Linux and SQL

Date: 03-23-2017

Abstract: This document provides information for interview purpose so the content is selective. Linux knowledge is not for system administration or basic usage of the windowing system or programming toolsor software. It is about system build-in commands and scripting capability.

Chapter 1 Linux Knowledge

1. Basic command checklist

Directory: ～，cd, rm, mkdir,mv, ls, cp, cat, chown, chgrp,chmod, ln

Process: **ps, kill**, at, bg,fg, crontab, nice, time, **id, whoami, who, what,** which,df, du,nohup

Text: touch, vi, wc, tr, split, cut, sort, join, file, head, tail, tee, uname

Permission: ulimit, umask,

Scripting: **alias, unalias,** sh, **sed, awk**, pr, sleep, test, read, diff, dirname, uniq, basename, nl, less, more, echo, printf

Searching: **find, grep, exec, xargs**

Network: netcat

Documentation: man

Software Development: ar, make, nm, yacc, lex, strip, ctags

System administration: mount, umount , lsof...etc

1. Disk and partition (24-30)

IDE:

SCSI:

Logical partition:

Extended partion:

Install linux to logical partition? (21)

1. Runlevel

3: multiuser without GUI

5: multiuser with GUI

(TODO: how to switch runlevel when login , where to set runlevel when boot in)

4/ Firewall (less important)

/etc/iptables

(TODO:)

1. Startup scripts

/etc/init.d/rc1.d, rc2.d...

/etc/rc.d/init.d

Redhat (fedora)

Canonical LTD (ubuntu)

iso, sun virtual box

1. Networking

34, 35, ifconfig

7. How DNS works

8. Profile files

/etc/profile

~/.bash\_profile

~/.bash\_rc

9/ /etc/passwd

/etc/shadow

/etc/fstab

(Need to write a little bit)

10.Server config

Ssh: /etc/ssh/sshd\_config

/etc/hosts.allow, /etc/hosts.deny

(TODO: write a little bit about it)

11. Security

12. Text process

How to kill one process with a name called java in one line?

A: kill -9 `ps axuw|grep java| grep -v grep| awk ‘{print $2}’`

How to kill all the processes with a name in one line? For example, kernels processes.

A:

*for pid in $(ps -ef | grep "some search" | awk '{print $2}'); do kill -9 $pid; done*

There are ways to make that more efficient,

*for pid in $(ps -ef | awk '/some search/ {print $2}'); do kill -9 $pid; done*

How to find out a file named “hello\*.txt”in all the directory underneath?

A: *find ~ -name “hello\*.txt”*

For example,

cd ~,

touch hello.txt; touch hello1.txt; touch hello2.txt

find ~ -name “hello\*.txt”

How to delete a file in all places with a name hello\*.txt in above example?

How to find out all the file names that contains “hello world”?

How to delete all the files that contain “hello world”?

grep “hello world” hello1.txt

(will show it contains or not, grep needs to specify a pattenr and files, note files can be empty or directory)

Find . -type f -exec grep “hello world” {} \;

(will show it in all the files found, but it is not these files, it just displays)

In order to show files, do this,

grep -rnw '/path/to/somewhere/' -e "pattern"

* -r or -R is recursive,
* -n is line number, and
* -w stands for match the whole word.
* -l (lower-case L) can be added to just give the file name of matching files.

Along with these, --exclude, --include, --exclude-dir or --include-dir flags could be used for efficient searching:

* This will only search through those files which have .c or .h extensions:

grep --include=\\*.{c,h} -rnw '/path/to/somewhere/' -e "pattern"

* This will exclude searching all the files ending with .o extension:

grep --exclude=\*.o -rnw '/path/to/somewhere/' -e "pattern"

Just like exclude files, it's possible to exclude/include directories through --exclude-dir and --include-dir parameter. For example, this will exclude the dirs dir1/, dir2/ and all of them matching \*.dst/:

grep --exclude-dir={dir1,dir2,\*.dst} -rnw '/path/to/somewhere/' -e "pattern"

* This works very well for me, to achieve almost the same purpose like yours.

How to show a pid and ppid of a process?

A: To see every process in standard way, ps -ef or ps -ely,

To see every process in bsd way, ps aux

12 VIM

How to replace a text once?

How to replace a text in all places?

How to go to the end or front of a large file.

How to go forward a page and backward a page?

How to copy and paste?

How to delete a line?

Chapter 2 SQL Knowledge

2.1 Terms

What is primary key?

A column or multiple columns that can be used to identify an unique record/row.

Primary key can not be null, and can only have one primary key set for a table.

Primary key is always indexed by default.

What is a unique key?

The value of this column is unique. The value can be null, once null it is not unique.

