udako
euskal unibertsitatea

NoSQL datu-baseak. MongoDB.

Zuhaitz Beloki
Ixa taldea, EHUko Informatika Fakultatea
zuhaitz.beloki@ehu.es

2014ko uztailak 2 eta 3

Indizea

1. **NoSQL datu-baseak**
   a. Ezaugarriak
   b. Sistema ezagunenak
   c. Eredu erlazionalekiko desberdintasunak
2. **MongoDB**
   a. Sarrera
   b. Datu-eredua
   c. CRUD eragiketak
   d. Indizeak eta neurketak
   e. GridFS
   f. Python driverra
   g. Erreplikazioa
   h. Partizionatze horizontala

Materiala

- Ikastaro honetan zehar hainbat ariketa praktiko egingo ditugu. Horretarako, zenbait datu-base eta bestelako material behar izango ditugu. Hurrengo estekan aurki dezakezu materiala:

https://www.dropbox.com/sh/kup4dk4ctkz8onk/AABSK7jgYfWal4dJCHmHE9usa

amazon
DynamoDB

mongoDB

Google Bigtable

Not
Only
SQL

redis

CouchDB
relax

riak

Cassandra

HBASE

NoSQL datu-baseak. MongoDB.

1.1 NoSQL datu-baseak: Ezaugarriak

NoSQL datu-baseak

- NoSQL: Not Only SQL.
- Datu-base sistemen familia bat da, betidanik ezagutzen ditugun sistema erlazionaletatik bereizten direnak, batez ere datu-ereduaren aldetik.
- Familia berean sartzen diren arren, elkarren artean oso desberdinak diren sistema asko daude: MongoDB, Cassandra, CouchDB, HBase…
- NoSQL terminoa 1998an sortu zen, baina ez zen zabaldu 2009 urtera arte.

NoSQL sistemen ezaugarriak

- NoSQL sistema batzuk oso desberdinak dira elkarren artean, baina zenbait ezaugarri komun dituzte denek:
  - Ez dute eredu erlazionala jarraitzen.
  - Datuek ez dute aurrez definitutako eskemarik jarraitzen.
  - Ez dago JOIN eragiketarik.
  - Helburua ez da ACID ezaugarriak betetzea.
  - Eskalagarritasun horizontala bermatzen dute.

NoSQL sistemen ezaugarriak

- NoSQL sistemen izaerari esker, datu-ereduen diseinuen sinpletasuna aipatu behar da.
- Sinpletasunak eraginkortasuna dakar. Gaur egun hain hedatuak dauden *big data* edo denbora errealeko sistemetan gero eta erabiliagoak urtetik urtera.
- CAP teorema: Datu-base sistema batek ezin ditu kontsistentzia, erabilgarritasuna eta erroreekiko tolerantzia, hirurak, bermatu.
    - NoSQL sistema gehienek, erabilgarritasunaren eta erroreekiko tolerantziaren mesedetan, kontsistentzia sakrifikatzen dute.

Eskalagarritasuna

- Ezaugarri garrantzitsuenetako bat eskalagarritasuna da. Gaur egun gero eta aplikazio gehiagok datu-kopuru erraldoiak maneiatzen dituzte (***big data***). Kopuru horiei aurre egiteko ezinbestekoa da datuak banatuta edukitzea, hau da, datu-baseak makina fisiko desberdinetan banatzea. Horrela, datu-kopurua handitu ahala, sistemari makina gehiago gehituko zaizkio eraginkortasuna linealki igoz (eskalagarritasun horizontala).

NoSQL datu-baseak. MongoDB.

1.2. NoSQL datu-baseak: Sistema ezagunenak

NoSQL sistemen sailkapena

- Ez da erraza NoSQL familiako sistemen sailkapen zuzen bat egitea, baina hau da onarpen handiena duena:
  - **Zutabeetan** oinarritutakoak: Cassandra, HBase…
  - **Dokumentuetan** oinarritutakoak: **MongoDB**, Clusterpoint, CouchDB…
  - **Gako-balio** pareetan oinarritutakoak: Redis, Dynamo, FoundationDB…
  - **Grafoetan** oinarritutakoak: Allegro, OrientDB, Virtuoso...

Sistema ezagunenak

1.2. NoSQL datu-baseak: Sistema ezagunenak

- NoSQL datu-base erabilienak, *DB-Engines*-en arabera, hauek dira gaur egun:
  1. MongoDB (dokumentuetan oinarritua)
  2. Cassandra (zutabeetan oinarritua)
  3. Redis (gako-balioetan oinarritua)
  4. HBase (zutabeetan oinarritua)
  5. Memcached (gako-balioetan oinarritua)
  6. CouchDB (dokumentuetan oinarritua)
- Horietatik, hauek aztertuko ditugu jarraian, labur:
  - CouchDB (dokumentuetan oinarritua)
  - Cassandra (zutabeetan oinarritua)
  - Redis (gako-balioetan oinarritua)
- Azkenik, hurrengo atalean, MongoDB sakonean aztertuko dugu.

221 systems in ranking, June 2014

Rank	Last Month	DBMS	Database Model	Score	Changes
1.	1.	Oracle	Relational DBMS	1500.92	-1.82
2.	2.	MySQL	Relational DBMS	1309.55	+0.46
3.	3.	Microsoft SQL Server	Relational DBMS	1223.79	+15.99
4.	4.	PostgreSQL	Relational DBMS	239.98	-0.66
5.	5.	MongoDB	Document store	231.44	+6.82
6.	6.	DB2	Relational DBMS	198.03	+11.56
7.	7.	Microsoft Access	Relational DBMS	142.36	-3.00
8.	8.	SQLite	Relational DBMS	89.18	-0.11
9.	9.	Cassandra	Wide column store	81.85	+0.12
10.	10.	Sybase ASE	Relational DBMS	80.70	+0.69
11.	11.	Solr	Search engine	67.88	+0.72
12.	↑ 13.	Redis	Key-value store	65.13	+3.09
13.	↓ 12.	Teradata	Relational DBMS	65.07	-0.46
14.	14.	FileMaker	Relational DBMS	53.78	-1.81
15.	15.	HBase	Wide column store	40.80	+0.54

CouchDB

- CouchDB terminoa ingelesezko *Cluster of Unreliable Commodity Hardware* akronimoetatik dator.
- 2005ean sortua.
- Dokumentuetan oinarritutako datu-baseak kudeatzeko sistema (DBKS) da. JSON dokumentuak gordetzen ditu.
- Apache lizentzia dauka, kode irekikoa.
- Gaur egungo webari erabat egokitutako sistema dela esaten da.

CouchDB

- Ezaugarri tekniko batzuk:
  - Kontsultak JavaScript lengoaiaz egiten dira.
  - MapReduce sistema erabiltzen da kontsultak egiteko.
  - Aplikazioen eta datu-basearen arteko komunikazioa RESTful interfazea erabiliz egiten da (HTTP hutsa).
  - Multi-master eredua erreplikaziorako: Datuak nodo batean baino gehiagotan kopiatuta eta sinkronizatuta mantendu daitezke, nodo guztiak nagusiak izanik (bai irakurtzeko eta bai idazteko eskuragarri).

CouchDB

- Helburu nagusietako bat eskalagarritasun horizontala da. Datu-base banatuak. Oso eskalagarria.
- Aurredefinitutako eskemarik ez datu-baseetan.
- JOIN eragiketarik ez.
- Datu-baseak kudeatzeko web-aplikazio bat dakar (phpMyAdmin-en antzeko zerbait).

CouchDB

Datu-eredua: CouchDB datu-base bat JSON dokumentuen bilduma bat da. Dokumentuak elkarrengandik independenteak dira, eta ez dute egitura finkorik. Elkarren artean oso desberdinak izan daitezke, nahiz eta normalean antzekoak izaten diren logikaz.

```
taldea = {
        'izena': 'Osasuna',
    'urtea': 1920,
    'kontaktua': [
        'www.osasuna.es',
        '@CAOsasuna',
        'www.facebook.com/C.A.Osasuna.oficial'
        ]
    'futbol-zelaia': {
        'izena': 'El Sadar',
        'sarrera': 19800,
        }
    'jokalariak': [
            {
            'izena': 'Asier Riesgo',
            'posizioa': 'atezaina'
            },
            …
        ]
    }
```

CouchDB

- Kontsultak egiteko modu nagusia: **bistak**.
- Bistak JavaScript lengoaiaz definitzen dira besterik ezean (badaude lengoaia gehiagotarako APIak). Bistek egiturarik gabeko dokumentuetatik egitura duten emaitzak ateratzen dituzte. Kontsulta bat inplementatzen du bista bakoitzak.

```
function(dok) {
  if (dok.Mota == "bezeroa") {
    emit(dok._id, {Abizena: doc.Abizena, Izena: doc.Izena, Helbidea: doc.Helbidea});
  }
}
```

Bista honek dokumentu guztietatik bezero motakoak bakarrik itzultzen ditu, bakoitza bere abizen, izen eta helbidearekin.

Cassandra

- Facebookek 2008an sortua.
- Gaur egun Apache Software Foundation barruan dagoen proiektua da.
- Zutabeetan oinarritutako DBKSa izan arren, badu gako-balioetan oinarritutakoen ezaugarririk.

Cassandra

- Cassandra ere eskalagarritasun horizontala lortzea helburu zutela garatu zuten. Datu-baseak hainbat makinatan banatzea helburu, eskalagarritasun eta abiadura handiak lortuz.
  - 2012ko Torontoko unibertsitatean eginiko ikerketa batek honela dio: "*In terms of scalability, there is a clear winner throughout our experiments. Cassandra achieves the highest throughput for the maximum number of nodes in all experiments.*"

Cassandra

- Ezaugarriak:
  - **Deszentralizatua**: Datuak hainbat makinatan banatuta daude, baina ez dago nodo nagusirik, edozein nodo baita gai edozein eskaerari aurre egiteko.
  - **Erreplikazioa**: Datuak hainbat zerbitzaritan kopiatuta mantendu daitezke, sinkronizatuta.
  - **Eskalagarritasuna**: Makina gehiago gehitu ahala, eraginkortasuna linealki hazten da.
  - **Erroreekiko tolerantzia**: Datuak automatikoki erreplikatzen direnez, nodo bat jokoz kanpo gelditu arren, behar diren datuak dauzkan beste batek erantzungo du.
  - **MapReduce**: Kontsultak eta eragiketak egiteko MapReduce eredua erabilgarri.
  - **Kontsultak**: CQL lengoaia (Cassandra Query Language), SQLren antzekoa.

