MongoDB schema design

PHP Benelux - Leuven, Belgium - Mar 28th, 2012
Derick Rethans - derick@10gen.com - twitter: @derickr
Derick Rethans

- Dutchman living in London
- PHP mongoDB driver maintainer for 10gen (the company behind mongoDB)
- Author of Xdebug
- Author of the mcrypt, input_filter, dbus, translit and date/time extensions
Database landscape

Scalability & performance vs. depth of functionality

Memcached

Key/Value

MongoDB

RDBMS
NoSQL

- **Key/value**
  - redis
  - riak
  - Memcached

- **Column**
  - Cassandra
  - HBase

- **Graph**
  - Neo4j

- **Document**
  - CouchDB
  - MongoDB
- JSON Document: the data (row)
- Collection: contains documents (table, view)
- Index
- Embedded Document (~join)
Documents

- Stored as BSON (Binary JSON)
- Can have embedded documents
- Have a unique ID (the `_id` field)
- Are schemaless

Simple document:

```json
{
    "_id" : ObjectId("4cb4ab6d7addf98506010001"),
    "handle" : "derickr",
    "name" : "Derick Rethans"
}
```

Document with embedded documents:

```json
{
    "_id" : "derickr",
    "name" : "Derick Rethans",
    "talks" : [
    { "title" : "Profiling PHP Applications",
    },
    { "title" : "Xdebug",
    }
    ]
}
```
RDBMS: Normalisation

- 1970 E.F. Codd introduces 1st Normal Form (1NF)
- 1971 E.F. Codd introduces 2nd and 3rd Normal Form (2NF, 3NF)
- 1974 Codd & Boyce define Boyce/Codd Normal Form (BCNF)
- 2002 Date, Darween, Lorentzos define 6th Normal Form (6NF)

Goals:
- Avoid anomalies when inserting, updating or deleting
- Minimize redesign when extending the schema
- Make the model informative to users
- Avoid bias towards a particular style of query
Blog in MongoDB

User Collection
- Name
- Email Address
- Author

Blog Collection
- Name
- URL
- Author
- Blog Entry
  - Article
  - Publish date
- Comment
  - Comment
  - Comment Date
  - Comment By
- Tag
  - Value
Schema considerations

- Access Patterns?
- Read / Write Ratio
- Types of updates
- Types of queries
- Data life-cycle

Considerations
- No Joins
- Document writes are atomic
Inheritance

Shape
- area

Circle
- radius

Square
- d

Rect
- length
- width
Single table inheritance - RDBMS
Single table inheritance - MongoDB

{ _id: "1", type: "circle", area: 3.14, radius: 1}
{ _id: "2", type: "square", area: 4, d: 2}
{ _id: "3", type: "rectangle", area: 10, length: 5, width: 2}
• Simple to query across sub-types
• Indexes on specialized values will be small
SELECT cpe.entity_id, value AS name
FROM catalog_product_entity cpe
INNER JOIN eav_attribute ea
    ON cpe.entity_type_id = ea.entity_type_id
INNER JOIN catalog_product_entity_varchar cpev
    ON ea.attribute_id = cpev.attribute_id AND cpe.entity_id = cpev.entity_id
WHERE ea.attribute_code = 'name'
EAV: Entity Attribute Value

SELECT entity_id, attribute_code, value
FROM catalog_product_entity_text cpev
JOIN eav_attribute ea ON cpev.attribute_id = ea.attribute_id;

<table>
<thead>
<tr>
<th>entity_id</th>
<th>attribute_code</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>description</td>
<td>Cute elephant</td>
</tr>
<tr>
<td>1</td>
<td>short_description</td>
<td>It’s cute</td>
</tr>
<tr>
<td>1</td>
<td>meta_keyword</td>
<td>NULL</td>
</tr>
</tbody>
</table>

SELECT entity_id, attribute_code, value
FROM catalog_product_entity_int cpev
JOIN eav_attribute ea ON cpev.attribute_id = ea.attribute_id;

