Introduction to Column Oriented Databases in PHP

By Slavey Karadzhov <slavey.karadzhov@zend.com>
About Me

- Name: Slavey Karadzhov (Славей Караджов)
- Programmer since my early days
- PHP programmer since 1999
- Open source fan and supporter
- Hobby (software) inventor
  - Won the best “IT Innovation of the Year 2004” from PC Magazine
- Working for Zend Technologies since 2008
- In Stuttgart, Germany
- As a programmer, consultant and trainer
Agenda

• How Do They Look?
• Why New Type Of Database?
• The Details
• Supported Operations
• Available Open Source CODB engines
• Putting Everything Together
• How To Use It From PHP
• Is CODB For Me?
• Questions
How Do They Look?

... and what is the difference to relational databases?
How do they look?

- **Representation in Relational (Row Oriented) DB**

<table>
<thead>
<tr>
<th>Row</th>
<th>ColumnAFamily</th>
<th>ColumnAQualifier</th>
<th>ColumnB</th>
<th>ColumnC</th>
</tr>
</thead>
<tbody>
<tr>
<td>row1</td>
<td>One</td>
<td>Едно</td>
<td>bg</td>
<td></td>
</tr>
<tr>
<td>row2</td>
<td>Two</td>
<td>Две</td>
<td>bg</td>
<td></td>
</tr>
<tr>
<td>row3</td>
<td>Three</td>
<td>Три</td>
<td>bg</td>
<td></td>
</tr>
<tr>
<td>row4</td>
<td>One</td>
<td>Eins</td>
<td>de</td>
<td>German</td>
</tr>
<tr>
<td>row5</td>
<td>Two</td>
<td>Zwei</td>
<td>de</td>
<td>German</td>
</tr>
<tr>
<td>row6</td>
<td>Three</td>
<td>Drei</td>
<td>de</td>
<td>German</td>
</tr>
</tbody>
</table>

- **Representation in Column Oriented DB (CODB)**

<table>
<thead>
<tr>
<th>Row</th>
<th>ColumnA</th>
<th>ColumnB</th>
<th>ColumnC</th>
</tr>
</thead>
</table>
| row1 | ColumnA: One=Едно
ColumnA: Two=Две
ColumnA: Tree=Три | ColumnB=bg | <no-memory-allocated-for-it> |
| row2 | ColumnA: One=Eins
ColumnA: Two=Zwei
ColumnA: Tree=Drei | ColumnC=de | German |
How do they look? (2)

• Table [http://labs.google.com/papers/bigtable.html](http://labs.google.com/papers/bigtable.html)
  
  ‣ Associative array
  ‣ Sparse
  ‣ Multidimensional
  ‣ Distributed
  ‣ Persistent
  ‣ Sorted

```php
<?php
$table = array (
    'row1' => array (
        'columnA' => array (
            'one' => 'едно',
            'two' => 'две',
            'three' => 'три',
        ),
        'columnB' => 'bg',
    ),
    'row2' => array (
        'columnA' => array (
            'one' => 'eins',
            'two' => 'zwei',
            'three' => 'drei',
        ),
        'columnB' => 'de',
        'columnC' => 'German',
    ),
);```
How do they look? (3)

- Column Families
  - Family - columnA, columnB, ...
  - Qualifier - one, two, three
  - Three dimensions + 1
    - Row, Family, Qualifier
    - +1 more dimension: time

