Ora2Pg: Outil de migration de bases Oracle vers PostgreSQL

Gilles Darold (Responsable Systèmes et Réseaux)
Concepteur/développeur du projet Ora2Pg

Jean-Paul Argudo (Responsable informatique)
Contributeur PostgreSQL/Ora2Pg

Historique

Ora2Pg est né suite à l'intérêt grandissant des détenteurs de bases Oracle pour PostgreSQL sur les mailings listes PG.

Il restait à fournir un outil de conversion semi-automatique leur permettant de mettre en oeuvre rapidement une architecture de test.

Ce fût chose faite en mai 2001 avec la première version de Ora2Pg.

Ora2Pg en est aujourd'hui à la version 3.0 (décembre 2004) enrichie de l'experience des nombreuses migrations et des fonctionnalités demandées par les utilisateurs.

Les évolutions futures d'Ora2Pg concernent principalement des mises à jour en relation avec les efforts des développeurs de PostgreSQL pour implémenter les fonctionnalités des bases Oracle dans PostgreSQL.

Principe

Le principe de base d'Ora2Pg est la découverte ou le scan automatique d'une base Oracle pour en extraire la structure ou les données et les convertir dans la syntaxe PostgreSQL.

La structure

Eléments de structure SQL exportés :

- Espace de nommage (NAMESPACE ou SCHEMA).
- Table, vue.
- Contrainte (clé unique, clés primaire, clé étrangère).
- Contrainte de valeur (CHECK).
- Index.
- Trigger.

- Séquence.
- Fonction, procédure, package.
- Droit (GRANT, ROLE, USER).
- Système de fichier (TABLESPACE).

La plupart de ces exportations ne nécessitent aucune reécriture mis à part les éléments :

- Fonction, procédure et package.
  En raison du langage procédural utilisé et du passage de paramètre des fonctions.

- Droit.
  En raison de la complexité des rôles et des différents utilisateurs "system" sous Oracle. L'implementation des rôles dans les prochaines versions de PostgreSQL devrait permettre une meilleure exportation de ces droits.

- Tablespace.
  En fonction du chemin sur le système de fichier. A noter que PostgreSQL prend un répertoire comme chemin du tablespace au lieu d'un fichier pour Oracle.

Les données

Ora2Pg permet aussi d'extraire les données d'une base Oracle en vue de leur chargement dans une base PostgreSQL.

La base PostgreSQL de destination peut être soit issue d'un premier export de la structure Oracle via Ora2pg, soit avoir une structure différente.

