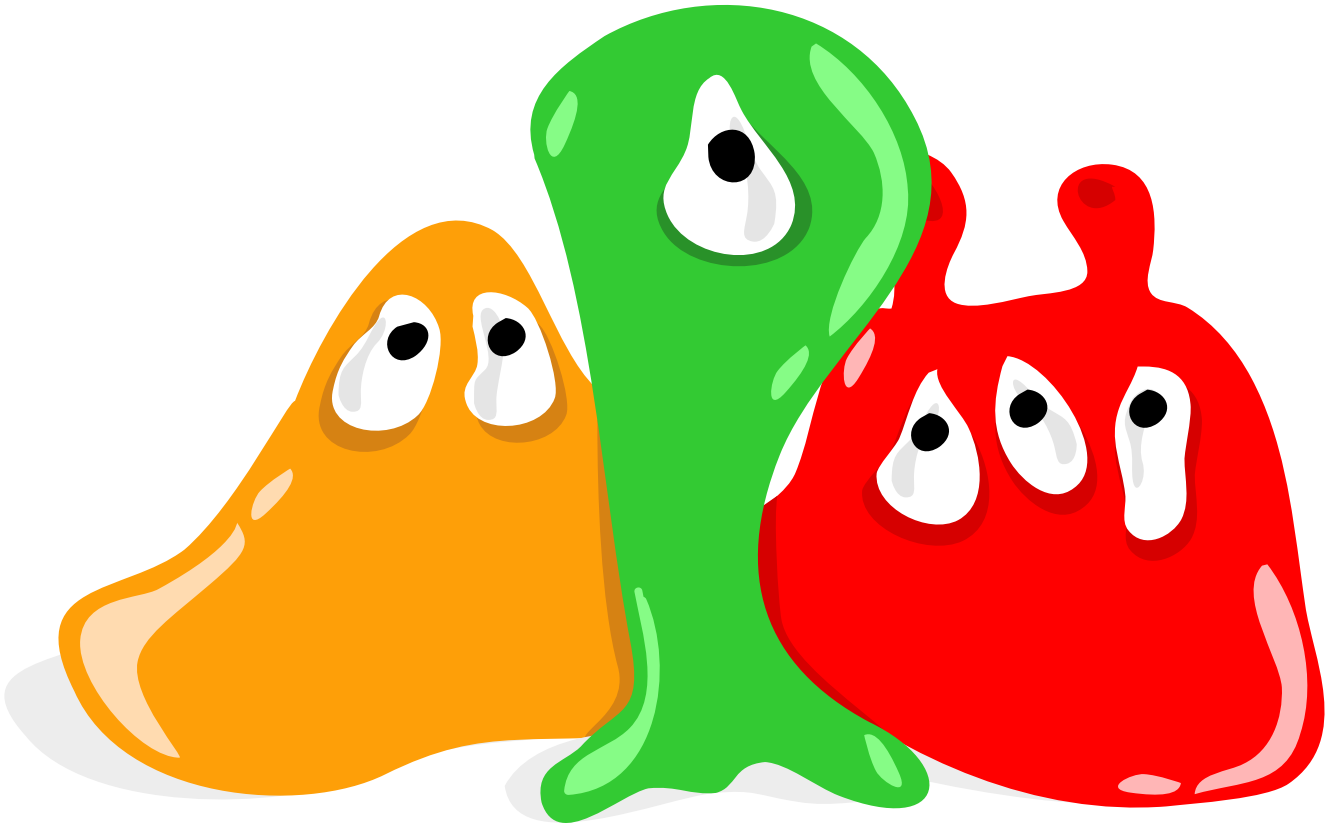
Сентябрь '04 № 6

PHPInside

электронный журнал для веб-разработчиков

Иные среди нас!



Oracle, Firebird, DB2, MS SQL и др.

Содержание

Обратная связь

Всем привет!

Еще не так давно на экранах отечественных кинотеатров отгремел блокбастер «Ночной дозор», согласно сценария которого среди людей встречаются «иные» - тоже по своей сути люди, но с другими возможностями. Вот и мы решили провести совершенно случайную параллель: а что, если в мире СУБД, используемых для веб-приложений, тоже существуют «иные» базы данных, отличные от знакомой нам MySQL. О некоторых из них и пойдет речь в этом номере.

Также в этом первом осеннем выпуске мы напомним вам (а если вы не знали, то расскажем) о том, что журнал PHP Inside совместно с некоторыми компаниями и интернет-ресурсами проводит конкурс статей «PHP Inside Art Contest» целью которого является появление на свет новых статей и авторов. Победителей и просто отличившихся ждут призы и «паблисити» - то, что мы называем «информационным освещением». Приняв участие в конкурсе, вы также поддержите наш журнал и поможете его развитию.

Читайте PHP Inside и пишите статьи!

Конкурс статей «PHP Inside Art Contest»

Конкурс «PHP Inside Art Contest» представляет собой состязание авторов статей, посвященных технологии веб-разработки PHP и связанным с ней технологиям (СУБД, XML, веб-серверы). Конкурс является открытым, и участие в нем может принять любой человек, имеющий возможность написать на любую из указанных тем на русском языке, вне зависимости от географии его проживания, возраста и рода деятельности. Прием статей на конкурс осуществляется с 20 сентября по 31 октября 2004.

Приглашаем вас. Проявите себя и получите призы! (http://phpinside.net/ac_index.php)

Номинации и призы

Журнал для веб-разработчиков «PHP Inside» и генеральный спонсор мероприятия компания ARBATEK (http://www.arbatek.ru), совместно с http://linuxcenter.ru и интернет-биржей проектов http://weblancer.net, учредили следующие призовые номинации и призы:

- «Авторские материалы: общие вопросы PHP». В эту номинацию попадают все авторские статьи о PHP и расширениях. Приз за первое место в данной номинации: три года хостинга от ARBATEK по тарифному плану «Элита» и годовая регистрация домена второго уровня, сертификат номиналом 1000 рублей, на которые можно купить любой товар в онлайн-магазине Линукс-Центра, и полугодовой аккаунт на интернет-бирже проектов weblancer.net со статусом «Профессионал».
- «Авторские материалы: PHP и смежные технологии». В эту номинацию попадают все авторские статьи о смежных с технологиях, таких как, например, СУБД (sql, проектирование, оптимизация), веб-серверы, программы и утилиты для PHP-разработчиков. Приз за первое место в данной номинации: три года хостинга от ARBATEK по тарифному плану «Элита» и годовая регистрация домена второго уровня, сертификат номиналом 1000 рублей, на которые можно купить любой товар в онлайн-магазине Линукс-Центра, и полугодовой аккаунт на интернет-бирже проектов weblancer.net со статусом «Профессионал».
- «Переводные материалы». В эту номинацию попадают все переводы, вне зависимости от их конкретной тематики. Приз за первое место в данной номинации: три года хостинга от ARBATEK по тарифному плану «Элита» и годовая регистрация домена второго уровня, сертификат номиналом 1000 рублей, на которые можно купить любой товар в онлайн-магазине ЛинуксЦентра, и полугодовой аккаунт на интернет-бирже проектов weblancer.net со статусом «Профессионал».

Каждый приз от компании ARBATEK эквивалентен примерно 350 у.е. Каждый приз от weblancer.net эквивалентен примерно 60 у.е. и открывает пользователям следующие возможности их сервиса:

- Общение с заказчиками в форуме проектов;

- Возможность видеть предложения (цена, сроки, комментарий) других разработчиков;
- Предложения профессиональных разработчиков располагаются выше предложений всех остальных разработчиков;
- Возможность участвовать во всех проектах, включая проекты с ограничением по статусу разработчиков.

Что еще, помимо призов?

Конкурс достаточно широко освещается в тематических электронных СМИ, поэтому участие в нем может прибавить известности его участникам. Имена победителей и других лучших участников конкурса будут обязательно опубликованы во всех наших финальных пресс-релизах (аудитория информационных спонсоров свыше 200 000 уникальных посетителей), победители и участники получат кнопки «Победитель конкурса PHP Inside Art Contest» и «Участник конкурса PHP Inside Art Contest» для своего сайта.

Для специалистов, возможно, отличие в конкурсе PHP Inside Art Contest будет дополнительной строкой в их резюме, а для компаний предоставляется еще одна возможность продемонстрировать клиентам уровень своих специалистов.

Как принять участие?

Если вы решили написать статью для конкурса, или у вас есть не публиковавшийся ранее в интернете и бумажной прессе материал, или вы готовы перевести интересную англоязычную статью, регистрируйтесь на нашем сайте (http://phpinside.net/ac_reg.php), подготавливайте статью (или статьи) и присылайте на адрес: artcontest@phpinside.net. Победители будут объявлены в ноябре 2004 года.

Благодаря кому этот конкурс проводится?

Генеральный спонсор конкурса, хостинг-провайдер ARBATEK был образован в 1999-м году. В 2003-м году компания объединила в себе несколько самостоятельных подразделений: 350mb.ru, HostPro.ru и Todos.ru, специализирующихся на виртуальном хостинге, аренде серверов и реселлерских услугах. Компания предоставляет практически полный спектр сервисов, известных на мировом рынке хостинга. На настоящий момент компания ARBATEK входит в двадцатку крупнейших хостинг-провайдеров России по числу зарегистрированных доменов в зоне .RU. ARBATEK сегодня — один из лидеров на рынке веб-хостинга, надежный партнер и сложившаяся команда профессионалов, ценящая каждого клиента!

Главная задача ЛинуксЦентра — продвижение операционной системы Linux в России. На сайте работает новостной канал, библиотека Линукс Центра, в рамках проекта создается уникальная «Виртуальная Энциклопедия Linux».

Линукс Центр издает в России дистрибутивы Linux, FreeBSD, NetBSD, OpenBSD, а также соответствующее программное обеспечение, игры под Linux, книги и аттрибутику. Также Линукс Центр распространяет в России коробочные продукты от компаний Линукс ИНК, Red Hat, Inc., SuSe Linux AG, ASPLinux и ALTLinux. Наши продукты продаются через дистрибьюторскую сеть (см. www.linuxcenter.ru/mag.phtml), партнерскую сеть фирмы 1С, интернет-магазины Ozon.Ru и Books.Ru, и конечно, через собственный интернет-магазин.

В интернет-магазине Линукс Центра собираются все достойные внимания дистрибутивы Linux, а также соответствующее ПО и обучающая литература. Все товары с виртуальной витрины есть на нашем складе. Доставка почтой по всей России. Оплата наложенным платежом, банковским переводом, WebMoney.

Сервис http://weblancer.net предназначен для предоставления пользователям Интернет удобного автоматизированного интерфейса, позволяющего пользователям (заказчикам и разработчикам) устанавливать контакты для выполнения различного рода проектов (работ, заданий).

Проект SOFT@Mail.Ru — результат сотрудничества портала Mail.ru и поставщика программного обеспечения компании SoftLine (http://www.softline.ru). Он представляет собой крупный каталог условно-бесплатных и бесплатных программ. Содержание сайта предлагает посетителям максимум возможностей для осуществления наилучшего выбора программ из более чем 8000 вариантов. Размещение своих продуктов и информации о них на SOFT@Mail.Ru доступно для любого независимого разработчика программного обеспечения при соблюдении условий законодательного и технического характера. Авторы могут получать доход от продажи условно-бесплатных программ, подписав договор сотрудничества.

«Сисадмин тоже человек». История этого проекта началась со странички конкурса, посвященного дню системного администратора. Прошло время, и страничка конкурса превратилась в сайт сообщества системных администраторов.

PHP Club — сообщество веб-разработчиков, которое существует уже более 5 лет. Задачи клуба — популяризация языка PHP и повышение качества проектов написанных на этом языке.

Linux.ru. Ежедневные новости об операционной системе Linux по-русски. Статьи, обзоры и документация. Форум.

Русские документы — компьютерная библиотека. Свежие статьи русских компьютерных изданий ежедневно.

Также вы всегда сможете найти информацию о конкурсе на сайтах: http://www.rusc.ru и http://forum.woweb.ru (специальный топик в разделе PHP).

