



POSTGRESQLを使ったチラシ・口コミサイトの構築・運営事例

—「とっとくねっつ」とPostgreSQLのかかわり—

オールクリエイター株式会社 橋本俊秀

電子チラシ&ロコサイト「とっとくねっ」について

- 会員2万人超の電子チラシ&商品クチコミサイトです。
<http://www.tottoku.net> (PC/携帯共通)
- 日本初の携帯3社対応電子チラシサイト(当社調べ)
- 2003年夏ごろより、橋本俊秀(筆者)が開発を開始。
2005年8月、アシストV株式会社のASP事業として商用準備。2006年1月実運用開始。同年春Vodafone(現softbankmobile)、au公式サイトとなる。2008年11月7日、アシストV株式会社より オールクリエイター株式会社へ運営会社変更。現在に至る。



なぜデータベースにPOSTGRESQLなのか？

- サーバーOSは、筆者の好みで、Linux！
- サーバーサイド開発言語は実績あるPerl5。
- 偶然、購入したPerl/CGI本に、PostgreSQL専用のPerlモジュールPgの使い方が手厚く載っていた。
- 筆者は昔、仕事でOracleとPostgreSQLのパフォーマンス比較をしたことあり
→ なので、“と・り・あ・え・ず”
PostgreSQLで仮決定でスタート！。。。ちょっと不安なので、標準SQLで作りこむ。
- その後、言語をPerlからPHPへ、LinuxをRedHatからVineへ変えたが、DBは浮気しなかった！



活用したテクニックとその功罪。。。。

- シリアル型の活用、タイムスタンプの内部取得
→ プログラム生産性を向上させてくれた。。。が、その後、pgpoolをつかう際の障害になってしまった。結局pgpoolはあきらめた
(T T)
- 標準SQL&シンプルSQLの組み合わせ
→ 標準SQLに従った短いSQL 文の組み合わせで、トランザクションを構成したので、見やすいソースにはなったが、サイトが活発になるにつれ、レスポンスの遅さが問題になってきた。
→ 別の開発メンバの手で、(1) 欲しい結果を1回のSQL発行で済ませるよう、joinを活用し、ソースを全面見直した。(2) window関数やPostgreSQL独自の文法を使い、DBの力をなるべく引き出すソースに改造した。(3) 複雑なSQL、対象データの多いSQLを組み込む時は、**explain analyze select * from....** をつかって、コストをかならず意識してコーディングするようになった。



○ カーソル関数とVIEW

→セオリーの的にカーソル関数をつかった。また筆者の経験からVIEWを使ってソースを見やすくした。。。つもりだった。しかし、やはり運用規模が大きくなるにつれ、性能の問題がでてきた。

→(1)カーソルはデータ件数が多い時、オーバーヘッドが大きかったので、カーソル関数を避ける改造をした。(2)VIEWは追加、更新、削除ルールを別に書いてDB内に収めてしまうので、ボトルネック調査で問題点が見つけずらかった。ので、追加のコードからはVIEWは使わない方針となった。



- JOINの使い方

→JOINをなるべく使うようにしたが、所々で、JOIN

のあるSQLの処理が重たくなった。調査したところ、

JOINする対象が4000件以上あると、コストがかかることが判明したので、JOINさせる時には、なるべく絞り込んでからJOINするようになった。

- テーブル定義時の厳密なカスケード指定

→最初のDB設計した時は、筆者1一人がテーブル間の関連を整理して、**cascade update, cascade delete**指定して定義していた。しかし、時間が進むにつれ、複数の開発者により諸々の追加改造が入り、テーブル数も当初の数
十テーブルから100をかるく超えるレベルになってきて、テーブル間の関係が複雑になってきた。

→実際に、運用中に、とある画像データを消したとき、関連がないと思われていた店舗テーブルのデータが消えた現象が発生してから、**cascade**
とくに **cascade delete**は指定なくなり、代わりに、定期的に
クリーンアッププログラムを実行するようになった。



後で別システムと繋いだ時のエピソード

- 事後的に、オープンソースの決済系システムを導入して、連携させることにした。が、それは、**Perl**、**SJIS**、**TextDB**というもので、簡単に移植できなかった。

そこで、私たちのとった対処策とは。。。。

- **database**を、**UNICODE**で生成しなおし、既存系はEUCで、決済系は**SJIS**のまま使えるようにした。
- 決済のTextDBのテーブルと同等のものを**create**した。
- TEXTDBの文法がSQLライクだったので、TEXTDBにアクセスする部分のインターフェイス・プログラムを書いて、実際には、PostgreSQLのデータベースへアクセスするように、改造した。
- 既存のテーブルと、決済系テーブル群とのデータ連携には、**PL/pgSQL**を使いデータベーストリガーで実現した。。。但し、そのあと苦勞することになった。汗；



POSTGRESQL 7から8へ変えた際のエピソード

- システムカタログ

当初のターゲットは、PostgreSQL7で、「PostgreSQLオフィシャルマニュアル」(インプレス)を参考にして作った。その際、システムカタログを参照している箇所が少なからずあった。が、その後、PostgreSQL8に上げる際の障害になった。最終的には、システムカタログ情報は一切見ない形に変えた

- バックアップとリストア

7から8への移行時、8系のDBサーバ(更新系)と7系のDBサーバ(参照系)が両方存在していたことがあった。で、7系で**binary**バックアップをとっていたため、8系でのリストアに失敗した。当たり前(汗汗;;)で、参照系DBもあわてて8系にあげる一幕があった。



そのほかのエピソード

- DBの複数台化、バックアップでバタバタ。。。一時期、PGクラスターを使っていた時期があったが、なぜかPGクラスターのサーバが時々止まった&文字化けを起こしていた。で、監視ツールを作るはめに。そして、レプリケートにずれが生じているのは見つかった。原因究明に手間取りそうだったので、結局、PGクラスターは使わなくなり、高性能DBサーバへの切り替えと、`pg_dump`の毎日実行にて対処した。※これは、たぶん、私たちの同ソフトの使い方に問題があったのだと思っています。誤解なきよう。



これから導入する方へのアドバイス？！

- 業務的なテーブル設計は、ほどほどに。。。→オープンソース系のCMSがやっているみたいに、キーには業務的な意味をもたせず、単なる機械的な一意キーとしておき、業務キーは別項目で非キーとして持つ方が、Web系システムだとラクかもしれませんよ。
- テーブル連携は、ほどほどがよし。
- サルでもできる性能対策を1つ→開発・試験は、「ぼろい」(=引退寸前の)サーバーで行い、そこで**explain analyze**してそこそこの値だったら、自信をもって「高性能」の運用サーバにリリースする。



最後に

- 弊社「オールクリエイター株式会社」について (<http://allcreator.net>)は、オープンソースCMSを使い、携帯サイト構築のお手伝いをするたテクニカル・ディレクティング専門会社です。CMSを活用した携帯サイト構築の際は、ぜひご相談くださいませ。

・謝辞

こんな素晴らしいオープンソース・データベースを作り上げたPostgreSQLの開発者、ユーザー会メンバに感謝します。また、PostgreSQLを使い「とっとくねっ」を開発運用するにあたり多大なる尽力をいただきました、アシストV株式会社市川晴夫社長、二瓶修さん、荒木秀之さん、山崎拓郎さんはじめ同社の皆様、コンテンツ運営に携わっていただいたスタッフの方々に、深く深くお礼申し上げます。

編集責任： オールクリエイター株式会社 橋本俊秀

編集協力： オールクリエイター株式会社 藤原りか