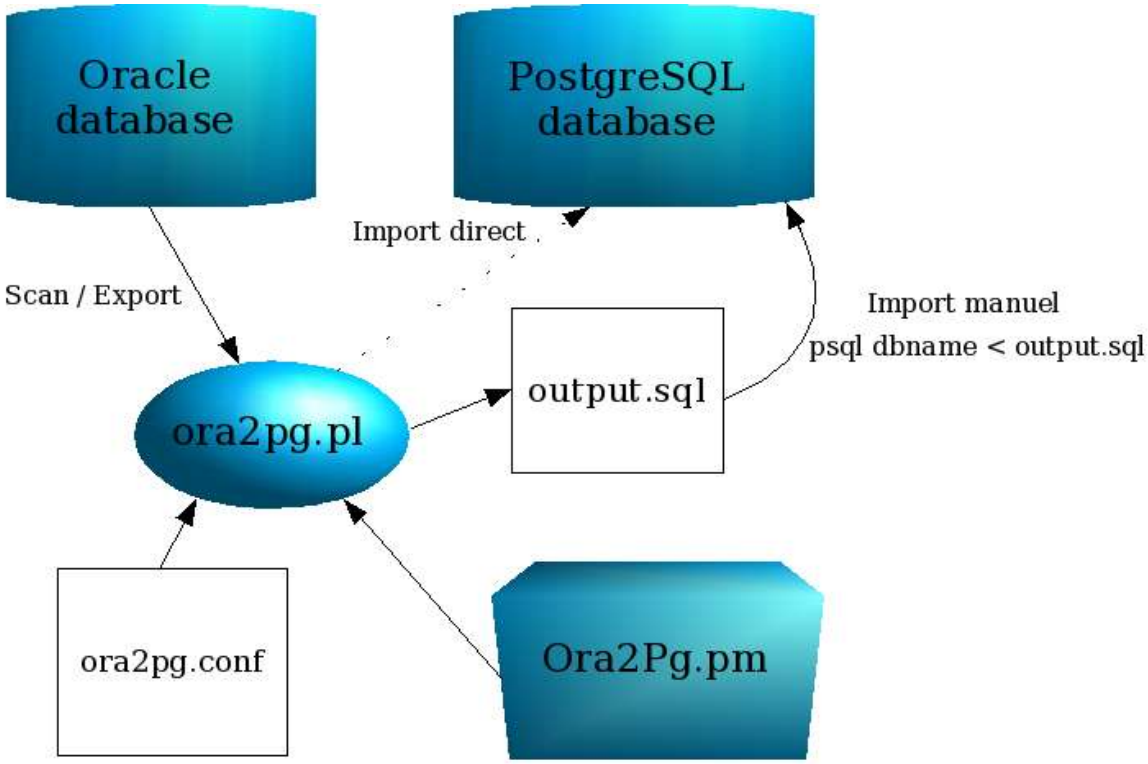
L'outil

Ora2Pg est composé d'un module Perl Ora2Pg.pm interfaçable à partir de n'importe quel script Perl.

La distribution inclut un script perl ora2pg.pl et un fichier de configuration ora2pg.conf permettant d'utiliser l'outil sans aucune connaissance Perl.

Architecture

Ora2Pg a été développé sous Linux SlackWare mais est compatible avec toutes les plateformes sous lesquelles Perl, PostgreSQL et Oracle sont disponibles.



Les pré-requis

Ora2pg nécessite en plus de Perl 5 plusieurs modules Perl tous disponibles à partir du CPAN :

DBI
DBD::Oracle
DBD::Pg (optionnel)
Compress::Zlib (optionnel)

Le module Perl DBD::Oracle necessite les librairies du client Oracle 8i ou 9i, les deux étant supportés par Ora2Pg.

Le module DBD::Pg recquiert l'installation préalable du client PostgreSQL. Il est optionnel car utilisé uniquement pour un chargement direct.

Le module Compress::Zlib est utilisé uniquement si ce type de compression est demandé (Cf. option de configuration OUTPUT).

Les bases de données peuvent être ou non sur la même machine que le logiciel, mais les librairies et modules Perl doivent, eux, être installés.

Configuration

La configuration d'Ora2Pg peut être aussi simple que :

- choisir la base de données Oracle à exporter,
- choisir le type d'export désiré,

ce qui peut être mis en oeuvre dans la minute.

Elle peut être aussi compliquée que :

- choisir la base à exporter,
- sélectionner certaines tables et champs à exporter,
- renommer certaines tables et champs lors de l'export,
- exporter les données en fonction d'une clause WHERE suivant les tables,
- retarder l'application des contraintes lors de l'insertion des données,
- compresser les données exportées pour préserver l'espace disque.

Toutes ces possibilités de configuration sont accessibles à partir du fichier de configuration ora2pg.conf et des valeurs données aux directives qu'il contient.

Configuration de connexion à la base Oracle

4 directives servent à paramétrer l'accès à la base de données Oracle à exporter :

ORACLE_HOME qui permet de positionner le chemin des librairies du client Oracle pour le module DBD::Oracle.
ORACLE_DSN qui permet de définir la source de la base de données sous la forme `dbi:Oracle:host=oradb_host;sid=TEST`
ORACLE_USER et ORACLE_PWD permettant de définir l'utilisateur et son mot de passe sous lequel la connexion sera réalisée.

Configuration du schéma de la base Oracle à exporter

Lors de l'export de la base l'extraction peut être limité à un certain schéma ou 'namespace' Oracle, notament en fonction de l'utilisateur suivant lequel se fait la connexion.

La directive SCHEMA permet de donner le nom du schéma sur lequel le travail doit se limiter.

Pour créer et utiliser ce même schéma au niveau de la base de données PostgreSQL de destination il faut donner la valeur 1 à la directive EXPORT_SCHEMA.

Par défaut, l'intégralité des objets de la base de données sont exportés hormis les objets appartenant aux rôles système suivants : SYS, SYSTEM, DBSNMP, OUTLN, PERFSTAT.