Отдельное спасибо Зеньковичу Денису (dizet@newmail.ru) за разработку логотипа и кнопок конкурса.

Состав жюри

- Войцеховский Александр, сотрудник компании ООО «Ди Джи» (г. Киев), ведущий сайта http://detail.phpclub.net/, член Team PHPClub, один из авторов русскоязычной документации: http://www.php.net/manual/ru/, автор статей: http://detail.phpclub.ru/author/young.

- Календарев Александр, специалист в области организации B2b, независимый разработчик (г. Санкт-петербург). Член Theam PHPClub с 2002 года, организатор проекта www.edocs.phpclub.net Статьи в печати: журнал «Открытые системы», «Программист». В интернете на сайтах: Королевство Дельфи, www.citforum.ru, www.edocs.phpclub.net, www.xmlportal.ru.

- Олищук Андрей, сотрудник компании «Аскон» (компания разработчик ведущей отечественной САПР «Компас», г. Коломна, Московская область), координатор проекта «Журнал для веб-разработчиков PHP Inside», координатор данного конкурса статей. Статьи в интернете: в журнале PHP Inside, на сайте http://detail.phpclub.ru.

- Чаплыгин Антон, в данный момент является сотрудником компании Verysell-6 (бывшая V6 Technologies, г. Москва). Энтузиаст движения Open Source. Занимается веб-технологиями, системным анализом, менеджментом проектов и IT-консультациями. Участвует в независимых проектах: Книга о программировании на языке Python (http://book.it-arts.ru/), журнал PHPInside, член редакции с 1-го номера. Является автором оригинальных учебных курсов по информатике и программированию для школьников, использованию и администрированию Linux и МСВС 3.0, а также соавтором учебника, посвященного OpenOffice.org. По выходным для души ведет компьютерный кружок в детском благотворительном фонде «Интеллект-Клуб».

Поддержка

Если у вас возникли дополнительные вопросы, то вы можете задать их редакции журнала по электронной почте nw@phpinside.net или по мобильному телефону московского региона, позвонив 8 (926) 560-25-74.

В фокусе

Жар-птица – кто ты?

Автор:
Александр Косарев

До недавнего времени под названием Firebird развивалось два проекта. Первый – СУБД, и второй – дочерний проект от Mozilla - лёгкий и быстрый браузер. Естественно, такая двоякость вносила путаницу в ряды новых пользователей, и, в конце концов, браузер переименовали в FireFox. Он является прекрасной альтернативой Internet Explorer. Для написания этой статьи я тестировал оба проекта под Win98, Win2k, Win2k3, Linux RedHat 9, FreBSD 4.9, 5.1, 5.2, и они показали себя с лучшей стороны.

Введение

Проект InterBase был начат в 1980 году. СУБД пережила долгий путь развития. В июле 2000 года компания Borland открыла код СУБД. Открытая тогда версия (6) доступна и сейчас. Она прекрасно работает на платформах Linux и Windows. Замечательно в этой СУБД то, что она не требовательна к ресурсам. В данный момент прекрасно справляется со всеми задачами на моём 166-ом пентиуме (64 RAM) на платформе FreeBSD. Читал также о её прекрасной работе на 75-ом пентиуме (платформа указана не была).

Для полноты картины добавлю, что проекты FireBird и InterBase являются «сестричками» и PHP между ними разницы не видит. Различие заключается в том, что компания Borland контролирует InterBase и версии выше шестой являются платными. В то же время открытый проект FireBird развивается по адресу: http://firebird.sourceforge.net.

Реализация

Для выполнения скриптов, приведённых в данной статье, вам придётся установить выбранную «сестричку» на свой сервер. Рекомендую Firebird. На http://firebird.sourceforge.net предлагаются две версии: классическая и «супер». С точки зрения PHP-разработчика нет разницы, с какой из них работать, а вот администраторам придётся почитать документацию и, как всегда, принять самое правильное решение.

На Unix-платформах вам придётся пересобрать PHP с опцией --with-interbase, выполнив следующие команды:

```
./configure --with-interbase=/opt/interbase
make
make install
```

Не забудьте при конфигурации указать и свои библиотеки.

В случае FireBird опция устанавливается та же. После сборки и установки PHP для полной поддержки работы с «Жар-Птицей» вам придётся поправить php.ini:

```
magic_quotes_sybase = On
```

Для счастливых пользователей Windows пересобирать абсолютно ничего не нужно, но тогда придётся проверить наличие библиотеки gds32.dll в системной папке SYSTEM32. При установленной «Жар-Птице» она там будет, ну а если не будет, то смело копируйте из папки dlls, находящейся в вашей корневой директории PHP.

По умолчанию логином для администрирования InterBase/FireBerd является SYSDBA с паролем masterkey – было бы неплохо их сменить, но об этом пусть болит голова у многоуважаемых администраторов. Для вас же достаточно будет просто найти утилиту gsec в bin папке СУБД (она поставляется в комплекте) и произнести:

```
gsec -user sysdba -password masterkey -add 'phptest' -pw 'phptest'
```

Перед gsec, конечно, необходимо указать путь к этой утилите. Теперь можно создавать базу данных – все таблицы СУБД хранит в одном файле. К сожалению, в этом расширении, команды, подобной `mysql_create_db()`, нет, поэтому придётся поднапрячься.

Создаём файл с содержанием:

```
SET SQL DIALECT 3;

CREATE DATABASE 'phptest.gdb'
PAGE_SIZE=8192
DEFAULT CHARACTER SET ISO8859_1;

CREATE TABLE ADDRESS
(
ID INTEGER NOT NULL,
NAME VARCHAR(100) NOT NULL,
CATEGORY INTEGER NOT NULL,
ADDRESS BLOB SUB_TYPE TEXT SEGMENT SIZE 100,
PRIMARY KEY (ID)
);

GRANT SELECT,DELETE,INSERT,UPDATE ON ADDRESS TO phptest;

commit;
```

Листинг 1. Файл create_db.sql

«Жар-Птица» показывает свой жар – она транзакциональна, поэтому обязательно наличие команды `commit`. Вообще-то по умолчанию СУБД произведет его самостоятельно, но только при безошибочном запросе, так как при наличии ошибки, откат затронет весь (!) запрос. Естественно, если файл БД создастся, то стираться он уже не будет, но вот создание таблицы и задание привелегий, могут откатиться.

Теперь нам придётся ещё раз воспользоваться консолью – утилита `isql` находится, опять же, в папке Bin. Итак, произносим:

```
isql -i create_db.sql -u sysdba -p masterkey
```

Подготовительные работы завершены, переходим к действию.

Запишем для начала конфигурационные параметры:

```
<?php
/*
Нужно показать полный путь к базе, при этом для windows c:\путь,
а для linux, естественно, /путь
*/
$base = 'c:\phptest.gdb';

//Если "Жар-Птица" находится на другом хосте
//то получаем xxx.xxx.xxx.xxx:/path
//можно, конечно, и так: examle.com:/path
//Имя пользователя по умолчанию
$user = 'SYSDBA';
//Пароль по умолчанию настоятельно рекомендую сменить
$pass = 'masterkey';

//существует также и ibase_connect - различие такое же как и в MySQL
$dbh = ibase_pconnect($base, $user, $pass);
//я предпочитаю сразу сообщать об ошибке
if ($err = ibase_errmsg()) die(__LINE__.':'.__FILE__.':'.$err);

?>
```

Листинг 2. Файл config.php

ВНИМАНИЕ!!! В этом примере подразумевается, что база phptest.gdb лежит в папке со скриптами – это неправильно, нужно её класть в какое-либо другое место, куда не прорвутся злые хакеры и, естественно, показывать полный путь к ней.

Излечение информации будет выглядеть так:

```
<?php
  include ('config.php');
  $sth = ibase_query('SELECT NAME,ADDRESS FROM ADDRESS ');
        if ($err = ibase_errmsg()) die(__LINE__.':'.__FILE__.':'.$err);

    ?><table border="1">
    <tr><td>Name</td><td>Address</td></tr>
    <?
        while ($row = ibase_fetch_row($sth))
        {
    ?>
    <tr><td>
    <?
                echo $row[0];
    ?>
    </td><td>
    <?
        ibase_blob_echo($row[1]);
    ?></td><?
        }
    ?></table><?

        ibase_free_result($sth);
        ibase_close($dbh);
?>
```

На данный момент наша база пустая, необходимо вставить информацию.

```
<?php
require_once('config.php');
if (isset($_REQUEST['ADDRESS'])
    && isset($_REQUEST['NAME'])
    && isset($_REQUEST['CATEGORY']))
{
    $blob_id = ibase_blob_create();
    ibase_blob_add($blob_id,$_REQUEST['ADDRESS']);
    $blob_id_str = ibase_blob_close($blob_id);

    $sth = ibase_prepare('INSERT INTO ADDRESS (ID,NAME,ADDRESS,CATEGORY)
VALUES (?,?,?,?)');
    $trans=ibase_trans();
    ibase_execute($sth,$_REQUEST['ID'],
    stripslashes(strip_tags($_REQUEST['NAME'])), $blob_id_str,$_REQUEST
['CATEGORY']);
    ibase_commit($trans);
    ibase_free_query($sth);
    ibase_close($dbh);
}
?>
<form method=post action="">
<table>
<tr>
        <td>ID</td>
        <td><input type="text" name="ID"></td>
</tr>
<tr>
        <td>NAME</td>
        <td><input type="text" name="NAME"></td>
</tr>
<tr>
        <td>CATEGORY</td>
        <td><input type="text" name="CATEGORY"></td>
</tr>
<tr>
        <td>ADDRESS</td>
        <td><textarea name="ADDRESS" rows="10"></textarea></td>
</tr>
<tr>
        <td colspan="2"><input type="submit" value="Пошел!"></td>
</tr>
</table>
</form>
```

Разучив и испробовав данные скрипты, вы (я очень на это надеюсь) сможете написать скрипт для редактирования записей, уже находящихся в базе. Почему я не привожу такого скрипта? По одной простой причине: «благими намерениями выложена дорога в ад». Изучая языки программирования, я пришёл к глубокому убеждению что практика – неотъемлемая часть обучения, и у меня есть совершенно коварное намерение создать вам небольшую трудность.

Для особо любопытных!

СУБД Firebird/InterBase не только тразакциональны, в них реализованы генераторы, триггеры и записываемые процедуры.

Как вы, наверное, заметили, при создании БД я не написал ничего хоть как-то напоминающего auto_increment в знакомом MySQL, по этому пришлось давать читателю возможность ставить номер самому. Думаю, что вы уже поэксперементировали и увидели ошибки совпадения ключей. В данных СУБД это не проблема.

Воизбежание дублирования применяют генератор в связке с триггером. Для их создания над данной БД нужно произнести:

```
CREATE GENERATOR GEN_ID
```

Таким образом, был создан генератор, который в данный момент равен нолику и будет стоять без триггера. Поэтому не забудьте завершающее заклятие.

```
SET TERM ^ ;
CREATE TRIGGER TRIG_NAME_ID FOR ADDRESS
ACTIVE BEFORE INSERT POSITION 0
AS

BEGIN

      NEW.ID = GEN_ID(GEN_ID_NAME, 1);
END
 ^
```

Здесь «SET TERM ^ ;» означает, что вы меняете завершающий знак для данного запроса, его просто необходимо будет поставить в конце получившейся заявки. Он обязателен для создания триггеров и процедур по простой причине – символ «\n» может быть использован в теле триггера(процедуры) как его составная часть. К примеру, если name содержит «\n», то даём ему чётное, ну а если нет, тогда нечётное.

GEN_ID – встроенная процедура, которая принимает переменную (в данном случае генератор) и число, являющееся шагом увеличения. Результат работы – увеличение (инкремент) при всяком вызове переменной на число и возврат полного значения. Процедура написана для того, чтобы не мучаться с определением текущего состояния генератора и корректно обрабатывать конфликты (про транзакции не забыли?).

ACTIVE BEFORE INSERT POSITION 0 – означает, что триггер активируется только в случае вставки. При попытке вставить что либо другое – другое будет сменено на результат работы GEN_ID

Ну а чем же отличаются триггеры от процедур? Триггеры привязаны к событиям, процедуры же приходится вызвать, даже иногда из триггеров – это существенное для нас с вами различие, остальное, как говорится, не суть важно.