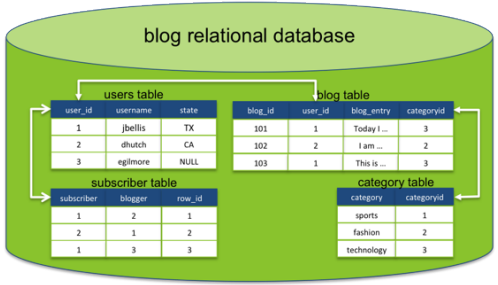
Cassandra

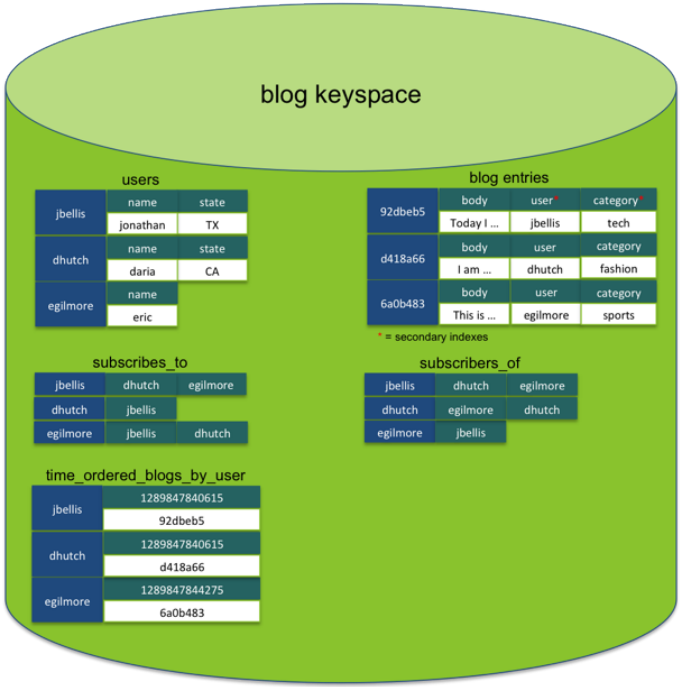
1.2. NoSQL datu-baseak: Sistema ezagunenak

Datu-eredua

Cassandrako zutabe familia bat DB erlazionaletako taula baten antzekoa da. Zutabe familia batek hainbat errenkada dauzka, bakoitza gako batez identifikatua. Errenkada bakoitzak gako-balioak ditu, horietako bakoitza zutabe bat bezala ikusten delarik. DBKS erlazionaletan ez bezala, zutabe familia bereko errenkadek zutabe kopuru desberdina izan dezakete.

Kontsulta mota bakoitzarentzat zutabe familia bat sortzea ohikoa da.





Redis

- Redis: REmote DIctionary Server.
- 2009an VMwarek garatua. DB-Engines-en arabera, munduan erabiliena den gako-balio motako datu-basea.

Redis

- Ezaugarriak:
  - **Datuen iraunkortasuna**: Redis datu-baseak memorian daude uneoro, eraginkortasunaren mesedetan. Kontsistentzia mantentzearren, idazketak egiten dira diskoan (2 segundoro, besterik ezean). Diskoan ez da aldiro informazio guztia kopiatzen. Aldiz, log fitxategi batean aldaketak idazten dira, eta, gero, log horretatik abiatuta datu-basea dagokion egoerara joango da.
  - **Erreplikazioa**: *Master/slave* eredua erabiltzen da erreplikazioa egiteko. Aldi berean, *slave* nodo bat beste *slave* baten *master*-a izan daiteke. Aplikazio bat edozein nodotara harpidetu daiteke, datuen fluxu osoa jasotzeko. Inkontsistentziak egon daitezke *master* eta *slave* nodoen artean.
  - **Eraginkortasuna**: Datuak memorian mantentzeak eraginkortasun aparta ematen dio sistemari, baina mugatua nolabait.
  - **Eskalagarritasuna**: Datuak partizionatzearen (datuak hainbat zerbitzaritan banatzea) espezifikazioa idatzita dagoen arren, oraindik ez dago inplementatuta.

Redis

Datu-eredua

Redisen datu-eredua, gako-balioetan oinarritutako datu-base sistema izanik, oso sinplea da. Dena gako-balioz osatuta dago. Hash taula erraldoi bat bezala ikus daiteke Redis datu-base bat, diferentzia bakarra iraunkortasuna izanik, hau da, datuak memorian egon arren, diskoan ere gordetzen direla. Gakoak beti string motakoak diren arren, balioak, string-ez gain, zerrendak, multzoak, ordenatutako multzoak etab. izan daitezke. Datu-mota bakoitzak bere eragiketa propioak eskaintzen ditu, atomikoak, erabiltzaile bat baino gehiagok aldi berean lan egin ahal izan dezaten datu berberen gainean.

Redis, askotan, funtzionalitate sinpleak inplementatzeko erabiltzen da: denbora-errealeko ranking sistema bat, ilarak, estatistika jakin batzuk lortzeko eta abar. Ohikoa da domeinuaren datu-biltegi nagusi gisa beste DBKS bat erabiltzea.

Eragiketak oso sinpleak dira:

```
> SET ald 10          // "foo" gakoari "10" balioa esleitu
> INCR ald            // "foo" gakoaren balioa atomikoki inkrementatu eta "11" izatera pasa
> GET ald             // Lortu "foo" gakoaren balioa: "11"
> LPUSH zerr a        // Gehitu "zerr" gakoa duen zerrendan "a" balioa ezker aldetik (atomikoki)
```

NoSQL datu-baseak. MongoDB.

1.3 NoSQL datu-baseak: Eredu erlazionalarekiko desberdintasunak

NoSQL *VS* DBKS erlazionalak

- Datu-base erlazionalek gure artean hainbeste urte daramatzatela jakinda, zergatik orain sistema berriekin hasi? Urte hauetan guztietan ez al gara, ba, ongi moldatu DB erlazionalekin?
- Jarraian, bi ereduen arteko desberdintasun nagusiak ikusiko ditugu.

NoSQL *VS* DBKS erlazionalak

Datuen izaera

Zenbaitetan, erabiliko diren datuen izaera erlazionala da. Entitate desberdin asko, entitateen artean lotura asko, entitate bakoitza atributu edo eremu finko eta zehatz batzuen bidez definitua… Betiko datu-base erlazionalak egokitzen dira hobekien kasu horietarako.

Beste batzuetan, aldiz, datuak heterogeneoagoak dira, ez dago hain garbi zein entitate dauzkagun, entitate bakoitzaren egitura aldatu egin daiteke objektu batetik bestera… Adibidez, hierarkizatutako egiturak, grafoak, XML dokumentuak etab. Demagun, esate baterako, egunkarietako artikuluak gorde nahi ditugula, izenburuen eta atalen egitura mantenduz. MongoDB bezalako datu-base batean, bere horretan gordeko genituzke artikuluak, nahiz eta artikulu batzuk elkarren artean oso desberdinak izan.

NoSQL *VS* DBKS erlazionalak

Eskalagarritasun horizontala

Aurrez aipatu dugun bezala, NoSQL datu-baseek horizontalki eskalatzen dute orokorrean. Horrek esan nahi du datuak hainbat zerbitzaritan banatu ditzakegula, hau da, datu-basea zatitan banatu dezakegula, eta makina bakoitzean zati bat gorde. Horrela, datuak asko hazten badira ere, programak, algoritmoak eta datu-baseak aldatu gabe, nahikoa izango da makina berriak erostearekin. Erreplikazioa ere eskaintzen dute, orokorrean, datuen iraunkortasuna eta eskuragarritasuna bermatzeko.

Datu-base erlazional gehien-gehienetan eskalatzeko modu bakarra makina hobea erostean datza, eta irtenbide hori oso mugatua da gaur egun, ez baitago milioika eta milioika datu modu eraginkorrean gorde eta tratatuko dituen makinarik.

NoSQL *VS* DBKS erlazionalak

Garapenerako malgutasuna

Zaila den arren NoSQL familiari ezaugarriak esleitzea, oso sistema desberdinak sartzen baitira bertan, orokorrean erabiltzeko oso sinpleak direla esan daiteke. MongoDB-ren kasuan, dokumentuak JSON formatuan gordetzen direnez, Python bezalako lengoaia batekin kontsulta egikaritzen den unean datu-baseko datuak gure programazio-lengoaiako objektuetan dauzkagu, erabiltzeko prest.

Datu-base erlazionaletan ere ezin esan lan hau nekeza denik, baina datu-basearen eredutik programazio-lengoaiaren objektuetara pasatzeak lan gehiago eskatzen du. Horregatik, eta datu-ereduaren malgutasunagatik, MongoDB-rekin prototipoak oso erraz sortzen dira.

NoSQL *VS* DBKS erlazionalak

Datuen aldakortasuna

Erabiltzen ohituta gauden sistema erlazionaletan, oso ongi pentsatu behar da datuen egitura zein izango den garapena hasi aurretik. Behin eskemak definituta, taula batean eremu bat gehitu edo kendu behar izateak buruhausteak ekar ditzake. Zer esanik ez taulen arteko erlazioa aldatu nahi bada, gako arrotzak ere aurrez definitzen baitira.

NoSQL datu-baseetan aurrez ez da eskemarik definitzen. Komenigarria da datuen izaera nolabait deskribatzea dokumentazio gisa, ideiak argitze aldera, baina DBKSak ez du datuen egitura definitzeko aukerarik ematen.

Ondorioak

- Hala ere, dena ez da dirudiena bezain polita.
- Datuen izaera erlazionala bada, datu-base erlazionalak erabiltzea egokiagoa da. Gainera, abiadura eta malgutasunaren mesedetan, NoSQL datu-baseek kontsistentzia gal dezakete (transakziorik ez, adibidez). Beraz: **Datu-kopuru handia daukagunean eta abiadura eta malgutasuna garrantzitsuagoa denean datuen kontsistentzia baino, NoSQL datu-baseak erabiltzea gomendatzen da. Bestela, ez.**

mongoDB

NoSQL datu-baseak MongoDB

2.1 MongoDB: Sarrera

Sarrera

- NoSQL familiako datu-baseak kudeatzeko sistema.
- Dokumentuetan oinarritua.
  - Datu-basean BSON dokumentuak gordetzen dira.
  - BSON JSONen oinarritutako formatua da, MongoDB-ren garatzaileek garatua.
- Ezaugarri nagusiak:
  - Oso eraginkorra.
  - Datu-eredu sinplea.
  - Oso eskalagarria.