A table can have many unique keys.

What is foreign key?

A column in this table but it is a primary key in another table.

Foreign key can be use in combination with other key as a primary key.

What is a key?

Key is a column used in concept with primary key and foreign key.

What is a index?

Index is some extra bit in the data structure of b/b+ tree of a column. This is to make the column searched faster.

\* Index makes where and groupby faster but make insert and udpate slower.

\* Index makes joins faster.

\* Index makes variable length column slower.

What is the difference between index and key?

Key is a column. Index is an extra bit in B/B+ tree.

What is group by?

The GROUP BY statement is often used with aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the result-set by one or more columns.

Syntax:

*SELECT column\_name(s)  
FROM table\_name  
WHERE condition  
GROUP BY column\_name(s)  
ORDER BY column\_name(s);*

For example, the following will show the number of customers in each country:

*Eg:*

*SELECT COUNT(CustomerID), Country  
FROM Customers  
GROUP BY Country;*

The above can be rewrite like this,so NUM is the column name,

*SELECT COUNT(CustomerID) as NUM, Country*

*FROM Customers*

*GROUP BY Country;*

The following SQL statement lists the number of customers in each country, sorted high to low:

Eg:

*SELECT COUNT(CustomerID), Country  
FROM Customers  
GROUP BY Country  
ORDER BY COUNT(CustomerID) DESC;*

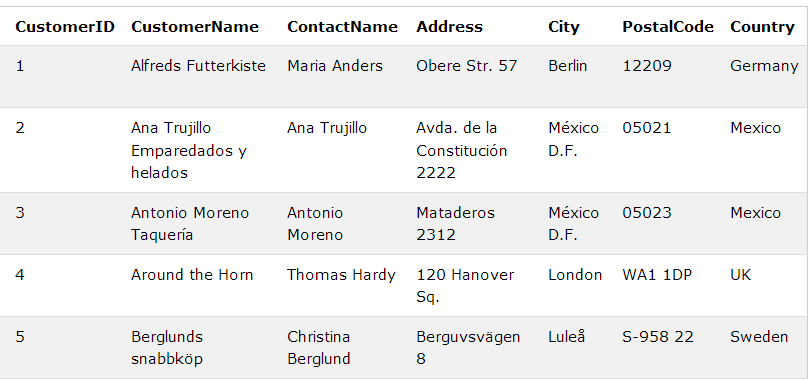
What is having? (ING, think about it)

The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions.

Syntax

*SELECT column\_name(s)  
FROM table\_name  
WHERE condition  
GROUP BY column\_name(s)  
HAVING condition  
ORDER BY column\_name(s);*

Example1,



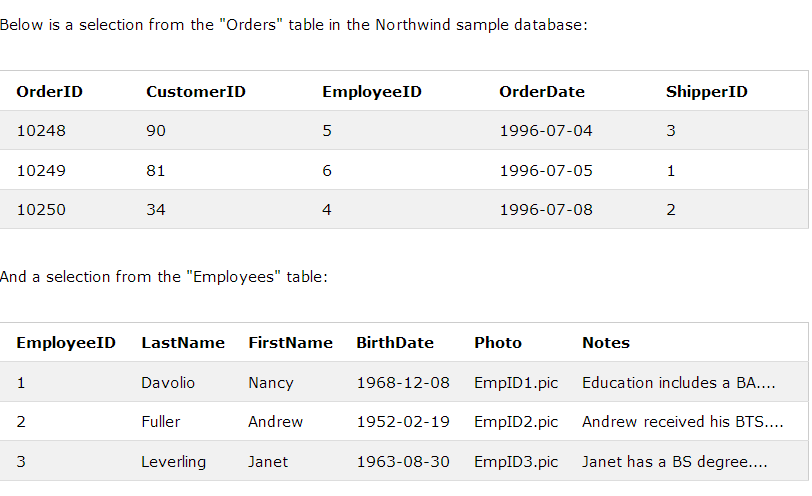
lists the number of customers in each country, sorted high to low (Only include countries with more than 5 customers):

*SELECT COUNT(CustomerID), Country  
FROM Customers  
GROUP BY Country  
HAVING COUNT(CustomerID) > 5;*

*,*

*SELECT COUNT(CustomerID), Country  
FROM Customers  
GROUP BY Country  
HAVING COUNT(CustomerID) > 5  
ORDER BY COUNT(CustomerID) DESC;*

Example 2



lists the employees that have registered more than 10 orders

*SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders  
FROM (Orders  
INNER JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID)  
GROUP BY LastName  
HAVING COUNT(Orders.OrderID) > 10;*

lists if the employees "Davolio" or "Fuller" have registered more than 25 orders:

*SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders  
FROM Orders  
INNER JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID  
WHERE LastName = 'Davolio' OR LastName = 'Fuller'  
GROUP BY LastName  
HAVING COUNT(Orders.OrderID) > 25;*

2.2 Optimization

2.2.1 How to optimiaze code with database as storage?

1. Less DB call: Code optimaztion, more oo design, save queried result in object and reduce dao.db call everywhere.
2. Faster query: Add index for frequently searched column, make query faster.