Кстати, после создания триггера не забудьте исправить post.php, и триггер сделает то, что необходимо.

Итак, на этом всё.

PHP и Oracle: создание приложений с низким ресурсопотреблением

Автор:
Джон Нейл
Перевод:
Данил Миронов
[patrungel]

PHP и Oracle являют собой превосходную связку для создания мощных и масштабируемых веб-приложений. В данной статье речь пойдёт о проблемах с производительностью, возникающих только в условиях высокого трафика. И всё будет изложено так, что вы сможете позаботиться об их устранении ещё до того, как они дадут о себе знать.

Если ваш PHP-сайт использует СУБД Oracle, то можно сказать, что у вас в руках один из самых мощных инструментов из всего арсенала веб-разработчика. Только представьте: у вас в руках одновременно оказались скорость и надёжность PHP-кода, помноженная на мощь и гибкость Oracle. Однако при несоблюдении некоторых правил вы рискуете значительно снизить производительность и защищённость вашего приложения, и тогда вашей системе придётся очень несладко.

В данной статье мы попытаемся описать некоторые приёмы, с помощью которых вы сможете заставить PHP и Oracle эффективно сотрудничать и тем самым минимизировать потребление ресурсов с обеих сторон. Иными словами, мы предлагаем вам описание инструментов и методик для написания «приложений с низким ресурсопотреблением».

Все приведённые здесь примеры взяты из реальной практики программирования, это наши кровь, пот и слёзы. Наши платформы – это несколько веб-серверов под Linux, вторая версия Apache и, как правило, последняя версия PHP. БД сервера с Oracle работают на аппаратной платформе Sun, под ОС Solaris (хотя у нас есть и несколько испытательных Oracle-серверов под Linux).

Итак, вначале мы расскажем, как можно в целом снизить нагрузку на ресурсы веб- и БД серверов. Затем мы рассмотрим примеры решения наиболее распространённых задач, о различных подходах в решении. В заключение мы сделаем обзор инструментов и методик, позволяющих вам отслеживать, насколько эффективно вы применили ресурсосберегающие приёмы при разработке приложений в связке PHP и Oracle.

Что такое «приложения с низким ресурсопотреблением»?

Разработка приложений с низким ресурсопотреблением – это, скорее, позиция разработчика, нежели навык. Это означает, что разработчик всегда стремится к тому, чтобы написанный им код и сделанные настройки потребляли как можно меньше ресурсов сервера. При этом разработчик пытается снизить ресурсопотребление всегда, независимо от того, как часто будет исполняться тот или иной кусок кода.

На практике разработка приложений с низким ресурсопотреблением означает ещё и возможность расширять ваши приложения и при этом не вводить сервер в долгий ступор. Кроме того, если разработчик сконцентрирован на том, чтобы для выполнения одних и тех же задач требовалось меньше ресурсов, то в результате его приложение будет более компактным и более устойчивым к сбоям.

И ещё один момент: при использовании таких средств, как PHP и Oracle, ввиду их «врождённой» скорости и надёжности у нас часто возникает иллюзия полной безопасности. Опасность состоит в том, что если проблемы с производительностью появляются, то они очень быстро приобретают нежелательный для вас масштаб.

Как использовать ресурсы по минимуму

При разработке сайта в связке PHP и Oracle количество требуемых ресурсов определяется несколькими факторами. Самыми простыми решениями в направлении ресурсосбережения можно назвать: использование постоянных соединений, отказ от фиксации транзакций Oracle, использование преимуществ SQL-кеша Oracle и снижение трафика передаваемых данных.

Постоянные соединения

Существует много аргументов за и против использования постоянных соединений. Единственный большой плюс использования постоянных подключений к базе данных в Oracle определяется тем фактом, что создание соединения с БД использует много процессорных ресурсов на обоих концах соединения. Наши тесты показали, что открытие нового соединения с БД Oracle отнимает от 0.25 до 0.5 секунд для каждой страницы. Использование постоянных соединений экономит нам это время при открытии почти каждой страницы.

Но если вы сделали свой выбор в пользу использования постоянных соединений, то вам нужно учесть некоторые моменты. Один из самых важных моментов: все ресурсы, открытые одним скриптом при постоянном соединении, остаются доступными для всех последующих скриптов, использующих то же соединение. Количество открытых ресурсов накапливается и может стать скрытой причиной замедления работы сервера и различных сообщений об ошибках.

Например, каждый созданный дескриптор оператора открывает курсор. В БД Oracle открытый курсор – указатель на место в памяти базы данных, а их количество в БД Oracle – ограниченное количество. И поскольку постоянное соединение закрывается (то есть когда закрываются все указатели в данном соединении) только с закрытием дочернего процесса Apache, на загруженном ресурсе количество открытых курсоров растёт, и через некоторое время вы начнёте получать сообщения об ошибках. Если вы планируете использовать постоянные соединения, то вы должны явно закрывать все созданные вами ранее дескрипторы операторов.

Ещё один класс проблем, который может свалиться на вас – это параметры сессии Oracle. Одним из наиболее часто используемых параметров является установка формата даты по умолчанию. Если для одного скрипта вы желаете иметь определённый формат даты и устанавливаете этот формат в качестве параметра сессии Oracle, то изменения будут действительны для всех скриптов, использующих то же соединение с БД. Это может привести к неправильной работе скриптов, причём без особых указаний причины происходящего.

Ещё одна опасность постоянных подключений заключается в том, что текущие версии PHP не очень хорошо справляются с перезапуском БД. Если вы используете постоянные подключения и перезапускаете базу данных Oracle, все открытые соединения от Apache отвалятся, но заново не откроются до следующей перезагрузки страницы. Это означает, что при каждом перезапуске БД Oracle вам нужно перезапускать и Apache, в противном случае ваши пользователи будут получать кучи сообщений об ошибках до тех пор, пока не истечёт время жизни старых соединений.

Итак, постоянные соединения действительно помогут вам снизить потребление ресурсов, если ваши PHP-скрипты выполняют следующие требования:

- каждый скрипт собирает за собой мусор;
- использование параметров для сессий Oracle чётко определено и согласовано со всеми участниками проекта.

Запрограммировать свои скрипты на сбор мусора достаточно просто. Если вы открываете дескриптор оператора, заройте его. Открываете любой новый дескриптор, закройте его. Если вы никогда не забываете закрывать все открытые вами ресурсы, то сервер Oracle вам ответит устойчивой и быстрой работой.

Существует не так уж и много полезных функций, которые в Oracle можно использовать через задание параметров для сессии, поэтому все эти параметры должны быть согласованы со всеми разработчиками в проекте и использоваться одинаково. Если какой-либо программист введёт новый параметр в сессию Oracle, то, трудно будет предсказать, какое действие этот параметр произведёт на код всех остальных участников. Более того, указание на тот или иной задействованный параметр сессии почти невозможно получить при отладке приложения.

Сокращение размера транзакций и их фиксаций

Каждый раз, когда Oracle делает фиксацию, он записывает на жёсткий диск всё содержимое буфера. Независимо от того, есть ли в нём данные для сохранения или нет. И естественно, каждая такая запись на диск отнимает время и ресурсы сервера Oracle.

И поскольку в PHP по умолчанию все функции Oracle настроены на автофиксацию, вы значительно увеличиваете количество ненужных записей на диск на стороне БД сервера.

Такие частые записи действительно не нужны, ибо почти все дескрипторы операторов, открытые PHP-сайтом, это запрос данных. За исключением случая, когда вы делаете выборку для обновления записей (select for update), select-у сохранять данные не требуется, нужно только чтение с диска.

Самый простой способ избежать постоянной фиксации, когда вам нужно только чтение – это применить опцию `OCI_DEFAULT` в ваших `OCIExecute`. Поведение функции изменится: вместо немедленной автофиксации будет требоваться отложенная фиксация. Необходимо помнить, что такая политика может привести к проблемам с сегментами отката, однако если вы следуете ранее данному совету о явном закрытии всех использованных ресурсов в вашем коде, то использование ресурсов будет минимальным.

Естественно, что при `insert`, `update` или `delete` вам необходимо провести фиксацию, иначе изменения не будут сохранены. Существуют ситуации, когда вам нужно отложить фиксацию и дождаться выполнения каких-либо действий, но чаще всего вам необходимо произвести запись как можно скорее. Только в некоторых случаях вам понадобится отложенная фиксация, это когда вы имеете дело с некоторыми типами связанных переменных (bind variables), например, при использовании больших объектов или PL/SQL.

И если вы делаете отложенную фиксацию при вставке, обновлении или удалении записей, то вам нужно убедиться в том, что у БД достаточно места в сегментах отката. Размер сегмента отката должен быть заведомо больше, чем может понадобиться для размещения самой большой транзакции, которая может иметь место при исполнении вашего скрипта. Если вы планируете работать с большими объектами, то убедитесь, что сегменты отката смогут вместить самый крупный большой объект, который вообще ожидается.

Управление SQL-кешем

Одним из важнейших преимуществ, которые вы получаете при использовании Oracle в качестве БД к вашему PHP-сайту, - это SQL движок, обладающий огромной мощностью в управлении данными. За эту мощь, однако, вам придётся заплатить всё более сложными и сложными запросами. И каждый раз, когда вы отправляете SQL-запрос серверу, происходит его разбор и создаётся модель исполнения. Каждый запрос должен пройти через стоимостной оптимизатор Oracle (CBO, Cost Based Optimizer), который определяет, какие индексы нужно задействовать и в каком порядке, как применить условия соединения, чтобы как можно быстрее выдать запрашиваемые данные.

Чтобы не обращаться без необходимости к оптимизатору, разработчики Oracle предусмотрели специальный буферный кеш, где хранятся результаты каждого разбора и исполнения. Однако этот кеш содержит полные запросы и чувствителен к регистру. То есть, чтобы грамотно управлять кешем и не заставлять по нескольку раз разбирать одни и те же запросы, весь ваш SQL должен быть приведён к определённому стандарту.

Для стандартизации вашего SQL-кода вам нужно придерживаться двух правил, очень простых: первое — выработать и придерживаться правил использования регистра символов, и второе — не разбивать лишний раз запрос на несколько строк. Хотя многие разработчики предпочитают использовать в своих SQL-запросах оба регистра, трудно найти двух людей, которые бы делали это одинаково. Такая разношёрстность в использовании регистров не позволяет вам эффективно использовать буферный кеш, и самый простой способ это обойти — оставаться в одном регистре. Конечно же, каждый волен решать, какой именно регистр использовать, но мы сделали выбор в пользу всех строчных букв, поскольку так легче печатать.

Ещё один способ повысить эффективность использования буферного кеша – создавать одинаковые запросы, отличающиеся только определёнными параметрами. Например, если вы запрашиваете ряды из одной таблицы, изменяя лишь первичный ключ, по которому производится выборка, вы можете заставить их использовать одно и то же вхождение кеша, применив связанную переменную для варьируемого параметра. Таким образом, все эти запросы будут на самом деле одним и тем же SQL-выражением за исключением значения переменной, которая и определяет, какие ряды вам нужны в данный момент.

Ещё один способ снизить нагрузку на сервер БД - это поместить самые сложные запросы в представления (view). Представления – это просто определённый заранее запрос. Преимущество представляемых таблиц состоит в том, что SQL разбирается и оптимизируется при их создании, а не когда выполняется связанный с ними запрос. В итоге к моменту, когда PHP-скрипт вызывает такое представление, сложные условия уже обработаны сервером БД.

Эффективное использование представлений, создающее, правда, дополнительные трудности для программистов при переходе на другие СУБД, значительно снижает использование ресурсов. Ещё одним преимуществом является то, что администратор БД может оптимизировать представления в системе, не влезая при этом в код PHP-разработчиков. При соблюдении таких правил, на загруженном ресурсе процент попаданий в буферный кеш (buffer cache hit ratio) может достигать 80%, а в некоторых условиях - ещё больше. Это существенно снижает нагрузку на оптимизатор и некоторые другие компоненты БД Oracle.