Puri-purian dagoen sistema

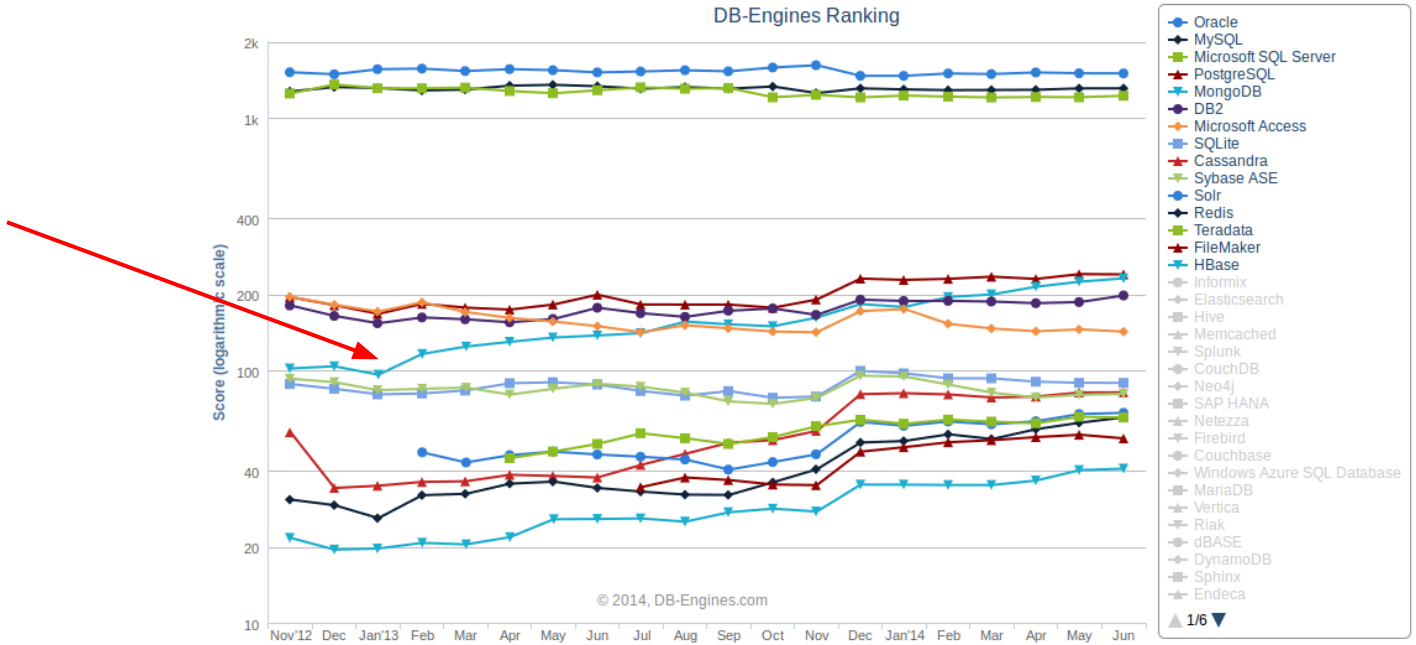
- NoSQL datu-base sistema erabiliena (*db-engines.com*).

221 systems in ranking, June 2014

Rank	Last Month	DBMS	Database Model	Score	Changes
1.	1.	Oracle	Relational DBMS	1500.92	-1.82
2.	2.	MySQL	Relational DBMS	1309.55	+0.46
3.	3.	Microsoft SQL Server	Relational DBMS	1223.79	+15.99
4.	4.	PostgreSQL	Relational DBMS	239.98	-0.66
5.	5.	MongoDB	Document store	231.44	+6.82
6.	6.	DB2	Relational DBMS	198.03	+11.56
7.	7.	Microsoft Access	Relational DBMS	142.36	-3.00
8.	8.	SQLite	Relational DBMS	89.18	-0.11
9.	9.	Cassandra	Wide column store	81.85	+0.12
10.	10.	Sybase ASE	Relational DBMS	80.70	+0.69

db-engines.com-etik hartua

- Denbora gutxian asko hazi den sistema.



db-engines.com-etik hartua

- Datu-base erlazionalak *vs* NoSQL datu-baseak:
  - Erlazionalak:
    - Oso egituratuak: egokia erabiliko diren datuen egitura bera ere konplexua denean.
    - Aspalditik oso erabiliak eta hedatuak => **sendoak**
    - Orokorrean, ez eskalagarriak.
    - Transakzioak onartzen dituzte => zenbait domeinutan **fidagarriagoak**
  - NoSQL:
    - Datu-eredu oinarrizkoa eta sinplea => **malgutasuna** eta **eraginkortasuna**
    - Sistema *gazteak* oraindik.
    - Oso **eskalagarriak.**
    - Ez dituzte transakzioak onartzen => eragiketa sinpleak baina **oso azkarrak.**

JSON eta BSON formatuak

- JSON: *Java Script Oriented Notation*.
  - Datuak makinen artean bidaltzeko formatu ona izateaz gain, datuak biltegiratzeko ere egokia.
  - XML lengoaiaren antzeko funtzioa, baina trinkoagoa.
  - Eskemarik gabea: egitura zurrunik ez.
- BSON: *Binary JSON.*
  - JSON formatuaren oso antzekoa, baina MongoDB eraginkorrago bilakatzen du.

Transakziorik ez

- MongoDB datu-baseek ez dituzte transakzioak onartzen.
- Transakzioen kudeaketa astuna izan daitekeenez, halakorik gabe askoz ere azkarrago egiten dira eragiketa gehien-gehienak.
- Eskalagarritasun horizontala bermatzen du, bertikala baino hobea gaur egun, emaitza hobeak lortzen baitira makina merke askorekin, garestia den makina bakarrarekin baino.
- Kontuz, eragiketa delikatuak egiteko (bankuko transakzioak, esaterako) agian MongoDB bezalako sistema bat hautatzea ez da gomendagarriena!

Kontsultak

- Kontsulta sistema landua dauka MongoDB-k, erabilerraza, ahaltsua eta azkarra.
- Adibidez:
  - `db.diskoak.find({"autorea": "Ruper Ordorika"})`
  - `db.autoreak.find({"diskoak": {$size: 2}})`
  - `db.autoreak.find().sort({"izena": 1})`
- Kontsulta arruntez gain, *MapReduce* motako kontsultak ere egin daitezke.

Indizeak

- Indizeen kudeaketa zabala.
- Dokumentu kopuru erraldoiekin lan egiteko sistema.
- Indize arruntak eta konposatuak.
- Balio bakarreko eremuen indizeak (*uniqueness*).
- Azpidokumentuak ere indexatu daitezke.

Erreplikazioa eta partizionatze horizontala

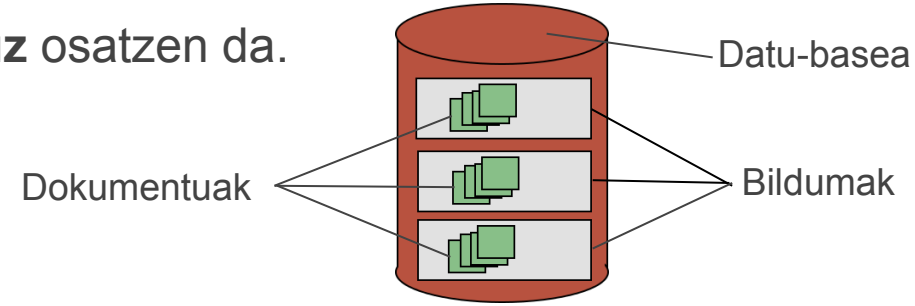
- MongoDB-ren funtsezko bi ezaugarri eskalagarritasunari begira:
- Erreplikazioa:
  - Datuen eskuragarritasuna bermatzeko modua.
  - **Datuen** hainbat **kopia** mantendu zerbitzari desberdinetan.
  - Informazio gehiago 5.7 atalean.
- Partizionatze horizontala (*sharding*):
  - **Datuak** hainbat zerbitzaritan **banatu** ditzakegu, erabiltzailearentzat (gizakia izan edo aplikazioa izan) gardena den modu batean.
  - Informazio gehiago 5.8 atalean.

NoSQL datu-baseak. MongoDB.

2.2 MongoDB: Datu-eredua

Datu-baseen egitura

- MongoDB dokumentuetan oinarritutako datu-baseak kudeatzeko sistema da.
- Ez dago ohituta gauden **taula** eta **tuplen** kontzepturik.
- Datu-baseak **bildumatan** (*collection*) banatzen dira, eta bilduma bakoitza **dokumentuz** osatzen da.



Datu-baseen egitura: Adibidea

- Adibidea: Pilota-txapelketen inguruko datuak gorde nahi ditugu datu-base batean:

BILDUMA	DOKUMENTUEN EGITURA
Pilotariak	Izena, abizenak, jaiotze-data, posizioa...
Txapelketak	Izena, urtea, sariak...
Partidak	Txapelketa (*ref*), frontoia (*ref*), pilotariak (*ref*), emaitza...
Frontoiak	Izena, herria, ikusle kopurua...

Datu-baseen egitura: Adibidea II

- Adibidea: Online denda baten eskaeren kudeaketa egin nahi da, informazioa biltegiratzeko MongoDB erabiliz:

BILDUMA	DOKUMENTUEN EGITURA
Erabiltzaileak	Izena, abizenak, e-mail...
Produktuak	Izena, mota...
Eskaerak	Erabiltzailea (*ref*), produktua (*ref*), data

Datu-baseen egitura

- MongoDB sistemako bildumak eta dokumentuak SQL sistemetako taula eta tuplen parekotzat har daitezke nolabait, baina ez dira inolaz ere baliokideak:
  - Tupla guztiek aurrez definitutako egitura izan behar dute derrigorrez, eremu jakin batzuk edukitzera behartuta daude. MongoDB-ko dokumentuen kasuan, bilduma bereko bi dokumentu oso desberdinak izan daitezke egiturari dagokionez.
  - Tupla bateko balioak zenbaki edo *string* bezalako oinarrizko motetara mugatuta daude. MongoDB-n, balio bat beste azpidokumentu bat edo lista oso bat izan daiteke.

Datu-baseen egitura: Adibidea III

- Adibidea: Pilota-txapelketen inguruko datuak gorde nahi ditugu datu-base batean, baina oraingoan bilduma bakarrean sartuko dugu dena:

BILDUMA	DOKUMENTUEN EGITURA

Txapelketak	- izena
	- urtea
	- partiduak: [{
	- pilotariak: [{
	- izena…
	}]
	}]
	- frontoia: { izena, herria… }

Dokumentuak

- Dokumentua da MongoDB datu-baseetako oinarrizko biltegiratze-unitatea:
  - Mundu errealeko objektu bakoitza, normalean, dokumentu batean definitzen da.
  - Dokumentu batek hainbat eremu izan ditzake.
    - Eremu bakoitza edozein motatakoa izan daiteke: zenbakiak, stringak, azpidokumentu oso bat, zerrendak…
- Dokumentuak BSON formatuan biltegiratzen dira, JSONen egokitzapena.
- Datu-motak: *string, integer, boolean, double, array, timestamp, object (azpidokumentuak), null, date, objectID, binary data, regular expression, javascript code...*

- Adibidea. Musika-disko baten informazioa:

```
{
    izena: “Kontzientzia ala infernua”,
    taldea: “Latzen”,
    urtea: 1996,
    abestiak: [ “Kontzientzia ala infernua”, “Bat gehiago”, “Ze ingo xu?”, … ],
    diskoetxea: { izena: “Oihuka”, e-mail: “oihuka@oihuka.com” },
}
```

Dokumentuen egitura librea

- Datu-baseek ez dute aurrez definitutako eskema zurrun bat jarraitzen, SQL datu-baseetan ez bezala.
  - Izaera bereko bi dokumentuk eremu desberdinak izan ditzakete.
  - Malgutasun handia.

