MongoDBS
MongoDoc
mongoQueries

A database for the modern web
10gen began work on a platform-as-a-service (PaaS)

MongoDB is a database management system designed to rapidly develop web appli□cations and internet infrastructure. The data model and persistence strategies are built for high read-and-write throughput and the ability to scale easily with automatic failover.

MongoDB stores its information in documents rather than rows.

```
{
_id: 10,
username: 'peter',
email: 'pbbakkum@gmail.com'
}
```

if you have multiple email

```
{
_id: 10,
username: 'peter',
email: [
'pbbakkum@gmail.com',
'pbb7c@virginia.edu'
]
}
```

And just like that, you've created an array of email addresses and solved your problem.

MongoDB's document format is based on JSON, a popular scheme for storing arbi□trary data structures. JSON structures consist of keys and values, and they can nest arbitrarily deep. A document-based data model can represent rich, hierarchical data structures. It's often possible to do without the multitable joins common to relational databases.

```
{ //_id field, primary key
_id: ObjectID('4bd9e8e17cefd644108961bb'),

title: 'Adventures in Databases',
url: 'http://example.com/databases.txt',

author: 'msmith',
vote_count: 20,

//Tags stored as array of b strings
tags: ['databases', 'mongodb', 'indexing'],

Attribute pointing to another document
image: {
url: 'http://example.com/db.jpg',
caption: 'A database.',
type: 'jpg',
size: 75381,
data: 'Binary'
},
Comments stored as array of comment objects
comments: [
{
user: 'bjones',
text: 'Interesting article.'
},
{
```

user: 'sverch',
text: 'Color me skeptical!'
}
]
}

Internally, MongoDB stores documents in a format called Binary JSON, or BSON.
When you query MongoDB and get results back, these will be translated into an easy-to-read data structure.
Where relational databases have tables, MongoDB has collections.
The data in a collection is stored to disk, and most queries require you to specify which collection you'd like to target.
The technique of separating an object's data into multiple tables like this is known as normalization. A normalized data set, among other things, ensures that each unit of data is represented in one place only.But strict normalization isn't without its costs. Notably, some assembly is required. To display the post you just referenced, you'll need to perform a join between the post and comments tables.

SCHEMA-LESS MODEL ADVANTAGES

This can speed up initial applica□tion development when the schema is changing frequently.
a schema-less model allows you to represent data with truly variable properties. For example, imagine

Ad hoc queries

To say that a system supports ad hoc queries is to say that it isn't necessary to define in advance what sorts of queries the system will accept.
example involving posts and comments. Suppose you want to find all posts tagged with the term politics having more than 10 votes.
The special $gt key indicates the greater-than condition:

db.posts.find({'tags': 'politics', 'vote_count': {'$gt': 10}});

Indexes

A critical element of ad hoc queries is that they search for values that you don't know when you create the database.Thus, you need a way to efficiently search through your data. The solution to this is an index.
ex.many books have indexes matching keywords to page numbers.
Indexes in MongoDB are implemented as a B-tree data structure.
With MongoDB, you can create up to 64 indexes per collection.The kinds of indexes supported.
ascending, descending, unique, compound-key, hashed, text, and even geospatial indexes.

Replication

MongoDB provides database replication via a topology known as a replica set. Replica sets distribute data across two or more machines for redundancy and automate failover in the event of server and network outages. Additionally, replication is used to scale database reads.

Replica sets consist of many MongoDB servers, usually with each server on a separate physical machine; we'll call these nodes. At any given time, one node serves as the replica set primary node and one or more nodes serve as secondaries.

Scaling

The easiest way to scale most databases is to upgrade the hardware.
The tech□nique of augmenting a single node's hardware for scale is known as vertical scaling, or scaling up.
Vertical scaling has the advantages of being simple, reliable, and cost-effective up to a certain point, but eventually you reach a point where it's no longer feasible to move to a better machine. It then makes sense to consider scaling horizontally, or scaling out.