```php
<?php
$table = array (
  'row1' => array (
    'columnA' => array (  
      'one' => 'едно',
      'two' => 'две',
      'three' => 'три'
    ),
    'columnB' => 'bg',
  ),
  'row2' => array (  
    'columnA' => array (   
      'one' => 'eins',
      'two' => 'zwei',
      'three' => 'drei'
    ),
    'columnB' => 'de',
    'columnC' => 'German'
  )
);
```
Why New Type Of Database?

... and aren’t the relational databases enough?
Why New Type Of Database?

2001 / 1 users

2002 / 2 PHP machines, 1 MySQL machine / 2k users

2003 / 4+ PHP machines, 1 MySQL machine / 20k users
Why New Type Of Database?(2)

• Problem 1:
  ‣ The Db is not scalable
  ‣ 2x more powerful machine brings less than 1.5x performance boost and costs a lot

• Solution 1:
  ‣ Sharding
    ‣ Move different tables to different DBs
    ‣ Splits one table by row into two or more tables on different DB machines
  ‣ Using commodity software
Why New Type Of Database?(3)

- Sharding has to be done on the client
  - MySQL and other relational DBs do not support sharding on the server side

- Problem 2:
  - The PHP code becomes complex
    - Difficult to use and understand
  - Resizing the shards is manual painful work
  - Sorting and merging of data becomes real problem
  - For speed more data needs to be redundant
  - Joins between shards are almost impossible
Why New Type Of Database?(4)

Example:

- Prerequisites:
  - Social website.
  - Every user has friends. Not all friends in one shard
  - Every user has activity: adding a comment, blog post, image upload, etc.
  - Every user has avatar icon and name
- Give me the last 20 activities from my friends
  - Sound simple, right?
Why New Type Of Database?(5)

• Example (cont’d):
  ‣ Give me the last 20 activities from my friends
    • Here is how it is done:
      ‐ Get the list of friends for an user
      ‐ Get information about the friends - name, avatar, shard
      ‐ Group the friends in shards
      ‐ For each shard get the last 20 activities per friend
      ‐ In PHP merge all the incoming data and sort it by date so that the first 20 show up
  ‣ For this 20 results the PHP needs (quite often)
    • 60 or more MB RAM
    • >10 seconds CPU time
Why New Type Of Database?(6)

- Activity table as a CODB table

<table>
<thead>
<tr>
<th>Row</th>
<th>Activity</th>
<th>Type</th>
<th>Friends</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>(MAXINT-timestamp())-userid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>9999999999999999999999999999999-000001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>999999999999999999999999999999977-000012</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- And in CODB you have to do
  - SCAN activity, \{ COLUMNS="friends:0000001", LIMIT=20 \}
- Be patient, the details are in the next slides
The Details

… must-know when designing a table
The Details

• There are **no joins** between tables.
• The row keys are always **sorted in alphabetical order**
• The type of the column value is always string, unless a counter
• The generation of the row key is very important
  - There is no auto-increment of the key
  - No key constraints (no primary, no unique, etc)
The Details (2)

- The generation of the row key is very important (2)
  - The fastest search is by using the row key
    - Secondary indexes are “almost” not supported
  - One cannot search in the value of the columns
    - Nothing like columnB="de"
  - But can search if a column family exists:
    - “columA:one”
Supported Operations

... list of the most important ones
Supported Operations

- **CREATE** - creating a table

  - `CREATE tableName, columnA, columnB, columnC`
  
  - Plus options to the table column families:
    - **Versions** - how many versions of the cell value to keep
    - **In Memory** - if the cell value should be kept in memory for faster search
    - **Counter** - if the column family will be used for autonomous increment and decrement
      - The value of the cell is integer and adding or decreasing the value is thread-safe (no race conditions)
    - **Compression** - good for storing, bad for searching
Supported Operations(2)

• **PUT** - for adding a cell
  - PUT table ‘rowKey’, ‘columnFamilyKey’, ‘value’
    - PUT table ‘row1’, ’columnA:one’, ‘eins’
    - PUT table ‘row1’, ’columnA:two’, ‘zwei’
    - PUT table ‘row1’, ’columnA:three’, ‘drei’
    - PUT table ‘row1’, ’columnB’, ‘de’
    - PUT table ‘row1’, ’columnC’, ‘German’

• **GET** - get row by its row key value
  - GET table, ‘rowKey’, { COLUMNS, VERSIONS}
    - GET table, ‘row2’, { ‘columnB:one’}
Supported Operations(3)

• There is no UPDATE!

• DELETE - for deleting a cell or complete row
  ‣ DELETE table, ‘rowKey’, ‘columnFamilyKey’ - for a cell
  ‣ DELETE table, ‘rowKey’

• SCAN - find rows
  ‣ SCAN table, {COLUMNS=array(), LIMIT=<int>, STARTROW=string, STOPROW}
    • SCAN table, { COLUMNS=[‘columnB:one’]}
      - Returns all rows that have the cell column family B and qualifier one
    • SCAN table, { STARTROW=‘ro’, STOPROW=‘row3’}
      - Returns all rows for which the row key starts with ro and ends with row3
Available Open Source CODB engines

... as of today
Available Open Source CODB engines

• Hbase
  ‣ Uses Hadoop as distributed storage backend
  ‣ Written in Java
  ‣ Large community base
  ‣ No single point of failure (ZooKeeper)

• Hypertable
  ‣ Also Hadoop
  ‣ Written in C
  ‣ May be faster and less memory/CPU intensive
  ‣ Smaller community base
  ‣ No single point of failure (Own manager for the Zoo)
Putting Everything Together

... without PHP
Putting Everything Together

- Activity table

<table>
<thead>
<tr>
<th>Row</th>
<th>Activity</th>
<th>Type</th>
<th>Friends</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>activity:title=Blog Post activity:content='This is the content of the post etc. etc.'</td>
<td>type:blog</td>
<td>friends:&lt;friend-id&gt;=0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>friends:&lt;friend-id&gt;=1</td>
</tr>
<tr>
<td>2</td>
<td>(MAXINT-timestamp())-userid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>activity:title=ColDB activity:content='Cause headache to my poor brain'</td>
<td>type:comment</td>
<td>friends:00000012=0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>friends:00000014=1</td>
</tr>
<tr>
<td>4</td>
<td>activity:title=ColDB activity:content='Cool DB'</td>
<td>type:comment</td>
<td>friends:00000002=0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>friends:00000001=1</td>
</tr>
</tbody>
</table>

- SCAN activity, { COLUMNS="friends:0000001", LIMIT=20 }

- The result
  - Sorted by key
  - Limited to 20 results
  - Taken, as if, from one server (no extra sharding logic)
Putting Everything Together(2)

- **SCAN activity**, `{ COLUMNS="friends:0000001", LIMIT=20 }

<table>
<thead>
<tr>
<th>Row</th>
<th>Activity</th>
<th>Type</th>
<th>Friends</th>
</tr>
</thead>
</table>
| 2   | (MAXINT-timestamp())-userid 
    activity:title=Blog Post 
    activity:content='This is the content of the post etc. etc.' | type:blog | friends:<friend-id>=0 
friends:<friend-id>=1 |
| 3   | 99999999999999999-000001 
    activity:title=ColDB 
    activity:content='Cause headache to my poor brain' | type:comment | friends:00000002=0 
friends:00000002=1 |
| 4   | 9999999999999999977-000012 
    activity:title=ColDB 
    activity:content='Cool DB' | type:comment | friends:00000002=0 
friends:00000001=1 |

- **What needs to be done additionally is:**
  - Get the user data for your friends and merge it in PHP
How To Use It From PHP

... some code examples
How To Use It From PHP

- Activity as an ActiveRecord

