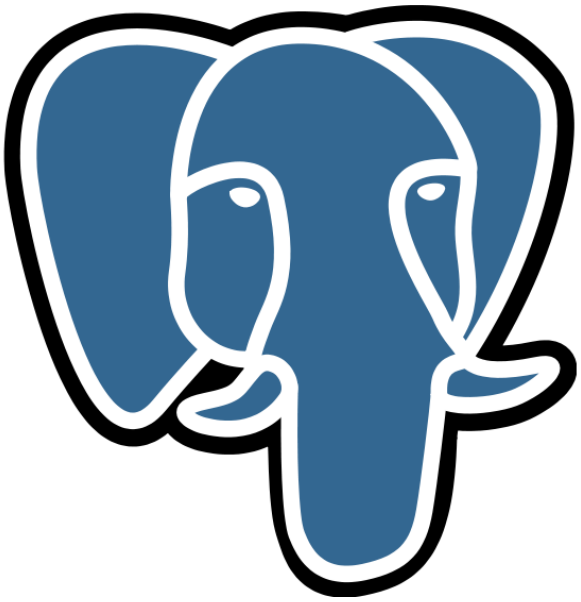
Table des matières

Sur ce document

Formation	Module N1
Titre	Plan de migration
Révision	24.04
PDF	https://dali.bo/n1_pdf
EPUB	https://dali.bo/n1_epub
HTML	https://dali.bo/n1_html
Slides	https://dali.bo/n1_slides
TP	https://dali.bo/n1_tp
TP (solutions)	https://dali.bo/n1_solutions

Vous trouverez en ligne les différentes versions complètes de ce document.

1/ Plan de migration



1.1 INTRODUCTION

Ce module est organisé en quatre parties :

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

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 permettre la génération d'un premier rapport d'évaluation.

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[1] pour que nous puissions améliorer ce support de formation.

[1] mailto:formation@dalibo.com

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 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 accessible 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 impliquées 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[2]
- Materialized Views that Really Work[3]

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[4] ainsi que le document Partitionnement (module DBA2)[5].

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.

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*

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

L'optimiseur Oracle supporte des *hints*, qui permettent au DBA de tromper l'optimiseur pour lui faire prendre des chemins que l'optimiseur a jugés 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, avec des règles avancées de reprise d'étapes en erreur ou de conditions à remplir pour déclencher d'autres actions (comme la création des index ou l'activation des contraintes).

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 réalise 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 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

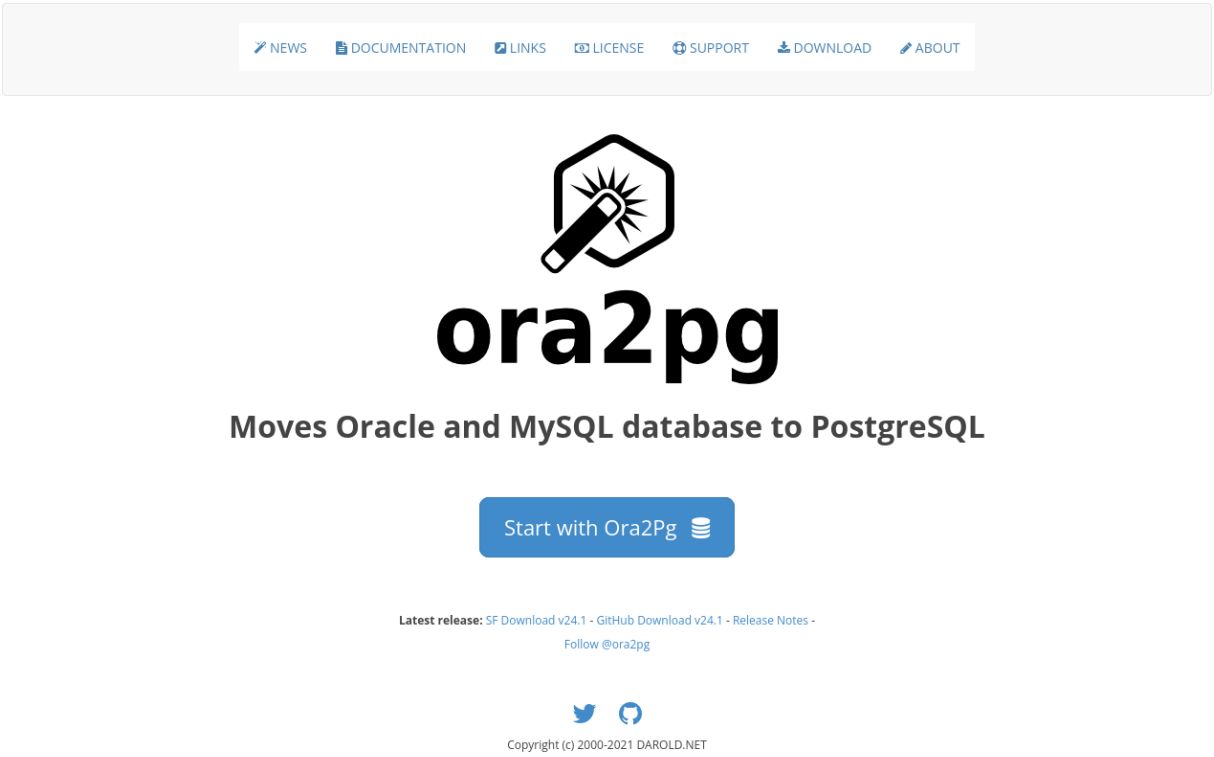
- Disponible via :
  - HTTP : https://ora2pg.darold.net/
  - Git : https://github.com/darold/ora2pg/releases
  - SourceForge : https://sourceforge.net/projects/ora2pg
- Dernière version : 24.1 (8 septembre 2023)

Les fichiers sources et les instructions de compilation sont accessibles depuis le site officiel du projet[6].