JavaScript shell

you can pick your database and then insert a simple document into the users collection like this:

use my_database

db.users.insert({name: "Kyle"})

To see the results
db.users.find().

count command:
db.users.count()

db.users.find({username: "jones"})

You can also specify multiple fields in the query predicate, which creates an implicit AND among the fields.

db.users.find({
... _id: ObjectId("552e458158cd52bcb257c324"),
... username: "smith"
... })

The three dots after the first line of the query are added by the MongoDB shell to indi□cate that the command takes more than one line.

You can also use MongoDB's $and operator explicitly. The previous query is identi□cal to.

db.users.find({ $and: [
... { _id: ObjectId("552e458158cd52bcb257c324") },
... { username: "smith" }
...] })

Selecting documents with an OR is similar:

db.users.find({ $or: [
... { username: "smith" },
... { username: "jones" }
...]})

Updating documents
All updates require at least two arguments. The first specifies which documents to update, and the second defines how the selected documents should be modified.
by default the update() method updates a sin□gle document.
There are two general types of updates, with different properties and use cases. One type of update involves applying modification operations to a document or documents, and the other type involves replacing the old document with a new one.

Suppose that user Smith decides to add her country of residence. You can record this with the following update:

db.users.update({username: "smith"}, {$set: {country: "Canada"}})

REPLACEMENT UPDATE

db.users.update({username: "smith"}, {country: "Canada"})

the document is replaced with one that only contains the country field, and the username field is removed because the first document is used only for match□ing and the second document is used for replacing the document that was previously matched.

Be sure to use the $set operator if you intend to add or set fields rather than to replace the entire document.

MORE ADVANCED UPDATES

db.users.update({"favorites.movies": "Casablanca"},
... {$addToSet: {"favorites.movies": "The Maltese Falcon"} },
... false,
... true)

third argument, false, controls whether an upsert is allowed. This tells the update operation whether it should insert a document if it doesn't already exist, which has different behavior depending on whether the update is an operator update or a replacement update.

fourth argument, true, indicates that this is a multi-update. By default, a Mon□goDB update operation will apply only to the first document matched by the query selector. If you want the operation to apply to all documents matched, you must be explicit about that. You want your update to apply to both smith and jones, so the multi-update is necessary.

Deleting data
If given no parameters, a remove operation will clear a collection of all its docu□ments.
db.foo.remove()

If you want to remove all users whose favorite city is Cheyenne, the expression is straightforward:

```
db.users.remove({"favorites.cities": "Cheyenne"})
```

Note that the remove() operation doesn't actually delete the collection; it merely removes documents from a collection.

If your intent is to delete the collection along with all of its indexes, use the drop() method:
```
db.users.drop()
```

Creating a large collection

```
for(i = 0; i < 20000; i++) {
db.numbers.save({num: i});
}
```

```
db.numbers.count()
db.numbers.find()
```

> it
The it command instructs the shell to return the next result set.
RANGE QUERIES

```
db.numbers.find( {num: {"$gt": 19995 }} )
```

You can also combine the two operators to specify upper and lower boundaries:

```
db.numbers.find( {num: {"$gt": 20, "$lt": 25 }} )
```

Explain
an invaluable tool for debugging or optimizing a query. EXPLAIN describes query paths and allows developers to diagnose slow operations by determining which indexes a query has used. Often a query can be executed in multiple ways, and sometimes this results in behavior you might not expect. EXPLAIN explains.

```
db.numbers.find({num: {"$gt": 19995}}).explain("executionStats")
```

The "execution Stats" keyword is new to MongoDB 3.0 and requests a different mode that gives more detailed output.

```
db.stats()
```
You can also run the stats() command on an individual collection:
```
db.numbers.stats()
```

Implementation

```
function ( obj , opts ){
if ( obj == null )
throw "can't save a null";
if ( typeof( obj ) == "number" || typeof( obj) == "string" )
throw "can't save a number or string"
if ( typeof( obj._id ) == "undefined" ){
obj._id = new ObjectId();
return this.insert( obj , opts );
}
else {
return this.update( { _id : obj._id } , obj , Object.merge({
upsert:true }, opts));
}
}
```