Configuration du type d'export à réaliser

L'action d'export à réaliser se fait par le biais de la directive TYPE. Voici les valeurs qu'elle peut prendre et leurs significations :

- TABLE : extrait la structure des tables avec index, clé primaire, clé unique et clé étrangère.
- VIEW : extrait uniquement les vues.
- GRANT : extrait les rôles (convertis en groupes PostgreSQL), les utilisateurs et les droits sur les différents objets .
- SEQUENCE : extrait les séquences.
- TABLESPACE : extrait les espaces de stockage (PostgreSQL >= 8)
- TRIGGER : extrait les triggers définis suivant les actions.
- FUNCTION : extrait les fonctions.
- PROCEDURE : extrait les procédures.
- PACKAGE : extrait les packages.
- DATA : extrait les données des tables en tant que lignes INSERT.
- COPY : extrait les données des tables en tant que blocs COPY.

On ne peut réaliser qu'un seul type d'export à la fois, l'utilisation de la directive TYPE doit donc être unique.

Certains types d'export ne peuvent pas être chargés directement dans la base PostgreSQL et nécessitent encore une édition manuelle : GRANT, TABLESPACE, FUNCTION, PROCEDURE et PACKAGE.

Limitation des objets exportés

Les actions d'exports peuvent être limités à certaines tables soit en précisant le nom des tables sur lesquelles doit se limiter le travail (directive TABLES) soit en précisant les tables à exclurent du travail d'exportation (directive EXCLUDE). Ces deux directives prennent pour valeur une liste de noms de tables séparés par un espace.

La limitation peut aussi se faire en utilisant l'id des tables dans le processus de scan. La directive MIN permet de préciser l'id de départ et la directive MAX l'id de fin. Les ids sont obtenus en exécutant le script une première fois avec la valeur 1 donnée à la directive SHOWTABLEID.

En ce qui concerne l'extraction des données, la directive DATA_LIMIT permet de scinder l'export en blocs de données de la taille de la valeur indiquée. Ceci dépend de la puissance de la machine sur laquelle vous travaillez et de la mémoire disponible. Par exemple 10 000 pour des petites machines et 1 000 000 si vous avez vraiment beaucoup de mémoire disponible.

La directive WHERE quant à elle permet d'ajouter une clause de sélection plus précise des données à extraire. La valeur de cette directive est une liste séparée par des espaces de valeurs définies comme suit :

```
NOM_TABLE[where clause]
```

Par exemple

```
T_TEST[ID1='001' AND ID1='002']
```

Si vous placez directement une clause dans cette liste sans la limiter à une table, cette clause s'appliquera à toute les tables. Par exemple :

```
T_TEST[ID1='001' AND ID1='002] DATE_CREATE > '2001-01-01'
```

Le critère de sélection sur la date de création sera appliquer sur l'ensemble des tables touchées par l'extraction (la table T_TEST incluse).

Modification de la structure des objets

L'un des détournements majeur d'Ora2Pg est son utilisation pour répliquer des bases Oracle dans des bases PostgreSQL dont les structures diffèrent.

Pour faciliter ce genre de processus 3 directives de configuration ont été ajoutées pour permettre le 'mappage' entre les différents objets.

Ces directives ne sont prises en compte que dans le cadre d'une extraction de données, c'est à dire lorsque la directive TYPE est positionnée à la valeur DATA ou COPY.

MODIFY_STRUCT : permet de limiter les champs à extraire suivant les tables. La valeur prise par cette directive est une liste séparée par des espaces ayant la syntaxe suivante :

```
NOM_TABLE(nomcol1,nomcol2,...) ...
```

Par exemple :

```
T_TEST(id1,dossier)
```

extraira uniquement les colonnes 'id1' et 'dossier' de la table T_TEST.

La directive de configuration REPLACE_TABLES permet de modifier le nom des tables lors de l'export. Elle prend pour valeur une liste séparée par des espaces ayant la syntaxe suivante :

```
NOM_TABLE_ORIGINE:NOM_TABLE_MODIFIE ...
```

Les noms des colonnes peuvent aussi être renommés pendant l'extraction par l'utilisation de la directive REPLACE_COLS. Elle prend pour valeur une liste séparée par des espaces ayant la syntaxe suivante :

```
NOM_TABLE_ORIGINE(NOM_COL_ORIGINE:NOM_COL_MODIFIE)
```

Par exemple :

```
T_TEST(dico:dictionary,dossier:folder)
```

remplacera la colonne 'dico' par 'dictionary' et la colonne 'dossier' par 'folder'. Le nom de la table doit toujours être le nom d'origine même s'il est modifié dans la directive REPLACE_TABLES.

