УДК 004.021

ОПТИМИЗАЦИЯ АРХИТЕКТУРЫ БАЗ ДАННЫХ С ЦЕЛЬЮ ПОВЫШЕНИЯ НАДЕЖНОСТИ И СКОРОСТИ РАБОТЫ

К.Х. ГРИГОРЯН
(Представлено: Д.В. ПЯТКИН)

Рассматриваются технологии и приемы, для реализации отказоустойчивости серверной части приложения и увеличения отклика базы данных. Проанализированы общие приемы оптимизации работы с базами данных.

Довольно часто основной проблемой и упущением при работе с большими объемами данных является плохо организованная часть приложения, отвечающая за хранение данных. Современные технологии требуют повышенного внимания к back-end части приложения. А разработка back-end'а начинается с базы данных. Базы данных, их устройство и архитектура играют ключевую роль в работоспособности приложения и целостности данных пользователей.

Репликация

Репликация позволяет создать полный дубликат базы данных. Так, вместо одного сервера в разрабатываемом продукте будет несколько. Чаще всего используют схему master-slave:

1. ***Master*** – это основной сервер БД, куда поступают все данные. Все изменения в данных (добавление, обновление, удаление) должны происходить на этом сервере.
2. ***Slave*** – это вспомогательный сервер БД, который копирует все данные с мастера. С этого сервера следует читать данные. Таких серверов может быть несколько.

Репликация позволяет использовать два или больше одинаковых серверов вместо одного. Операций чтения (SELECT) данных часто намного больше, чем операций изменения данных (INSERT/ UPDATE). Поэтому репликация позволяет разгрузить основной сервер за счет переноса операций чтения на Slave.

Для организации технологии репликации, в программном коде продукта необходимо объявить два соединения с базой данных. Одно – для мастера, одно – для Slave. Пример реализации предоставлен в листинге 1.

Листинг 1 – Несколько подключений в виде репликации.

```
<?
$master = mysql_connect('10.10.0.1', 'root',);
$slave = mysql_connect('10.10.0.2', 'root',);
  // исполняемый код
$q = mysql_query('INSERT INTO users ...', $master);
// другая часть приложения
$q = mysql_query('SELECT * FROM users WHERE...', $slave);
```

Репликация обычно поддерживается самой СУБД (например, MariaDb) и настраивается независимо от приложения.

Следует отметить, что репликация сама по себе не очень удобный механизм масштабирования. Причина тому – рассинхронизация данных и задержки в копировании с Master на Slave. Однако это отличное средство для обеспечения отказоустойчивости. Вы всегда можете переключиться на слейв, если мастер ломается и наоборот. Чаще всего репликация используется совместно с шардингом, именно из соображений надежности.

Шардинг

Шардинг – это другая техника масштабирования работы с данными. Суть его в разделении базы данных на отдельные части так, чтобы каждую из них можно было вынести на отдельный сервер.

Есть несколько типов шардинга – вертикальный и горизонтальный.

Горизонтальный шардинг – это разделение одной таблицы на разные сервера. Это необходимо использовать для таблиц, которые хранят внушительные объемы информации и не умещаются на одном сервере. Разделение таблицы на куски делается по такому принципу:

1) на нескольких серверах создается одна и та же таблица (только структура, без данных);
2) в приложении выбирается условие, по которому будет определяться нужное соединение (например, четные на один сервер, а нечетные – на другой);
3) перед каждым обращением к таблице происходит выбор нужного соединения.

Предположим, что есть приложение, которое работает с высоконагруженной таблицей, хранящей фотографии пользователей. Также имеется два сервера (обычно они называются шардами) для этой таблицы. Возможный сценарий использования – для нечетных пользователей хранить и извлекать информацию, используя первый сервер, а для четных – второй. Таким образом, на каждом из серверов будет только часть всех данных о фотографиях пользователей. Пример представлен в листинге 2.