Снижение трафика передаваемых данных

Ещё одна скрытая причина медленной работы БД заключается в том, что между веб-сервером Apache и сервером БД Oracle передаётся слишком много данных. К счастью, веб- и БД-серверы обычно расположены близко друг к другу (предпочтительно в одном сегменте сети). Однако, большие объёмы — часто просто без надобности – передаваемых данных могут вызвать замедления работы и увеличить время отклика.

Одним из способов снизить трафик — это создать представляемые таблицы, содержащие только нужные вам столбцы. Например, если у вас в таблице содержатся поля с большими объектами, а вам они в не нужны, исключите соответствующие столбцы из запроса. Если же вы настаиваете на применении синтаксиса `select * from ...`, то создайте представление, содержащее всё, кроме столбцов LOB.

Ещё один способ — это писать такие запросы, чтобы к каждой строке полученной выборки не приходилось применять ещё один, внутренний запрос. Ибо если вы в своём коде для каждого ряда полученной выборки применяете внутренний запрос, то вы создаёте большой трафик между Oracle-сервером и PHP-скриптом. Помните, что благодаря способности Oracle-процессора исполнять сложные, многомерные запросы, почти всегда есть возможность поместить несколько вложенных запросов в один.

Ещё один маршрут ненужного трафика проходит в самой базе данных, он тоже снижает производительность. Нередко оптимизатору сложно определить, что данное условие соединения является отношением внешний ключ/первичный ключ. Если вы знаете, что в результате наложения вашего условия для соединения будет возвращён один единственный ряд, то вы можете поделиться вашим знанием с оптимизатором, передав ему директиву SQL-компилятора FIRST_ROWS, как показано в Листинге 1 (см. в приложении к журналу). Так оптимизатор узнает, что применяемое условие соединения – это отношение внешнего ключа к первичному.

Ещё один случай, когда излишний поток данных может снизить скорость работы – это использование связей (database link) между несколькими Oracle-серверами. Если вы делаете соединения через связи или делаете соединения на дальнем конце связи, локальный оптимизатор очень сильно потрудится, связывая все данные. Вы обнаружите замедление работы как локального, так и удалённого Oracle-сервера, поскольку передача данных через связи съедает ресурсы на обеих системах.

Оптимизация в решениях распространённых задач в связке PHP и Oracle

Существует ряд весьма распространённых задач, при решении которых выбор, сделанный программистом, может накопить значительное влияние на работу системы. Среди таких задач мы можем назвать постраничный вывод, подсчёт промежуточных и итоговых сумм, условное суммирование и поиск средних значений, задание временных интервалов в запросах. В каждом случае существует несколько решений. Мы продемонстрируем, какие из решений, по нашим наблюдениям, потребляют наименьшее количество ресурсов на наших серверах.

Постраничный вывод

Разработчиков нередко просят выводить результаты запросов или отчёты постранично. Например, кому-то понадобится выводить результаты поиска по 20 вхождений на странице.

В интернет-среде это позволяет избежать ситуаций, когда по не очень удачному запросу сервер вернёт тысячи результатов, а пользователя интересуют лишь несколько первых.

В некоторых реляционных СУБД, включая MySQL и в некоторой степени Microsoft SQL Server, встроенные расширения SQL позволят вам легко и просто это сделать. К сожалению или к счастью, в Oracle есть лишь половина того, что вам нужно для ограниченного вывода, причём ограничение рядов выполняется перед их сортировкой. Итак, принимая во внимание вышеуказанные ограничения, вам предстоит решить, будете ли вы вырезать нужный вам интервал средствами PHP или заставите Oracle возвращать только требуемое количество рядов.

Чтобы понять, в чём же заключаются эти ограничения в Oracle, мы предлагаем вам составить запрос постраничного вывода как бы «изнутри». Предположим, вам нужно получить все записи о сотрудниках, сортируя их по фамилии, а затем по имени. Запрос из Листинга 2 (см. приложение к журналу) отлично с этим справляется. И пока в таблице содержатся записи о нескольких десятках сотрудников, нас не беспокоит, сколько рядов мы получим, и как это всё нарисуется в браузере.

Но как только несколько десятков вырастет до нескольких сотен, нам, вероятно, потребуется вывести результат на нескольких страницах. Если вам нужно получить первые двадцать рядов, и вы немного почитали руководство по SQL-диалекту Oracle, то вы попытаете счастья с запросом из Листинга 3 (см. приложение к журналу). К сожалению, оптимизатор сначала выполнит ограничение по количеству рядов, и только потом применит механизмы сортировки. В целом вы получите неправильный результат. Всё встанет на свои места, если запрос из Листинга 2 поместить в подзапрос, как в Листинге 4 (см. приложение к журналу).

Такой вот простой приём позволит вам создавать запросы, отрезающие ряды, которые находятся дальше заданного максимального значения. Для вывода первой страницы этого вполне хватает. Но если мы пожелаем вывести вторую страницу, то к результату придётся применить ещё один запрос. Если изменить запрос из Листинга 4, чтобы он также выдавал и количество возвращаемых рядов, то мы можем поместить его ещё в один подзапрос, ограничив вывод рядами, чей номер не ниже заданного нами минимального значения. Запрос в Листинге 5 (см. приложение к журналу) демонстрирует, как мы можем запросить только с 20 по 39 ряды из полученной выборки.

Будет ли постраничный вывод реализован средствами PHP или через разобранный нами запрос - ваш выбор определится тем, что будет съедать меньше ресурсов: работа оптимизатора или трафик данных. Наш опыт показывает немного лучшие результаты вывода первых 3-4 страниц при работе с запросом. С ростом общего количества рядов в полученной выборке и ростом номера страницы, на которой находится пользователь, разница в ресурсопотреблении стремится к нулю.

Использование статистических запросов

Ещё одной распространённой задачей при генерировании отчётов является подсчёт промежуточных и итоговых сумм. Когда ваши вычисления ограничиваются суммированиями и подсчётом количества рядов, вопрос о выборе между статистическими функциями Oracle или переменными в PHP почти несущественен. Однако, если вам нужно вычислить средние значения или другие функции, то вам очень пригодится большая коллекция статистических функций в SQL от Oracle.

Возможности вычисления статистических функций в Oracle SQL в основном реализованы в виде опций оператора GROUP BY и функции GROUPING. Пример запроса с использованием этой функции приведён в Листинге 6 (см. приложение к журналу). Этот запрос подсчитывает минимум, максимум, среднее значение и суммы зарплат сотрудников с выделением промежуточных результатов по отделам и итогового результата по всем рядам.

Чтобы знать, в каких рядах находятся промежуточные итоги для каждого столбца, мы включаем значения из функции GROUPING в оператор SELECT. Главное, что вам нужно запомнить о функции GROUPING – это то, что работает совсем не так, как вы думаете. Более того, если определённый столбец выделен для хранения промежуточных итогов, то в рядах выборки он содержит NULL. И вы не можете рассчитывать на эти значения NULL для определения столбца с промежуточными итогами, если NULL также может быть и в полях выборки как таковой.

В Листинге 7 (см. приложение к журналу) приведён полный пример PHP-кода, делающего запрос из Листинга 6 с применением функции GROUPING. Результат работы этой функции используется для отображения промежуточных и окончательного итогов.

Использование Case и NVL

Существует класс случаев, где функции Case и NVL из SQL-диалекта Oracle позволяют вам более эффективно решить задачу. Нередко вам нужно суммировать или выполнить другие действия с данными только при выполнении заданного условия. Кроме того, иногда требуется повернуть таблицу с полученными данными, изменив таким образом измерения таблицы, предоставленной при выборке. И, наконец, вам может понадобиться произвести одни действия, если в поле содержится значение, и другие, если значение нулевое.

Для условной обработки данных вы можете использовать накопительные переменные PHP или же сделать всё на стороне Oracle. Преимущество варианта с PHP состоит в том, что оптимизатор Oracle менее нагружен разбором запроса. Однако, использование кеша, индексов и, возможно, представлений также может помочь вам в том же направлении, хоть и не так эффективно. А самым большим недостатком PHP-варианта можно назвать большой и бесполезный трафик между сервером Oracle и Apache. К тому же, этот вариант потребляет на сервере Apache много памяти, а на загруженных машинах этот ресурс всегда в большом дефиците.

Допустим, вам нужно подсчитать зарплаты всех менеджеров. Как это делается средствами PHP, показано в Листинге 8 (см. приложение к журналу). Код получает все данные из базы и принимает решение, какие данные использовать в суммировании, а какие – нет. Или же можно сделать один запрос, приведённый в Листинге 9 (см. приложение к журналу), и сразу получить ответ на поставленный вопрос.

Нередко, когда нужно транспонировать или иначе повернуть таблицу с полученными данными, вам приходится получать сначала данные из базы, затем помещать их в массивы в PHP и работать с ними. Как правило, это самое эффективное решение. Однако, если условия позволяют, вы можете использовать функцию CASE и с её помощью получить поля для их индивидуальной обработки. Наша практика показала, что эта возможность оказывается чрезвычайно полезной, когда нам нужно получить сведения о проведённых транзакциях с разбитием на сегодня, вчера, эту неделю, месяц.

Применение функции CASE для разбития значения даты на нужные интервалы делает сам процесс получения статистики предельно прозрачным. Если вы обратитесь к Листингу 10 (см. приложение к журналу), то сможете ознакомиться со скриптом, вывод которого представлен на рисунке 1. Как и в предыдущем примере, результаты получены при минимальном трафике.

И, наконец, если мы хотим выполнить с полями выборки разные действия в зависимости от того, являются их значения NULL или нет, то функция NVL значительно ускорит этот процесс. Существует множество применений этой функции. Например, пусть у нас в таблице предусмотрен счётчик следования, который применяется для сортировки записей, и, как правило, нам хочется, чтобы новые записи появлялись в конце этой последовательности. Запрос, приведённый в Листинге 11 (см. приложение к журналу), добавляет в таблицу новую запись и при этом гарантированно ставит её в конце списка.

Ещё одним полем применения функции NVL является работа со сроками действия. В этих случаях вас интересует, находится ли текущая дата в интервале между указанными начальным и конечным сроками. В Листингах 12 и 13 (см. приложение к журналу) приведены два запроса, вывод которых ничем не отличается. На их работу ресурсов уходит примерно одинаково, это скорее пример того, как можно творчески подойти к использованию функции NVL.

Быстрые функции Oracle для работы с данными

Новичков в работе с Oracle нередко смущает тип DATE, поскольку его формат на самом деле содержит и дату, и время. В Oracle не предусмотрено отдельного типа данных для даты и отдельного типа - только для времени, как это реализовано в других реляционных СУБД. Внутренний формат хранения даты в Oracle – это число с плавающей точкой. Целая часть - это количество дней, начало отсчёта установлено на 1 января 2000 года до н. э. В дробной части содержится истёкшая часть суток. Таким образом, число 10.5 указывает на 10 января 2000 года до н. э., полдень.

Такой способ представления даты говорит о том, что у Oracle имеются в наличии очень быстрые функции для выполнения определённых действий с датами. Например, если вам нужно узнать, сколько дней между двумя заданными датами (не принимая во внимание время), вы можете применить функцию TRUNC, как показано в Листинге 14 (см. приложение к журналу). Если вам нужно узнать, не приходится ли значение поля LAST_DATE на сегодня, то вам нужно сравнить TRUNC(LAST_DATE) и TRUNC(SYSDATE). Функция TRUNC, получая один параметр, приводит дату, то есть число с плавающей точкой, к целому. Эффект этого приведения: дата и время конвертируется в полночь того же дня.

Второй параметр в функции TRUNC позволит вам проводить более тонкие манипуляции с датами. Например, чтобы узнать, принадлежат ли две даты одному месяцу, вам нужно сравнить TRUNC (date1, 'MONTH") и TRUNC(date1, 'MONTH'). Чтобы определить, попадают ли две заданные даты на одну и ту же неделю, сравните TRUNC(date1, 'IW") и TRUNC(date2, 'IW'). Заметьте, что мы используем IW, а не WW, поскольку в Oracle параметр IW соответствует спецификации недели в формате ISO, где неделя всегда начинается в понедельник. Если вы используете WW, то неделя начинается с того дня, на которое пришлось 1 января того года.

