



The PHP Company

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

	A	B	C	D	E
1	Row	ColumnAFamily	ColumnAQualifier	ColumnB	ColumnC
2	row1	One	Едно	bg	
3	row2	Two	Две	bg	
4	row3	Three	Три	bg	
5	row4	One	Eins	de	German
6	row5	Two	Zwei	de	German
7	row6	Three	Drei	de	German

- Representation in Column Oriented DB (CODB)

	A	B	C	D
1	Row	ColumnA	ColumnB	ColumnC
2	row1	ColumnA:One=Едно ColumnA:Two=Две ColumnA:Tree=Три	ColumnB=bg	<no-memory-allocated-for-it>
3	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>

- ▶ Associative array
- ▶ Sparse
- ▶ Multidimensional
- ▶ Distributed
- ▶ Persistent
- ▶ Sorted

```
1 <?php
2 $table = array (
3     'row1' => array (
4         'columnA' => array (
5             'one' => 'одно',
6             'two' => 'две',
7             'three' => 'три'
8         ),
9         'columnB' => 'bg',
10    ),
11    'row2' => array (
12        'columnA' => array (
13            'one' => 'eins',
14            'two' => 'zwei',
15            'three' => 'drei'
16        ),
17        'columnB' => 'de',
18        'columnC' => 'German'
19    )
20 );
```

How do they look? (3)

- Column Families

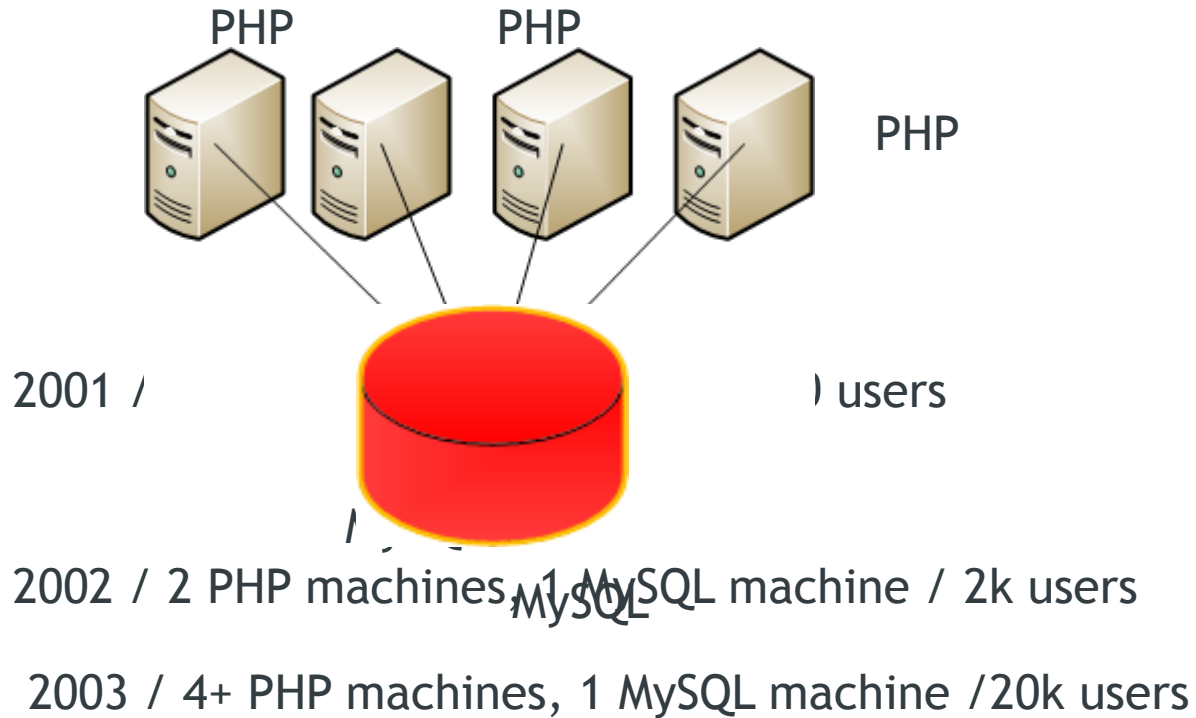
- ▶ Family - columnA, columnB,...
- ▶ Qualifier - one, two, three
- ▶ Three dimensions + 1
 - Row, Family, Qualifier
 - +1 more dimension: time

```
1 <?php
2 $table = array (
3     'row1' => array (
4         'columnA' => array (
5             'one' => 'одно',
6             'two' => 'две',
7             'three' => 'три'
8         ),
9         'columnB' => 'bg',
10    ),
11    'row2' => array (
12        'columnA' => array (
13            'one' => 'eins',
14            'two' => 'zwei',
15            'three' => 'drei'
16        ),
17        'columnB' => 'de',
18        'columnC' => 'German'
19    )
20 );
```

Why New Type Of Database?

... and aren't the relational databases enough?

Why New Type Of Database?



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 web site.
 - 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

	A	B	C	D
1	Row	Activity	Type	Friends
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	99999999999999999999999999999999-000001	activity:title=CoIDB activity:content='Cause headache to my poor brain'	type:comment	friends:00000012=0 friends:00000014=1
4	99999999999999999999999999999977-000012	activity:title=CoIDB activity:content='Cool DB'	type:comment	friends:00000002=0 friends:00000001=1

- And in CODB you have to do

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

- Be patient, the details are in the next slides



The PHP Company

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:
 - `"columnA:one"`



The PHP Company

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



The PHP Company

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)



The PHP Company

Putting Everything Together

... without PHP

Putting Everything Together

- Activity table

	A	B	C	D
1	Row	Activity	Type	Friends
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	99999999999999999999999999999999-000001	activity:title=CoIDB activity:content='Cause headache to my poor brain'	type:comment	friends:00000012=0 friends:00000014=1
4	99999999999999999999999999999977-000012	activity:title=CoIDB activity:content='Cool DB'	type:comment	friends:00000002=0 friends:00000001=1

- 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}

	A	B	C	D
1	Row	Activity	Type	Friends
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	99999999999999999999999999999999-000001	activity:title=CoIDB activity:content='Cause headache to my poor brain'	type:comment	friends:00000012=0 friends:00000014=1
4	99999999999999999999999999999977-000012	activity:title=CoIDB 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



The PHP Company

How To Use It From PHP

... some code examples

How To Use It From PHP

- Activity as an ActiveRecord

```
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

```
1 <?php
2 abstract class Zend_Cdb_ActiveRecord implements ArrayAccess {
3     protected $_table;
4     protected $_data = array();
5     protected $_changedData = array();
6
7
8     * @param offset[]
9
10    public function offsetExists ($offset) {[]
11
12
13
14
15    * @param offset[]
16
17    public function offsetGet ($offset) {[]
18
19
20
21
22    * @param offset[]
23
24    public function offsetSet ($offset, $value) {[]
25
26
27
28
29
30
31    * @param offset[]
32
33    public function offsetUnset ($offset) {[]
34
35
36
37
38
39    * @param string row[]
40
41    abstract public function get($row);
42
43
44    abstract public function setRow($row);
45
46
47
48    * @param string $startRow[]
49
50    abstract public function scan($startRow, $stopRow=null, $columns=array(), $limit=null);
51
52
53
54
55    abstract public function save();
56 }
```

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



The PHP Company

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/>
- Sharding:
[http://en.wikipedia.org/wiki/Shard_\(database_architecture\)](http://en.wikipedia.org/wiki/Shard_(database_architecture))
- Active Record Pattern:
http://en.wikipedia.org/wiki/Active_record_pattern