3. Less join: To many join, relation complex, use big flat table, reduce join.

4. Network, bandwidth, switch...etc.

5. Architecture, distributed database.

6. Load balancing: DB shading, keep certain amount of records in one db, and move other records to different db or physical location.

2.2.2 What is sharding?

Sharding is a type of database [partitioning](http://searchstorage.techtarget.com/definition/partition) that separates very large databases the into smaller, faster, more easily managed parts called data shards. The word shard means a small part of a whole.

Technically, sharding is a synonym for **horizontal partitioning**. In practice, the term is often used to refer to any database partitioning that is meant to make a very large database more manageable.

The governing concept behind sharding is based on the idea that as the size of a database and the number of transactions per unit of time made on the database increase linearly, the response time for querying the database increases exponentially.

Additionally, the costs of creating and maintaining a very large database in one place can increase exponentially because the database will require high-end computers. In contrast, data shards can be distributed across a number of much less expensive commodity servers. Data shards have comparatively little restriction as far as hardware and software requirements are concerned.

Sharding strategy

1. Geographically

Customers located on the East Coast can be placed on one server, while customers on the West Coast can be placed on a second  server. Assuming there are no customers with multiple locations, the split is easy to maintain and build rules around.

2. Normal horizontal partition

Horizontal partitioning is a design principle whereby rows of a database table are held separately, rather than splitting by columns (as for normalization). Each partition forms part of a shard, which may in turn be located on a separate database server or physical location. The advantage is the number of rows in each table is reduced (this reduces index size, thus improves search performance). If the sharding is based on some real-world aspect of the data (e.g. European customers vs. American customers) then it may be possible to infer the appropriate shard membership easily and automatically, and query only the relevant shard.

This will not break business logic or MVC. The work of determining the correct shard where to store the data would be transparently done by your data access layer.

Long story short: Sharding is basically the process of distributing tables onto different servers in order to balance the load onto both equally.

2.2.3 Database partition

1/ Vertical partition

Vertical partitioning is very domain specific. You draw a logical split within your application data, storing them in different databases. It is almost always implemented at the **application level** — a piece of code routing reads and writes to a designated database.

**Vertically partition by functionality.** Binary blobs tend to occupy large amounts of space and are isolated within your application. Storing files in S3 can reduce storage burden. Other functionalities such as full text search, tagging, and analytics are best done by separate databases.

2/ Horizontal partition

In contrast, sharding splits a homogeneous type of data into multiple databases. You can see that such an algorithm is easily generalizable. That’s why sharding can be implemented at either the application or **database level**. In many databases, sharding is a first-class concept, and the database knows how to store and retrieve data within a cluster. Almost all modern databases are natively sharded. Cassandra, HBase, HDFS, and MongoDB are popular distributed databases. Notable examples of non-sharded modern databases are [Sqlite](http://www.sqlite.org/" \t "https://medium.com/@jeeyoungk/_blank), Redis ([spec in progress](http://redis.io/topics/cluster-spec" \t "https://medium.com/@jeeyoungk/_blank)), [Memcached](http://memcached.org/" \t "https://medium.com/@jeeyoungk/_blank), and [Zookeeper](http://zookeeper.apache.org" \t "https://medium.com/@jeeyoungk/_blank).

There are pros and cons of sharding,

Cons:

* Cross-partition operations: inefficient, also redistribution is a problem.
* Resharding is hard
* Hotspots: uneven distribution of data and operations. How to balance them?
* Sharding adds additional [programming and operational complexity](http://www.percona.com/blog/2009/08/06/why-you-dont-want-to-shard/" \t "https://medium.com/@jeeyoungk/_blank) to your application.

2.2.4 How sharding works?

Shard or Partition Key is a portion of primary key which determines how data should be distributed. A partition key allows you to retrieve and modify data efficiently by routing operations to the correct database. Entries with the same partition key are stored in the same node. A logical shard is a collection of data sharing the same partition key. A database node, sometimes referred as a physical shard, contains multiple logical shards.

Summary:

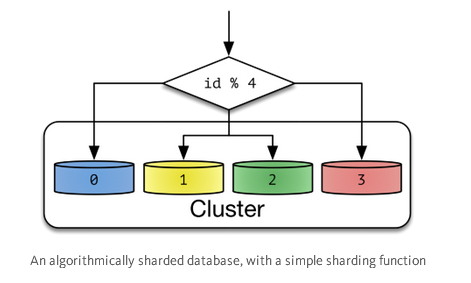
Shard: partition key to route to different db.