Если вы ещё раз обратитесь к Листингу 10, то вы увидите пример эффективного использования функции TRUNC при работе с датами. Такие функции работают намного быстрее, нежели сравнения TO_CHAR или сравнения BETWEEN. Кроме того, поскольку тип DATE в Oracle содержит и время, то применение функции TRUNC при работе с ними оградит вас от проблем, возникающих при использовании BETWEEN. В качестве примера допустим, что вам нужно получить список всех транзакций, имевших место между 1 августа и 31 августа 2003 года. Если вы ограничились запросом из Листинга 15 (см. приложение к журналу), то в ваш отчёт не попали транзакции, проведённые 31 августа. Однако, если мы используем запрос из Листинга 16 (см. приложение к журналу), то все транзакции от 31 августа независимо от времени проведения будут включены в наш отчёт.

Настройка и мониторинг

В идеальном мире все PHP-программисты умеют правильно и грамотно составлять SQL-запросы. Если бы все делали всё правильно, то мониторинг баз данных был бы вовсе не нужен. Однако, вроде бы никто из нас не живёт в этом прекрасном месте, поэтому, дабы не терять контроль над потерями в производительности, нам постоянно приходится следить за тем, какие ресурсы базы данных и -сервера используются и какими запросами.

Настройка и мониторинг – это набор задач, чаще всего, конечно, выполняемых администратором БД. Однако иногда возникают ситуации, когда PHP-программист тоже может принять участие в мероприятиях по настройке и мониторингу системы. Как правило, администратор БД без проблем может определить запрос, снижающий работоспособность системы, более того, он в состоянии его исправить, однако он может потратить уйму времени на поиск скрипта, где находится этот запрос.

Кроме того, администратору БД и PHP-программисту нужно работать в одной команде, чтобы и после отладки эти запросы давали правильный вывод.

Для программиста основные задачи в настройке системы сводятся к двум видам: поиск неудачного (или не очень удачного) SQL и создание инструментов для мониторинга.

Поиск неудачных SQL-запросов

Словарь данных Oracle ведёт учёт использования системных ресурсов для каждого запроса. В любое время вы можете обратиться к этим системным таблицам за различного рода информацией о производительности. Существует несколько критериев оценки производительности, основные из них - это обращения к буферу, запросы на разбор и чтение данных с диска. Эти параметры отражают использование разных ресурсов сервера БД Oracle и наибольшим образом влияют на скорость обработки запроса. Во всех случаях меньшие значения означают лучшую производительность.

Для обнаружения запросов с плохими показателями обращения к буферу вам нужно будет отправить серверу запрос из Листинга 17 (см. приложение к журналу). Обращения к буферу отражают степень нагрузки на ЦП сервера. Если вам нужно получить данные только по нескольким пользователям, то вы можете ввести дополнительные ограничения в запрос. Посмотрим, как вычисляется выдаваемый результат, это поможет вам его правильно интерпретировать. В нашем случае вычисляется отношение загрузки ЦП к числу выполнений каждого запроса. Благодаря этому новые запросы (которые ещё так много раз не исполнялись) уходят на верхние позиции в списке, а «долгожители» остаются внизу.

Ещё один показатель неудачного запроса – это количество разборов оптимизатором. Чтобы найти рекордсменов по ресурсопотреблению в этой категории, обратитесь к Листингу 18 (см. приложение к журналу). Наибольшие показатели означают, что запрос надо поместить в виртуальное представление или оптимизировать его какими-либо другими средствами, но избежать слишком частого разбора оптимизатором. И хотя хранение запросов в виде представлений не влияет на количество частичных разборов (soft parse), оно значительно сокращает количество полных разборов (hard parse). Опять же, поделив полученные значения на количество выполнений запроса, мы отправляем новые запросы наверх.

И последний из основных критериев: мы отследим, какие запросы больше остальных производят чтение с диска. Для их поиска используйте запрос из Листинга 19 (см. приложение к журналу). Высокий показатель будет означать, что при каждом исполнении такого запроса производится большой объем чтений с диска. Такие запросы – кандидаты номер один для оптимизации с помощью дополнительных условий WHERE или прочих приёмов, которые сократят объём передаваемых данных.

Более подробные сведения об интерпретации полученных значений вы можете найти в документации Oracle, руководство «Oracle 8i Designing and Tuning for Performance». Обычно это PDF-документ, поставляемый на диске с дистрибутивом.

Мониторинг системных ресурсов

Когда вы начнёте отслеживать ресурсопотребление запросов, то наверняка через некоторое время вам захочется наладить мероприятия систематического контроля над использованием ресурсов, чтобы оперативно решать связанные с этим проблемы. Когда это время придёт, вам понадобится библиотека запросов и среда для регулярного мониторинга.

Если вышеприведенные запросы вы поместите в регулярно исполняемый скрипт, то сможете узнать, какие запросы исполняются чаще всего. На самом деле это просто бесценный инструмент для программистов, если им нужно выяснить, где предпринятые меры по оптимизации были наиболее эффективны. Кроме того, такой инструмент позволяет избежать ситуаций, когда уйма времени убивается на оптимизацию запроса, который исполняется всего раз в сутки, а до запросов, встречающихся чуть ли не на каждой странице, руки не дошли. Ещё одной областью для мониторинга является веб-сервер Apache. Здесь смотрится, сколько дочерних процессов Apache активно в данный момент времени, загрузка памяти, общее число процессов в системе, вычисляется средняя нагрузка. Все эти данные помогают устранять проблемы до того, как они начнут выходить из-под контроля.

В наших системах мы проводим регулярный мониторинг ресурсов. Основные инструменты для его проведения - это скрипты, ежедневно выявляющие слишком частые обращения к буферу, запросы на разбор и чтение данных с диска. Также мы проводим мониторинг ресурсов на наших веб-серверах: считаем среднюю нагрузку, количество процессов, загрузку памяти и заносим эти данные RRD (Round-Robin Database). Далее мы регулярно составляем графики использования ресурсов системы на разные интервалы времени.

Итак, действительно ясной картины можно добиться, только прилагая определённые усилия по всем этим направлениям. Так вы будете в курсе того, какие проблемы возникли у вас сегодня и какие вероятнее всего возникнут завтра.

Заключение

Работа с такой мощной системой, как СУБД Oracle - просто клад для разработчика. Гибкость и мощь Oracle окажут вам действительную помощь при создании сложных и компактных сайтов. Однако, если вы соответствующим образом не подготовитесь, возникнут проблемы с ресурсопотреблением и застанут вас врасплох.

К счастью, эта статья может стать точкой отправления в вашей подготовке, и вы будете в состоянии использовать все достоинства Oracle в полную силу и при этом не причинять ущерба системе. Все данные здесь рекомендации являются результатом непосильных трудов нашей команды в критических ситуациях. Может быть, наш опыт поможет вашим веб- и БД серверам хорошо сработаться.

Оригинал статьи находится по адресу: http://phparch.com/sample.php?mid=32

Сравнение PostgreSQL, MySql и коммерческих БД

Автор:
Тим Конрад
Перевод:
Вера Козлова

Можете ли вы полагаться на ведущие open source СУБД PostgreSQL и MySQL, когда нужны производительность и возможности, которых повсеместно добиваются Oracle, SQL server и DB2? Пока ещё нет, но они могут предложить достаточно, чтобы удовлетворить ваши потребности. Выясним, как они соотносятся друг с другом, а также с коммерческими альтернативами.

На сегодняшний день сервер баз данных – это постоянный элемент почти каждого бизнеса. Широко распространённые коммерческие базы данных, такие как Oracle, Microsoft SQL Server и сервер DB2 от IBM, включают в себя много возможностей, на которые люди полагаются, делая свои серверы баз данных «солидными». Эти возможности включают усовершенствованное хранение баз данных, средства управления данными, репликацию информации и инструменты для резервного копирования.

За прошедшие десять лет сообщество open source улучшило качество своего программного обеспечения, сделав его более пригодным для использования в корпорациях. Как результат, в последние годы коммерческие предприятия продемонстрировали свою заинтересованность в переходе с лицензированного коммерческого программного обеспечения на open source. Например, предприятия всего мира наиболее часто используют Linux, язык программирования Perl, веб-сервер Apache и два ведущих open-source СУБД: PostgreSQL и MySQL.

Эта статья сравнивает PostgreSQL и MySQL между собой, а также с их коммерческими аналогами. Вместо того, чтобы рассматривать MySQL MAX, основанный на СУБД от SAP, статья рассматривает более широко используемый "оригинальный" MySQL.

Вопросы по MySQL и PostgreSQL часто связаны со скоростью. И хотя последние релизы Postgre стали намного быстрее, более ранние версии, были известны своей медлительностью. Но скорость это ещё не всё, когда надо выбрать хорошую СУБД. Это сравнение основано на возможностях, а не на скорости. Если всё, что вам надо – это чистая скорость, вы можете получить её другими способами.

Как всё начиналось

История PostgreSQL

Система управления реляционными базами данных (СУРБД) PostgreSQL произошла от проекта POSTGRES в Калифорнийском университете в Беркли. Профессор Майкл Стоунбрейкер (Michael Stonebraker) запустил проект в 1986 г., чтобы заменить устаревшие СУРБД Ingres. Его спонсировали Управление перспективных исследований Министерства обороны США, (DARPA), Национальный научный фонд (the National Science Foundation), Армейское Исследовательское управление (Army Research Office) и ESL Inc.

С тех пор, БД, известная как проект POSTGRES, брала на себя различные функции в разных организациях, включая БД перемещений астероидов, систему анализа фининсовых данных и инструмент образования.

POSTGRES изначально использовал для доступа к информации баз данных язык, называемый PostQUEL. В 1994 году Эндрю Йу (Andrew Yu) и Джолли Чен (Jolly Chen) добавили интерпретатор POSTGRES SQL, первоначально известный как Postgres95. Postgres95 был затем перелицензирован в соответствии с лицензией Беркли на использование программного обеспечения и вскоре после этого переименован в PostgreSQL.

Краткая история MySQL

До создания MySQL люди,которые его написали, использовали mSQL, чтобы подключиться к своим собственным низкоуровневым структурам данных. Они обнаружили, что mSQL не хватает тех возможностей и скорости, которые им требовались, и решили написать вместо него свою собственную интерфейсную часть. Так началась жизнь MySQL как продукта.

Особенности выдачи лицензии

И MySQL, и PostgreSQL имеют разные лицензии, и важно понимать, как они работают, когда эти продукты включаются в проекты предприятия. Разные лицензии удовлетворяют разные потребности и содержат разные условия.

MySQL AB, компания-владелец и производитель MySQL, имеет два возможных лицензионных соглашения для своей БД:

- GNU General Public License (GPL) – лицензия для некоммерческих (GPL) проектов. Если ваш проект на 100 процентов распространяется по GPL, то вы можете использовать эту лицензию. Чтобы полностью соответствовать требованиям, вы должны распространять ваше приложение вместе с исходным кодом. Вы также можете пользоваться этой лицензией, если вы не собираетесь распространять ваш проект.
- Коммерческая лицензия (Commercial License) для коммерческих приложений. Пример использования этой лицензии – когда вы не хотите распространять исходный код вашего приложения. Это также распространяется на драйверы БД. Вы можете использовать драйверы БД MySQL с коммерческими приложениями, только если они вместе распространяются согласно GPL лицензии, или если у вас есть коммерческая лицензия.

У PostgreSQL схема лицензирования намного проще. Она выдаётся в соответствии с лицензией Беркли, которая позволяет любое использование, пока продукт содержит копию лицензии Беркли. Это значит, что вы можете выпускать коммерческий продукт, который использует PostgreSQL или является производным от PostgreSQL без включения исходного кода.

Возможности, на которые вы привыкли рассчитывать.

Сравнение БД сводится к возможностям, которые предоставляет каждая СУБД. Администраторы БД, которые работали с коммерческими СУБД, такими как Oracle, DB2, или MS-SQL, опирались на довольно широкий набор возможностей. Этот раздел сравнивает перечисленные коммерческие БД с open source БД.