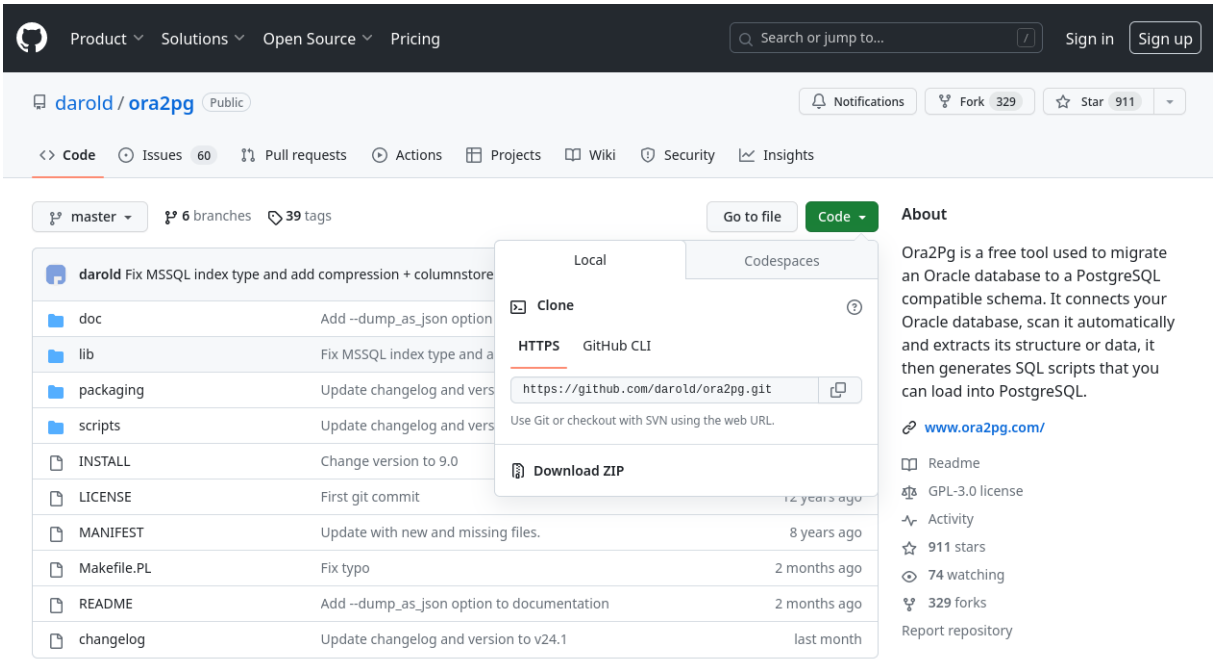
Il est très important de toujours télécharger la dernière version, car l'ajout de fonctionnalités et les corrections de bogues sont permanentes. En effet, ce projet bénéficie d'améliorations et de corrections au fur et à mesure des retours d'expérience des utilisateurs. Il est constamment mis à jour.

La dernière distribution officielle peut être téléchargée directement depuis GitHub.com[7], où la dernière version est disponible aux formats **zip** ou **tar.gz**.

[6] https://ora2pg.darold.net/
[7] https://github.com/darold/ora2pg/releases/



Pour obtenir le dernier code en développement, il faut aller sur la page du dépôt GitHub (https://github.com/darold/ora2pg) et cliquer sur le bouton *Download ZIP* de la branche de développement. 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`)
- `libaio` (Redhat) ou `libaio1` (Debian)
- 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 `libaio` . 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[8] nécessitera les outils de compilation pour l'installation de `DBD::Oracle` alors que l'utilisation de la distribution libre d'ActiveState[9]

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

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`.

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
- `DBD::ODBC` pour migrer les bases Microsoft SQL Server

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 permet de migrer les bases de données MySQL depuis la version 16.0 et les bases de données SQL Server depuis la version 24.0. Tout comme pour Oracle, Ora2Pg a besoin de se connecter à l'instance distante au travers d'un pilote Perl. C'est le rôle des modules Perl `DBD::MySQL` ou `DBD::ODBC`.

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.

1.6 ÉVALER UNE MIGRATION

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.
-C | --cdc_file file: File used to store/read SCN per table during export.
                      default: TABLES_SCN.log in the current directory. This
                      is the file written by the --cdc_ready option.
-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.
-M | --mssql      : Export a Microsoft SQL Server database.
-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.
  -S | --scn    SCN : Allow to set the Oracle System Change Number (SCN) to
                      use to export data. It will be used in the WHERE clause
                      to get the data. It is used with action COPY or INSERT.
  -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.
  -W | --where clause : Set the WHERE clause to apply to the Oracle query to
                      retrieve data. Can be used multiple time.
  --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_json     : As above but force ora2pg to dump report in JSON.
--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,
                      TEST_COUNT and SHOW_TABLE actions.
--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.
--blob_to_lo       : export BLOB as large objects, can only be used with
```

```
                      action SHOW_COLUMN, TABLE and INSERT.
   --cdc_ready        : use current SCN per table to export data and register
                        them into a file named TABLES_SCN.log per default. It
                        can be changed using -C | --cdc_file.
   --lo_import        : use psql \lo_import command to import BLOB as large
                        object. Can be use to import data with COPY and import
                        large object manually in a second pass. It is recquired
                        for BLOB > 1GB. See documentation for more explanation.
   --mview_as_table str: Comma separated list of materialized views to export
                        as regular table.
   --drop_if_exists   : Drop the object before creation if it exists.
   --delete clause    : Set the DELETE clause to apply to the Oracle query to
                        be applied before importing data. Can be used multiple
            time.
```

1.6.1 Rapport d'évaluation - 1

- Rapport exhaustif du contenu de la base Oracle