Egitura librea: Adibidea

- Multimedia bilduma bat egin nahi dugu, gure musika eta film guztiak katalogatzeko. SQL sistemetan bi aukera: bi taula desberdin (musika, filmak) edo taula bakarrean eremu guztiak (*null* balio asko).
- MongoDB datu-base batean hurrengo bi dokumentuak gorde ditzakegu bilduma berean:
  - `{ izena: “Crush”, taldea: “Bon Jovi”, abestiak: [“It’s My Life”, “Say it isn’t so”, …] }`
  - `{ izena: “Gattaca”, aktoreak: [“Ethan Hawke”, Uma Thurman”, …], zuzendaria: “Andrew Niccol” }`

Indizeak

- Bilduma mailako indizeak definitu daitezke, dokumentuen edozein eremuren gainean.
  - Azpidokumentuen eremuen gaineko indizeak ere definitu daitezke.
- *2.4* atalean sakonago aztertuko ditugu indizeak.

_id eremua

- Dokumentu guztiek besterik ezean daukaten eremua, dokumentuaren identifikadore unibokoa izango dena.
- Edozein balio esleitu diezaiokegu, bilduma bereko beste dokumentu batek esleituta eduki ezean.
- Baliorik esleitzen ez bazaio, lehenetsitako balio uniboko bat izango du.

  Adibidez: `{ _id: ObjectId("539dcaebd884cd82cc2b5996"), … }`

Dokumentuen hazkuntza

- Dokumentu bat sortzean, diskoaren zati bat gordetzen du beretzat. Zati hau, normalean, dokumentuak une horretan behar duen byte kopurua baino altuagoa izaten da (*padding*). Horrela, etorkizunean dokumentua hazten bada, nahikoa toki izango du.
- Dokumentua gehiegi hazten bada, diskoaren beste espazio bat esleitu behar zaio eta informazio guztia diskoan birkokatu (*reallocation*). Beraz, kontuz honekin!

Azpidokumentuak *vs* erreferentziak

- SQL sistemetan gako arrotzen bidez **beste tauletako objektuak erreferentziatzen** ditugun moduan, MongoDB-n ere gauza berbera egiteko aukera daukagu.
- Alabaina, erreferentziatuko genukeen **objektua dokumentu horretan bertan txertatu** dezakegu azpidokumentu gisa.
- Askotan ez da erraza erabakitzen dokumentu batek bestea barruan txertatuta izan behar lukeen edo erreferentzia egin behar liokeen.

Azpidokumentuak *vs* erreferentziak

- Dokumentuak txertatzearen alde onak eta txarrak:
  - **Kontsulta eraginkorragoak**: beharrezko datu guztiak batera daude.
  - **Idazketa atomikoak**: dokumentu bakarra modu atomikoan eguneratu daiteke, bi ez.
  - **Erredundantzia**: dokumentu berbera toki askotan errepikatzeko arriskua.
  - **Dokumentuen hazkuntza**: arrisku gehiago dokumentuak gehiegi hazteko (*reallocation*).
- Dokumentuen erreferentziak gordetzearen alde onak eta txarrak:
  - **Erredundantziarik ez**: datuak normalizatuta gordetzen dira.
  - **Dokumentuen hazkuntza txikiagoa**.
  - **Informazio konplexuak adierazteko erraztasuna**: grafoak, zuhaitzak…
  - **Kontsulta ez eraginkorrak**: ez dago *join*ik MongoDB-n.
  - **Idazketa ez atomikoak**: ez dago transakziorik.

Azpidokumentuak *vs* erreferentziak

- Noiz txertatu azpidokumentuak:
  - ***one to many* edo *contains* motako erlazioetan**: dokumentu bat gehien-gehienetan beste baten baitan agertuko da, ez du zentzu handirik dokumentua independenteki erabiltzeak.
- Noiz gorde erreferentziak:
  - ***Many to many* erlazioetan**: dokumentuak txertatuz gero, erredundantzia handia izan daiteke.
  - **Datu-egitura konplexuetan**: grafoak, zuhaitzak…
- Erreferentzia bidez gordetzeko arrazoi berezirik ezean (erredundantzia handia...), azpidokumentu gisa txertatzea gomendatzen da.

Ariketa

Enpresa bat hainbat sailetan banatzen da. Sail bakoitzean langile kopuru handia dago, baina kopuru hori nahiko egonkorra da . Bai sailen eta bai langileen inguruko informazioa gorde nahi dugu MongoDB datu-base batean. Kontsulta gehienak sail mailakoak dira: sail bateko langileak lortu, sail bateko langilerik zaharrenak lortu, sail baten azkeneko proiektuak lortu…

Ariketa II

Futbol-liga baten informazioa gorde nahi da. Bereziki, talde (izena, entrenatzailea, bazkide kopurua, jokalariak…), jokalari (izena, posizioa, jaiotze-data…) eta partiden (taldeak eta emaitza) inguruko datuak gorde nahi dira.

Zer gertatzen da partiduetan goleatzaileen informazioa ere gorde nahi bada?

Ariketa III

Herrialde oso bateko zinema-aretoen informazioa gorde nahi da, bakoitzean eskaintzen diren pelikulekin batera. Pelikulen kasuan, izenaz gain, bestelako informazioa ere gorde nahi da (zuzendaria, aktoreak…).

Eta, areto bakoitzean pelikula bakoitza zein egunetan estreinatu den ere gorde nahiko bagenu?

Ariketa IV

Webgune bat daukagu eta bertako orri bakoitzak jasotzen dituen bisiten informazioa gorde nahi dugu (IP helbidea, data…). Webguneak bisita asko jasotzen ditu egunero (ehunka mila).

Ariketa V

Musikaren katalogo bat egin nahi dugu. Taldeen informazioa (izena, nongoak diren, diskografia…), diskoena (izena, urtea, abestiak…) eta abestiena (izena, pista-zenbakia…) gorde nahi dugu.

NoSQL datu-baseak. MongoDB.

2.3 MongoDB: CRUD eragiketak

CRUD eragiketak

- CRUD eragiketak: Create, Read, Update, Delete.
- Ez da SQL lengoaia erabiltzen, MongoDB-ren funtzioak baizik.
- Eragiketak komando-lerrotik edo programazio-lengoaien bidez (lengoaia ugaritarako APIak daude)
- Adibidez:
  - `db.jokalariak.find({posizioa: "aurrelaria"})` *// Lortu aurrelari guztiak*
  - `db.jokalariak.insert({izena: "Gaizka", abizena: "Toquero", posizioa: "aurrelaria"})` *// Sartu Gaizka Toquero izeneko aurrelari bat*
  - `db.jokalariak.find({posizioa: "aurrelaria"}).sort({abizena: 1})` *// Lortu aurrelari guztiak, abizenaren arabera ordenatuta*

Mongo shell

- MongoDB datu-baseak kudeatzeko tresna erabilgarriena, aplikazio mailan lan egin nahi ez bada, *Mongo shell*-a da.
- Komando-lerrotik egikaritzen da, JavaScript lengoaiaz lan egiten da bertan eta datu-baseen gaineko edozein eragiketa egin daiteke modu sinplean, kontsultak egitetik indizeak definitzeraino.
- Exekutatzeko, terminala ireki eta “mongo” komandoa egikaritu.

Mongo shell


```
zuhaitz@Alienware:~$ mongo
MongoDB shell version: 2.6.1
connecting to: test
> show dbs
admin       (empty)
eng_tokens  0.453GB
kk          0.078GB
local       0.078GB
> use eng_tokens
switched to db eng_tokens
> show collections
system.indexes
tokens
>
```

Mongo shell

- *Shell-a* exekutatzeko (*mongo* prozesua), *mongod* prozesuak martxan egon behar du.
  - Normalean besterik ezean ordenagailuarekin batera jartzen da martxan *mongod* prozesua.
  - mongod prozesuaren konfigurazioa fitxategi batean definitu daiteke (instalazio batzuetan, besterik ezean `/etc/mongodb.conf` fitxategia erabiltzen da). Fitxategia exekuzio garaian aukeratzeko: `mongod -f /patha/mongodb.conf` .

Datu-baseak aztertzen

- Datu-baseak zerrendatzeko:

```
> show dbs
```

- Datu-base bat aukeratzeko (hortik aurrera, eragiketa guztiak DB horren gainean egingo dira):

```
> use euskal_taldeak
```

- Aukeratutako DBaren bildumak zerrendatzeko:

```
> show collections
```

Datuak sartzen

- BSON dokumentuak sartzen dira. Bi pausotan sartu daiteke dokumentu berri bat, lehenik dokumentua definituz eta gero txertatuz:

```
> dokumentua = ({
      izena: “Su Ta Gar”, herria: “Eibar”, urtea: 1987
      diskografia: [{izena: “Jaiotze Basatia”, abesti_kop: 12}, {izena: “Hortzak Estuturik”, abesti_kop: 12}, ...]
      })
> db.taldeak.insert(dokumentua)
```

- **Oharra:** agindu gehienek db.*bilduma*.*agindua*(*argumentuak*) formatua dute (hasierakoa beti da `db` karaktere-parea, ez da datu-basearen izena!).

Kontsultak: `find()`

- Kontsultak egiteko oinarrizko funtzioa: `find()`. Funtzio honek, **bilduma jakin batetik,** eskatutako dokumentuak itzultzen ditu.

```
> db.taldeak.find({ izena: “Su Ta Gar” })
> db.taldeak.find({ izena: “Su Ta Gar”, herria: “Eibar” })
```

- Kontsultak, SQL lengoaiaren ordez, JSONen oso antzeko formatu batean egiten dira.

Kontsultak: `find()`

- `find()` funtzioaren lehenbiziko argumentuak zein dokumentu eskuratu nahi diren definitzen du:
  - Adibidez: itzuli Eibarko talde guztiak:

```
> db.taldeak.find({ herria: "Eibar" })
```

- Bigarren argumentuak, aldiz, eskuratuko diren dokumentuen zein eremu jaso nahi diren adierazten du:
  - Adibidez: itzuli taldeen izenak soilik:

```
> db.taldeak.find({ herria: "Eibar" }, { izena: 1 })
```

Kontsultak: `find()`

- Adibide gehiago:
  - Itzuli talde guztien identifikadoreak:

```
> db.taldeak.find({}, { _id: 1 })
```

  - Su Ta Gar taldearen informazioa, diskografia izan ezik:

```
> db.taldeak.find({ izena: "Su Ta Gar" }, { diskografia: 0 })
```

  - Su Ta Gar taldearen informazioa, diskografia eta identifikadorea izan ezik:

```
> db.taldeak.find({ izena: "Su Ta Gar" }, { diskografia: 0, _id: 0 })
```

Kontsultak: azpidokumentuak

- Azpidokumentuen informazioa ere kontsulten baldintzetan sartzeko puntua (.) erabiltzen da:
  - Itzuli Jaiotze Basatia diskoaren taldea

```
> db.taldeak.find({ “diskografia.izena”: “Jaiotze Basatia” })
```

- Ez dago modurik `find()` funtzioarekin azpidokumentuak itzultzeko. Baldintzak edozein direla ere, baldintzak betetzen dituzten dokumentu **osoak** itzuliko dira.
- Hori Aggregation Framework delakoarekin lor daiteke (aurrerago ikusiko dugu)

Kontsultak: findOne()

