Bisoños Usuarios de GNU/Linux de Mallorca y Alrededores | Bergantells Usuaris de GNU/Linux de Mallorca i Afegitons

Reducción del tiempo de respuesta de Bulma optimizando las consultas a Postgres (12862 lectures)

Per **Ricardo Galli Granada**, ***gallir*** (http://mnm.uib.es/gallir/)

Creado el 19/04/2001 13:02 modificado el 19/04/2001 13:02

*Aunque ya habíamos optimizado bastante los select usados para generar las páginas de Bulma, cuando me puse a probar la función microtime() del PHP para medir el tiempo (real, de reloj) necesario para generar las páginas encontré que los resultados eran "inaceptables" desde mi punto de vista (**0.3** segundos para la página índice). En este artículo comento brevemente los tres trucos que permitieron bajar los tiempos a casi un tercio del original.*

Los primeros resultados mostraban que el tiempo necesario para generar la página índice (index.phtml) eran de unos 0.28 a 0.31 segundos. La generación de las páginas de artículos (body.phtml) necesitaban de **0.08** a **0.11** segundos. Evidentemente se estaba gastando un tiempo precioso en consultas a la base de datos.

La solución consistió en:

1. Reducir al mínimo el ORDER BY en las clausulas SELECT: redujo el tiempo en unos 0.13 segundos.
2. Usar el *pg_fetch_array()* en vez de hacer un ciclo para leer cada uno de los campos con el *pg_result()*: redujo el tiempo en 0.04 - 0.05 segundos.
3. Evitar leer los contenidos de los campos en el Recordset->Exec si el SQL no es un SELECT: redujo el tiempo en 0.01 segundos aproximadamente.

Con estas simples técnicas hemos podido reducir el tiempo empleado en la página índice a unos **0.11** segundos (una reducción de más de un 60%) y a **0.06** segundos en las páginas de los artículos (una reducción de un 30% aproximadamente).

NOTA: tened en cuenta que hablamos de tiempo total usado, en horas "reloj", y no de tiempo de CPU usado, ya que la mayoría de uso de CPU para las consultas a la base de datos se consume en los procesos del Postgres y no en el PHP de los procesos del Apache.

A continuación explico en qué consistió cada uno de los cambios.

Reducción del ORDER BY

cada vez que se muestra una lista de artículos de las distintas secciones, usábamos un doble ordenamiento en la cláusula SELECT:

```
ORDER BY
  fecha_modificacion_noticia desc,
  id_noticia desc
```

Como se observa, ordenábamos por fecha de modificación (un TIMESTAMP) y luego por el identificador de noticia, siendo éste último una clave única ascendente. El doble ordenamiento no tenía sentido desde el momento que empezamos a usar TIMESTAMP, porque es prácticamente imposible que dos artículos tuviesen la misma fecha y hora de modificación, por lo que el ordenamiento por *id_noticia* no tiene efecto. Y aunque lo tuviese, no es importante si el primero sale inmediatamente después del segundo o viceversa.

La solución fue quitar el ordenamiento por *id_noticia* en todos los SELECTs y dejarlos de la siguiente forma:

```
ORDER BY
  fecha_modificacion_noticia desc
```

Sólo con estas modificación se redujeron los tiempos en un 40% aproximadamente.

Moraleja: el Postgres es bastante ineficiente (comparado a otras bases de datos, incluida MySQL) a la hora de ordenar los resultados. Por lo tanto conviene minimizarlos e intentar en lo posible que tengan un índice que ayude al Postgres (en versiones anteriores a la 7.1 no surge efecto).

Uso del *pg_fetch_array()*

La abstracción de base de datos implementada en recordset.php almacena los resultados de los campos de la consulta en un array del tipo *hash* que permite la lectura de los resultados mediante una simple instrucción:

```
$rs->fields["nombre_campo"]
```

Para rellenar los campos de *fields* después de un SELECT, se usaba el siguiente código:

```
if (!$this->Eof()) {
  for ($i=0; $i < pg_NumFields($this->rs); $i++) {
    $strFieldName = pg_FieldName($this->rs, $i);
    $this->fields[$strFieldName] =
      pg_Result($this->rs, $this->nActRow, $strFieldName);
  }
}
```

Es decir, se recuperaban los datos uno a uno y se almacenaban en la posición correspondiente de *fields*. Esto era de más ineficiente ya que las funciones de PHP4 para Postgres ya tienen una función que permite hacer eso con sólo una llamada:

```
$this->fields = pg_fetch_array($this->rs, $this->nActRow);
```

Con este cambio se redujeron los tiempos en un 17% (0.04 a 0.05 segundos) aproximadamente.

Moraleja: usad las funciones nativas de base de datos y evitad los ciclos de lecturas de datos.

No intentar rellenar los campos si el SQL no es un SELECT

La función Exec del recordset.php ejecuta una instrucción SQL y luego carga los resultados en el array *fields* (según lo explicado en el apartado anterior). Pero no tiene sentido hacerlo con SQL que no sean SELECT ya que el Postgres retorna siempre un error. Para ellos agregué una simple expresión regular que se asegura que el SQL del Exec haya sido siempre un SELECT:

```
if (eregi("^[ \n\r\t]*select", $strQuery)) {
  ...
  $this->MoveNext();
  ...
```

La función *eregi()* (expresión regular POSIX, insensitiva a mayúsculas-minúsculas) lo que hace en nuestro caso es verificar que la primera palabra del SQL haya sido un SELECT. Los primeros caracteres de la expresión regular permite que aparezcan (cero o más) espacios, tabulaciones y *newlines* antes de la palabra, esto es muy importante ya que es bastante común el uso de esos caraceteres durante la programación. Ejemplo:

```
$con->Exec("
        SELECT xxx, yyyy
...
```

En este caso el tiempo no se redujo considerablemente debido a que la mayoría de los SQL para la generación de las páginas son SELECT, a excepción de la lectura de los artículos donde se hace un UPDATE para incrementar el contador de lecturas y en la inserción de artículos y comentarios.

E-mail del autor: gallir _ARROBA_ uib.es
Podrás encontrar este artículo e información adicional en: http://bulma.net/body.phtml?nIdNoticia=611