```
ora2pg -t SHOW_REPORT
ora2pg -t SHOW_REPORT --dump_as_html
```

Rapport sur le contenu de la base Oracle

Ora2Pg dispose d'un mode d'analyse du contenu de la base Oracle afin de générer un rapport sur son contenu et présenter ce qui peut ou ne peut pas être exporté.

L'outil parcourt l'intégralité des objets, les dénombre, extrait les particularités de chacun d'eux et dresse un bilan exhaustif de ce qu'il a rencontré. Pour activer le mode « analyse et rapport », il faut utiliser l'export de type `SHOW_REPORT` par la commande suivante :

```
ora2pg -t SHOW_REPORT
```

Par défaut, le rapport se présente au format texte sur la sortie standard. Pour proposer un format plus lisible, il est **recommandé** d'activer l'option `--dump_as_html` qui crée un rapport au format HTML. D'autres formats existent, comme le CSV (`--dump_as_csv`) ou le JSON (`--dump_as_json`), si la centralisation et la consolidation de rapports sont nécessaires.

Voici un exemple de rapport obtenu avec cette commande :

```
-------------------------------------------------------------------------------
Ora2Pg v20.0 - Database Migration Report
-------------------------------------------------------------------------------
Version Oracle Database 12c Enterprise Edition Release 12.1.0.2.0
Schema  HR
Size    28.56 MB
```

```
-------------------------------------------------------------------------------
Object  Number  Invalid Comments    Details
-------------------------------------------------------------------------------
DATABASE LINK   2   0   Database links will be exported as SQL/MED PostgreSQL's
                Foreign Data Wrapper (FDW) extensions using oracle_fdw.
GLOBAL TEMPORARY TABLE  0   0   Global temporary table are not supported by
                PostgreSQL and will not be exported. You will have to
                rewrite some application code to match the PostgreSQL
                temporary table behavior.
INDEX   28  0   19 index(es) are concerned by the export, others are automatically
                generated and will do so on PostgreSQL. Bitmap will be
                exported as btree_gin index(es) and hash index(es) will be
                exported as b-tree index(es) if any. Domain index are exported
                as b-tree but commented to be edited to mainly use FTS.
                Cluster, bitmap join and IOT indexes will not be exported at all.
                Reverse indexes are not exported too, you may use a trigram-based
                index (see pg_trgm) or a reverse() function based index and search.
               Use 'varchar_pattern_ops', 'text_pattern_ops' or 'bpchar_pattern_ops'
                operators in your indexes to improve search with the LIKE operator
                respectively into varchar, text or char columns.
                3 function based b-tree index(es).
                13 b-tree index(es).
                3 spatial index index(es).
INDEX PARTITION 2   0   Only local indexes partition are exported, they are
                build on the column used for the partitioning.
JOB 0   0   Job are not exported. You may set external cron job with them.
PROCEDURE   3   1   Total size of procedure code: 1870 bytes.
SEQUENCE    3   0   Sequences are fully supported, but all call to
                sequence_name.NEXTVAL or sequence_name.CURRVAL will be
                transformed into NEXTVAL('sequence_name') or
                CURRVAL('sequence_name').
SYNONYM 0   0   SYNONYMs will be exported as views. SYNONYMs do not exists with
                PostgreSQL but a common workaround is to use views or set the
                PostgreSQL search_path in
                your session to access object outside the current schema.
TABLE   22  0    2 check constraint(s). 1 unknown types. Total number of rows: 421.
                Top 10 of tables sorted by number of rows:.
                sg_infrastructure_route has 188 rows.
                employees has 107 rows. departments has 27 rows. countries has
                25 rows.
                locations has 23 rows. jobs has 19 rows. job_history has 10 rows.
                error_log_sample has 6 rows. emptyclob has 4 rows. regions has
                4 rows.
                Top 10 of largest tables:...
TABLE PARTITION 5   0   Partitions are exported using table inheritance and
                check constraint.
                Hash and Key partitions are not supported by PostgreSQL and will not
                be exported.
                5 RANGE partitions..
TRIGGER 3   1   Total size of trigger code: 736 bytes.
VIEW    1   0   Views are fully supported but can use specific functions.
-------------------------------------------------------------------------------
Total   69  2
-------------------------------------------------------------------------------
```

D'autres paramètres ne peuvent pas être analysés par Ora2Pg comme l'usage de l'application. Il existe

aussi d'autres objets qui ne sont pas exportés directement par Ora2Pg comme les objets `DIMENSION` des fonctionnalités `OLAP` d'Oracle dans la mesure où ils n'ont pas d'équivalent dans PostgreSQL.

1.6.2 Rapport d'évaluation - 2

- Estimation du coût de migration

```
ora2pg -t SHOW_REPORT --estimate_cost
ora2pg -t SHOW_REPORT --estimate_cost --dump_as_html
```

Évaluer la charge de migration d'une base Oracle

Pour déterminer le coût en jours/personne de la migration, Ora2Pg dispose d'une directive de configuration nommée `ESTIMATE_COST`. Celle-ci peut aussi être activée en ligne de commande : `--estimate_cost`. Cette fonctionnalité n'est disponible qu'avec le type d'export `SHOW_REPORT`.

```
ora2pg -t SHOW_REPORT --estimate_cost
```

Le rapport généré est identique à celui généré par `SHOW_REPORT`, mais cette fonctionnalité provoque en plus l'exploration des objets de la base de données, du code source des vues, triggers et routines stockées (fonctions, procédures et paquets de fonctions), puis donne un score à chaque objet et à chaque routine suivant le volume de code et la complexité de réécriture manuelle de ce code. En effet, la réécriture d'une routine comportant un `CONNECT BY` ne prend pas le même temps que la réécriture d'une routine comportant des appels à `GOTO`.