Import direct dans une base PostgreSQL

Par défaut, l'export des objets ou données de la base Oracle est enregistré dans un fichier. Pour renvoyer le flux des données exportées directement dans la base PostgreSQL de destination il existe 3 directives de configuration :

- PG_DSN : chaine de connexion à la base PostgreSQL sous la forme `dbi:Pg:dbname=pgdb;host=localhost;port=5432`
- PG_USER: utilisateur PostgreSQL utilisé pour la connexion.
- PG_PWD : mot de passe de l'utilisateur.

Autres directives de configuration

- IMPORT : permet l'inclusion de fichier de configuration commun.
- DEBUG : affiche les informations sur l'export en cours.
- SKIP : permet d'exporter le schema d'une base de données sans certaines contraintes (clés primaires, clé uniques, clés étrangères, indexes, test de valeurs ou toutes).
- CASE_SENSITIVE : par défaut tous les noms des objets sont convertis en minuscule (comportement de PostgreSQL par défaut). En positonnant cette directive a 1, les noms gardent exactement leur syntaxe - Déconseillé.
- OUTPUT : permet de spécifier le fichier de sortie des extractions. Si le fichier donné porte l'extension .gz ou .bz2, il sera automatiquement compressé.
- BZIP2 : permet de modifier le chemin par défaut de l'exécutable bzip2.
- USER_GRANTS : permet d'extraire les informations des tables Oracle ayant le préfix ALL_ plutôt que DBA_ par défaut. Ceci est à utiliser si les droits de l'utilisateur sont insuffisant.
- GEN_USER_PWD : lors de l'exportation des utilisateurs, si cette directive a la valeur 1, un mot de passe aléatoire sera généré pour chacun d'eux. Sinon le mot de passe par defaut ('change_my_secret') est utilisé.
- FKEY_DEFERRABLE : force les clés étrangères à être exportées en mode "deferrable". Par défaut elles sont exportées comme elles ont été créées.
- DEFER_FKEY : force le report du test des contraintes en fin de

transaction lors de l'export des données.

- PG_NUMERIC_TYPE: les types numériques Oracle sont exportés par défaut en tant que numeric(p,s). Ce type de données est très lent sous PostgreSQL. En donnant la valeur 1 à cette directive, les types numeriques seront convertis en type PostgreSQL approchant (smallint, integer, bigint, real et float).

Exécution d'Ora2Pg

Une fois la configuration terminée il ne reste qu'à executer le script ora2pg.pl en lui précisant le fichier de configuration à utiliser.

```
#> ora2pg.pl ora2pg.conf
```

Voici ce qu'affiche ora2pg.pl lors de l'exportation de la structure d'une base de données (option TYPE => TABLE, NUMERIC => 1) avec l'option DEBUG activée.

```
[1] Scanning OBJ_M ( BEA OBJ_M TABLE Generic Parameters)...
      M_ID => type:NUMBER , length:22, precision:10, scale:0, nullable:N ,
default:
      M_APPLID => type:NUMBER , length:22, precision:10, scale:0,
nullable:N , default:
      M_USERID => type:NUMBER , length:22, precision:10, scale:0,
nullable:N , default:
      M_TYPE => type:NUMBER , length:22, precision:10, scale:0,
nullable:Y , default:
      M_VALUE => type:NUMBER , length:22, precision:10, scale:0, nullable:N
, default:
      M_LABEL => type:VARCHAR2 , length:254, precision:, scale:, nullable:Y
, default:

[2] Scanning A_FORM ( BIA A_FORM TABLE )...
      DICO => type:VARCHAR2 , length:15, precision:, scale:, nullable:N ,
default:
      DOSSIER => type:VARCHAR2 , length:15, precision:, scale:,
nullable:N , default:
      ID1 => type:VARCHAR2 , length:36, precision:, scale:, nullable:N ,
default:
      DATAPP => type:DATE , length:7, precision:, scale:, nullable:N ,
default:
      NO => type:NUMBER , length:22, precision:, scale:, nullable:N ,
default:
      RUB => type:VARCHAR2 , length:4, precision:, scale:, nullable:N ,
default:
      TYP => type:VARCHAR2 , length:1, precision:, scale:, nullable:Y ,
default:
      FLAG => type:VARCHAR2 , length:7, precision:, scale:, nullable:Y ,
      ETR => type:DATE , length:7, precision:, scale:, nullable:Y ,
default:
      OPA => type:VARCHAR2 , length:4000, precision:, scale:, nullable:Y ,
default:


Dumping table OBJ_M...
Dumping table A_FORM...
```