Хранение данных

MySQL обладает несколькими возможными механизмами хранения данных. Первоначально использовался ISAM/MyISAM (индексно-последовательный доступ), который был затем заменён более новым InnoDB. Другие механизмы хранения возможны, но данное исследование сконцентрировано на использовании таблиц InnoDB, так как обычно он обладает наиболее широким набором возможностей и является типом таблиц по умолчанию в MySQL версии 4.x. Выбирая механизм хранения данных в MySQL удостоверьтесь, что вы изучили все возможности, которые вы планируете реализовать.

Пока я собирал материал для этой статьи, я обнаружил, что некоторые возможности существуют в одном механизме хранения, и не существуют в остальных. Особенно это заметно в том, что InnoDB и BDB – это единственные типы транзакционно-безопасных таблиц. С другой стороны, PostgreSQL использует только один механизм хранения данных, точно названый системой хранения Postgre.

Целостность данных

Одной из самых важных возможностей любой СУБД является целостность данных. Соответствие ACID (Atomic – Элементарность, Consistent – Непротиворечивость, Isolated – Автономность и Durable – Долговечность) – это характеристика, гарантирующая целостность данных. ACID по существу означает, что когда внутри БД совершается транзакция, любая транзакция будет полностью успешной, и информация записана в БД или ничего не записано. И PostgreSQL и MySQL поддерживают соответствующее ACID-функционирование транзакций.

Обе БД также поддерживают частичную отмену транзакций, и они знают, как обращаться с deadlock. MySQL использует традиционное блокирование на уровне строк. PostgreSQL по умолчанию использует нечто, названное Multi Version Concurrency Control (MVCC) («Параллельное управление множеством версий» или «Контроль конкурирующих версий»). MVCC слегка отличается от блокирования на уровне строк тем, что транзакции в БД выполняются на моментальном снимке данных и затем преобразуются в последовательную форму. Новые версии PostgreSQL поддерживают стандартное блокирование на уровне строки как опциональное, но MVCC является предпочтительным методом.

Продвинутые возможности

PostgreSQL обладает многими возможностями БД, которые есть у Oracle, DB2 или MS-SQL, включая триггеры, представления, наследование, правила, хранимые процедуры, курсоры и определяемые пользователем типы данных. Опытная версия MySQL, версия 5.0, поддерживает представления, хранимые процедуры и курсоры. Будущая версия MySQL, версия 5.1, будет поддерживать триггеры. MySQL, тем не менее, поддерживает продвинутую возможность разделения данных в пределах БД. А PostgreSQL - нет.

Хранимые процедуры и триггеры

В то время как PostgreSQL поддерживал хранимые процедуры и триггеры почти с самого начала, MySQL поддерживает их только в опытных версиях 5.0 и дальше. Язык запросов PostgreSQL, PL/pgSQL очень похож на PL/SQL Oracle. К тому же процедуры и триггеры PostgreSQL можно также писать и на других языках, таких как PL/TCL, PL/perl и PL/python. Эти дополнительные языки разделяются на две основные категории: безопасные и опасные. Безопасные позволяют использовать в языке программирования только те вещи, которые не влияют на систему отрицательно, как при прямом доступе к файловой системе.

Индексы

Oracle известен предоставлением большого количества настроек для БД, особенно в том, что касается индексирования. В целом, опытные пользователи Oracle, возможно, найдут методы индексирования, применяемые в этих open source БД, слегка примитивными. И PostgreSQL и MySQL поддерживают одностолбцовые, многостолбцовые, уникальные и первичные индексы. MySQL замечательно поддерживает полнотекстовые индексы, и PostgreSQL также может поддерживать полнотекстовые индексы после некоторых изменений в БД, которые включаются в исходники.

Типы данных

БД содержат информацию, и типы информации, которые БД может содержать, называются типами данных. И PostgreSQL и MySQL поддерживают большинство стандартных типов данных. В последние несколько лет поддержка больших объектов популярна всё больше и больше, и обе БД также их поддерживают. PostgreSQL поддерживает типы данных, определяемые пользователем, в то время как MySQL - нет. И MySQL и PostgreSQL поддерживают хранение географических элементов, известных как GIS (Geographic Information System – Географическая Информационная Система). PostgreSQL дополнительно поддерживает сетевые типы данных, которые понимают Ipv4 и Ipv6.

Репликация

Другая важная характеристика БД уровня предприятия - это поддержка репликаций. И MySQL и PostgreSQL поддерживают single-master/multi-slave сценарии репликации. Этот базовый уровень репликации входит в дистрибутивы ПО, и его исходный код открыт.

PostgreSQL предлагает дополнительную поддержку для multi-master/multi-slave репликации от независимого производителя, а также дополнительные методы репликации.

Поддержка платформ

В то время как Oracle и DB2 работают на разнообразных платформах, Microsoft SQL Server ограничен Windows. MySQL и PostgreSQL поддерживают много различных платформ, включая Windows, Linux, FreeBSD и MacOSX. MySQL использует потоковую модель для процессов сервера, в котором все пользователи подключены к одному демону БД. PostgreSQL использует непотоковую модель, где каждое новое подключение к БД создаёт новый процесс БД.

Методы интерфейса БД

PostgreSQL и MySQL – оба поддерживают ODBC и JDBC для сетевых соединений, а также собственные методы доступа к БД. Эти собственные методы обеспечивают доступ через сеть с помощью простого текста, а для большего уровня секретности - с помощью SSL-кодирования.

Другой важной частью интерфейса является аутентификация в БД. MySQL использует простой метод - хранит всю свою аутентификационную информацию в таблице. Когда пользователи предпринимают попытку доступа к БД, MySQL сравнивает их полномочия в этой БД, проверяя, с какой машины пользователь может подключиться и к каким ресурсам у него есть доступ.

PostgreSQL может использовать такой же метод, но у него также есть некоторые другие. Например, он может использовать файл хоста для доступа к БД, чтобы определить, какие удалённые пользователи к какой БД могут подключаться. Также можно использовать локальные системы аутентификации для доступа к БД (т. е. ваш пароль Unix может быть также вашим паролем PostgreSQL).

Множество методов программирования также предоставляет пути доступа к этим БД. PostgreSQL и MySQL – оба поддерживают доступ с помощью C/C++, Java, Perl, Python и PHP. У PostgresSQL есть также внутренние языки программирования для написания хранимых процедур и триггеров, в их числе pl/pgsql, pl/tcl и pl/perl.

Резервное копирование

Когда доходит до резервного копирования, open source БД могут не полностью удовлетворять ваши потребности. Обе БД идут со скриптами, облегчающими выполнение простого текстового дампа данных вашей БД и их схемы. Решения обеих БД предоставляют ещё и методы для горячего резервного копирования БД или резервирования вашей БД без её закрытия. У многих коммерческих средств резервирования, таких как Vertias NetBackup или Tivoli TSM, есть программы-агенты, которые предоставляют онлайновое резервирование коммерческих БД. Быстрый веб-поиск выдал всего несколько производителей, которые создают агентов для PostgreSQL и MySQL. Всего их не так уж много.

Средства резервного копирования также позволяют выполнять простое восстановление БД при сбоях программы, таких как повреждение БД или неожиданные перебои в электросети. PostgreSQL использует систему, называемую Write Ahead Logging, для проверки на непротиворечивость БД. У MySQL есть проверка на непротиворечивость только для типа таблиц InnoDB.

Средства GUI

Многие люди используют средства GUI, чтобы управлять их БД. Многие такие средства – и open source, и коммерческие – доступны для MySQL и PostgreSQL. Эти инструменты могут быть или приложениями, которые выполняются в собственной системе команд вашей операционной системы или доступными через сеть средствами. Многие из этих инструментов сделаны точно по образцу аналогичных средств из коммерческих БД.

Перенос данных

И у MySQL и у PostgreSQL есть утилиты переноса данных, чтобы помочь перенести данные из коммерческих БД. Эти утилиты представляются третьими лицами как для open source, так и для коммерческих средств. С PostgreSQL идут утилиты, помогающие перенести данные из Oracle и MySQL. Очевидно, что чем сложнее ваша схема, тем сложнее будет перенос, и некоторые из этих утилит не смогут в совершенстве перенести все данные.

Обучение и поддержка

Осуществление поддержки облегчает переход на open source ПО в коммерческих организациях. Часто люди не знают, что для многих open source продуктов эта поддержка доступна за рамками веб-сайтов и списков рассылки. MySQL AB предоставляют поддержку MySQL, и несколько компаний, включая Command Prompt, Inc. и PostgreSQL, Inc., предоставляет поддержку для PostgreSQL. Эти предложения включают тот же уровень поддержки, что и альтернативные коммерческие БД, многие предоставляют поддержку в режиме 365x24

Обучение также доступно по широкому ряду тем для PostgreSQL и MySQL. MySQL AB предоставляет обучение в городах по всему миру, по темам от администрирования до написания веб-приложений, используя MySQL. Обучение PostgreSQL также предоставляют dbExperts и Big Nerd Ranch.

Кто ещё их использует?

Множество больших компаний используют различным образом обе open source БД. У обеих СУБД есть довольно большие использующиеся инсталляции БД. Я использую слово «довольно», так как хранение данных – это относительное понятие. Oracle и DB2 могут довольно просто масштабироваться до терабайтов хранимых данных. Про MySQL и PostgreSQL известно, что они хорошо доходят до сотен гигабайтов, но некоторые компании используют БД, превышающие эти порядки.

Cox Communications используют MySQL, чтобы управлять информацией, связанной с бизнесом в области кабельных модемов. NASA использует MySQL, чтобы хранить информацию о государственных заказах. Slashdot, широко известное онлайн-издание, использует MySQL, чтобы хранить всю информацию, связанную с их сайтом.

Associated Press использует MySQL, чтобы обрабатывать различные типы информации, включая доступ к переписи населения США (U.S. Census) и результам Олимпийских игр.

Возможно, вы используете PostgreSQL косвенно на совершенно законном основании. Afilias, который управляет регистрацией на.ORG, использует PostgreSQL чтобы хранить всю информацию о регистрации на .ORG. Американское химическое общество (The American Chemical Society) использует PostgreSQL, чтобы хранить документы, которые существуют только в этой БД. BASF использует PostgreSQL в торговом программном комплексе для сельскохозяйственной продукции.

The World, медиа-компания, построила большую часть своих инфраструктур с использованием PostgreSQL.

Если не ломается, не расширяй.

Аксиома open source ПО состоит в том, что разработчики пишут его, чтобы «почесать там, где чешется». Это хорошая привычка, когда «место, где чешется» - это что-то вроде сбойного диска, который заставляет разработчика улучшать программы, работающие на этом диске. Однако, в отношении БД, чесаться начинает только тогда, когда данные превышают определённые пределы, такие как размер или сложность.

PostgreSQL и MySQL широко используются для относительно небольших БД (менее 100Гб, например). Как только данные вырастают за пределы приблизительно 100Гб, количество пользователей резко падает. В этот момент разрешить вопрос, связанный с большими БД, становится проблемой.

Аксиома «чесания» применяется при работе в некоторых наиболее «загруженных терминами» областях. Наиболее продвинутые возможности в open source БД, такие как репликация, рядом не лежали по сравнению с тем, что мы можем обнаружить у коммерческих альтернатив.

В самом деле, большинству пользователей не нужна репликация на таком уровне, который предлагают Oracle, DB2 или MS-SQL, поэтому разработчики PostgreSQL и MySQL «не чешутся», чтобы улучшить их.

Впрочем, самое важное в open source ПО – это то, что его достаточно просто испытать, и имеется много свободно доступной онлайн документации, помогающей вам изучать продукт. Несмотря на то, что эти БД, может быть не оптимальны для любого проекта, они очень хорошо работают с остальными.

Если вам интересно, и эта статья не ответила на определённые вопросы в применении, попробуйте MySQL или PostgreSQL и посмотрите, соответствуют ли они вашим потребностям.