```php
class Activity extends Zend_Cdb_ActiveRecord {
    //...

    $activity = new Activity();
    $activity->setRow('99999999999999-1');
    $activity['columnA:one'] = 'eins';
    $activity['columnA:two'] = 'zwei';
    $activity['columnB'] = 'de';
    $activity->save();

    $friendActivities = $activity->scan(null, null, 'friends:00001',20);
    foreach ($friendActivities as $activity) {
        print $activity['activity:title']."="."$activity['activity:content']."\n";
    }
```
How To Use It From PHP(2)

- ActiveRecord

```
<?php
abstract class Zend_Cdb_ActiveRecord implements ArrayAccess {
    protected $_table;
    protected $_data = array();
    protected $_changedData = array();

    * @param offset
    public function offsetExists ($offset) {}

    * @param offset
    public function offsetGet ($offset) {}

    * @param offset
    public function offsetSet ($offset, $value) {}

    * @param offset
    public function offsetUnset ($offset) {}

    * @param string row
    abstract public function get($row);

    abstract public function setRow($row);

    * @param string $startRow
    abstract public function scan($startRow, $stopRow=null, $columns=array(), $limit=null);

    abstract public function save();
}
```
How To Use It From PHP (3)

• The PHP code is much simpler

• Resizing the shards is done automatically and transparently
  ‣ If the load gets high just add new CODB server

• Failover, no single point of failure

• Sorting is fast

• Merging of the data, if the table is well-designed, is also fast
Is CODB For Me?

... who should use it
Is CODB For Me?

• Require a lot of memory and hard disk space
  ▶ More redundant, more de-normalized
  ▶ Rational DBs have decades of optimizations behind them
  ▶ Does not makes sense to use it for small websites
    ▶ Although this may change if CODB becomes main stream

• Not for a bank system
  ▶ No transaction safety
  ▶ Can be partially/eventually consistent

• Works fine if the final result is a relatively small set of data
Thank you very much

... questions anyone?
References

- BigTable paper: http://labs.google.com/papers/bigtable.html
- Hypertable: http://hypertable.org/
- Apache Hbase: http://hbase.apache.org/
- Apache Hadoop: http://hadoop.apache.org/
- Active Record Pattern: http://en.wikipedia.org/wiki/Active_record_pattern