- `find()` funtzioak dokumentu zerrenda bat itzultzen du. Zenbaitetan, nahiago izango dugu dokumentu bakarra jaso. Horretarako dago `findOne()` funtzioa. Sintaxia `find()`-en berbera da, baina baldintzak betetzen dituen lehenbiziko dokumentua itzultzen du.
  - Itzuli Jaiotze Basatia diskoaren taldea

```
> db.taldeak.findOne({ "diskografia.izena": "Jaiotze Basatia" })
```

Kontsultak: `sort()`, `limit()`, `skip()`

- `sort()`: find() bidez lortutako dokumentu zerrenda bat ordenatzen du:
  - Itzuli talde guztiak, izenaren arabera ordena gorakorrean ordenatuta:

```
> db.find().sort({ izena: 1 })
```

- `limit()`: kontsulta batek itzulitako dokumentu kopurua mugatzen du:
  - Itzuli, izenaren arabera alfabetikoki ordenatuta, lehenbiziko hiru taldeak:

```
> db.find().sort({ izena: 1 }).limit(3)
```

- `skip()`: dokumentu zerrendako lehenbiziko *n* dokumentuak saltatzen ditu:
  - Itzuli, izenaren arabera alfabetikoki ordenatuta, hirugarren taldetik hamargarrenera:

```
> db.find().sort({ izena: 1 }).limit(8).skip(2)
```

Kontsultak: `count()`

- `count()`: Kontsulta batek itzulitako dokumentu kopurua zenbatzen du:
  - Bildumako talde kopurua:

    ```
    > db.taldeak.find().count()
    ```

  - Eibarko talde kopurua:

    ```
    > db.taldeak.find({ herria: "Eibar" }).count()
    ```

- **Oharra**: `skip()` eta `limit()` funtzioen eragina ez du kontuan hartzen besterik ezean, horretarako argumentu bezala `true` balioa pasatu behar zaio:

  ```
  > db.taldeak.find().limit(10).count() => talde kopuru totala

  > db.taldeak.find().limit(10).count(true) => 10
  ```

Kontsultak: `distinct()`

- **`distinct()`:** Agindutako eremuaren balio desberdin guztiak itzultzen ditu, zerrenda batean, errepikatu gabe:

```
> db.taldeak.distinct("herria") => Herri guztien zerrenda itzuliko du
> db.taldeak.distinct("diskografia.izena") => Talde guztien disko guztien zerrenda
```

Kontsultak: `$gt`, `$lt`, `$gte`, `$lte`

- “handiago” eta “txikiago” motako eragileak ere defini daitezke:
  - 10 abesti baino gehiagoko diskoren bat duten taldeak:

```
> db.taldeak.find({ “diskografia.abesti_kop”: {$gt: 10} })
```

  - Bost abesti edo gehiago, eta hamar edo gutxiago dituen diskoren bat duten taldeak:

```
> db.taldeak.find({ “diskografia.abesti_kop”: {$gte: 5, $lte: 10} })
```

Kontsultak: `$size`

- Emaitzak array baten osagai kopuruaren arabera filtratzen ditu:
  - Itzuli disko bakarra duten taldeak:

```
> db.taldeak.find({ diskografia: {$size: 1} })
```

  - Itzuli disko bat baino gehiago dituzten taldeak:

```
> db.taldeak.find({ diskografia: {$size: {$gt: 1}} })
```

Kontsultak: `$in`, `$nin`, `$ne`

- Baldintza gisa eremu bati balio zehatz bat esleitu ordez, balio posibleen multzo bat esleitu dakioke `$in` erabiliz:
  - Itzuli Donostiako eta Iruñeko talde guztiak:

```
> db.taldeak.find({ herria: {$in: ["Donostia", "Iruñea"]} })
```

- `$nin`-ek (*not in*) kontrakoa egiten du, adierazitako baliorik ez duten dokumentuak itzultzen ditu.
- `$ne`-k (*not equal*) `$nin`-en gauza bera egiten du, baina balio bakarrarekin:
  - Itzuli Eibarkoak ez diren talde guztiak:

```
> db.taldeak.find({ herria: {$ne: "Eibar"} })
```

Kontsultak: $elemMatch

- Zer gertatzen da array bateko elementu jakin batek baldintza bat baino gehiago betetzea nahi dugunean?
  - Adibidez: Demagun orain taldeen diskoek beste eremu bat dutela, argitaratze-urtea. 10 abesti dituen eta 2000 urtean argitaratu den diskoren bat duten taldeak lortu nahi ditugu:

    ```
    > db.taldeak.find({ “diskografia.abesti_kop”: 10, “diskografia.urtea”: 2000})
    ```

    => **Gaizki!** Baldintza bakoitza zerrenda osoan begiratzen da, ez elementu berean!

    ```
    > db.taldeak.find({ diskografia: {$elemMatch: {abesti_kop”: 10, urtea: 2000}}
    })
    ```

    => **Ongi!** Modu honetan, baldintza guztiak elementu berari aplikatzen zaizkiola ziurtatzen dugu.

Kontsultak: `$or`

- `$or`: Hainbat baldintzaren artean gutxienez bat bete behar du.
  - Itzuli Eibarkoa izateaz gain, disko bakarra duten edo 1999 urtean sortu ziren taldeak:

```
> db.taldeak.find({ herria: "Eibar",
                    $or: [ {diskografia: {$size: 1}}, {urtea: 1999}]
                    })
```

Kontsultak: `$exists`

- Eremu jakin bat definituta duten dokumentuak bakarrik eskuratzea ahalbidetzen du `$exists` eragileak:
  - Itzuli sortze-urtea definituta duten taldeak:

```
> db.taldeak.find({ urtea: {$exists: true} })
```

Kontsultak: `$not`

- `$not`: Eragile honek edozein baldintza ukatzen du.
  - Adibidez: Herria definituta ez duten taldeak itzuli:

```
> db.taldeak.find({ herria: {$exists: false} })
```

- Beste modu bat, `$not` erabiliz:

```
> db.taldeak.find({ herria: {$not: {$exists: true}} })
```

Kontsultak: Espresio erregularrak

- Kontsulta baten definizioan espresio erregularrak erabili daitezke:
  - Izenean “eta” hitza daramaten taldeak:

```
> db.taldeak.find({ izena: / eta / })
```

  - Izenaren hasieran zenbaki bat dutenak:

```
> db.taldeak.find({ izena: /^\d/ })
```

Eguneraketak: `update()`

- Bilduma bateko dokumentuak eguneratzeko oinarrizko funtzioa `update()` da. Bi argumentu hartzen ditu: aldatuko diren dokumentuak aukeratzeko baldintzak eta datu berriak.
- Besterik ezean dokumentu osoa berridazten denez, zehazten ez diren eremuak ez dira mantenduko, ezabatu egingo dira.
- Eskatutako dokumentua aurkitu ezean, berria sortzeko eskatzeko aukera ere badago (upsert).

Eguneraketak: `update()`

- Adibideak:
  - Aldatu Su Ta Gar taldearen herria, eta jarri Gernika:

    ```
    > db.taldeak.update({ izena: "Su Ta Gar" }, { herria: "Gernika"})
    ```
    =>

    Gaizki! Herria (eta "_id") izan ezik, beste eremu guztiak galdu egingo dira!

  - Hesian izeneko taldeari aldatu sortze-urtea, eta jarri 2000. Taldea datu-basean ez badago, sortu dokumentu berria:

    ```
    > db.taldeak.update({ izena: "Hesian" }, { urtea: 2000, izena: "Hesian",
    herria: "Sakana"}, { upsert: true })
    ```

Eguneraketak: `$set`, `$unset`

- Eremu baten balioa aldatu nahi dugunean, ez dugu eremu guztien balioak berriz definitzen hasi nahi (`update()`-ren besterik ezeko portaera). Aldiz, `$set` eragilearekin eremu bakarra eguneratu dezakegu.
  - Aldatu Su Ta Gar taldearen herria, eta jarri Gernika:

```
> db.taldeak.update({ izena: "Su Ta Gar" }, {$set: {herria: "Gernika"}})
```

- `$unset`-ekin kontrakoa egin dezakegu, eremu bat kendu. Kontuz! Eremu bat kentzeak ez du esan nahi `Null` balioa ematea, errotik kentzea baizik.

```
> db.taldeak.update({ izena: "Su Ta Gar" }, {$unset: {herria: 1}})
```

Eguneraketak: $push, $addToSet

- **$push**: Eragile honek array batean balio berri bat gehitzeko aukera ematen du.
  - Gehitu disko berri bat Hesian taldearen diskografian:

```
> db.taldeak.update({ izena: "Hesian" },
                    {$push: { diskografia:{ izena: "Hitzetik", abesti_kop: 11}}})
```

- **$addToSet**: Modu berean erabiltzen da, baina elementua aurrez zerrendan badago ez du berriro gehitzen.

Eguneraketak: `$pop`, `$pull`, `$pullAll`

- **`$pop`**: Ezabatu zerrendako lehen edo azken elementua.

```
> db.taldeak.update({ izena: "Hesian" }, {$pop: {kideak: 1}}) // Azkeneko kidea kendu
```

- **`$pull`**: Ezabatu zerrendatik balio jakin bat duten elementuak.

```
> db.taldeak.update({ izena: "Hesian" }, {$pull: {kideak: "Fran"}})
```

- **`$pullAll`**: Ezabatu zerrendatik hainbat balio eragiketa bakarrean.

```
> db.taldeak.update({ izena: "Hesian" }, {$pullAll: {kideak: ["Fran", "Zuriñe"]}})
```

Eguneraketak: `$inc`

- `$inc`: Zenbakizko eremu baten balioa inkrementatzen du.
  - Gehitu bi urte Berri Txarrak taldearen sortze-urteari:

```
> db.taldeak.update({ izena: "Berri Txarrak" }, {$inc: {urtea: 2}})
```

Eguneraketak: “$” eragilea

- Nola eguneratu array bateko elementu jakin bat? `$` eragileak parekatzen dugun elementuaren indizea adierazten du, gero parekatutako elementu hori atzitzeko eta eguneratzeko.
  - Berri Txarrak taldearen Libre diskoaren argitaratze urteari urte bat gehitu:

```
> db.taldeak.update({ “diskografia.izena”: “Libre” },
                      {$inc: {“diskografia.$.urtea”: 1}})
```

Dokumentuak ezabatzen: `remove()`

- Dokumentuak ezabatzeko, besterik gabe, `remove()` funtzioa erabiliko dugu, argumentu bezala atzitu nahi ditugun dokumentuak definituko dituzten baldintzak zehaztuz.
  - Ezabatu Bilboko talde guztiak:

    ```
    > db.taldeak.remove({ herria: "Bilbo" })
    ```

  - Ezabatu bildumako talde guztiak:

    ```
    > db.taldeak.remove({})
    ```

Bildumak eta datu-baseak ezabatzen

- **`drop()`**: Bilduma bat ezabatzen du.

```
> db.taldeak.drop()
```

- **`renameCollection()`**: Bilduma bati izena aldatzen dio.

```
> db.taldeak.renameCollection("euskal_musika_taldeak")
```