Logical shard: data sharing the same partition key.

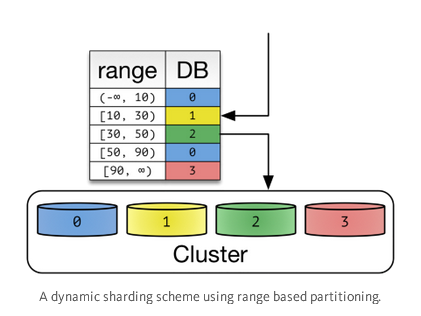
Physical shard: a db node, contains multiple logical shards.

**[Algorithmic](http://blog.clustrix.com/2013/01/17/sharding-in-theory-and-practice-part-two/" \t "https://medium.com/@jeeyoungk/_blank) sharding:** use a sharding function (partition\_key) -> database\_id to locate data. “hash(key) % NUM\_DB” will get the id. Resharding is hard, as the resharding function and all the data has to be redone again.

Cons: sharding function is very important



**Dynamic sharding:** an external **locator service** determines the location of entries. It can be implemented in multiple ways. If the cardinality of partition keys is relatively low, the locator can be assigned per individual key. Otherwise, a single locator can address a range of partition keys.In the example of range-based partition keys, range queries are efficient because the locator service reduces the number of candidate databases.



Cons:

Dynamic sharding is more resilient to nonuniform distribution of data. Locators can be created, split, and reassigned to redistribute data. However, relocation of data and update of locators need to be done in unison. This process has many corner cases with a lot of interesting theoretical, operational, and implementational challenges.

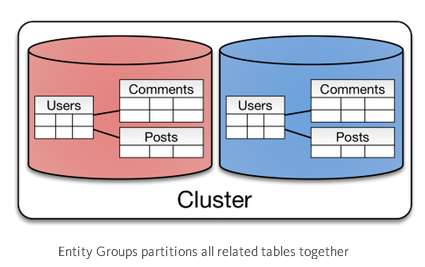
The locator service becomes a single point of contention and failure. Every database operation needs to access it, thus performance and availability are a must. However, locators cannot be cached or replicated simply. Out of date locators will route operations to incorrect databases. Misrouted writes are especially bad — they become undiscoverable after the routing issue is resolved.

Since the effect of misrouted traffic is so devastating, many systems opt for a high consistency solution. Consensus algorithms and synchronous replications are used to store this data. Fortunately, locator data tends to be small, so computational costs associated with such a heavyweight solution tends to be low.

Good news is that dynamic sharding is widely used in Hadoop HBase and mongoDB.

Apache HBase splits row keys into ranges. The range server is responsible for storing multiple regions. Region information is stored in Zookeeper to ensure consistency and redundancy. In MongoDB, the [ConfigServer](http://docs.mongodb.org/manual/core/sharded-cluster-config-servers/" \l "sharding-config-server" \t "https://medium.com/@jeeyoungk/_blank) stores the sharding information, and mongos performs the query routing. ConfigServer uses synchronous replication to ensure consistency. When a config server loses redundancy, it goes into read-only mode for safety. Normal database operations are unaffected, but shards cannot be created or moved.

### **Entity Groups**



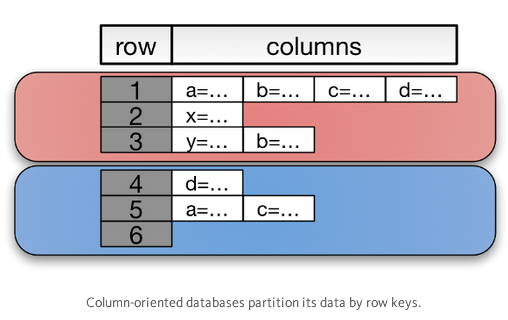
The concept of entity groups is very simple. Store related entities in the same partition to provide additional capabilities within a single partition. Specifically:

1. Queries within a single physical shard are efficient.
2. Stronger consistency semantics can be achieved within a shard.

Entity groups can be implemented either algorithmically or dynamically. They are usually implemented dynamically since the total size per group can vary greatly. The same caveats for updating locators and moving data around applies here. Instead of individual tables, an entire entity group needs to be moved together.

Other than sharded RDBMS solutions, Google Megastore is an example of such a system. Megastore is publicly exposed via Google App Engine’s [Datastore API](https://cloud.google.com/appengine/docs/python/datastore/entities" \t "https://medium.com/@jeeyoungk/_blank).