Тим Конрад (Tim Conrad) живёт и работает в Нью Йорке, где он сопровождает инфраструктуру сети в компании, предоставляющей услуги образования. В свободное время он любит ходить на концерты, пить пиво, играть на гитаре и отлично проводит время со своими компьютерами.

Оригинал статьи находится по адресу: http://www.devx.com/dbzone/Article/20743

Идеи

Использование разделяемой памяти в PHP

Автор:
Александр Прохоренко
Перевод:
Кирилл Крушняков

IPC ("Inter-Process Communication" - межпроцессное взаимодействие) - одна из важнейших особенностей ОС семейства UNIX. Она позволяет различным процессам взаимодействовать между собой. В этой статье речь пойдёт о двух технологиях System V IPC (System V – одна из ключевых версий ОС UNIX компании AT&T - прим. пер.): о семафорах и разделяемой памяти. System V IPC впервые появилась в SVR2 (System V Release 2 - прим. пер.). System V IPC, однако, была реализована многими разработчиками. Она также доступна в SVR4.

Концепция IPC слагается из нескольких компонентов. Термин подразумевает различные механизмы обмена данными между процессами, стартовавшими в одной системе. IPC позволяет избежать создания огромного приложения с большим набором функций всех различных назначений и заменить его на использование отдельных, малых приложений, способных обмениваться данными между собой. Традиционный подход Unix заключается в том, чтобы позволить многопроцессорным ситемам запускать приложения в отдельных процессах (threads) для сокращения времени, требуемого на выполнение специфических задач.

На высоком уровне мы можем разделить межпроцессное взаимодействие на следующие наиболее крупные и важные разделы:

- Сообщения: каналы (pipes) и очереди сообщений (pipes and message queues);
- Разделяемая память (Shared memory);
- Удаленный вызов процедур - RPC (remote procedure calls);
- Синхронизация: семафоры и любые виды блокирования;
- Сетевое взаимодействие (API сокетов).

В этой статье речь идет о разделяемой памяти и синхронизации (семафорах). Детальное описание этих методов займет слишком много времени - существует огромное количество различного материала, посвященного этой теме. Если вы заинтересованы в подробном изучении, обратитесь ко множественным книгам по межпроцессному взаимодействию.

Идентификаторы IPC

Каждый объект IPC (очередь ли сообщений, семафор, или сегмент разделяемой памяти) обладает уникальным идентификатором (Id), который позволяет ядру ОС идентифицировать этот объект. К примеру, для того, чтобы сослаться на определенный сегмент разделяемой памяти, вам всего лишь необходимо знать уникальный идентификатор, назначенный этому сегменту.

Учтите, что идентификатор объекта IPC является уникальным только для одного типа объектов. Другими словами, только одна очередь сообщений может иметь идентификатор 12345, хотя номер 12345 может также использоваться семафорами и/или сегментами разделенной памяти.

Ключи IPC

Как создается идентикфикатор IPC? Для этого необходим ключ. Первым шагом при создании среды взаимодействия между приложениями является координирование использования ключей. Представьте себе это так: для того, чтобы позвонить кому-либо, вы должны знать его телефонный номер. Телефонная компания должна знать, как переслать ваш звонок абоненту. И только, когда он отвечает, происходит соединение.

В случае применения System V IPC «телефон» соединяет объекты одного типа. Телефонной компанией или способом маршрутизации является «ключ» IPC.

Приложения должны генерировать свои собственные ключи. Фунция ftok() делает это и для клиента и для сервера. Значение ключа, возвращаемое функцией ftok(), зависит от значения inode и младшего значения идентификатора устройства для первого аргумента - файла – и символа – второго аргумента. Это не обеспечивает уникальности, но приложения могут проводить проверки на предмет коллизий и генерировать новые ключи по мере необходимости.

```
key_t mykey;
mykey = ftok ("/tmp/myapp", 'a');
```

В примере, приведенном выше, директория /tmp/myapp комбинируется с литеральным идентификатором a для генерации ключа. Другим распространенным примером является использование текущей директории:

```
key_t mykey;
mykey = ftok(".", 'a');
```

Обязанность проектирования алгоритма генерации ключей лежит на программисте прикладного приложения. Любой из этих методов должен учитывать конкурирующие состояния процессов и меры предотвращений «тупиковых» ситуаций. Для достижения наших демонстрационных целей мы ограничимся функцией `ftok()`. Если мы условимся, что каждый процесс-клиент будет запущен из собственного уникального домашнего каталога, то условие уникальности будет соблюдено полностью.

Следующие системные вызовы IPC используют значение возвращаемого ключа для создания или изменения доступа к объектам IPC. Команда ipcs отображает состояние всех объектов System V IPC.

- `ipcs -q`: показывать только очереди сообщений;
- `ipcs -s`: показывать только семафоры;
- `ipcs -m`: показывать только разделяемую память;

- `ipcs --help`: для любознательных.

По умолчанию показываются все три категории объектов. Рассмотрим пример простейшего вывода команды `ipcs`:

```
------ Shared Memory Segments --------
shmid owner     perms bytes     nattch status
------ Semaphore Arrays --------
^semid owner     perms nsems status
------ Message Queues --------
msqid owner     perms used-bytes messages
0     root      660   5          1
```

Мы видим единственную очередь сообщений с идентификатором 0. Она принадлежит пользователю root и обладает правами доступа 660, или `-rw-rw---`. Очередь содержит одно 5-тибайтовое сообщение.

Команда `ipcs` является мощным механизмом для мониторинга памяти ядра объектов IPC.

Команда ipcrm

ipcrm удаляет объекты IPC из ядра. Однако, поскольку объекты IPC могут быть удалены с помощью системных вызовов из прикладной программы, необходимость удалять их вручную возникает редко. Команда очень простая:

```
ipcrm - type (тип) id (номер)
```

Необходимо обозначить тип удаляемого объекта параметром. Обратитесь за подробностями к справке man. Идентификатор IPC можно определить с помощью команды `ipcs`. Помните, что идентификатор уникален только для объектов одного типа. Вот почему необходимо указывать тип объекта при его удалении.

Семафоры

Лучше всего семафоры представить, как счетчик доступа к публичным ресурсам. Обычно они используются, как замки, не позволяя одному процессу получить доступ к чему-либо, уже используемому другим процессом. Семафоры также могут предоставить эксклюзивный доступ к ресурсам данной машины или ограничить количество процессоров, использующих ресурс одновременно.

Этот механизм также обеспечивает функции для работы с разделяемой памятью System V. Разделяемая память обеспечивает доступ к глобальным переменным для различных процессов. Демоны httpd и даже другие программы (на Perl, C и других языках) могут получить доступ к этим данным с целью глобального обмена данными. Помните, однако, что разделяемая память не защищена от одновременного доступа.

Таблица 1 демонстрирует переменные Unix, которые определяют параметры ограничений разделяемой памяти. Каждый подвид Unix обладает своей документаций по этим переменным. К примеру, во FreeBSD они являются частью конфигурации ядра. Вам потребуется перекомпилировать ядро для принятия изменений.

Таблица 1. Переменные разделяемой памяти Unix.

SHMMAX	Максимальное количество разделяемой памяти, обычно 131072 байт
SHMMIN	Минимальное количество разделяемой памяти, обычно 1 байт
SHMMNI	Максимальное количество сегментов разделяемой памяти в системе, обычно 100
SHMSEG	Максимальное количество сегментов разделяемой памяти для одного процесса, обычно 6

Функции сообщений могут посылать сообщения одним процессам и принимать сообщения от других. Они являются простым и эффективным способом обмена данными между процессами без необходимости использования системы сокетов Unix.

Использование разделяемой памяти и семафоров

Изучая что-то новое, каждый разработчик хочет начать использовать эту технологию на практике, хотя бы написав простую программу «Hello, world!». Это нормально, поэтому приведем минимальные сведения, необходимые для того, чтобы вы могли создать этот простой тестовый пример.

Разделяемая память является быстрейшим видом IPC, но она требует синхронизации между сохранением и извлечением данных. Каков же тогда алгоритм использования разделяемой памяти? Попросту говоря, он заключается в следующем:

- Получить доступ к разделяемой памяти, используя семафор;
- Записать данные в сегмент разделяемой памяти;
- После завершения записи уведомить об этом другие программы, используя семафор.

Синхронизация с использованием семафора необходима, поскольку использование ресурса разделяемой памяти практически аналогично использованию файлового ресурса, где временное блокирование и разблокирование доступа позволяет избежать потери данных.

Теперь возникает вопрос, как использовать семафор. На самом деле это весьма просто и понятно из приведенного примера кода. Единственная тонкость заключается в том, что, поскольку семафоры могут блокировать и разблокировать ресурсы, они также могут блокировать друг друга.

Без аккуратного проектирования процессы могут попытаться овладеть одним и тем же семафором одновременно, вызывая долгие задержки при ожидании доступа. И, даже при должном проектировании, всегда будет верхний предел производительности в мультипроцессорных системах. За подробностями обратитесь к любой книге, посвященной многопроцессорным системам.

Здесь приводится несколько примеров того, как следует использовать различные функции семафоров и разделяемой памяти. В конце статьи также прилагается более длинный пример кода. Вот эти функции в порядке их появления.

```
int sem_get (int key [, int max_acquire [, int perm]])
```

Эта функция возвращает id семафора. Он будет положительным в случае успеха и равен `FALSE` в случае ошибки. Используйте id для доступа к семафору V с помощью ключа key.

Если это необходимо, семафор может обладать правами доступа, указанными в параметре perm. Значение по умолчанию: 0666. Параметр `max_acquire` управляет количеством процессов, которые могут одновременно получить доступ к семафору. По умолчанию он равен 1.

Вторичный вызов функции `sem_get()` для того же ключа вернет другой id, но они оба будут указывать на один и тот же семафор.

```
bool sem_acquire(int sem_identifier)
```

Эта функция возвращает TRUE в случае успеха и FALSE при неудаче. Она заблокируется, если необходимо, до тех пор, пока не займет запрошенный семафор. Процесс попытки занять запрошенный семафор будет длиться бесконечно, если он превысит значение параметра max_acquire.

Все семафоры, используемые в процессе, но не освобожденные явно, будут закрыты автоматически. Однако, это породит предупреждение.

```
int shm_attach(int key [, int memsize [, int perm]])
```

Эта функция создает или открывает сегмент разделяемой памяти. `shm_attach()` возвращает идентификатор для использования при получении доступа к сегменту с помощью заданного ключа. Размер сегмента в байтах будет равен `mem_size`. Значение по умолчанию равно sysvshm. init_mem в файле конфигурации PHP (или 10,000 байт, если оно не установлено в файле). Необязательный параметр perm определяет права доступа и равен 0666 по умолчанию.

Первый вызов с заданным ключом создаст сегмент. Второй вызов с тем же ключом вернет другой идентификатор, игнорируя два других параметра. Оба идентификатора предоставляют доступ к одному и тому же сегменту разделяемой памяти.

```
mixed shm_get_var(int id, int variable_key)
```

Эта функция возвращает переменную, идентифицируемую параметром variable_key, из сегмента разделяемой памяти с идентификатором id. Переменная остается в разделяемой памяти.

```
int shm_put_var(int shm_identifier, int variable_key, mixed variable)
```

Эта функция добавляет или обновляет значение переменной, заданной параметром variable_key, в сегменте разделяемой памяти, заданном параметром shm_identifier. Она позволяет работать с переменными любых типов.

```
bool sem_release(int sem_identifier)
```

Эта функция освобождает семафор, если он занят текущим процессом. В противном случае выдает предупреждение (warning). Возвращает TRUE в случае успеха и FALSE в случае ошибки.

После освобождения семафора для того, чтобы его повторно использовать, вызовите sem_acquire().

```
int shm_remove(int shm_identifier)
```

Эта функция удаляет сегмент разделяемой памяти и все содержащиеся в нем данные.

```
bool sem_remove(int sem_identifier)
```

Эта функция удаляет семафор, заданный с помощью sem_identifier, если он был создан с помощью sem_get. Возвращает TRUE в случае успеха и FALSE в случае ошибки. Если семафора с указанным идентификатором не существует, она породит предупреждение (warning). После удаления семафор недоступен.