- **`dropDatabase()`**: Aukeratuta daukagun datu-basea ezabatzen du.

```
> db.dropDatabase()
```

Aggregation Framework

- Kontsultak egiteko beste sistema bat da, 2.2 bertsioarekin batera agertu zena.
- Helburua zera da: kontsulten *pipeline* bat definitzea, bakoitzaren emaitza hurrengoaren sarrera izanik. Honela, kontsulta konplexuak modu erraz eta txukun batean egin ditzakegu.
- **Adibidea**: Itzuli 2000. urtetik aurrera kaleratutako diskoak:

```
> db.taldeak.aggregate(
    {$unwind: "$diskografia"},
    {$match: {"diskografia.urtea": {$gte: 2000}}})
```

Aggregation Framework: `$match`

- Dokumentuak filtratzen ditu, *pipeline*aren hurrengo fasera baldintzak betetzen dituzten dokumentuak bakarrik pasatuz.
  - Lortu Tolosako taldeak soilik:

```
> db.taldeak.aggregate({$match: {herria: "Tolosa"}})
```

Aggregation Framework: `$sort`

- **`$sort:`** dokumentuak ordenatzen ditu agindutako eremuen arabera.
  - Zerrendatu disko guztiak, argitaratze urtearen arabera ordenatuta (berrienetik zaharrenera):

```
> db.taldeak.aggregate({$unwind: "$diskografia"},
                        {$sort:  {"diskografia.urtea": -1}})
```

Aggregation Framework: `$limit`

- Jasotako dokumentuen lehenbiziko n dokumentuak bakarrik pasatzen dira hurrengo fasera:
  - Bost talderik zaharrenak itzuli:

```
> db.taldeak.aggregate({$sort: {"urtea": 1}},
                       {$limit: 5})
```

Aggregation Framework: `$unwind`

- Array bateko elementu guztiak dagokion dokumentutik “askatzen” ditu, dokumentu independente bihurtuz azpidokumentu guztiak.
  - Lortu disko guztiak:

```
> db.taldeak.aggregate({$unwind: “$diskografia”})
```

  - Lortu hamar abesti dituzten disko guztiak:

```
> db.taldeak.aggregate({$unwind: “$diskografia”},
                                {$match: {“diskografia.abesti_kop”: 10}})
```

Aggregation Framework: `$project`

- **`$project`** eragileari esker, dokumentuei eremu berriak gehitu diezazkiokegu, eta baita aurretik bazeudenak kendu ere.
  - Kalkulatu talde bakoitzaren urte kopurua (2014. urtean):

```
> db.taldeak.aggregate({$project:
        {"_id": 1, "urte_kopurua": {$subtract: [2014, "$urtea"]}}
        })
```

- Oharra: `$subtract` eragileak emandako zerrendako elementuen arteko kenketa egiten du.

Aggregation Framework: `$group`

- **`$group`:** SQLko GROUP BY egituraren antzekoa. Dokumentuak eremu baten edo gehiagoren arabera multzokatzen ditu, eta multzo bakoitzeko batezbestekoak etab. kalkulatu.
  - Urte bakoitzean sortutako talde kopurua, urte emankorrenetik emankortasun gutxienekora. ordenatuta:

```
> db.taldeak.aggregate({ $group: {_id: "$urtea", "talde_kop": {$sum: 1}} },
                        { $sort: {talde_kop: -1}})
```

- `$group`-en barruan erabili daitezkeen eragileak: `$sum`, `$addToSet`, `$first`, `$last`, `$max`, `$min`, `$avg`, `$push`.

NoSQL datu-baseak. MongoDB.

2.4 MongoDB: Indizeak eta neurketak

Exekuzio-denborak neurtzen

- Bi tresna exekuzio-denborak eta eraginkortasuna neurtzeko: MongoDB Profiler tresna eta `explain()` funtzioa.
- **MongoDB Profiler**: Egokia motel exekutatzen ari diren kontsultak aurkitzeko.
- **`explain()`**: Egokia kontsulta jakin baten eraginkortasuna neurtzeko.

MongoDB Profiler

- *Profiler*-a gaitzeko bi modu:
  - mongod instantzia osoari eragiten diona: mongod exekutatzean --profile eta --slowms parametroak pasa. Adibidez:

    ```
    $ mongod --profile=1 --slowms=30
    ```

    Goiko aginduak analisi maila 1 dela definitzen du, eta kontsulta moteltzat hartzen ditu 30 ms baino gehiagoko exekuzio-denbora dutenak.
  - Guk nahi dugunean gaitu eta desgaitu dezakegu *profiler*-a:

    ```
    > db.setProfilingLevel(1) // Gaitu
    > db.setProfilingLevel(0) // Desgaitu
    ```

MongoDB Profiler

- Hiru analisi-maila desberdin daude burutu nahi den analisi motaren arabera:
  - **0. maila**: *Profiler*-a desaktibatu, ez egin analisirik.

    > db.setProfilingLevel(0)
  - **1. maila**: *Profiler*-a gaitu, eta zehaztu zenbat ms-tik gorako kontsultak gorde logean.

    > db.setProfilingLevel(1, 30)
  - **2. maila**: Kontsulta guztiak analizatu eta gorde.

    > db.setProfilingLevel(2)

MongoDB Profiler

- *Profiler*-ak `system.profile` bilduman gordetzen du kontsulten inguruko informazioa, beste edozein BSON dokumenturen formatu berean.
- Informazio hori kontsultatzeko, bilduma arrunt batean egingo genukeen bezala egingo dugu.
- Adibidez: Erakutsi 15 ms baino gehiago behar izan duten kontsulten informazioa, motelenetik azkarrenera ordenatuta:

```
> db.system.profile.find({ millis: {$gt: 15} }).sort({millis: -1})
```

MongoDB Profiler

- `system.profile` bildumako dokumentuen eremu garrantzitsuenak hauek dira:
  - op: Eragiketa mota (query, update…).
  - query: Burututako kontsulta bera.
  - nreturned: Itzulitako dokumentu kopurua.
  - nscanned: Sistemak kontsulta burutzeko aztertu behar izan duen dokumentu kopurua.
  - responseLength: Itzulitako dokumentuen byte kopuru totala.
  - millis: Kontsulta burutzeko behar izan den milisegundo kopurua.
  - ts: Kontsulta egikaritu den unea (*timestamp*).

```
{
    "op" : "query",
    "ns" : "eng_tokens.tokens",
    "query" : {
       "length" : {
             "$gt" : 5
       }
    },
    "cursorid" : 87673938946,
    "ntoreturn" : 0,
    "ntoskip" : 0,
    "nscanned" : 440,
    "nscannedObjects" : 440,
    "keyUpdates" : 0,
    "numYield" : 0,
    "lockStats" : {
       "timeLockedMicros" : {
             "r" : NumberLong(525),
             "w" : NumberLong(0)
       },
       "timeAcquiringMicros" : {
             "r" : NumberLong(3),
             "w" : NumberLong(3)
       }
    },
    "nreturned" : 101,
    "responseLength" : 5914,
    "millis" : 0,
    "execStats" : {
       "type" : "COLLSCAN",
       "works" : 441,
       "yields" : 3,
       "unyields" : 3,
       "invalidates" : 0,
       "advanced" : 101,
       "needTime" : 340,
       "needFetch" : 0,
       "isEOF" : 0,
       "docsTested" : 440,
       "children" : [ ]
    },
    "ts" : ISODate("2014-06-24T08:30:06.409Z"),
    "client" : "127.0.0.1",
    "allUsers" : [ ],
    "user" : ""
}
```

`explain()` funtzioa

- MongoDB Profiler-arekin lortzen den antzeko informazioa lortu daiteke `explain()` funtzioa kontsulta jakin bati aplikatuz.
- Informazioa unean bertan itzultzen da, ez da `system.profile` bilduman gordetzen.
- Adibidez:
  - `db.taldeak.find({urtea: 2014}).explain()`

```
> db.tokens.find({length: {$gt: 5}}).explain()
{
    "cursor" : "BasicCursor",
    "isMultiKey" : false,
    "n" : 269787,
    "nscannedObjects" : 1159874,
    "nscanned" : 1159874,
    "nscannedObjectsAllPlans" : 1159874,
    "nscannedAllPlans" : 1159874,
    "scanAndOrder" : false,
    "indexOnly" : false,
    "nYields" : 10161,
    "nChunkSkips" : 0,
    "millis" : 1892,
    "server" : "Alienware:27017",
    "filterSet" : false
}
```

Indizeak

- Indizeak ezinbestekoak dira kontsulten eraginkortasuna bermatzeko.
- Indizeek kontsultei mesede egiten dieten bezala, kalte egiten die idazketei, indizeen egitura eguneratu behar baita idazketa bakoitzean.
- Bilduma mailan definitzen dira indizeak.
- `sort()` funtzioak errorea ematen du ordenatze-prozesuan darabilen bufferra (32 MB) betetzen bazaio, oso ohikoa dokumentu asko dauzkagunean. Hori ekiditeko, indizeak ezinbestekoak dira.

Indizeak

- Definitutako indize guztien definizioak `system.indexes` bilduman gordetzen dira. Beste edozein bildumatan bezala egin ditzakegu kontsultak hemen ere.

```
> db.system.indexes.find()
```

  - Goiko eragiketak bilduma guztietako indizeak erakusten ditu. Bilduma jakin baten indizeak ikusteko:

```
> db.taldeak.getIndexes()
```

- `_id` eremuaren gainean indizea sortzen da beti.

Indizeak sortzen

- **Indize arruntak**: eremu bakarraren gainean definitzen direnak.

```
> db.taldeak.ensureIndex({urtea: 1})
> db.taldeak.ensureIndex({"diskografia.urtea": 1})
```

- **Indize konposatuak**: eremu baten baino gehiagoren gainean definitzen direnak. Lehenbiziko eremuaren arabera ordenatzen dira dokumentuak, baina balio berbera duten dokumentuak bigarren eremuaren arabera ordenatzen dira.

```
> db.taldeak.ensureIndex({urtea: 1, izena: 1})
```

Indizeen parametroak

- **`{background: true}`:** Indizea bilduman dokumentu asko daudenean definitzen bada, denbora asko behar izan lezake sortzen bukatzeko. Parametro honekin definitzen bada, datu-basea erabiltzen jarraitu dezakegu indizea sortzen den bitartean.
  - Indizea sortzen ari dela prozesua akabatu nahi badugu, honela egin dezakegu:

    `> db.killOp(<eragiketaren IDa>)` // IDak ikusteko: `db.currentOp()`

- **`{unique: true}`**: Eremu bateko balioak unibokoak direla bermatzen du indize honek. Aurrez badagoen balio bat dokumentu berri batean sartzen saiatzen bagara, errorea ematen du. Indizea definitzean ere errorea emanen du balioak errepikatuta badaude.

Indizeen parametroak

- `{sparse: true}`: Eremu jakin baten balioak definituta dituzten dokumentuak bakarrik sartzen dira indizean aukera honekin.