### **Hierarchical keys & Column-Oriented Databases**



Column-oriented databases are an extension of key-value stores. They add expressiveness of entity groups with a **hierarchical primary key**. A primary key is composed of a pair (row key, column key). Entries with the same partition key are stored together. Range queries on columns limited to a single partition are efficient. That’s why a column key is referred as a range key in DynamoDB. Column-oriented databases can model a problem such as [time series](http://planetcassandra.org/getting-started-with-time-series-data-modeling/" \t "https://medium.com/@jeeyoungk/_blank) efficiently. The term column database is losing popularity.

Ref:

https://medium.com/@jeeyoungk/how-sharding-works-b4dec46b3f6

http://www.agildata.com/database-sharding/

Chapter 3 DBMS

3.1. MySQL and MariaDB (free open source version MySQL since MySQL has commercial version.)

3.1 Creating table, primary key and foreign key

3.1.0 Data types

Text types: Tinytext, Text, Mediumtext, Longtext. Db2 and Oracle use clob.

Numeric types: Tinyint, Smallint, Mediumint, Int, Bigint, Float, Double

Temporal Data: This is information about dates and times such as Date,Datetime, Timestamp, year.

Blob vs text: there are four blob types and four text types, there are corresponding to each other.

They are: TINYTEXT, TEXT, MEDIUMTEXT, and LONGTEXT, and TINYBLOB, BLOB, MEDIUMBLOB, and LONGBLOB. BLOB values are treated as binary strings (byte strings). They have the binary character set and collation, and comparison and sorting are based on the numeric values of the bytes in column values.In most respects, you can regard a BLOB column as a [VARBINARY](https://dev.mysql.com/doc/refman/5.7/en/binary-varbinary.html" \o "12.4.2 The BINARY and VARBINARY Types) column that can be as large as you like. Similarly, you can regard a TEXT column as a [VARCHAR](https://dev.mysql.com/doc/refman/5.7/en/char.html" \o "12.4.1 The CHAR and VARCHAR Types) column.

* For indexes on BLOB and TEXT columns, you must specify an index prefix length. For [CHAR](https://dev.mysql.com/doc/refman/5.7/en/char.html" \o "12.4.1 The CHAR and VARCHAR Types) and [VARCHAR](https://dev.mysql.com/doc/refman/5.7/en/char.html" \o "12.4.1 The CHAR and VARCHAR Types), a prefix length is optional.
* BLOB and TEXT columns cannot have DEFAULT values.

Char vs varchar: char is fixed size and max at 255 and varchar is variable size. Char is static memeory allocation, and is 50% faster. Varchar is dyanmic memory allocation and supports 65535 characters after MySQL 5.0.3. If your content is fixed size and will not exceed some length limit, then char will get better performance.

**3.1.1 Creating**

Example1,

*create table person (  
person\_id SMALLINT UNSIGNED,  
frame varchar(20),  
lname varchar(20),  
gender ENUM('M','F'),  
birth\_date DATE,  
address varchar(30),  
city varchar(20),  
state varchar(20),  
country varchar(20),  
postal\_code varchar(20),  
constraint pk\_person primary key (person\_id)  
);*

To check the table,

*desc person* or *describe person*

Example2,

*create table favorite\_food(  
person\_id smallint unsigned,  
food varchar(20),  
constraint pk\_favorite\_food primary key(person\_id, food),  
constraint fk\_person\_id foreign key(person\_id) references person (person\_id)  
);*

**3.1.2 nonexistent foreign key**

*insert into faverite\_food(person\_id, food) values('999', 'lasa')*

**3.1.3 show warnings**,

*show warnings;*

eg:

*update person set gender = 'z' where person\_id = 1;*

*show warnings;*

eg:

*update person set birth\_date = 'DEC-21-1980' where person\_id = 1;*

*show warnings;*

**3.1.4 Column Aliases**

eg:

*select version() version, user() user, database() databasexxxx;*

*select version() version,'Active', user() user, database() databasexx;*

**3.1.5 Subquery-generated tables**

*SELECT e.emp\_id, e.fname, e.lname FROM (select emp\_id, fname, lname, start\_date, title FROM employee) e;*

*SELECT account\_id, product\_id, cust\_id, avail\_balance FROM account WHERE product\_cd IN ('CHK', 'SAV', 'CD', 'MM')*

*SELECT account\_id, product\_id, cust\_id, avail\_balance FROM account WHERE product\_cd IN (SELECT product\_cd from product WHERE product\_type\_cd = 'ACCOUNT')*

**3.2 View**

*create view employee\_vw AS select emp\_id, fname, lname, YEAR(start\_date) start\_year from employee;*

