Document Stores

- originates to Damien Katz and Lotus Notes, CouchDB
- the responsibility for the schema is moved from the database towards the application:
  - loss of enforcement of normalization and referential integrity
  - gain of flexibility and schema modifications at run-time for the application
- data mostly stored as JSON
MongoDB

- Document Store
- try to close the gap between classic RDBMS and Key/Value stores
- supported by a number of successful internet companies (10gen, ...)
- good integration with programming languages: C++, C#, Java, JavaScript, PHP, Ruby, Perl, Python
JSON Example

```json
{  
  "id" : "1",  
  "name" : "Ipanema",  
  "description" : "Eine alkoholfreie Variante für den Caipirinha-Abend",  
  "ingredients" : {  
    "Limette" : { "amount" : 1, "preparation" : "Achteln" },  
    "Brauner Zucker" : { "amount" : 2, "unit" : "TL" },  
    ...  
  },  
  "preparation" : "Limetten und Zucker in einem Glas mörsern, mit crushed ice bedecken und den Flüssigkeiten auffüllen. Mit einem Strohhalm servieren",  
  "links" : {  
    "linktypes/publish" : "http://cocktails.example.com/publish/1",  
    "linktypes/edit" : "http://cocktails.example.com/cocktails/1",  
    "linktypes/delete" : "http://cocktails.example.com/cocktails/1"  
  }  
}
```
JSON in MongoDB

- each document needs a special ID field: _id
- the _id values has to be unique
- can be anything
- automatic default (if not given):
  - automatic 12-byte number:
    - 4 byte time stamp
    - 3 byte client machine id
    - 2 byte process id
    - 3 byte counter
Terms

- Database: a physical container for collections
- Collection:
  - a group of MongoDB documents / eq. to a table
  - exists in a single MongoDB database
  - does not enforce a schema
  - typically similar structured documents
- Document:
  - set of key/value pairs
  - value can also be a list or a nested document
Design

- Design the schema according to your requirements
- Combine objects into one document if you will use them together (joins are not supported)
- Do joins in your application, not in the database
- Optimize for frequent use cases
Commands

- use *database_name* to select a certain database (default database is test)
- check your database with `db`
- `show dbs` shows all available (non-empty) databases
- after database selection you directly refer to collections and document e.g.: `db.genomes.insert({"seq":"acgtaattggc"})` would insert a new document into the collection genomes
Collection Command

- db.dropDatabase(): obvious ;-) 

- db.createCollection(name, options):
  - used to create a collection, allowing options
  - name: name of the collection
  - Options: document to set options in the key:value style, like autoindexing, size limit, etc
  - option: is optional

- db.collection_name.drop(): obvious
**insert Command**

- `insert() / save()``
- `db.collection_name.insert(document[s])``
- multiple documents are given in list of documents, i.e. embraced with brackets ( [ ] )``
- if `_id` is not given, it is auto-generated by the database``
- if `_id` is given and already exits in the database, save() overwrites the existing document, otherwise it is like insert()```
find Command

- `db.collection_name.find({criteria}, {fields_to_show})`
- `find()`: has attribute `pretty()` to print out nicely formatted
- print out the whole collection if no filters are given
- `db.mycoll.find({},{"title":1,_id:0})`
<table>
<thead>
<tr>
<th>Operation</th>
<th>Syntax</th>
<th>Example</th>
<th>RDBMS Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equality</td>
<td><code>{&lt;key&gt;: &lt;value&gt;}</code></td>
<td><code>db.mycol.find({&quot;by&quot;: &quot;tutorials point&quot;}).pretty()</code></td>
<td><code>where by = 'tutorials point'</code></td>
</tr>
<tr>
<td>Less Than</td>
<td><code>{&lt;key&gt;: {$lt: &lt;value&gt;}}</code></td>
<td><code>db.mycol.find({&quot;likes&quot;: {$lt:50}}).pretty()</code></td>
<td><code>where likes &lt; 50</code></td>
</tr>
<tr>
<td>Less Than Equals</td>
<td><code>{&lt;key&gt;: {$lte: &lt;value&gt;}}</code></td>
<td><code>db.mycol.find({&quot;likes&quot;: {$lte:50}}).pretty()</code></td>
<td><code>where likes &lt;= 50</code></td>
</tr>
<tr>
<td>Greater Than</td>
<td><code>{&lt;key&gt;: {$gt: &lt;value&gt;}}</code></td>
<td><code>db.mycol.find({&quot;likes&quot;: {$gt:50}}).pretty()</code></td>
<td><code>where likes &gt; 50</code></td>
</tr>
<tr>
<td>Greater Than Equals</td>
<td><code>{&lt;key&gt;: {$gte: &lt;value&gt;}}</code></td>
<td><code>db.mycol.find({&quot;likes&quot;: {$gte:50}}).pretty()</code></td>
<td><code>where likes &gt;= 50</code></td>
</tr>
<tr>
<td>Not Equals</td>
<td><code>{&lt;key&gt;: {$ne: &lt;value&gt;}}</code></td>
<td><code>db.mycol.find({&quot;likes&quot;: {$ne:50}}).pretty()</code></td>
<td><code>where likes != 50</code></td>
</tr>
</tbody>
</table>
conjunction/ disjunction

```javascript
> db.mycol.find({
  
  $and: [
    
    {key1: value1}, {key2:value2}
  
  
  ]

});.pretty()

> db.mycol.find({
  
  $or: [
    
    {key1: value1}, {key2:value2}
  
  
  ]

});.pretty()
```
update Command

- `db.collection_name.update(SELECTION_CRITERIA, UPDATED_DATA)`
  
  ```
  db.mycol.update({'title':'MongoDB Overview'},
  {$set:{'title':'New MongoDB Tutorial'}})
  ```

- to update multiple documents set parameter `multi` to true (`{multi: true}`)

- in comparison to `save()` it only updates the specified fields
remove Command

- used to remove documents
- `db.collection_name.remove(DELETION_CRITERIA)`
Demo

- Check out the command line and python tutorial under:
  http://api.mongodb.com/python/current/tutorial.html

- get a toy mongodb server for free at:
  https://mlab.com