Indizeak ezabatzen

- Bilduma bateko indize guztiak ezabatzeko:

```
> db.taldeak.dropIndexes()
```

- Indize jakin bat ezabatzeko, sortzeko garaian zehaztutako definizioa eman behar zaio berriz, indizeek ez baitute identifikadorerik!

```
> db.taldeak.dropIndex({urtea: 1, izena: 1})
```

Indizeen aukeraketa

- MongoDB kontsulta bakoitza egikaritzeko, indizerik egokiena aukeratzen saiatzen da. Hala ere, zenbaitetan guk adierazi nahi izango diogu zein indize erabili zehazki, **`hint()`** funtzioa erabiliz:

```
> db.taldeak.find({urtea: {$gte: 2000}}).hint({urtea: 1})
```

- Indizerik ez erabiltzea behartzeko **`$natural`** eragilea erabili:

```
> db.taldeak.find({urtea: {$gte: 2000}}).hint({$natural: 1})
```

- Kontsulta bakoitzaren kasuan zein indize aukeratu duen ikusteko, atal honen hasieran aipatutako **`explain()`** funtzioa erabili daiteke.

NoSQL datu-baseak. MongoDB.

2.5 MongoDB: GridFS

Sarrera

- MongoDB-ren dokumentuen gehienezko tamaina 16 MBekoa da.
- Gaur egun, tamaina hori motz gelditzen da irudiak, bideoak eta bestelakoak gordetzeko.
- MySQL datu-baseetan BLOB mota dago irudiak, bideoak etab. gordetzeko, baina ez da oso soluzio “dotorea”.
- **GridFS**: Fitxategi handi bat, modu transparentean, MongoDB-ko hainbat dokumentutan banatuta gordetzeko sistema.

GridFS: Eragiketak

- Komando lerrotik GridFS erabiltzeko tresna bat dakar MongoDB-k: mongofiles.
- Demagun oporretan_12.jpg irudia daukagula /home/erab/irudiak karpetan. Honela gehituko genuke `argazkiak` izeneko datu-basean:

```
$ mongofiles put /home/erab/irudiak/oporretan_12.jpg -d argazkiak
```

- Datu-baseko fitxategiak ikusteko, aldiz:

```
$ mongofiles list -d argazkiak
```

GridFS: Eragiketak

- Fitxategi bat datu-basetik berreskuratzeko:

```
$ mongofiles get /home/erab/argazkiak/oporretan_12.jpg -d argazkiak
```

- **Oharra**: Kontuz eragiketa honekin, jatorrizko direktorioan sortzen baitu fitxategia (kasu honetan, /home/erab/argazkiak), eta aurrez izen berarekin beste zerbait bazegoen, gainidatzi egiten baitu.

- Fitxategi bat datu-basetik ezabatzeko:

```
$ mongofiles delete /home/erab/argazkiak/oporretan_12.jpg -d argazkiak
```

GridFS: Barne egitura

- Bi ataletan banatzen da GridFS. Batetik, datuak gordetzen diren `chunks` bilduma, eta, bestetik, metainformazioa gordetzen den `files` bilduma.
- Fitxategi bat GridFS sistema erabiliz gordetzen dugunean, honela antolatzen da informazioa:

db.fs.files	db.fs.chunks
- **_id**: fitxategiaren identifikadorea - **filename**: fitxategiaren *path*-a eta izena - **chunkSize**: zatien tamaina - **uploadDate**: sortze-data - **md5**: fitxategiaren MD5a - **length**: fitxategiaren tamaina bytetan	- **_id**: zatiaren identifikadorea - **files_id**: fitxategiaren identifikadorea - **n**: zenbatgarren zatia den

GridFS: Bestelako eragiketak

- `mongofiles` tresna erabiliz bilaketak egin ditzakegu fitxategien edukietan, eta tresnak eduki hori zein fitxategitan aurkitu duen esango digu:

```
$ mongofiles search kaixo!
```

- Noski, argazkiak gordetzen baditugu ez dauka zentzurik testu-bilaketak egitea, baina gogoratu edozein fitxategi mota gorde dezakegula, testu-fitxategiak barne.

- `mongofiles` tresnarekin baino erabilgarriago suertatzen da GridFS programazio-lengoaia batekin erabiltzea, web-aplikazio baten barruan esate baterako.

NoSQL datu-baseak. MongoDB.

2.6 MongoDB: Python *driver*-a

Python eta MongoDB

- MongoDB programazio-lengoaia askorekin erabil daiteke, baina errazen eta intuitiboen, segur aski, Pythonekin suertatzen da (PyMongo). Batez ere, bien arteko sintaxiaren antzekotasun handiari esker.

Hau da MongoDB-ko dokumentu baten sintaxia:

```
taldea = {
        "izena": "Sorotan Bele",
        "herria": "Hondarribia",
        "kideak": ["Mikel Errazkin", "Mikel Izulain", …]
        "diskografia": [
                {"izena": "Sorotan Bele", "urtea": 1992},
                {"izena": "Mundu Hegian", "urtea": 1994}
        ]}
```

Eta hau da dokumentu beraren sintaxia Pythonen:

```
taldea = {
        "izena": "Sorotan Bele",
        "herria": "Hondarribia",
        "kideak": ["Mikel Errazkin", "Mikel Izulain", …]
        "diskografia": [
                {"izena": "Sorotan Bele", "urtea": 1992},
                {"izena": "Mundu Hegian", "urtea": 1994}
        ]}
```

MongoDB-rekin konektatzen

- PyMongo-ren MongoClient klasea erabiliko dugu konektatzeko:

```
>>> from pymongo import MongoClient
>>> mongo = MongoClient()  // baliokidea: MongoClient("localhost", 27017)
```

- Behin MongoDB-rekin konektatu garela, datu-base bat aukeratu eta aldagai batean erreferentzia gordeko dugu:

```
>>> db = mongo.euskal_musika
```

- Azkenik, bilduma baten erreferentzia gordeko dugu, eragiketa gehienak bertan egingo baititugu:

```
>>> bilduma = db.taldeak
```

Dokumentu berriak sartzen

- MongoDB-ren kasuan komando-lerrotik sartzen genituen bezalaxe sartzen dira dokumentuak Python-ekin ere:

```
>>> taldea = {"izena": "Sorotan Bele", "urtea": 1992}
>>> bilduma.insert(taldea)
```

- Era berean, dokumentu zerrenda bat ere sar dezakegu eragiketa bakarrean:

```
>>> taldeak = [{"izena": "Latzen", "urtea": 1993}, {"izena": "Hemendik At!", "urtea": 1995}]
>>> bilduma.insert(taldeak)
```

Kontsultak

- Bilduma gorde dugun aldagaia erabiliko dugu kontsultak egiteko:

```
>>> taldea = bilduma.find_one({"izena": "Sorotan Bele"})
```

- Dokumentu zerrenda bat jasotzen dugunean, erraz iteratu dezakegu elementu guztietan zehar:

```
>>> for taldea in bilduma.find():
...         print taldea
```

Kontsultak

- Eremuak filtratu nahi ditugunean, `fields` parametroa definitu behar dugu:

```
>>> talde_izenak = bilduma.find({"urtea": 2000}, fields={"izena": True})
```

- Bestelako funtzioen erabilera (`sort()`, `limit()`, `count()`, `$gt`, `$lt`, `$in`...) ia berdin mantentzen da:

```
>>> talde_kopurua = bilduma.find().count()
>>> talde_berriak = bilduma.find({"urtea": {"$gt": 2010}}).
                          sort("urtea", pymongo.ASCENDING)
```

Aggregation Framework

- *Aggregation Framework*-a ere modu berean erabiltzen da ia. Batzuetan, modulu gehigarri batzuk inportatu behar izango ditugu:

```
>>> from bson.son import SON
>>> db.taldeak.aggregate(
                { '$group': {'_id': '$urtea', 'talde_kop': {'$sum': 1}} },
                { '$sort': SON([('talde_kop': -1)])}
            )
```

Espresio erregularrak

- Espresio erregularrentzako Python-en lehenetsitako modulua `re` da.
- Pymongo-rako ere hori erabiliko dugu:

```
>>> import re
>>> bilduma.find({'izena': re.compile(' eta ')})
```

Eguneraketak

- Eguneraketak egiteko modua ere ez da asko aldatzen:

```
>>> bilduma.update({'izena': 'Sorotan Bele'}, {'$inc': {'urtea': 1}})
```

- Dokumentuak, bildumak eta datu-baseak ezabatzeko:

```
>>> bilduma.remove({'izena': 'Sorotan Bele'})

>>> bilduma.drop()

>>> mongo.drop_database("euskal_musika")
```

Dokumentuen arteko erreferentziak

- Dokumentuen erreferentziak egiteko bi modu daude:
  1. Erreferentzia eskuz egitea, hau da, 1-n erlazio batean, n aldeko elementuei 1 aldeko elementuaren identifikadorea gehitzea (gako arrotza).
  2. Gauza bera egitea, baina DBRef() sistema erabilita:

```
>>> from bson.dbref import DBRef
>>> talde_id = 't123'  // taldeak 'taldeak' izeneko bilduman dauzkagu
>>> disko_berria = {   // diskoak 'diskoak' izeneko bilduman dauzkagu
     'izena': 'Jonen Kezkak',
     'urtea': 1996,
     'taldea': DBRef('taldeak', talde_id) }
```

Dokumentuen arteko erreferentziak

- Ondoren, disko baten taldea eskuratzeko (2. modua erabili badugu):

```
>>> db.dereference(diskoa["taldea"])
```

  - Modu honetan, taldea zein bildumatan dagoen jakin gabe ere lortu dezakegu erreferentzia.

- Erreferentzia eskuz egin badugu (1. modua), honela lortuko genuke disko baten taldea:

```
>>> db.taldeak.find_one({'_id': diskoa["taldea"]})
```

  - Horretarako, noski, diskoaren taldea `taldeak` bilduman dagoela jakin beharko genuke.

NoSQL datu-baseak. MongoDB.

2.7 MongoDB: Erreplikazioa

Sarrera

- **Erreplikazioa**: Datu-baseetako informazioaren kopia bakarra mantendu ordez, informazioa zerbitzari gehiagotan kopiatuta eta denbora ia errealean sinkronizatuta mantentzea.
- Partizionatze horizontalarekin batera, *big data* eredurako ezinbesteko funtzionalitatea.

Erreplikazioaren helburuak

- **Eskalagarritasuna**: Erredundantzia hobetzen da, erreplikazioari esker datuen kopia lokalak hainbat zerbitzaritan mantendu baitaitezke. Horrela, aplikazioaren erabiltzailea gertuen daukan zerbitzarira bideratuko da, latentzia txikituz. Bestalde, eskaerak hainbat zerbitzaritan banatu daitezke, paralelizazio maila handituz.
- **Iraunkortasuna** eta **fidagarritasuna**: Datuen kopiak zerbitzari ugaritan mantenduz, backup-zerbitzari gisa erabil litezke hauetako bat edo gehiago. Horrela, zerbitzari batek huts egingo balu, datuak beste zerbitzari batetik berreskura daitezke. Gainera, zerbitzarietan mantentze-lanak egin behar direnerako, manipulatu nahi den zerbitzaria sistematik deskonektatu daiteke, besteak martxan mantenduz.
- **Isolamendua**: Datu-basearen gainean prozesaketa astun bat egin nahi bada (backup bat, estatistikak atera…), zerbitzari bakar baten gainean egin daiteke dena, erreplikatutako beste zerbitzariak eskuragarri utziz, sistema osoaren eraginkortasunari eragin gabe.