```
-------------------------------------------------------------------------------
Ora2Pg v20.0 - Database Migration Report
-------------------------------------------------------------------------------
Version Oracle Database 12c Enterprise Edition Release 12.1.0.2.0
Schema  HR
Size    28.56 MB

-------------------------------------------------------------------------------
Object  Number  Invalid Estimated cost  Comments    Details
-------------------------------------------------------------------------------
DATABASE LINK   2   0   6   Database links will be exported as SQL/MED PostgreSQL's
                    Foreign Data Wrapper (FDW) extensions using oracle_fdw.
GLOBAL TEMPORARY TABLE  0   0   0   Global temporary table are not supported by
                    PostgreSQL and will not be exported. You will have to
                    rewrite some application code to match the PostgreSQL
                    temporary table behavior.
INDEX   28  0   5.3 19 index(es) are concerned by the export, others are
                    automatically
                    generated and will do so on PostgreSQL. Bitmap will be
                    exported as btree_gin index(es) and hash index(es) will be
```

```
                    exported as b-tree index(es) if any. Domain index are exported
                    as b-tree but commented to be edited to mainly use FTS.
                   Cluster, bitmap join and IOT indexes will not be exported at all.
                    Reverse indexes are not exported too, you may use a
                    trigram-based index (see pg_trgm) or a reverse() function
                    based index and search.  Use 'varchar_pattern_ops',
                    'text_pattern_ops' or 'bpchar_pattern_ops' operators in
                    your indexes to improve search with the LIKE operator
                    respectively into varchar, text or char columns.
                    3 function based b-tree index(es). 13 b-tree index(es).
                    3 spatial index index(es).
INDEX PARTITION 2   0   0   Only local indexes partition are exported, they are
                    build on the column used for the partitioning.
JOB 0   0   0   Job are not exported. You may set external cron job with them.
PROCEDURE   3   1   16  Total size of procedure code: 1870 bytes.
                    add_job_history: 3. test_dupl_vazba: 7. secure_dml: 3.
SEQUENCE    3   0   1   Sequences are fully supported, but all call to
                    sequence_name.NEXTVAL or sequence_name.CURRVAL will be
                    transformed into NEXTVAL('sequence_name')
                    or CURRVAL('sequence_name').
SYNONYM 0   0   0   SYNONYMs will be exported as views. SYNONYMs do not exists
                    with PostgreSQL but a common workaround is to use views
                    or set the PostgreSQL search_path in
                    your session to access object outside the current schema.
TABLE   22  0   2.4  2 check constraint(s). 1 unknown types. Total number of
                    rows: 421.
                    Top 10 of tables sorted by number of rows:.
                    sg_infrastructure_route
                    has 188 rows. employees has 107 rows. departments has 27 rows.
                    countries has 25 rows. locations has 23 rows. jobs has 19 rows.
                    job_history has 10 rows. error_log_sample has 6 rows.
                    regions has 4 rows.
                    emptyclob has 4 rows. Top 10 of largest tables:.
TABLE PARTITION 5   0   1   Partitions are exported using table inheritance
                    and check constraint.
                    Hash and Key partitions are not supported by PostgreSQL
                    and will not be exported.
                    5 RANGE partitions..
TRIGGER 3   1   9.3 Total size of trigger code: 736 bytes.  cisvpolpre_bi:
                    3.3. update_job_history: 3.
VIEW    1   0   1   Views are fully supported but can use specific functions.
-------------------------------------------------------------------------------
Total   69  2   42.0042.00 cost migration units means approximatively 1 man-day(s).
                    The migration unit was set to 5 minute(s)

-------------------------------------------------------------------------------
Migration level : B-5
-------------------------------------------------------------------------------

Migration levels:
    A - Migration that might be run automatically
    B - Migration with code rewrite and a human-days cost up to 5 days
    C - Migration with code rewrite and a human-days cost above 5 days
Technical levels:
    1 = trivial: no stored functions and no triggers
    2 = easy: no stored functions but with triggers, no manual rewriting
```

```
    3 = simple: stored functions and/or triggers, no manual rewriting
    4 = manual: no stored functions but with triggers or views with code rewriting
    5 = difficult: stored functions and/or triggers with code rewriting
-------------------------------------------------------------------------------

Details of cost assessment per function
Function test_dupl_vazba total estimated cost: 7
    CONCAT => 9 (cost: 0.1)
    TEST => 2
    SIZE => 1
    TO_CHAR => 1 (cost: 0.1)
    PRAGMA => 1 (cost: 3)
Function add_job_history total estimated cost: 3
    TEST => 2
    SIZE => 1
Function secure_dml total estimated cost: 3
    TEST => 2
    SIZE => 1
-------------------------------------------------------------------------------

Details of cost assessment per trigger
Trigger cisvpolpre_bi total estimated cost: 3.3
    CONCAT => 3 (cost: 0.1)
    TEST => 2
    SIZE => 1
Trigger update_job_history total estimated cost: 3
    TEST => 2
    SIZE => 1
-------------------------------------------------------------------------------
```

En fin de rapport, Ora2Pg affiche le nombre total d'objets rencontrés, les objets invalides et un nombre correspondant au nombre d'unités de coût de migration qu'il aura estimé nécessaire en fonction du code détecté (voir l'*Annexe 3 : Méthode de valorisation de la charge de migration*). Cette unité vaut par défaut 5 minutes, cela correspond au temps moyen que mettrait un spécialiste pour porter le code. Dans l'exemple ci-dessus, on a donc une estimation par Ora2Pg d'une migration ayant un coût de 162,5 unités multipliées par 5 minutes, ce qui correspond en gros à 2 jours/personne.

L'ajustement de cette valeur est à faire en fonction de l'expérience de l'équipe en charge de la migration. Pour la première migration, il est tout à fait raisonnable de doubler ce coût dans le fichier de configuration `ora2pg.conf` :

```
COST_UNIT_VALUE       10
```

ou en ligne de commande :

```
ora2pg -t SHOW_REPORT --estimate_cost --cost_unit_value 10
```

Dans ce mode de rapport, Ora2Pg affiche aussi les détails du coût de migration estimé par routine.

Il est possible d'obtenir un rapport au format HTML en activant la directive `DUMP_AS_HTML` :