Command-line tools

mongodump and mongorestore—Standard utilities for backing up and restoring a database. mongodump saves the database's data in its native BSON format and thus is best used for backups only; this tool has the advantage of being usable for hot backups, which can easily be restored with mongorestore.

mongoexport and mongoimport—Export and import JSON, CSV, and TSV7 data; this is useful if you need your data in widely supported formats. mongoimport can also be good for initial imports of large data sets, although before importing, it's often desirable to adjust the data model to take best advantage of MongoDB. In such cases, it's easier to import the data through one of the drivers using a custom script.

mongosniff—A wire-sniffing tool for viewing operations sent to the database. It essentially translates the BSON going over the wire to human-readable shell statements.

mongostat—Similar to iostat, this utility constantly polls MongoDB and the system to provide helpful stats, including the number of operations per second (inserts, queries, updates, deletes, and so on), the amount of virtual memory allocated, and the number of connections to the server.

mongotop—Similar to top, this utility polls MongoDB and shows the amount of time it spends reading and writing data in each collection.

mongoperf—Helps you understand the disk operations happening in a running MongoDB instance.

mongooplog—Shows what's happening in the MongoDB oplog.

Bsondump—Converts BSON files into human-readable formats including JSON.

Aggregation

use myaggregate
Now, it will create a new database and if existing then switch to that database.
Next step is to create a mongo collection
db.createCollection("stocks")

db.stocks.insertMany([
{ name: "Infosys", qty: 100, price: 800 },
{ name: "TCS", qty: 100, price: 2000 },
{ name: "Wipro", qty: 2500, price: 300 }
])

db.stocks.find().pretty()

Mongodb Matching Documents
The first stage of a pipeline is matching, and that allows to filter out the documents so that we are only manipulating the documents that we care about. The matching expression looks and acts much like the MongoDB find function

db.stocks.aggregate([
{ $match: { "price": 2000 } }
])
The first stage of a pipeline is matching, and that allows to filter out the documents so that we are only manipulating the documents that we care about.
MongoDB Grouping Documents
Once we've filtered out the records we don't want, we can start grouping together the ones that we do into useful subsets. We can also use groups to perform the operations across the common field in all documents, such as calculating the sum of a set of transactions and counting documents.
db.stocks.aggregate([{$group : {_id : "$qty", same_qty : {$sum : 1}}}])

Add sorting with a $sort

The $sort takes a document that specifies the field(s) to sort by and the respective sort order. "sort order"Can have one of the following values:
1 to specify the ascending order.
-1 to specify descending order.
db.stocks.aggregate([
{ $group : {_id : "$qty", total : { $sum : 1 } } },
{ $sort : {total : -1} }
]);

Join Collections

MongoDB is not a relational database, but you can perform a left outer join by using the $lookup stage.
The $lookup stage lets you specify which collection you want to join with the current collection, and which fields that should match.

Orders
[
{ _id: 1, product_id: 154, status: 1 }
]
Products
[
{ _id: 154, name: 'Chocolate Heaven' },
{ _id: 155, name: 'Tasty Lemons' },
{ _id: 156, name: 'Vanilla Dreams' }
]
var MongoClient = require('mongodb').MongoClient;

```
var url = "mongodb://127.0.0.1:27017/";

MongoClient.connect(url, function(err, db) {
if (err) throw err;
var dbo = db.db("mydb");
dbo.collection('orders').aggregate([
{ $lookup:
{
from: 'products',
localField: 'product_id',
foreignField: '_id',
as: 'orderdetails'
}
}

]).toArray(function(err, res) {
if (err) throw err;
console.log(JSON.stringify(res));
db.close();
});
});
```

output

```
[
{ "_id": 1, "product_id": 154, "status": 1, "orderdetails": [
{ "_id": 154, "name": "Chocolate Heaven" } ]
} ]
```

Drop Collections

*db.COLLECTION_NAME.drop()*OrdersJoin