Erreplika multzoa

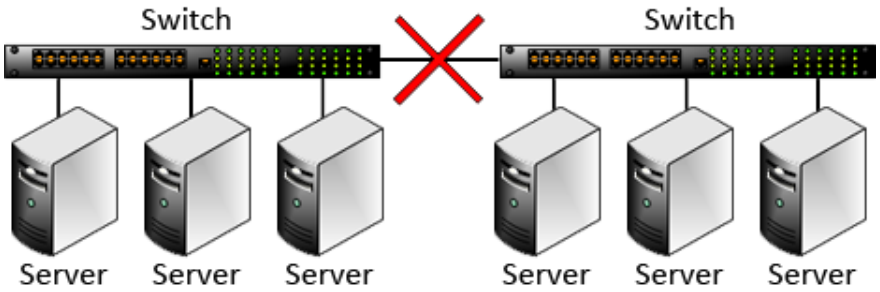
- Datuen kopiak mantentzen dituzten zerbitzarien multzoari erreplika multzoa (***replica set***) deitzen zaio.
- Erreplika multzo bakoitzean nodo nagusi bat eta bigarren mailako hainbat nodo edo arbitro-nodo daude.
- **Nodo nagusia**n bakarrik egin daitezke idazketak, eta bigarren mailako nodoek bertatik kopiatuko dituzte datuak.
- Nodo nagusia zein izan bozketa bidez erabakitzen da.

Erreplika multzoa

- **Bigarren mailako nodo**etatik irakurri bakarrik egin daiteke. Nodo hauek nodo nagusiarekin sinkronizatzen dira. Nodo nagusiarekin arazoren bat dagoenean, bigarren mailako bat bihurtzen da nagusi, bozketa bidez.
- **Arbitro**ek ez dute datuen kopiarik mantentzen, bozketan parte hartzen dute bakarrik.
    - Normalean *Split Brain* arazoa ekiditeko erabiltzen dira.

Split Brain arazoa

- Nodo batek, nagusi izateko, erreplika multzoko nodoen gehiengoa ikusteko gai izan behar du. Hori dela eta, erreplika multzoetan nodo kopuruak bakoitia izan behar du. Hori lortzeko arbitroak gehitu daitezke.



Split brain arazoa.

Oplog

- Datuen sinkronizazioa burutzeko ***oplog*** (*operation log*) delakoa erabiltzen da. Nodo nagusiak logean idazten ditu egikaritzen dituen eguneratze-eragiketa guztiak. Horrela, bigarren mailako nodoek *oplog*-a irakurri eta bertako aginduak erreplikatzen dituzte. Bigarren mailako nodoren bat denbora-tarte batez sistematik deskonektatuta egon bada ere, *oplog*-ari esker sinkronizatu daiteke.

Erreplika multzo bat eraikitzen

Erreplika multzo bat martxan jartzeko:

1. `mongod` prozesua martxan jarri makina bakoitzean, erreplika multzoaren izena adieraziz:

   ```
   $ mongod --replSet probamultzoa
   ```

2. Hasiera batean nodo nagusia izango denera konektatu:

   ```
   $ mongo host:portua
   ```

3. Erreplika multzoa hasieratu eta ezarri:

   ```
   > rs.initiate()
   ```

4. Gainontzeko partaideak gehitu:

   ```
   > rs.add("host:portua")
   ```

5. Egiaztatu dena ongi joan dela:

   ```
   > rs.status()
   ```

NoSQL datu-baseak. MongoDB.

2.8 MongoDB: Partizionatze horizontala (*sharding*)

Sarrera

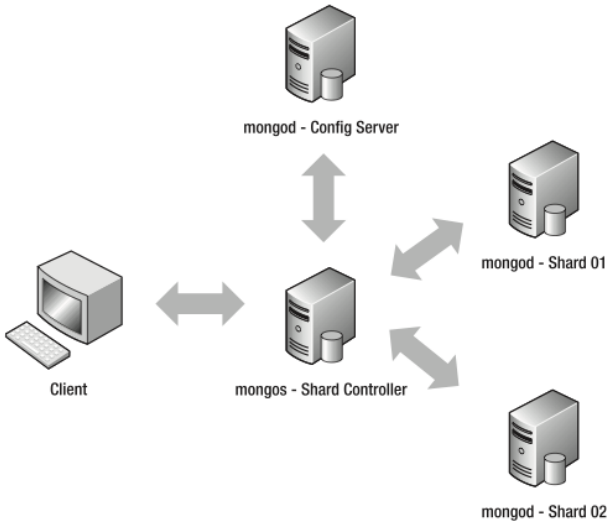
- Eskalagarritasuna bermatzeko ezinbesteko ezaugarria.
- **Eskalagarritasun bertikala**: Eraginkortasuna hobetze aldera, eskura daukagun makina hobetzean datza, ahaltsuago bihurtuz (memoria gehiago, prozesamendu-ahalmen handiagoa…). Irtenbide hau oso mugatua da.
- **Eskalagarritasun horizontala**: Eraginkortasuna hobetze aldera, hainbat makina erabiltzean datza, nahiz eta bakoitza ez izan oso ahaltsua. Aplikazio eta software mailan eredu-aldaketa eskatzen du, baina askoz ere malguagoa eta ahaltsuagoa da.

Sarrera

- **Datuen partizionatze bertikala**: Datu-base erlazionaletan datuak ongi definitutako errenkada eta zutabeetan antolatzen dira. Datuak partizionatzeko zutabe mailan banatzen dira taulak, taula berriak sortuz.
- **Datuen partizionatze horizontala**: MongoDB datu-baseetan, dokumentuek elkarren artean duten izaera desberdina dela-eta, ezinezkoa da zutabe mailan partizionatzea. Aldiz, bildumak dokumentu multzotan banatzen dira. Horrela, zerbitzari bakoitzak bildumaren zati bat gordeko du. Bildumak **partizionatze-gakoa**ren arabera zatitzen dira.

Partizionatze horizontala MongoDB-n

Partizionatzea egiteko zerbitzari batek baino gehiagok hartzen dute parte. Horietako bat kontrolatzailea izango da, eta gainerakoek datu-baseen bilduma zatiak izango dituzte. Kontrolatzailea arduratzen da eskaera bakoitza dagokion zerbitzariari bidaltzen, konfigurazio-zerbitzaritik jasotzen baitu zerbitzari bakoitzak dituen datuen informazioa. Horrela, aplikazioa kontrolatzailearekin konektatuko da eta hari egingo dizkio eskaera guztiak, partizionatzearen kontu guztiak erabiltzailearentzat gardenak izanik.



Irudia *The definitive guide to MongoDB* liburutik hartua.

Zerbitzari motak

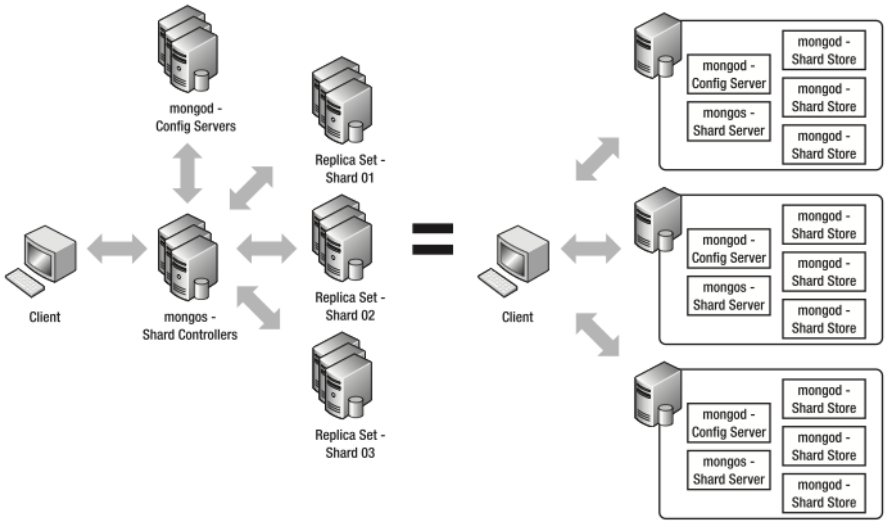
- ***Shard* zerbitzariak**: Datuak, dokumentuak, gordeta dituzten zerbitzariak. Bilduma bakoitzaren zati bat besterik ez dute edukiko.
- ***Config* zerbitzariak**: *Shard* zerbitzari bakoitzak zein bilduma-zati dituen kontrolatzen du.
- **Controller zerbitzariak**: Aplikazioa zerbitzari hauekin soilik komunikatuko da. Eskaerak hauei egingo zaizkie, eta hauek, *config* zerbitzariei informazioa eskatuz, datuak jasoko dituzte dagozkien shard zerbitzarietatik.

Arkitektura

2.8 MongoDB: Partizionatze horizontala (sharding)

Aurreko atalean ikusitako erreplikazioa eta partizionatze horizontala konbinatuz, arkitektura ahaltsuak eraiki daitezke, ondoko irudian ikus daitekeenez.

Makina fisiko bakoitzak ez du zertan prozesu bakarra izan. Makina bakoitzean hainbat prozesu elkartuz, topologia konplexuak eraiki ditzakegu makina fisiko gutxitan.



Irudia *The definitive guide to MongoDB* liburutik hartua.

Datu-base bat partizionatzen

Hauek dira ezagutu beharreko komandoak eta funtzioak:

1. Konfigurazio zerbitzari bat martxan jartzeko:

```
$ mongod --configsvr
```

2. Partizionatzearen zerbitzari kontrolatzaile bat martxan jartzeko:

```
$ mongos --configdb confighost:configport
```

3. *Shard* zerbitzari bat gehitzeko:

```
$ mongod
$ mongo kontrolhost:kontrolport
> sh.addShard("host:port")
```

4. "db" datu-basearen "bild" bilduma "_id" eremuaren arabera partizionatzeko:

```
> sh.enableSharding("db")
> sh.shardCollection("db.bild", {"_id": 1})
```

Kredituak eta Baimenak

Egilea: Zuhaitz Beloki

Data: 2014ko uztailaren 1a.

Baimena: Creative Commons Aitortu-PartekatuBerdin 3.0

Oharra: material hau 'NoSQL datu-baseak. MongoDB.' ikastaroko ikasleen esku jartzen da Creative Commons Aitortu-PartekatuBerdin 3.0 lizentziarekin. Lizentzia honekin edukia kopiatu, banatu eta erakutsi ahal izango dituzu, ondorengo baldintzak beteaz:

- Edukiaren jatorrizko egilea aipatu behar duzu.
- Lanaren kopia zein banaketa askea da.
- Lan eratorriak, jatorrizko egiletza aitortzeaz gainera, baimen (lizentzia) berdina izan beharko du.