*select emp\_id, start\_year from employee\_vw;*

**3.3 Table Alias**

*select employee.emp\_id, employee.fname, employee.lname, department.name dept\_name from employee INNER JOIN department ON employee.dept\_id = department.dept\_id;*

*select e.empy\_id, e.fname, e.lname, d.name dept\_name from employee e INNER JOIN department d ON e.dept\_id = d.dept\_id;*

**3.4 Sorting Order**

eg:

*select account\_id, product\_cd, open\_date, avail\_balance from account order by avail\_balance DESC*

The following sort by the last 3 digits from right,

*select cust\_id, cust\_type\_cd, city, state, fed\_id from customer ORDER BY RIGHT(fed\_id,3).*

The following sort by the 2,5th column,

*select emp\_id, title, start\_date, fname, lname from employee order by 2,5;*

**3.6 Filtering**

**Where**

eg:

*where end\_date IS NULL AND (title = 'Teller' OR start\_date < '2003-01-01')*

*where end\_date IS NULL AND NOT (title = 'Teller' OR start\_date < '2003-01-01')*

***NOT IN***

*… WHERE product\_cd NOT IN ('CHK', 'SAV', 'CD', 'MM')*

**Matching**

*… WHERE lname LIKE '\_a%e%'*

**Range**

*… WHERE start\_date BETWEEN '2003-01-01' AND '2001-01-01'*

*… WHERE start\_date >= '2003-01-01' AND start\_date <= '2001-01-01'*

*… WHERE fed\_id LIKE '\_\_\_-\_\_-\_\_\_\_'*

\_ is one letter, % is any number of letter including zero.

**Regular Expression**

*… WHERE lname REGEXP '^[FG]'*

**NULL**

*… WHERE superior\_emp\_id IS NULL;*

*… WHERE superior\_emp\_id = NULL*

**JOIN [ON]**

CROSS JOIN (cartesian product, permutations of different table records, rarely used)

eg:

SELECT e.fname, e.lname, d.name FROM employee e JOIN department d

INNER JOIN (most commonly used)

eg:

*SELECT e.fname, e.lname, d.name FROM employee e JOIN department d ON e.dept\_id = d.dept\_id;*

or the following, if you don't specify INNER, it is INNER by default,

*SELECT e.fname, e.lname, d.name FROM employee e INNER JOIN department d ON e.dept\_id = d.dept\_id;*

or the following to use “using” subclause than the “on” subclause.

*SELECT e.fname, e.lname, d.name FROM employee e INNER JOIN department d USING(dept\_id);*

**3.7 character set (TODO)**

show character set;

eg: *create database foreign\_sales character SET utf8*

***3.8 MySQL backup***

*Back entire db, -d means not to output the data.*

*C:\Users\linfeng>mysqldump -d -h localhost -uroot -p123456 words > dumpfile.sql*

*If you want to backup the entire db with data, without -d*

*C:\Users\linfeng>mysqldump -h localhost -uroot -p123456 words > dumpfile.sql*

3.2. HBase

Distributed database used by hadoop and hdfs. Please check other documentation to fill out this part.

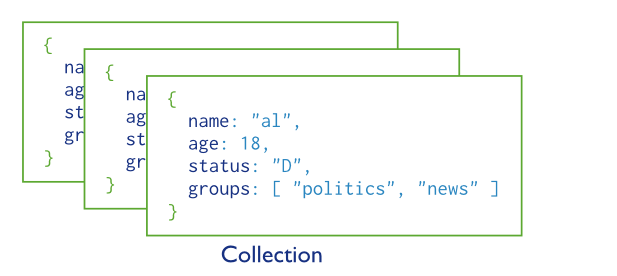
Chapter 4 NoSQL

4.1 MongoDB

MongoDB stores all documents in [collections](https://docs.mongodb.org/manual/reference/glossary/" \l "term-collection). A collection is a group of related documents that have a set of shared common indexes. Collections are analogous to a table in relational databases and each record is implemented in a way called BSON, binary json.

Collection: table

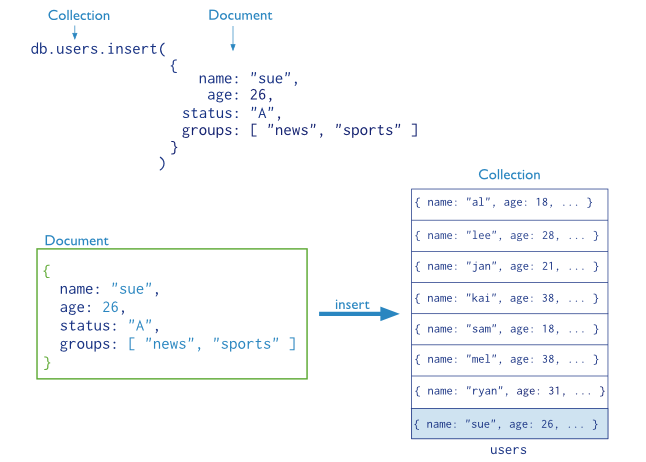
Document: record (BSON)