```
DUMP_AS_HTML       1
```

ou en utilisant l'option `--dump_as_html` en ligne de commande :

```
ora2pg -t SHOW_REPORT  --estimate_cost --cost_unit_value 10 --dump_as_html
```

Ora2Pg propose un exemple de rapport en HTML sur son site : Ora2Pg - Database Migration Report[10]

Par défaut, Ora2Pg affiche les dix tables les plus volumineuses en termes de nombre de lignes et le top dix des tables les plus volumineuses en taille (hors partitions). Le nombre de table affichées peut être contrôlé avec la directive de configuration `TOP_MAX`.

L'action `SHOW_REPORT` renvoie le rapport sur la sortie standard (`stdout`), il est donc conseillé de renvoyer la sortie dans un fichier pour pouvoir le consulter dans l'application adaptée à son format. Par exemple :

```
ora2pg -t SHOW_REPORT --estimate_cost --dump_as_html > report.html
```

[10] https://ora2pg.darold.net/report.html

1.7 CONCLUSION

Points essentiels :

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

1.7.1 Questions

N'hésitez pas, c'est le moment !

1.8 QUIZ

https://dali.bo/n1_quiz

1.9 INSTALLATION DE POSTGRESQL DEPUIS LES PAQUETS COMMUNAUTAIRES

L'installation est détaillée ici pour Rocky Linux 8 et 9 (similaire à Red Hat et à d'autres variantes comem Oracle Linux et Fedora), et Debian/Ubuntu.

Elle ne dure que quelques minutes.

1.9.1 Sur Rocky Linux 8 ou 9

ATTENTION : Red Hat, CentOS, Rocky Linux fournissent souvent 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 fortement conseillée.

Installation du dépôt communautaire :

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

- la version majeure de PostgreSQL (ici la 16) ;
- la distribution (ici Rocky Linux 8) ;
- l'architecture (ici x86_64, la plus courante).

Les commandes sont à lancer sous **root** :

```
# 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 16 (client, serveur, librairies, extensions) :

```
# dnf install -y postgresql16-server postgresql16-contrib
```

Les outils clients et les librairies nécessaires seront automatiquement installés.

Une fonctionnalité avancée optionnelle, le JIT (*Just In Time compilation*), nécessite un paquet séparé.

```
# dnf install postgresql16-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-16/bin/postgresql-16-setup initdb
# cat /var/lib/pgsql/16/initdb.log
```

Ce dernier fichier permet de vérifier que tout s'est bien passé et doit finir par :

```
Success. You can now start the database server using:

    /usr/pgsql-16/bin/pg_ctl -D /var/lib/pgsql/16/data/ -l logfile start
```

Chemins :

Objet	Chemin
Binaires	`/usr/pgsql-16/bin`
Répertoire de l'utilisateur **postgres**	`/var/lib/pgsql`
`PGDATA` par défaut	`/var/lib/pgsql/16/data`
Fichiers de configuration	dans `PGDATA/`
Traces	dans `PGDATA/log`

Configuration :

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

Commandes d'administration habituelles :

Démarrage, arrêt, statut, rechargement à chaud de la configuration, redémarrage :

```
# systemctl start postgresql-16
# systemctl stop postgresql-16
# systemctl status postgresql-16
# systemctl reload postgresql-16
# systemctl restart postgresql-16
```

Test rapide de bon fonctionnement et connexion à psql :

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

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

```
# systemctl enable postgresql-16
```

Ouverture du *firewall* pour le port 5432 :

Voir si le *firewall* est actif :

```
# systemctl status firewalld
```

Si c'est le cas, autoriser un accès extérieur :

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

(Rappelons que `listen_addresses` doit être également modifié dans `postgresql.conf`.)

Création d'autres instances :

Si des instances de *versions majeures différentes* doivent être installées, il faut d'abord installer les binaires pour chacune (adapter le numéro dans `dnf install …`) et appeler le script d'installation de chaque version. l'instance par défaut de chaque version vivra dans un sous-répertoire numéroté de `/var/lib/pgsql` automatiquement créé à l'installation. Il faudra juste modifier les ports dans les `postgresql.conf` pour que les instances puissent tourner simultanément.

Si plusieurs instances d'une *même version majeure* (forcément de la même version mineure) doivent cohabiter sur le même serveur, il faut 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 nouveau sous-répertoire de `/var/lib/pgsqsl/16/` (ou l'équivalent pour d'autres versions majeures).

Pour créer une seconde instance, nommée par exemple **infocentre** :

- Création du fichier service de la deuxième instance :

```
# cp /lib/systemd/system/postgresql-16.service \
        /etc/systemd/system/postgresql-16-infocentre.service
```

- Modification de ce dernier fichier avec le nouveau chemin :

```
Environment=PGDATA=/var/lib/pgsql/16/infocentre
```

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

```
# export PGSETUP_INITDB_OPTIONS='--data-checksums --lc-messages=C'
# /usr/pgsql-16/bin/postgresql-16-setup initdb postgresql-16-infocentre
```

- Option 2 : restauration d'une sauvegarde : la procédure dépend de votre outil.
- Adaptation de `/var/lib/pgsql/16/infocentre/postgresql.conf` (port surtout).
- Commandes de maintenance de cette instance :

```
# systemctl [start|stop|reload|status] postgresql-16-infocentre
# systemctl [enable|disable] postgresql-16-infocentre
```

- Ouvrir le nouveau port dans le firewall au besoin.

1.9.2 Sur Debian / Ubuntu

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

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

Installation du dépôt communautaire :

L'installation des dépôts du PGDG est prévue dans le paquet Debian :

```
# apt update
# apt install -y  gnupg2  postgresql-common
# /usr/share/postgresql-common/pgdg/apt.postgresql.org.sh
```

Ce dernier ordre créera le fichier du dépôt `/etc/apt/sources.list.d/pgdg.list` adapté à la distribution en place.