<table>
<thead>
<tr>
<th>entity_id</th>
<th>attribute_code</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>status</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>visibility</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>tax_class_id</td>
<td>2</td>
</tr>
</tbody>
</table>
In MongoDB

{
    '_id': 1,
    'name': 'Elephpant',
    'url_key': 'elephpant',
    'description': 'Cute elephpant',
    'short_description': "It's cute",
    'status': 1,
    'visibility': 4,
    'tax_class_id': 2,
}


One to Many relationships can specify:
• degree of association between objects
• containment
• life-cycle
Embedded Array / Array Keys

- slice operator to return subset of array
- some queries harder e.g find latest comments across all documents

```json
blogs: {
  author: "Hergé",
  date: "Tue Mar 28 2012 12:41:29 GMT",
  comments: [
    {
      author: "Kyle",
      date: "Tue Mar 28 2012 12:41:54 GMT",
      text: "great book"
    }
  ]
}
```
One to Many (1:n)

Embedded Tree
• single document
• natural
• hard to query

blogs: {
  author: "Hergé",
  date: "Tue Mar 28 2012 12:41:29 GMT",
  comments: [
    {
      author: "Kyle",
      date: "Tue Mar 28 2012 12:41:54 GMT",
      text: "great book"
    }
  ]
}
Normalised (with two collections)

- most flexible
- more queries

blogs: {
    author: "Hergé",
    date: "Tue Mar 28 2012 12:41:29 GMT",
    comments: [
        { comment: ObjectId("1") }
    ]
}

comments: {
    _id: "1",
    author: "Kyle",
    date: "Tue Mar 28 2012 12:41:54 GMT",
}
Many to Many (n:m)

- products can be in many categories
- category can have many products
Many to Many (n:m)

products:
{ _id: 10, name: "Blue elephant", category_ids: [ 4, 7 ] }
{ _id: 11, name: "Pink elephant", category_ids: [ 4, 8 ] }

categories:
{ _id: 4, name: "toys", product_ids: [ 10, 11 ] }

All categories for a given product (pink elephant):
   db.categories.find( { product_ids: 11 } );
All products for a given category (toys):
   db.products.find( { category_ids: 4 } );
Updates need to be done in two collections
products:
{ _id: 10, name: "Blue elephant", category_ids: [ 4, 7 ] }
{ _id: 11, name: "Pink elephant", category_ids: [ 4, 8 ] }

categories:
{ _id: 4, name: "toys" }
{ _id: 8, name: "everything pink" }

All categories for a given product (pink elephant):
product = db.products.find( { category_ids: 4 } );
db.categories.find( { _id: { $in: product.category_ids } } );

All products for a given category (toys):
db.products.find( { category_ids: 4 } );
products:
{ _id: 10, name: "Blue elephant" }
{ _id: 11, name: "Pink elephant" }

categories:
{ _id: 4, name: "toys",   product_ids: [ 10, 11 ] }  

All categories for a given product (pink elephant):
```
db.categories.find( { product_ids: 11 } );
```

All products for a given category (toys):
```
category = db.categories.find( { category_ids: 4 } );
db.products.find( { _id: { $in: category.product_ids } } );
```
Embedding

- Simple data structure
- Limited to 16MB
- Larger documents
- How often do you update?
- Will the document grow and grow?