BSON



Architecture,



4.1.1 Database Binary:

Mongod: server

Mongo: client

4.1.2 Start server in windows:

1. Automatic creation of windows service

"C:\mongodb\bin\mongod.exe" --config "C:\mongodb\mongod.cfg" --install

"C:\mongodb\bin\mongod.exe" --remove

Net start MongoDB

Net stop MongoDB

4.2 Usage and syntax

1/ create collection

*C:\Users\linfeng>mongo*

*2017-03-24T15:47:53.437-0400 I CONTROL [main] Hotfix KB2731284 or later update*

*is not installed, will zero-out data files*

*MongoDB shell version: 3.2.4*

*connecting to: test*

*>*

The following creates table and insert a record,

**CREATE** **TABLE** users (

id MEDIUMINT **NOT** **NULL**

AUTO\_INCREMENT,

user\_id Varchar(30),

age Number,

status char(1),

**PRIMARY** **KEY** (id))

**db**.**users**.**insert**( { **user\_id**: "abc123", **age**: 55, **status**: "A" } )

Output:

*connecting to: test*

*>*

*> CREATE TABLE users (*

*... id MEDIUMINT NOT NULL*

*... AUTO\_INCREMENT,*

*... user\_id Varchar(30),*

*... age Number,*

*... status char(1),*

*... PRIMARY KEY (id))*

*2017-03-24T15:49:53.726-0400 E QUERY [thread1] SyntaxError: missing ; before*

*statement @(shell):1:7*

*> db.users.insert( { user\_id: "abc123", age: 55, status: "A" } )*

*WriteResult({ "nInserted" : 1 })*

*>*

*When you do this, it warns you that the collection already existed.*

*> db.createCollection("users")*

*{ "ok" : 0, "errmsg" : "collection already exists", "code" : 48 }*

*The above sql creates not only a table similar to the collection.*

*If you don’t specify a primary, an auto id will be set as primary key.*

2/ ALTER TABLE users ADD join\_date DATETIME

Collections do not describe or enforce the structure of its documents; i.e. there is no structural alteration at the collection level.

However, at the document level, [update()](https://docs.mongodb.org/manual/reference/method/db.collection.update/" \l "db.collection.update" \o "db.collection.update()) operations can add fields to existing documents using the [$set](https://docs.mongodb.org/manual/reference/operator/update/set/" \l "up._S_set" \o "$set) operator.

db.users.update(

{ },

{ $set: { join\_date: new Date() } },

{ multi: true })

3/ ALTER TABLE users DROP COLUMN join\_date

Collections do not describe or enforce the structure of its documents; i.e. there is no structural alteration at the collection level.

However, at the document level, [update()](https://docs.mongodb.org/manual/reference/method/db.collection.update/" \l "db.collection.update" \o "db.collection.update()) operations can remove fields from documents using the [$unset](https://docs.mongodb.org/manual/reference/operator/update/unset/" \l "up._S_unset" \o "$unset) operator.

db.users.update(

{ },

{ $unset: { join\_date: "" } },

{ multi: true }

)

4/ DROP TABLE users

db.users.drop()

5/ **CREATE** **INDEX** idx\_user\_id\_asc**ON** users(user\_id)

**db**.**users**.**createIndex**( { **user\_id**: 1 } )

**6/ CREATE** **INDEX**

idx\_user\_id\_asc\_age\_desc**ON** users(user\_id, age **DESC**)

**db**.**users**.**createIndex**( { **user\_id**: 1, **age**: -1 } )

7/ delete records

A. **DELETE** **FROM** users**WHERE** status = "D"

**db**.**users**.**remove**( { **status**: "D" } )

B. **DELETE** **FROM** users

**db**.**users**.**remove**({})

8/ update records

A. **UPDATE** users **SET** status = "C" **WHERE** age > 25

**db**.**users**.**update**( { **age**: { **$gt**: 25 } }, { **$set**: { **status**: "C" } }, { **multi**: **true** })

B. **UPDATE** users **SET** age = age + 3 **WHERE** status = "A"

**db**.**users**.**update**( { **status**: "A" } , { **$inc**: { **age**: 3 } }, { **multi**: **true** })

9/ Select

<https://docs.mongodb.org/manual/reference/sql-comparison/>

4.3 Import data

C:\Program Files\MongoDB\dataset.jason

Ref: <https://docs.mongodb.org/getting-started/java/import-data/>

import sample database to Mongodb

Eg:

C:\Users\linfeng>mongoimport --db test --collection restaurants --drop --file "c

:\Program Files\MongoDB\dataset.json"

2016-03-25T00:11:24.643-0500 connected to: localhost

2016-03-25T00:11:24.646-0500 dropping: test.restaurants

2016-03-25T00:11:25.884-0500 imported 25359 documents

4.4 java driver and bson jar

mongo-java-driver-3.0.4.jar and bson-3.0.4.jar in C:\Program Files\MongoDB.