Installation de PostgreSQL 16 :

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

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-16 postgresql-client-16
```

La première instance est automatiquement créée, démarrée et déclarée comme service à lancer au démarrage du système. Elle porte un nom (par défaut `main`).

Elle est immédiatement accessible par l'utilisateur système **postgres**.

Chemins :

Objet	Chemin
Binaires	`/usr/lib/postgresql/16/bin/`
Répertoire de l'utilisateur **postgres**	`/var/lib/postgresql`
PGDATA de l'instance par défaut	`/var/lib/postgresql/16/main`
Fichiers de configuration	dans `/etc/postgresql/16/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, qui demandent en paramètre la version et le nom de l'instance :

```
# pg_ctlcluster 16 main [start|stop|reload|status|restart]
```

Démarrage de l'instance avec le serveur :

C'est en place par défaut, et modifiable dans `/etc/postgresql/16/main/start.conf`.

Ouverture du firewall :

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

Statut des instances du serveur :

```
# pg_lsclusters
```

Test rapide de bon fonctionnement et connexion à psql :

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

Destruction d'une instance :

```
# pg_dropcluster 16 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 :

- optionnellement, `/etc/postgresql-common/createcluster.conf` permet de mettre en place tout d'entrée les *checksums*, les messages en anglais, le format des traces ou un emplacement séparé pour les journaux :

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

- créer une instance :

```
# pg_createcluster 16 infocentre
```

Il est également possible de préciser certains paramètres du fichier `postgresql.conf`, voire les chemins des fichiers (il est conseillé de conserver les chemins par défaut) :

```
# pg_createcluster 16 infocentre \
  --port=12345 \
  --datadir=/PGDATA/16/infocentre \
  --pgoption shared_buffers='8GB' --pgoption work_mem='50MB' \
  --  --data-checksums --waldir=/ssd/postgresql/16/infocentre/journaux
```

- adapter au besoin `/etc/postgresql/16/infocentre/postgresql.conf` ;
- démarrage :

```
# pg_ctlcluster 16 infocentre start
```

1.9.3 Accès à l'instance depuis le serveur même (toutes distributions)

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 (16.0)
Type "help" for help.
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 (16.0)
Type "help" for help.
postgres=#
```

Une authentification par mot de passe est plus sécurisée :

- dans `pg_hba.conf`, paramétrer une authentification par mot de passe pour les accès depuis `localhost` (déjà en place sous Debian) :

```
# IPv4 local connections:
host    all             all             127.0.0.1/32            scram-sha-256
# IPv6 local connections:
host    all             all             ::1/128                 scram-sha-256
```

(Ne pas oublier de recharger la configuration en cas de modification.)

- ajouter un mot de passe à l'utilisateur `postgres` de l'instance :

```
dalibo:~$ sudo -iu postgres psql
psql (16.0)
Type "help" for help.
postgres=# \password
Enter new password for user "postgres":
Enter it again:
postgres=# quit

dalibo:~$ psql -h localhost -U postgres
Password for user postgres:
psql (16.0)
Type "help" for help.
postgres=#
```

- Pour se connecter sans taper le mot de passe à une instance, 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/16/data/log` ou `/var/log/postgresql/`) ;
- toute modification de `pg_hba.conf` ou `postgresql.conf` impliquant de recharger la configuration peut être réalisée par une de ces trois méthodes en fonction du système :

```
root:~# systemctl reload postgresql-16

root:~# pg_ctlcluster 16 main reload

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

1.10 TRAVAUX PRATIQUES

Environnement

- Installer les paquets `perl`, `perl-DBI` requis pour le contexte d'exécution d'Ora2Pg ainsi que `cpan`, `gcc`, `make`, `libaio-devel`, `perl-devel` pour l'étape de compilation.

(Optionnel) 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.

Prise en main de l'instance Oracle

- Démarrer l'instance Oracle.

- Autoriser l'utilisateur `hr` à se connecter.

- Vérifier la connexion à la base Oracle avec `sqlplus`.

- IP de l'instance : *demander au formateur*
- Utilisateur : `hr`
- Mot de passe : `phoenix`
- Nom de service : `hr`

Création du rapport d'évaluation

- Définir une configuration globale avec un fichier `/etc/ora2pg/ora2pg.conf`.

- Créer un rapport d'évaluation `rapport.html` de la base `HR` en activant l'estimation de le charge de migration.

1.11 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.

Les commandes suivantes sont déstinées à être exécutées sur un serveur Rocky Linux 8.

Environnement

- Installer les paquets `perl`, `perl-DBI` requis pour le contexte d'exécution d'Ora2Pg ainsi que `cpan`, `gcc`, `make`, `libaio-devel`, `perl-devel` pour l'étape de compilation.

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

```
# dnf install -y perl perl-DBI
# dnf install -y cpan gcc make perl-devel libaio-devel
```

(Optionnel) 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/1917000
wget $D/oracle-instantclient19.17-basic-19.17.0.0.0-1.x86_64.rpm
wget $D/oracle-instantclient19.17-devel-19.17.0.0.0-1.x86_64.rpm
wget $D/oracle-instantclient19.17-sqlplus-19.17.0.0.0-1.x86_64.rpm
```

Installer ensuite les paquets téléchargés :

```
# dnf install -y oracle-instantclient19.17-basic-19.17.0.0.0-1.x86_64.rpm \
                 oracle-instantclient19.17-devel-19.17.0.0.0-1.x86_64.rpm \
                 oracle-instantclient19.17-sqlplus-19.17.0.0.0-1.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 *Oracle Database* (lorsqu'une instance Oracle est présente sur le serveur) :

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

- Avec les librairies *Instant Client* (en cas d'absence des binaires Oracle sur le serveur) :

```
# export ORACLE_HOME=/usr/lib/oracle/19.17/client64
# export LD_LIBRARY_PATH=$ORACLE_HOME/lib
```

- Installation du driver

L'option `-T` de `cpan` permet d'ignorer la phase de tests des fichiers compilés.

```
# cpan install -T 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 :

```
# dnf 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 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/v24.1.tar.gz
# tar xvf v24.1.tar.gz
# cd ora2pg-24.1
```

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

Compilation et installation d'Ora2Pg

[11] http://search.cpan.org/search?query=DBD::Pg
[12] https://github.com/darold/ora2pg/releases

- 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.

Prise en main de l'instance Oracle

- Démarrer l'instance Oracle.

En se connectant à l'utilisateur **oracle** pour la première fois, il est nécessaire de démarrer correctement l'instance de démonstration. Il s'agit d'une installation en mode *container*.