За более подробной информацией об этих или других функциях обращайтесь к официальной документации PHP. Примеры кода смотрите в приложении к журналу (директория shm).

Alexander Prohorenko является системным администратором UNIX и администратором сетевой безопасности.

Оригинал статьи находится по адресу:
http://www.onlamp.com/pub/a/php/2004/05/13/shared_memory.html
Перевод предоставлен сайтом: http://detail.phpclub.ru

Люди

Фриланс в родном отечестве

Интервью брал:
Андрей Олищук [nw]

Профессия веб-программиста, как и любого другого разработчика программного обеспечения, очень тяготеет ко всякого рода подработкам. Порой подработки даже перерастают в уход с основной работы на вольные хлеба, а оттуда уже не далеко и до собственного дела. Ханс Заунер, президент NYPHP, в интервью для пилотного выпуска нашего журнала сказал, что большинство американских программистов все же совмещают основную работу с подработками. О том, как обстоят дела в родном отечестве нам поведал руководитель проекта weblancer.net Егор Ливитин (далее ЕЛ). Сервисы weblancer.net позволяют встречаться заказчикам и исполнителям заказов в одном месте.

nw: **Расскажите о вашем проекте: какие сервисы он предлагает, много ли у вас активных пользователей, велика ли команда, ведущая проект? Также хотелось бы узнать о вас: является ли weblancer.net вашей постоянной работой или это дополнительный проект лично для вас?**

ЕЛ: Наш сервис представляет собой интернет-биржу проектов. Цель данного сервиса - максимально упростить и ускорить процесс подбора Заказчиками исполнителей для выполнения различного рода проектов, работ, заданий. Для достижения этой цели основной акцент делается на автоматизацию всех процессов в сервисе, хотя, безусловно, служба поддержки необходима, как и для любого другого сервиса.

На данный момент в нашем сервисе зарегистрировано более 700 заказчиков и более 2000 разработчиков. К сожалению, необходимо признать, что активную деятельность проявляют лишь чуть более половины разработчиков. В связи с этим мы постоянно пытаемся совершенствовать наш сервис, делая его более доступным и понятным для большего числа пользователей. Над усовершенствованием нашего сервиса в данный момент работают 2 программиста и 1 веб-дизайнер. Службу поддержки формируют 2 человека.

Weblancer.net является не единственным проектом, разработанным и поддерживаемым нашей группой, но лично мне, как руководителю, этот сервис очень интересен, и я стараюсь уделять ему как можно больше внимания.

nw: **Среди фрилансеров существует большое количество «некомпьютерных» специалистов, например, журналистов. Почему вы выбрали сферу программирования и веб-дизайна? Нет ли у вас стремления в будущем расширить сферу деятельности?**

ЕЛ: Мы выбрали сферу программирования и веб-дизайна прежде всего потому, что потребность в разработке именно программных продуктов и сопутствующих им графических элементов очень велика, и, на мой взгляд, процент «компьютерных» специалистов в интернет-аудитории всё еще подавляющий.

Процент «компьютерных» специалистов в интернет-аудитории всё еще подавляющий

Но, тем не менее, мы планируем добавить такие сферы, как «перевод текстов», «написание статей», «администрирование сайтов» и т.д.

nw: **Я не заметил у вас английской версии сайта, вы принимаете только русскоязычные проекты? Не планируете ли в будущем проводить тендеры для зарубежных компаний? Что вам мешает это делать сейчас, и какие могут возникнуть проблемы?**

ЕЛ: Английская версия сервиса предусматривалась изначально и обязательно будет запущена в конце сентября - начале октября.

nw: **Что необходимо веб-разработчику, чтобы стать вашим пользователем? Велики ли затраты пользователей на услуги вашего сайта (комиссия, вступительные или иные взносы)?**

ЕЛ: Наш сервис является частично бесплатным. Для заказчиков в нашем сервисе комиссия отсутствует. Учетная запись разработчика также является бесплатной, однако, для использования всех возможностей нашего сервиса разработчики могут оплатить комиссию (от $1 до $10 в месяц) для изменения статуса учетной записи.

Таким образом, пользователи могут использовать наш сервис абсолютно без затрат. В свою очередь, все средства, которые поступают к нам от пользователей, изменивших свой статус в сервисе, мы расходуем на рекламу и продвижение нашего сервиса, что способствует увеличению числа заказов, размещаемых заказчиками.

nw: **Финансовый вопрос: сколько (примерно) могут зарабатывать активные участники через ваш сервис? Сколько в среднем тратят заказчики на один их проект? Как вы считаете, что более доходно для программистов: искать заказчиков самим или участвовать в проводимых вами тендерах?**

ЕЛ: Исходя из анализа работы нашего сервиса, можно сказать, что в среднем заказчики готовы тратить на разработку проектов от $500 до $1000. Это касается проектов средней сложности. Заработок же разработчиков зависит лишь от их умения находить общий язык с заказчиками и, конечно же, от качества и скорости выполнения работ.

Заработок разработчиков зависит лишь от их умения находить общий язык с заказчиками и, конечно же, от качества и скорости выполнения работ.

Безусловно, программистам необходимо сочетать самостоятельный поиск заказов с использованием нашего сервиса. Такой подход, на наш взгляд, является оптимальным.

nw: **В ваших тендерах со стороны разработчиков участвуют только фрилансеры, или бывают и фирмы? У кого из них, по вашему мнению, больше шансов получить проект и почему? Насколько высок уровень разработчиков, пользующихся вашим сервисом?**

ЕЛ: В нашем сервисе зарегистрировано более 500 различных фирм и студий, занимающихся разработкой программных продуктов и изготовлением графических элементов. Шансы всех разработчиков уравновешены, есть лишь разница в подходе к работе с нашим сервисом.

Так, например, фирмы предпочитают оплатить комиссию нашего сервиса и использовать все возможности, что, чаще всего, позволяет лучше обсудить заказ и, следовательно, увеличить шансы на победу в тендере.

nw: **Как вы в целом оцениваете рынок фриланса среди веб-разработчиков в России? Насколько, по вашему мнению, велик спрос на независимых разработчиков? Каковы тенденции этого рынка (уменьшение/увеличение спроса на независимых разработчиков)?**

ЕЛ: Рынок «фриланса» очень трудно оценить (измерить). Не секрет, что в зарубежных странах этой сфере уделяется всё больше внимания. По данным информационных агентств в странах бывшего СНГ, в частности в России, этот рынок вырос в несколько раз за последние несколько лет.

Тенденция к росту рынка фриланса всё же есть

Не берусь давать какие-либо прогнозы, но, учитывая рост числа различного рода онлайн-сервисов, мне кажется, что спрос на независимых разработчиков не будет снижаться.

nw: **Что, по вашему мнению, выгоднее в наше время в России/СНГ/Прибалтике: быть штатным специалистом в организации или быть фрилансером? Где проходит грань между фрилансером и частным предпринимателем? Или такой грани нет?**

ЕЛ: На мой взгляд грань между фрилансером и частным предпринимателем - сугубо субъективный психологический барьер :). Даже не знаю, как ответить на этот вопрос :)

По поводу "штатных специалистов" и "фрилансеров". Фрилансеры определенно обладают большей свободой действий. Имеют право выбора, что зачастую положительно сказывается на качестве выполнения работ.

nw: **Сейчас принято считать, что работа на зарубежные компании (для них это аутсорсинг) гораздо выгоднее работы на отечественных клиентов. Почему тогда ваш сервис жив и пользуется определенной популярностью, ведь большинство заказов отечественного происхождения?**

ЕЛ: Вероятно это связано с тем, что тенденция к росту рынка фриланса всё же есть. Хотя стоимость работ на нашем рынке оценивается гораздо ниже, чем на западном.

nw: **И, наконец, что вы могли бы пожелать читателям нашего журнала?**

ЕЛ: Хочется, конечно же, пожелать успехов в работе на рынке фриланса. Ведь успех каждого разработчика напрямую зависит от его стремления найти своё место в этой нише.

Мы же, в свою очередь, постараемся оказать максимальное содействие в этом всем вашим читателям и нашим пользователям :)

nw: **Спасибо!**

Интервью: CMS и открытый код

Интервью брал:
Андрей Козак

Недавно, Андрей Козак, который знаком нашим читателям по обзору «Свободно распространяемые», опубликованном в одном из прошлых номеров PHP Inside, взял отдельные экспресс-интервью у представителей двух отечественных компаний-разработчиков CMS: Романа Овчинникова (президента «Individ Company», CMS «Saitistika») и Натальи Грихиной (менеджера по рекламе и PR «Битрикс», CMS «Bitrix»). Им были заданы следующие вопросы, касающиеся представляемых ими разработок. Соответственно, Роман Овчинников рассказывал о CMS «Saitistika», а Наталья Грихина вела речь о CMS «Bitrix».

Андрей Козак (АК): **Ваши продукты занимают одно из лидирующих мест на рынке CMS в странах СНГ и в частности в России. Эти программы поставляются с открытыми исходными кодами?**

Роман Овчинников (РО): Примерно 95% кода открыто.

Наталья Грихина (НГ): Да, коммерческие версии продукта поставляются в исходных текстах.

АК: **Что способствовало тому, что вы оставили большое количество исходников открытыми?**

РО: Причин открытости кода много, и они все традиционные - пользователи хотят контролировать и изменять поведение отдельных функций системы, так проще накатывать изменения от производителя и т.д.

НГ: Продукт поставляется с открытыми исходниками, чтобы предоставить свободу действий разработчикам. С исходными текстами разработчики смогут в случае необходимости подстраивать продукт под свои требования, создавать новые модули для продукта. Кроме того, с открытыми исходниками нет необходимости устанавливать на сервере дополнительное оборудование для раскодирования.

АК: А как вы боретесь с проблемой кражи кода?

РО: Пока таких прецедентов не обнаружено. В случае обнаружения будем обращаться в суд.

НГ: С подобными действиями можно бороться только организационными методами. Защита кода от несанкционированного доступа обеспечивается хостером. Если же сам пользователь продукта попытается скопировать исходники и использовать их, то мы будем действовать в соответствии с законодательством об авторских правах.

АК: **В процессе разработки у вас не возникало идей воспользоваться программой-кодировщиком?**

РО: Если имеется ввиду софт, который преобразовывает код в непонятный для человека, то не возникало. Мы не стремимся от покупателя что-то скрыть.

НГ: Пробная версия нашего продукта закодирована с помощью Zend Optimizer. Пользователи могут скачивать продукт с нашего сайта и использовать его 30 дней. По функциональным возможностям пробная версия не ограничена.

АК: А в самом коде привязки к определенному доменному имени или IP-адресу не имеется?

РО: Нет.

НГ: Нет, в коде нет подобных привязок. Однако каждый лицензионный ключ регистрируется на определенный домен. Если пользователь будет использовать этот же лицензионный ключ для другого сайта, ключ будет блокирован.

АК: А если с юридической позиции - Вы заключаете договор или что-то в этом роде?

РО: Поставка продукта происходит на основе договора поставки, в котором оговорены все условия. Основным же документом служит лицензионное соглашение, где регламентированы все права и обязанности сторон.

НГ: Каждый пользователь продукта принимает условия лицензионного соглашения.

АК: Спасибо!

Команда этого выпуска

Авторы и переводчики

- Александр Косарев
- Данил Миронов [patrungel]
- Вера Козлова
- Кирилл Крушняков
- Андрей Козак
- Андрей Олищук [nw]

Редакционная коллегия, корректировка

- Александр Смирнов [PHPclub];
- Елена Тесля [Lenka];
- Александр Войцеховский [young];
- Антон Чаплыгин;
- Андрей Олищук [nw], координатор проекта PHP Inside;
- Дмитрий Попов;
- http://phpclub.ru

Подготовка обложки, верстка

- Денис Зенькович;
- Антон Чаплыгин;
- Андрей Олищук [nw].

Спасибо всем участникам проекта!

Координаты редакции журнала:

http://phpinside.net

nw@phpinside.net

Обсуждаем номер на

http://phpclub.ru/talk

Для заметок