

Extrait du Easter-eggs - Spécialiste GNU/Linux

<http://www.easter-eggs.com>

Retour sur l'optimisation des performances d'une base de données PostgreSQL

- Études de cas - Développement applicatif -

Date de mise en ligne : vendredi 27 février 2009

Easter-eggs - Spécialiste GNU/Linux

Sommaire

- [Problématique](#)
- [Optimisation de la base \(...\)](#)
- [Javascript](#)
- [Systèmes de Cache](#)
- [Réduction de la taille des \(...\)](#)
- [Outils utilisés](#)
- [Liens utiles pour l'optimisatio](#)

La base de données PostgreSQL du projet [MAVISE](#) fournit les données de base sur l'ensemble du marché audiovisuel accessible dans l'Union européenne. Elle a été développée par Easter-eggs en collaboration avec l'Observatoire Européen de l'Audiovisuel, pour le compte de la direction générale de la communication de la Commission européenne. Nous allons décrire dans cet article les différentes méthodes que nous avons mises en place pour optimiser les performances de cette application.

Problématique

La base de données contient un grand nombre d'informations :

- la description du paysage audiovisuel de 30 pays ;
- des fiches sur 5 000 chaînes de télévision ;
- des fiches sur 4 000 entreprises ;
- des fiches sur 8 000 programmes de télévision ;

L'ensemble de ces données ne sont pas accessibles au grand public.

D'un point de vue technique, la base de données PostgreSQL contient :

- 115 tables ;
- 100 vues ;
- 220 Mo de données.

Cela donne une petite idée de la taille de l'application.

La complexité de l'application, la quantité de données disponibles et leur affichage ont soulevé certaines problématiques pour gérer les transferts entre le client et le serveur Web et les requêtes complexes.

Optimisation de la base de données

Pour l'optimisation de la base de données, il faut être vigilant sur les points suivants :



L'[ORM \(DB DataObject\)](#) dans notre cas) ne doit faire aucune opération de jointure . Il utilise toujours une vue dédiée pour chaque fonctionnalité ;

- ▶ Chaque vue ne calcule que les champs qui sont nécessaires à l'affichage ;
- ▶ Le plan d'exécution de chaque vue est optimisé à l'aide d'index. Pour trouver quels sont les index à créer, il faut utiliser l'instruction EXPLAIN ANALYSE , comme décrit sur le wiki de Postgres : [Using EXPLAIN](#).

Le travail principal consiste donc à exécuter toutes les vues de l'application pour optimiser les plans d'exécution à l'aide d'index. Pour mieux comprendre la sortie de l'instruction EXPLAIN ANALYSE, nous utilisons l'outil suivant : [explain-analyze.info](#).

Pour savoir quelles sont les optimisations les plus pertinentes, il est intéressant d'analyser les logs du serveur PostgreSQL à l'aide du logiciel [Practical Query Analysis](#).

Il faut aussi "tuner" la configuration du serveur de base de données pour utiliser le maximum de la mémoire disponible sur le serveur.

Javascript

Nous avons utilisé le framework Javascript [Mootools](#), le contrôleur de tableau [dhtmlxGrid](#) et l'éditeur de contenu [TinyMCE](#).

Dans chaque page de l'application, il faut veiller à ne charger que les fichiers nécessaires. En effet, ces outils, bien que très pratiques, sont très volumineux et augmentent la taille des pages dans des proportions importantes.

Dans certains cas, nous avons dépassé plus de 640 Ko de code Javascript dans une page !

Les solutions que nous avons mises en place pour réduire la taille de ces fichiers :

- La compression Gzip, avec laquelle nous avons malheureusement rencontré des problèmes très importants avec Internet Explorer ;
- Nous avons ensuite opté pour [YUI Compressor](#) qui fonctionne bien mieux .

Systèmes de Cache

L'application dispose de nombreux systèmes de cache :

- Le système de cache du gestionnaire de template [Smarty](#) ;
- Notre propre système de cache pour le contenu des tableaux, réalisé à l'aide du module [Pear Cache Lite](#) ;
- L'[API de XCache](#), pour certains calculs et [XCache](#) pour les opcodes PHP ;
- Le système de cache d'[Apache Cocoon](#) pour le système d'impression ;
- La cache du navigateur des utilisateurs. Il est possible de bien contrôler le contenu du navigateur des clients à l'aide du module [mod_expires](#) d'Apache.

Réduction de la taille des pages

Nous avons travaillé pour réduire au maximum la taille des échanges entre le navigateur web et le serveur. Pour cela nous avons utilisé la compression Gzip sur :

- Le contenu des tableaux ;
- Les pages produites par le système de template ;

Celle-ci est réalisée par le gestionnaire de tampon de PHP, à l'aide de la fonction [ob_gzhandler](#) .

Outils utilisés

L'optimisation d'une application est toujours empirique, il est nécessaire d'avoir de bons outils pour observer les effets des modifications. Pour cela nous avons utilisé :

- Le module [Firebug](#) pour Firefox ;
- Et son add-ons [YSlow](#).

Liens utiles pour l'optimisation des performances d'une application Web

Pour finir, voici quelques liens très intéressants sur l'optimisation d'une application Web :

- Le blog [Performance web](#) ;
- Sermon de l'équipe de Yahoo : [Exceptional Performance](#).