Linking

- More complex data structure
- Unlimited data size
- More, smaller documents
- What are the maintenance needs?
Don't do:

```javascript
{  
  temperature: {  
    _id: 42,  
    points: [  
      { 1332942067: 17.3 },  
      { 1332942118: 17.5 }  
    ]  
  }  
}
```

instead, do:

```javascript
{  
  temperature: {  
    _id: 42,  
    points: [  
      { ts: 1332942067, temp: 17.3 },  
      { ts: 1332942118, temp: 17.5 }  
    ]  
  }  
}
```

Define and document your keys!
A bit more work when updating, but a lot easier to retrieve
Indexes

- Just like a relational database, MongoDB also benefits from indexes.
- Every collection has (automatically) an index on _id.
- Indexes can be on single or multiple fields.
- `MongoCursor->explain()`.

```php
<?php ini_set('xdebug.var_display_max_depth', 1);
$m = new Mongo;
$c = $m->demo->elephpants;
$c->drop();
$c->insert( array( '_id' => 'ele1', 'name' => 'Jumbo' ) );
$c->insert( array( '_id' => 'ele2', 'name' => 'Tantor' ) );
var_dump( $c->find( [ '_id' => 'ele1' ] )->explain() );
?>```
<?php ini_set('xdebug.var_display_max_depth', 1);
$m = new Mongo;
$c = $m->demo->elephpants;
$c->drop();
$c->insert([ '_id' => 'ele1', 'name' => 'Jumbo' ]);
$c->insert([ '_id' => 'ele2', 'name' => 'Tantor' ]);
$c->insert([ '_id' => 'ele3', 'name' => 'Stampy' ]);
var_dump( $c->find([ 'name' => 'Jumbo' ])->explain() );
?>
<?php ini_set('xdebug.var_display_max_depth', 1);
$m = new Mongo;
$c = $m->demo->elephpants;
$c->drop();
$c->ensureIndex( [ 'name' => 1 ] );
$c->insert( [ '_id' => 'ele1', 'name' => 'Jumbo' ] );
$c->insert( [ '_id' => 'ele2', 'name' => 'Tantor' ] );
$c->insert( [ '_id' => 'ele3', 'name' => 'Stampy' ] );
var_dump( $c->find( [ 'name' => 'Jumbo' ] )->explain() );
?>
More about indexes

- **Compound indexes:**
  
  \$myCol->ensureIndex( [ _id: 1, ts: -1 ] )

- **Searching with regexp:** \^:
  
  \$myCol->find( [ 'name' => new MongoRegex( '/^tan/i' ) ] )

- **2d index wants longitude, latitude (as in GeoJSON):**
  
  \$myCol->insert( [ _id: 42, loc: [ 6.43, 52.1233 ] ] );
  \$myCol->insert( [ _id: 42, loc: { long: 6.43, lat: 52.1233 } ] );
  \$myCol->insert( [ _id: 42, loc: { latitude: 6.43, longitude: 52.1233 } ] );
Geospatial Indexes

Helps you with finding POIs (pubs!) in a 2D space

```php
<?php
$m = new Mongo; $c = $m->demo->pubs; $c->drop();

$c->ensureIndex( array( 'l' => '2d' ) );
$c->insert([ 'name' => 'Betsy Smith', 'l' => [ -0.193, 51.537 ] ]);  
$c->insert([ 'name' => 'London Tavern', 'l' => [ -0.202, 51.545 ] ]);  

$closest = $m->demo->command( [ 'geoNear' => 'pubs', 'near' => [ -0.198, 51.538 ], 'spherical' => true, ] );

foreach ( $closest['results'] as $res ) {
    printf( "%s: %.2f km\n", $res['obj']['name'], $res['dis'] * 6378 );
}
?>
```
- Failover/Availability
- Scaling reads
- Primaries, secondaries and arbiters
- Odd number to prevent split brain
- Scaling writes and reads
- Config servers, router (mongos) and replica sets
Who uses MongoDB?

craigslist
CHECK24
sourceforge
Pinterest
HM Government
shutterfly
the guardian
The New York Times
Chicago Tribune
Grooveshark
edelight
Viber
Eventbrite
Wordz
sk songkick
Uber
SAP
the national archives
floxe
Trello
CustomInk

Resources

- Slides: http://derickrethans.nl/talks/:::talk_id:::
- Contact me: Derick Rethans: @derickr, derick@10gen.com
- Feedback: