MongoDB

scalable, high-performance,
open source NoSQL database

Document Store

Full Index Support

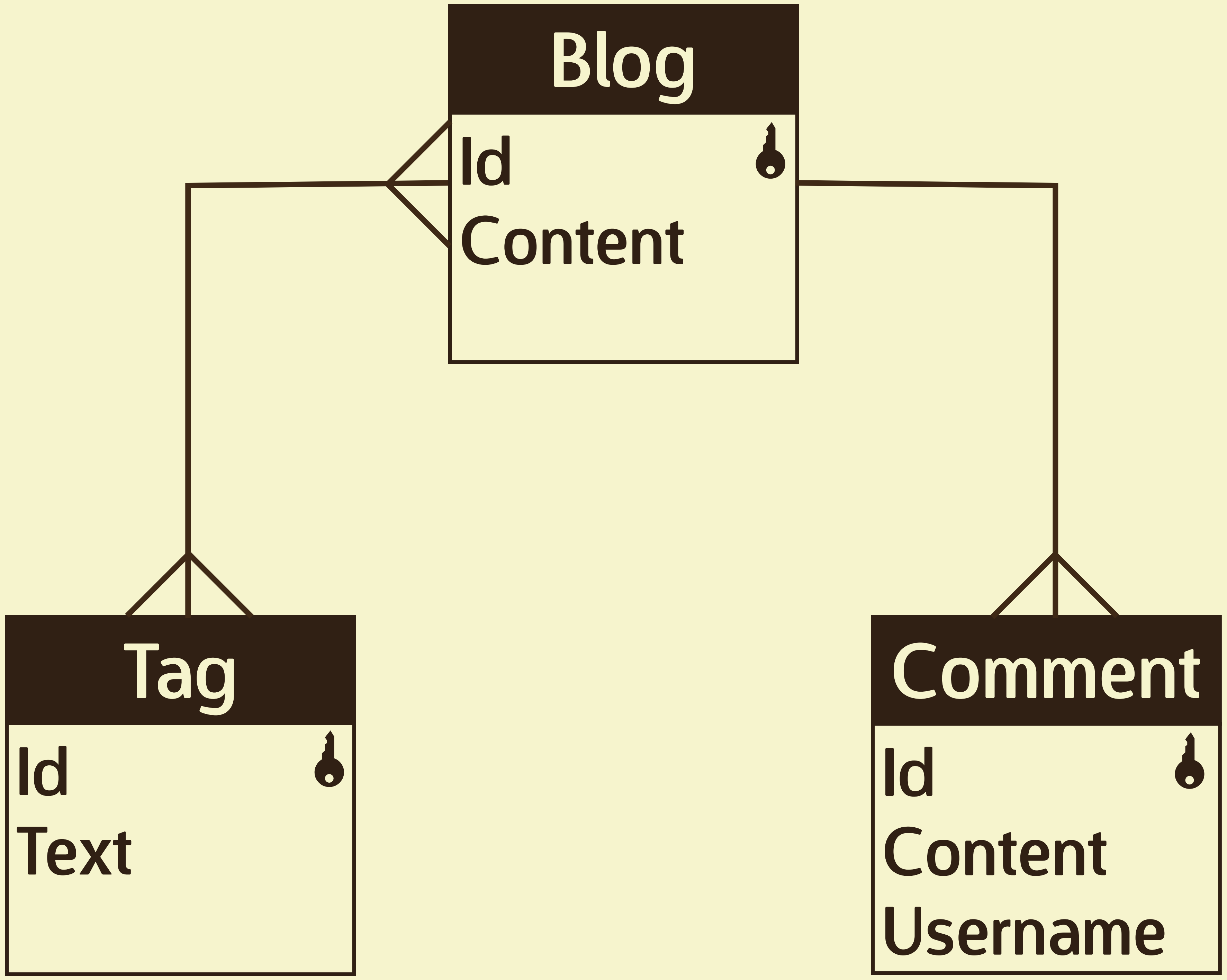
Replication

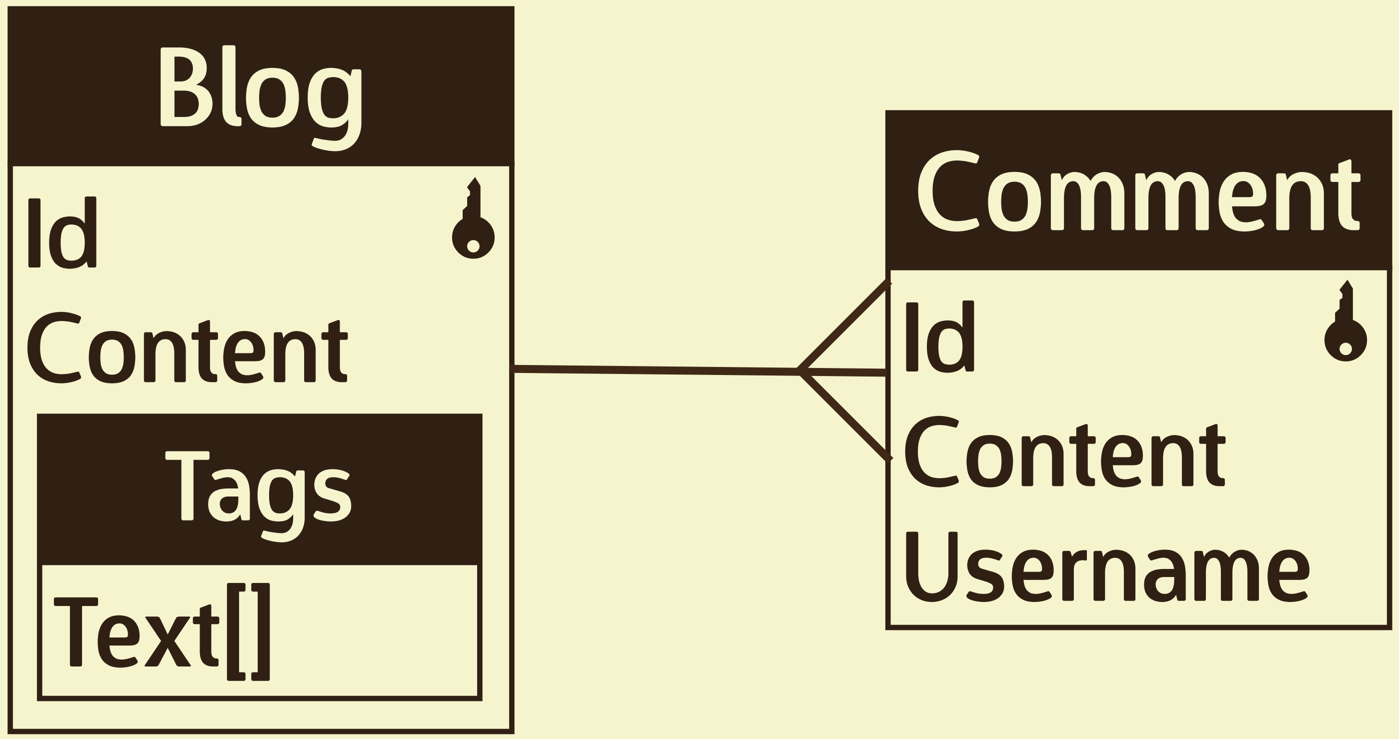
High Availability

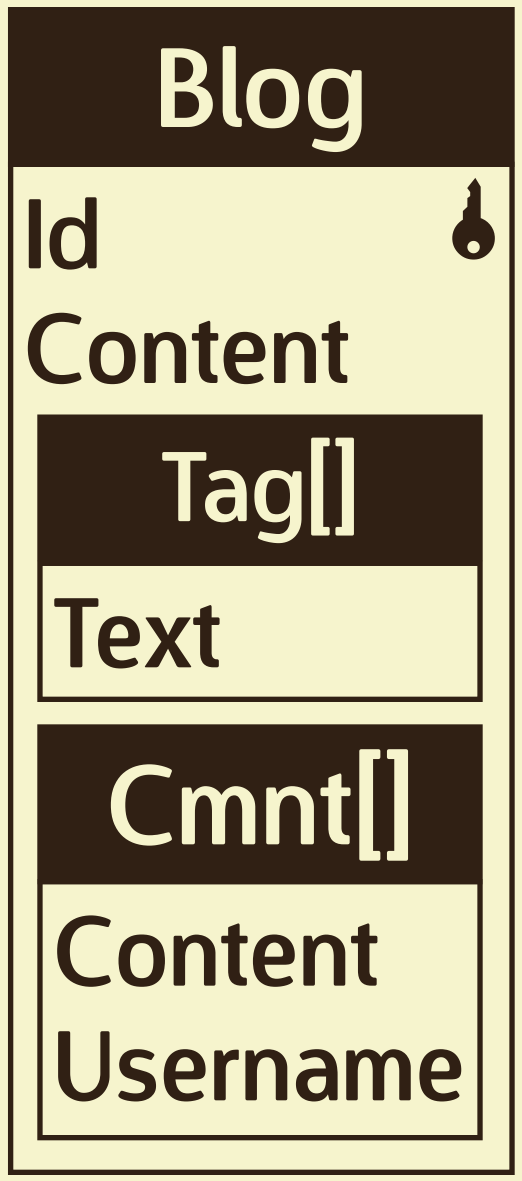
Querying

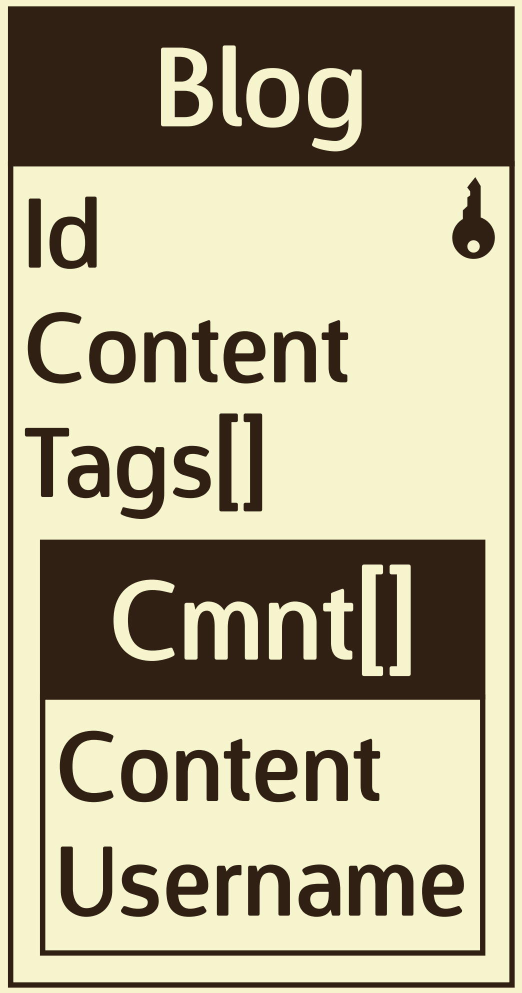
Auto-Sharding

Map/Reduce


Blog
Id
Content
Tag
Id
Text
Comment
Id
Content
Username


Blog
Id
Content
Tags
Text[]
Comment
Id
Content
Username


Blog
Id
Content
Tag[]
Text
Cmnt[]
Content
Username


Blog
Id
Content
Tags[]
Cmnt[]
Content
Username

```
{
  _id:      new ObjectId(),
  content:  ‘lorem ipsum...’,
  tags:     [‘technical’,‘mongo’],
  comments: [{
    _id:      new ObjectId(),
    content:  ‘Best post EVAR!’,
    username: ‘jameshu’
  },{
    _id:      new ObjectId(),
    content:  ‘Worst post EVAR!’,
    username: ‘anotherp’
  }]
}
```

SQL	Mongo
database	database
table	collection
row	document
column	field
index	index
primary key	_id

Working with Collections

Inserts, Updates and Deletes

```
> use awesomedb

> db.blogs.insert({content: “hello”, tags: [“test”]})
> db.blogs.insert({content: “hi”})
> db.blogs.insert({content: “goodbye”, tags: [“test”]})
```

```
-- insert blog
INSERT INTO blogs(id, content) VALUES(1, ‘hello’);

-- add tag references
INSERT INTO tags(id, text) VALUES(1, ‘test’);
INSERT INTO blogs_tags(blog_id, tag_id) VALUES(1,1);
```

```
> use awesomedb

> db.blogs.update({_id: new ObjectId("...")},
... {$set: {content: "changed content"}})
> db.blogs.update({tags: "rant"},
... {$set: {content: "REDACTED"}}, { multi: true })
```

```
-- update a single entity
UPDATE blogs SET content = 'changed content' WHERE id = 1

-- update multi (automatic)
UPDATE blogs SET content = 'REDACTED'
WHERE content LIKE '%s**t%'
```

```
> use awesomedb

> db.blogs.remove({_id: new ObjectId("...")})
> db.blogs.remove({content: /s**t/})
```

```
-- update a single entity
DELETE FROM blogs WHERE id = 1

-- delete multi (automatic)
DELETE FROM blogs WHERE content LIKE '%s**t%'
```

Querying

Finding data from collections

```
> use awesomedb

> db.blogs.find()
> db.blogs.findOne()
> db.blogs.find({}, {content: 1})
```

```
-- get all blog entries
SELECT * FROM blogs

-- get first blog entry
SELECT * FROM blogs LIMIT 1

-- get the contents column
SELECT content FROM blogs
```

```
> use awesomedb

> db.blogs.find({rating: 5})
> db.blogs.find({rating: 5}).sort(author: 1)
> db.blogs.find({rating: { $gt: 3}})
```

```
-- get all blog entries with a rating of 5
SELECT * FROM blogs WHERE rating = 5

-- get entries with 5 rating orderd by author
SELECT * FROM blogs WHERE rating = 5 ORDER BY author

-- get all blog entries with a rating of 3 or greater
SELECT * FROM blogs WHERE rating > 3
```

$gt
$gte
$lt
$lte
$ne
$in

$nin
$mod
$all
$size
$exists
$type

$not
$where
$elemMatch
$regex
$and
$or ....

Indexes

Creating Performant Queries

```
> use awesomedb

> db.blogs.ensureIndex({author: 1})
> db.blogs.ensureIndex({title: 1}, {unique: true})
> db.blogs.ensureIndex({slug: 1}, {
... unique: true, sparse: true})
```

Map/Reduce

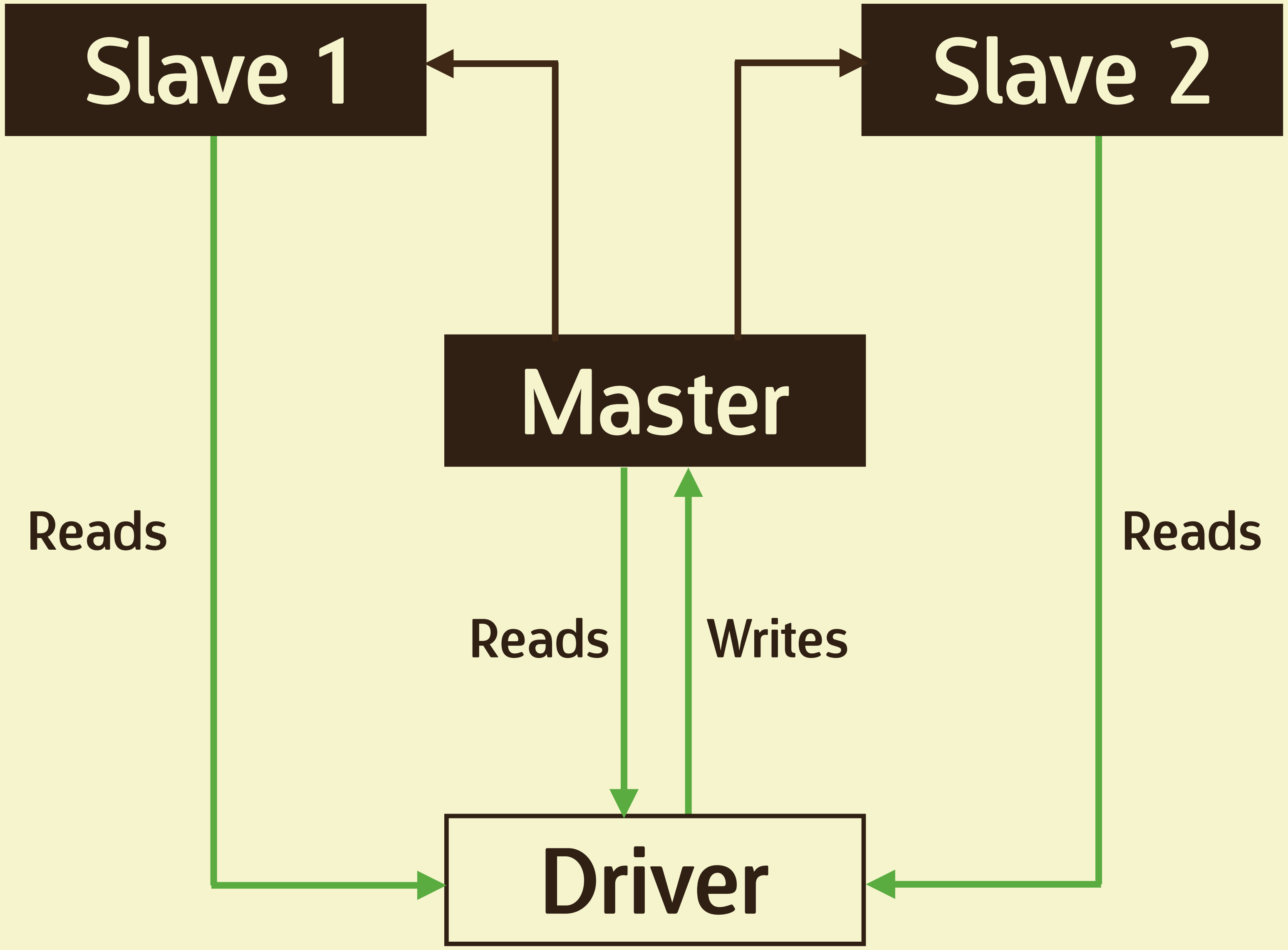
“BigData” Analysis

```
// MAP FUNCTION
var m = function(){
  if(this.tags){
    this.tags.forEach(function(t){
      emit(t, 1)
    });
  }
}

// REDUCE FUNCTION
var r = function(key, values){
  return values.length;
}

db.blogs.mapReduce(m, r, {out: {inline : 1}})
```

```
{
  "results" : [
    {"_id" : "mongo",     "value" : 1},
    {"_id" : "technical", "value" : 2}
  ],
  "timeMillis" : 0,
  "counts" : {
    "input"  : 3,
    "emit"   : 3,
    "reduce" : 1,
    "output" : 2
  },
  "ok" : 1,
}
```

Replica Sets

Failover, Recover and Scalability


Slave 1
Slave 2
Master
Reads
Reads
Writes
Reads
Driver

Write Concern

None
Normal
Safe
Journal Safe
FSync

MongoDB

scalable, high-performance, open source NoSQL database