```
# changer d'utilisateur linux
sudo -iu oracle

lsnrctl start
rlwrap sqlplus / as sysdba
```

Dans l'invite de commandes, déclencher le démarrage de l'instance.

```
SQL> startup
```

- Autoriser l'utilisateur `hr` à se connecter.

Se positionner dans la base `HR` pour réactiver le compte.

```
SQL> ALTER SESSION SET container = hr;
SQL> ALTER PROFILE default LIMIT password_life_time UNLIMITED;
SQL> ALTER USER hr IDENTIFIED BY "phoenix";
```

- Vérifier la connexion à la base Oracle avec `sqlplus`.

Ajouter les variables d'environnements dans le fichier `.bash_profile` de l'utilisateur **dalibo**.

```
$ cat ~/.bash_profile
export ORACLE_HOME=/opt/oracle/product/21c/dbhomeXE
export LD_LIBRARY_PATH=$ORACLE_HOME/lib
export PATH=$PATH:$ORACLE_HOME/bin:/usr/local/bin
```

Charger les variables :

```
source $HOME/.bash_profile
```

Ouvrir une session sur l'instance Oracle et se déconnecter :

```
sqlplus hr/phoenix@localhost:1521/hr
```

```
SQL> exit
```

Création du rapport d'évaluation

- Définir une configuration globale avec un fichier `/etc/ora2pg/ora2pg.conf`.

Lors de l'installation, il se peut que ce fichier soit absent. Il est nécessaire de copier le fichier de la distribution vers le fichier de configuration global.

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

Pour la suite des exercices, renseigner les paramètres de connexions à l'instance Oracle dans la première section du fichier de configuration.

```
# Set Oracle database connection (datasource, user, password)
ORACLE_DSN      dbi:Oracle://localhost:1521/hr
ORACLE_USER     hr
ORACLE_PWD      phoenix
```

- Créer un rapport d'évaluation `rapport.html` de la base `HR` en activant l'estimation de le charge de migration.

La commande suivante permet de créer le rapport d'évaluation :

```
ora2pg -t SHOW_REPORT -n hr --estimate_cost --dump_as_html > rapport.html
```

L'en-tête du rapport fait l'état de la version du serveur Oracle et du nom du schéma qui a été analysé. La volumétrie est calculée d'après la somme des segments (tables, index, vues matérialisées, etc.) présents dans le schéma.

Le tableau principal du rapport contient le décompte des objets identifiés dans le schéma, comme les tables ou les index. Pour chaque catégorie, le coût estimé permet de comprendre rapidement où se situe l'effort requis pour réaliser la migration. Une colonne de commentaire donne des éléments supplémentaires sur le décompte et restitue des conseils génériques sur la migration de tel ou tel composant.

Object	Number	Invalid	Estimated cost	Comments	Details
DATABASE LINK	0	0	0.00	Database links will be exported as SQL/MED PostgreSQL's Foreign Data Wrapper (FDW) extensions using oracle_fdw.	
FUNCTION	2	0	8.50	Total size of function code: 1322 bytes.	▸ See details
GLOBAL TEMPORARY TABLE	0	0	0.00	Global temporary table are not supported by PostgreSQL and will not be exported. You will have to rewrite some application code to match the PostgreSQL temporary table behavior.	
INDEX	19	0	3.00	11 index(es) are concerned by the export, others are automatically generated and will do so on PostgreSQL. Bitmap will be exported as btree_gin index(es). Domain index are exported as b-tree but commented to be edited to mainly use FTS. Cluster, bitmap join and IOT indexes will not be exported at all. Reverse indexes are not exported too, you may use a trigram-based index (see pg_trgm) or a reverse() function based index and search. Use 'varchar_pattern_ops', 'text_pattern_ops' or 'bpchar_pattern_ops' operators in your indexes to improve search with the LIKE operator respectively into varchar, text or char columns.	▸ See details
JOB	0	0	0.00	Job are not exported. You may set external cron job with them.	
PACKAGE BODY	2	0	42.00	Total size of package code: 3712 bytes. Number of procedures and functions found inside those packages: 10.	▸ See details
PROCEDURE	2	0	8.20	Total size of procedure code: 1130 bytes.	▸ See details
SEQUENCE	3	0	1.00	Sequences are fully supported, but all call to sequence_name.NEXTVAL or sequence_name.CURRVAL will be transformed into NEXTVAL('sequence_name') or CURRVAL('sequence_name').	
SYNONYM	0	0	0.00	SYNONYMs will be exported as views. SYNONYMs do not exists with PostgreSQL but a common workaround is to use views or set the PostgreSQL search_path in your session to access object outside the current schema.	
TABLE	7	0	1.20	2 check constraint(s).	▸ See details
TRIGGER	2	0	5.00	Total size of trigger code: 0 bytes.	▸ See details
VIEW	1	0	1.00	Views are fully supported but can use specific functions.	
Total	38	0	69.90	69.90 cost migration units means approximatively 1 person-day(s). The migration unit was set to 5 minute(s)	

Le score de complexité (de **A-1** à **C-5**) permet de classifier la migration d'un schéma selon plusieurs niveaux de complexité. Ici, le score du schéma HR vaut **B-5** :

- La migration prévoit de la réécriture de code et le coût est inférieur à 5 jour/personne ;
- La difficulté est maximale avec la présence de procédures stockées ou de triggers avec de la réécriture de code.

Migration level: B-5

- Migration levels:
  - A - Migration that might be run automatically
  - B - Migration with code rewrite and a human-days cost up to 5 days
  - C - Migration with code rewrite and a human-days cost above 5 days
- Technical levels:
  - 1 = trivial: no stored functions and no triggers
  - 2 = easy: no stored functions but with triggers, no manual rewriting
  - 3 = simple: stored functions and/or triggers, no manual rewriting
  - 4 = manual: no stored functions but with triggers or views with code rewriting
  - 5 = difficult: stored functions and/or triggers with code rewriting

Enfin, la dernière partie apporte un détail plus précis sur le calcul du coût par functions, procédures et triggers. Par défaut, le coût d'une réécriture d'une méthode, même simple, vaudra 3 unités, soit l'équivalent de 15 minutes.

Details of cost assessment per function

▾ Show

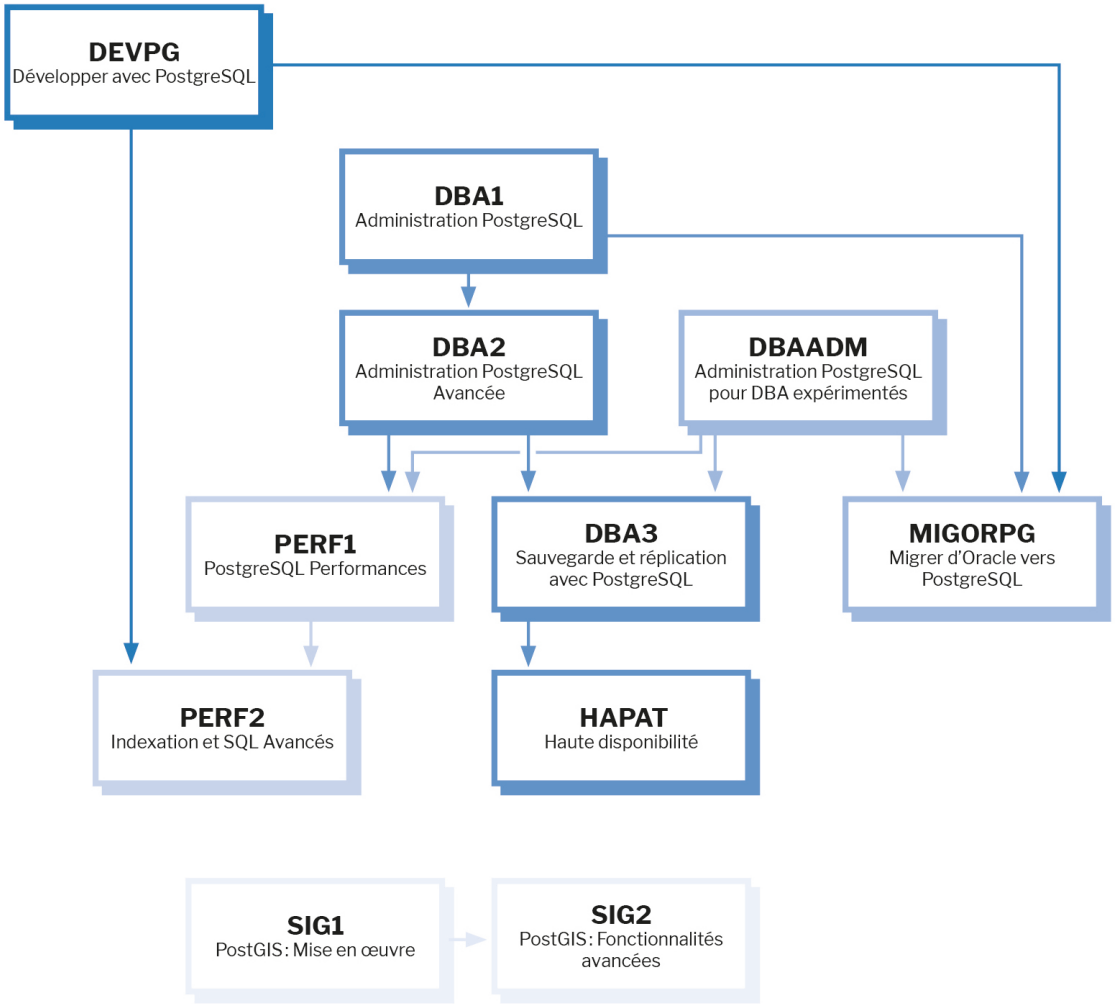
- Function last_first_name total estimated cost: 3.5
  - CONCAT => 5 (cost: 0.1)
  - TEST => 2
  - SIZE => 1
- Function emp_sal_ranking total estimated cost: 3
  - TEST => 2
  - SIZE => 1

Les formations Dalibo

Retrouvez nos formations et le calendrier sur https://dali.bo/formation

Pour toute information ou question, n'hésitez pas à nous écrire sur contact@dalibo.com.

Cursus des formations



Retrouvez nos formations dans leur dernière version :

- 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

Les livres blancs

- Migrer d'Oracle à PostgreSQL
  https://dali.bo/dlb01
- Industrialiser PostgreSQL
  https://dali.bo/dlb02
- Bonnes pratiques de modélisation avec PostgreSQL
  https://dali.bo/dlb04
- Bonnes pratiques de développement avec PostgreSQL
  https://dali.bo/dlb05

Téléchargement gratuit

Les versions électroniques de nos publications sont disponibles gratuitement sous licence open source ou sous licence Creative Commons.