Et voici le résultat de l'extraction écrit dans le fichier output.sql :

```
BEGIN TRANSACTION;

CREATE TABLE "obj_m" (
      "m_id" bigint NOT NULL,
      "m_applid" bigint NOT NULL,
      "m_userid" bigint NOT NULL,
      "m_type" bigint,
      "m_value" bigint NOT NULL,
      "m_label" varchar(254)
);
CREATE UNIQUE INDEX pk_m ON "obj_m" ("m_applid","m_id","m_userid");

CREATE TABLE "a_form" (
      "dico" varchar(15) NOT NULL,
      "dossier" varchar(15) NOT NULL,
      "id1" varchar(36) NOT NULL,
      "datapp" timestamp NOT NULL,
      "no" float NOT NULL,
      "rub" varchar(4) NOT NULL,
      "typ" varchar(1),
      "etr" timestamp,
      "opa" varchar(4000),
      PRIMARY KEY ("dico","dossier","id1","datapp","no","rub")
);
CREATE INDEX a_form_id1 ON "a_form" ("id1");
CREATE INDEX a_form_dosid1 ON "a_form" ("dossier","id1");
CREATE INDEX a_form_etr ON "a_form" ("dico","etr","rub");

END TRANSACTION;
```

Ora2pg en action

Il est extrêmement difficile de donner des chiffres et encore plus de donner des informations sur l'utilisation d'Ora2Pg et donc des migrations Oracle vers PostgreSQL. Il semble que très peu d'entreprises communiquent sur ce type de projet.

Cependant les échanges avec les utilisateurs font apparaitre trois types d'utilisations de cet outil :

- Test de PostgreSQL.
- Réplication de tout ou partie de base données Oracle vers une base PostgreSQL. Principalement dans le cadre de données Intranet publiées sur Internet.
- Migration effective de base de données.
- Reverse engeneering.

Pour ce qui est des entreprises et organisations utilisant Ora2Pg il s'agit principalement de SSII proposant de l'Open Source, donc des migration vers PostgreSQL, mais aussi des universités et des organisations gouvernementales

comme des ministères.

Ora2Pg génére 1100 à 1200 visites par mois sur son site d'hébergement à l'adresse : http://www.samse.fr/GPL/ora2pg/.

Lors des sorties des nouvelles versions le nombre de visite monte à 1500 voire 1600.

On peut en déduire qu'il y a actuellement 400 à 500 utilisateurs fidèles d'Ora2Pg dont principalement des SSII vu la cible de cet outil.

Conclusion

Si Ora2Pg sert à encourager les directions informatiques à faire le pas à moindre frais d'un portage de leurs applications "maison" sous base de données PostgreSQL, il aura atteint son but.

S'il sert aux éditeurs de solutions logicielles à proposer à leurs clients une alternative de base de données Open Source, c'est encore mieux !

Remerciements

Je tiens particulièrement à remercier tous les développeurs qui ont participer à la conception et la réalisation de PostgreSQL et ce depuis le début (Postgres95). Merci pour cette perle rare !

Le Groupe SAMSE et ses DBA Oracle Josian Larcheque et Stephane Silly pour avoir mis à ma disposition des bases Oracle et leurs connaissances de ce SGBD.

Ainsi que tous les contributeurs au projet Ora2Pg :

Stephane Schildknecht
Jean-Paul Argudo
Jan Kester
Paolo Mattioli
Mike Wilhelm-hiltz
Jefferson Medeiros
Ian Boston
Thomas Wegner
Andreas Haumer
Marco Lombardo
Adam Sah and Zedo Inc
Antonios Christofide and National Technical University of Athens

Un grand merci à eux !