Листинг 2 – Несколько подключений, в стиле горизонтальный шардинг.

```
<?
$image_connections = [
        '1' => '10.10.0.1',
        '2' => '10.10.0.2',
];

$user_id = $_SESSION['user_id'];
// получение фотографий для пользователя $user_id
$connection_num = $user_id % 2 == 0 ? 1 : 2;
$connection =
mysql_connect($image_connections[$connection_num],'root', '');
$q = mysql_query('SELECT * FROM images WHREE user_id = ' . intval($user_id), $connection);
```

Результат операции $user_id % 2 будет остатком от деления на 2. То есть для четных чисел – 0, а для нечетных – 1.

Горизонтальный шардинг – это очень мощный инструмент масштабирования данных.

Вертикальный шардинг – это выделение таблицы или группы таблиц на отдельный сервер.

С целью применения такой разновидности шардинга, как вертикальный, необходимо использовать соответствующее соединение для работы с каждой таблицей.

Пример представлен в листинге 3.

Листинг 3 – Использование нескольких подключений в стиле вертикальный шардинг.

```
<?
$users_connection = mysql_connect('10.10.0.1', 'root', '');
$image_connection = mysql_connect('10.10.0.2', 'root', '');

$q = mysql_query('SELECT * FROM users WHERE ...', $users_connection);
$q = mysql_query('SELECT * FROM images WHERE...', $image_connection);

$q = mysql_query('SELECT * FROM albums WHERE...', $image_connection);
```

В отличие от репликации, мы используем разные соединения для любых операций, но с определенными таблицами.

Не следует применять технику шардинга ко всем таблицам. Правильный подход – это поэтапный процесс разделения растущих таблиц. Следует задумываться о горизонтальном шардинге, когда количество записей в одной таблице является заметным фактором на производительность работы приложения. Шардинг и репликация зачастую используются совместно. Пример такой реализации представлен в листинге 4.

Листинг 4 – Совместное использование шардинга и репликации

```
<?
$image_connections = [
  '1' => [
        'master' => '10.10.0.10',
        'slave' => '10.10.0.11',
  ],
  '2' => [
        'master' => '10.10.0.20',
        'slave' => '10.10.0.21',
  ],
];

$user_id = $_SESSION['user_id'];
```

```
// Читаем данные со слейвов
$connection_num = $user_id % 2 == 0 ? 1 : 2;
$connection = mysql_connect($image_connections[$connection_num]['slave'], 'root', '');
$q = mysql_query('SELECT * FROM image WHREE user_id = ' . intval($user_id), $connection);

//исполняемый код...
//Изменение данных происходит на мастерах
$image_id = 7;
$connection_num = $user_id % 2 == 0 ? 1 : 2;
$connection = mysql_connect($image_connections[$connection_num]['master'], 'root', '');
$q = mysql_query('UPDATE images SET views = views + 1 WHREE image_id = ' . intval($image_id),
$connection);
  '2' => '10.10.0.2',
```

Такая схема часто используется не только для масштабирования, но и для обеспечения отказоустойчивости. Так, при выходе из строя одного из серверов шарда, всегда будет запасной.

Заключение

Итогом данной работы можно выделить рассмотренные некоторые возможности для грамотной реорганизации баз данных с целью улучшения оптимизации и увеличения отказоустойчивости.

ЛИТЕРАТУРА

1. Semantic UI [Электронный ресурс]. – Режим доступа: https://semantic-ui.com/. – Дата доступа: 26.09.2017.
2. Semantic UI Introduction [Электронный ресурс]. – Режим доступа: http://learnsemantic. com/developing/ customizing.html#the-future-of-themes. – Дата доступа: 26.09.2017.
3. Semantic UI tutorials [Электронный ресурс]. – Режим доступа: https://github.com/Semantic-Org/ Semantic-UI/wiki/Tutorials. – Дата доступа: 26.09.2017.