4.5 connect to mongodb and insert data

Import com.mongodb.MongoClient;

Import com.mongodb.client.MongoDatabase;

import java.text.DateFormat;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Locale;

import static java.util.Arrays.asList;

MongoClient mongoClient = new MongoClient();

MongoDatabase db = mongoClient.getDatabase("test");

DateFormat format = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss'Z'", Locale.ENGLISH);db.getCollection("restaurants").insertOne(

new Document("address",

new Document()

.append("street", "2 Avenue")

.append("zipcode", "10075")

.append("building", "1480")

.append("coord", asList(-73.9557413, 40.7720266)))

.append("borough", "Manhattan")

.append("cuisine", "Italian")

.append("grades", asList(

new Document()

.append("date", format.parse("2014-10-01T00:00:00Z"))

.append("grade", "A")

.append("score", 11),

new Document()

.append("date", format.parse("2014-01-16T00:00:00Z"))

.append("grade", "B")

.append("score", 17)))

.append("name", "Vella")

.append("restaurant\_id", "41704620"));

4.6 find and query

import org.bson.Document;

import com.mongodb.Block;

import com.mongodb.client.FindIterable;

import static com.mongodb.client.model.Filters.\*;

import static com.mongodb.client.model.Sorts.ascending;

import static java.util.Arrays.asList;

Query for All Documents in a Collection

FindIterable<Document> iterable = db.getCollection("restaurants").find();

iterable.forEach(new Block<Document>() {

@Override

public void apply(final Document document) {

System.out.println(document);

}});

Specify Equality Conditions

If the <field> is in an embedded document or an array, use [dot notation](https://docs.mongodb.org/manual/reference/glossary/" \l "term-dot-notation" \o "(in mongodb-manual v3.2)) to access the field.

To help specify the query condition, the Java driver also provides the [Filters](http://api.mongodb.org/java/3.0/com/mongodb/client/model/Filters.html) class. The class contains various static methods to simplify building the query predicates, including the [eq](http://api.mongodb.org/java/3.0/com/mongodb/client/model/Filters.html" \l "eq-java.lang.String-TItem-) method:

eq(<field>, <value>)

**Query by a Top Level Field**

FindIterable<Document> iterable = db.getCollection("restaurants").find(

new Document("borough", "Manhattan"));

Iterate the results and apply a block to each resulting document.

iterable.forEach(new Block<Document>() {

@Override

public void apply(final Document document) {

System.out.println(document);

}});

Using the static [Filters](http://api.mongodb.org/java/3.0/com/mongodb/client/model/Filters.html) helper(s), you can also specify the query as follows:

db.getCollection("restaurants").find(eq("borough", "Manhattan"));

More:

<https://docs.mongodb.org/getting-started/java/query/>

4.7 update data

4.8 remove data

4.9 data aggregation

4.10 FAQ

1. MongoDB does not support SQL
2. MongoDB does not support transaction
3. MongoDb handles cache
4. To list a collection’s indexes, use the [db.collection.getIndexes()](https://docs.mongodb.org/manual/reference/method/db.collection.getIndexes/" \l "db.collection.getIndexes" \o "db.collection.getIndexes()) method.
5. > db.users.getIndexes()

[

{

"v" : 1,

"key" : {

"\_id" : 1

},

"name" : "\_id\_",

"ns" : "test.users"

}

]

6. How can I see the size of an index?

The [db.collection.stats()](https://docs.mongodb.org/manual/reference/method/db.collection.stats/" \l "db.collection.stats" \o "db.collection.stats()) includes an [indexSizes](https://docs.mongodb.org/manual/reference/command/collStats/" \l "collStats.indexSizes" \o "collStats.indexSizes) document which provides size information for each index on the collection.

Depending on its size, an index may not fit into RAM. An index fits into RAM when your server has enough RAM available for both the index and the rest of the [working set](https://docs.mongodb.org/manual/reference/glossary/" \l "term-working-set). When an index is too large to fit into RAM, MongoDB must read the index from disk, which is a much slower operation than reading from RAM.

Cassendra

Chapter 5 New DBMS

OrientDB

InfluxDB (graph db)