

PostgreSQL : deux projets satellites

pgloader et prefix

Dimitri Fontaine

29 janvier 2008

Organisation du projet PostgreSQL

core, contribs et pgfoundry

- 1 Développement central, -core
- 2 Projets proches du coeur, contribs
- 3 Projets annexes, pgfoundry ou sourceforge

Quelques projets pgfoundry (1/2)

- PostgreSQL Installer
- phppgadmin
- skytools, pgbouncer, plproxy
- pgcluster et cybercluster

Quelques projets pgfoundry (2/2)

- Npgsql
- pgpool et pgpool-II
- pl/java, pl/sh, pl/lua
- pgfouine
- pgloader et prefix

PgLoader

Présentation

Outil ETL : *Extract Transform Load*

Extract Formats texte « simple », CSV, et *blobs*

Transform Ordre des colonnes, sélection d'une partie seulement des colonnes, reformatage à la volée, etc

Load Utilisation de COPY avec gestion d'erreur par dichotomie

Chargement de données

Formats texte et CSV

- 1 lecture de lignes « *physiques* » d'un fichier plat
- 2 constitution de lignes « *logiques* »
- 3 traitement de ces lignes
 - 1 restrictions de colonnes,
 - 2 ordre des colonnes,
 - 3 ajout de constantes,
 - 4 reformatage
- 4 COPY tous les commit_every lignes *logiques*

Historique

- La version actuelle de `pgloader` est une réécriture en *python* d'un projet initialement en *TCL*.
- Développée pour répondre à un besoin précis : import des données issues d'une base Informix.
- Proposé ensuite sur `pgfoundry` où le projet évolue depuis
- Utilisé aujourd'hui dans le cadre de migrations depuis MySQL.

Fonctionnement Général

- Fichier de configuration au format INI, très souple

```
only_cols    = 1-3, 5
columns      = x, y, a, b, d :6, c :5
reformat     = dt_cx :mysql :timestamp
```

- Suite à une demande d'un utilisateur, la verbosité est facilement contrôlable

Exemple d'utilisation

Depuis le répertoire des exemples fournis :

```
examples # pgloader -Tqsc pgloader.conf
```

```
examples # pgloader -Tqsc pgloader.conf cluttered
```

Souplesse du format texte

pgloader sait lire des formats de fichiers de données *torturés*

- 1 lignes *logiques* contenant des retours chariots non échappés (`field_count`)
- 2 fichiers dont les lignes se terminent par le séparateur de champs (`trailing_sep`)
- 3 traitement des échappements arbitraires des fins de lignes au sein des données (`newline_escapes`)

Gestion des erreurs

- Le comportement de COPY
- Continuer en cas d'erreur
- 2 fichiers de rejets
 - 1 journaux d'erreurs
 - 2 données

```
reject_data = /tmp/example.rej  
reject_log  = /tmp/example.rej.log
```

Colonnes utilisateurs

`pgloader` peut ajouter du contenu constant à la volée. Use Case :

- fédération de données, importées par *batch* sur un serveur central
- 1 fichier par serveur d'origine
- ajout d'une colonne serveur en fonction du fichier (`pgloader.conf`)

Reformatage à la volée

- Support de modules utilisateurs de reformatage
`reformat_path =`
`. :/home/dim/.../pgloader/reformat`
- `def bar(reject, input) :`
- `reject.log(messages, data = None)`
- Exemple : réécriture à la volée des Timestamp MySQL

Exemples fournis

Tests unitaires

pgloader est fourni avec un répertoire d'exemples

- `examples/README`
- `examples/pgloader.conf`

Les exemples servent de jeux de tests !

Indexer les recherches par préfixes

Un exemple valant mieux qu'un long discours :

```
SELECT * FROM table  
WHERE colonne LIKE 'prefix%';
```

```
SELECT * FROM table  
WHERE prefix @> 'recherche';
```

Objectif

On recherche le plus grand préfixe en base pour un numéro de téléphone donné :

```
SELECT *  
FROM   prefixes  
WHERE  prefix @> '0123456789' ;
```

On veut pouvoir utiliser un index *spécialisé* pour accélérer la recherche.

GiST

- Opérateurs @> (*contient, est un préfixe de*),
COMMUTATOR = '<@'
- 7 méthodes à implémenter
consitent, penalty, union, picksplit,
same, compress, decompress
- Association à un ensemble d'opérateurs via une
opclass
- Compilation du module avec pgxs
- Installation dans la base de données cible par
chargement de fichier SQL (ordres DDL)

Création de l'index

Cela se fait tout simplement :

```
CREATE OPERATOR CLASS gist__prefix__ops FOR  
TYPE text USING gist ...
```

```
dim=# CREATE INDEX idx__prefix ON prefixes  
USING gist(prefix gist__prefix__ops) ;
```

Particularités

et difficultés

Indexer des préfixes est relativement simple

- `consistent(a, b) == a @> b`
- `union()` est le plus grand préfixe

Deux fonctions difficiles

- `penalty()`
- `picksplit()`

La communauté PostgreSQL

- Une communauté très active
- Aide extraordinaire sur IRC, *#postgresql*
- Par l'auteur du module IP4R (types de données et *opclass*) en particulier

Questions ?

Et n'oubliez pas, le stand PostgreSQL vous accueille ;)