|  |  |
| --- | --- |
| **MONGODB** | **MYSQL** |
| Semi-structured Database //////un🡪video,image | Structured Database |
| Multi model—can store structured as well as unstructured | Single Model |
| Schema dynamic | Schema static |
| Cannot handle complex queries  Database have multiple nodes known as shards | Easily handle complex queries |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| DB//Show current DB show dbs ///all databases except blank database | SHOW DATABASES//show all databases |
| Use Db\_name//if database not present create new and switch | Use Db\_name//Not create New One and Create Database db\_name is used to create database |
| Table | Collections |
| Row | Documents |
| Columns | Field |
| Inserting one row in a table  Db.Collection\_name.insert({Document})  e.g.==>db.Studentds.insert({name:”manvendra”});  //if Collection or table is not present it will create | Insert into table\_name(column) values(record)  Insert into(name) Students values(“manvendra”)  If table is not there throw error |
| Db.dropdatabase()//drop current database | Drop database database\_name |
| Show collections | Show tables |
| Db.createCollection(collection\_name,<option>)//to create table or Collection  Options can be🡺{capped:true}=means fixed size  Size🡺to give size  {autoindexid:true}🡺 equivalent to autoincreament in sql  Max🡺for maximum no of documents allowed  db.createCollection**(**"XYZ"**,{**size**:**10**,**max**:**100**})** | Create table table\_name(column\_name column\_datatype); |
| Db.collection\_name.drop()//delete table  e.g.=>db.Students.drop() | Drop table table\_name  Drop table students |
| Db.collection\_name.insert([{--------},{======}]);//multiple  e.g.=>db.SAtudents.insert([{name:”manu”},  {name:”narendra”}]) | INSERT INTO tasks(title, priority)  VALUES  ('My first task', 1),  ('It is the second task',2),  ('This is the third task of the week',3); |
| Db.students.find() | Select \* from Students; |
| Db.collection\_name.update(  {Condition},{$set:{value to be set}})  e.g.=>db.student.update({id:1},{$set{name:”xyz”}}) | UPDATE table\_name SET field1 = new-value1, field2 = new-value2  [WHERE Clause]  e.g.=>Update Students set name=”manu” where uid=? |
| Only changes the first record having that uid  To change all documents with same uid  We need a extra bracket having multi=>true  db.student.update({id:1},{$set:{name:”xyz”}},{“multi”:”true”})  or we can use .updateMany()  e.g.=> db.student.updateMany({id:1},{$mul:{salary:1.02}})  we can use $mul(multiply)  $currentDate🡺 to set current date in any field  **The following operation updates the lastModified field to the current date, the "cancellation.date" field to the current timestamp as well as updating the status field to "D" and the "cancellation.reason" to "user request".**  **db.users.update(**  **{ \_id: 1 },//multiple updating**  **{**  **$currentDate: {**  **lastModified: true,**  **"cancellation.date": { $type: "timestamp" }**  **},**  **$set: {**  **status: "D",**  **"cancellation.reason": "user request"**  **}**  **}**  **)**  **The updated document would resemble:{**  **"\_id" : 1,**  **"status" : "D",**  **"lastModified" : ISODate("2014-09-17T23:25:56.314Z"),**  **"cancellation" : {**  **"date" : Timestamp(1410996356, 1),**  **"reason" : "user request"**  **}**  **}**  $inc🡺takes+ and –ve and perform incr/decre on field  $min🡺**db.scores.update( { \_id: 1 }, { $min: { lowScore: 150 } } ) check if value of lowscore is greater than 150 update**  **$rename🡺db.students.update( { \_id: 1 }, { $rename: { "name.first": "name.fname" } } )**  **// rename the field name**  **{upsert:true}//if update check condition if it does not find it instead of creating a error it will insert a new Document**  $setoninsert🡺**db.products.update(**  **{ \_id: 1 },**  **{**  **$set: { item: "apple" },**  **$setOnInsert: { defaultQty: 100 }**  **},**  **{ upsert: true }**  **)**  //if we r updating and it is inserting new document(upsert:true)//than it is used  $unset🡺to remove values  db.products.update(  { sku: "unknown" },  { $unset: { quantity: "", instock: "" } }  ) //removing value of quantity and instock | Above statement change name of all students with that uid  Update table Student set salary=salary\*0.2 where id=? |
| Db.getCollection(“collection\_name”) //to get collection | NO such Command |
| Db.collection\_name.remove({condition},options)  e.g=>db.students.remove({“id”:”1233443”}),🡺deletemany()  Remember=>it deletes all with that id same like mysql  If we don’t want to delete all and some specific one  db.students.remove({“id”:”1233443”},1 or true)🡸🡺deleteOne() | DELETE FROM table\_name [WHERE Clause]  Eg=>DELETE FROM tutorials\_tbl WHERE tutorial\_id = 3 |
| Db.students.remove({}) // truncate table | TRUNCATE TABLE [database\_name.]table\_name; |
| Db.students.find({“name”:”manu”}) | Select \* from students where name=”manu” |
| Db.students.find({“name”:{$gte:”manu”}})  Gte,lte,gt,lt,ne(not equals) | Select \* from students where name>”manu” |
| Db.students.find({“name”:”manu”,”age”:”12”}) | Select \* from students where name=”manu” And age=13 |
| Db.students.find({$or:[{“name”:”manu”}{“age”:”13”}]}) | Select \* from students where name=”manu” OR age=13 |
| db.inventory.find( { status: { $in: [ "A", "D" ] } } ) | Select \* from *nventory where*  status in(“A”,”D”) |
| Db.students.find({},{“name”:1,”age”:1}) //projections | Select name,age from students |
| Db.students.find({},{“name”:1,”age”:1}).skip(2)//skip first two records | SELECT \* FROM foo LIMIT 2, n //skip first two and print others |
| Db.students.find({},{“name”:1,”age”:1}).limit(2)// | SELECT \* FROM foo LIMIT 2 |
| Db.students.find({},{“name”:1,”age”:1}).sort({name:1})// | Select \* from students orderby name; |
| Db.students.find({},{“name”:1,”age”:1}).sort({name:-1})// | Select \* from students orderby name desc; |
| db.inventory.find( { "size.uom": "in" } )  find in nested where uom =”in” |  |
| The following example queries for all documents where the field tags value is an array with exactly two elements, "red" and "blank", in the specified order:  db.inventory.find( { tags: ["red", "blank"] only |  |
| If, instead, you wish to find an array that contains both the elements "red" and "blank", without regard to order or other elements in the array, use the [$all](https://docs.mongodb.com/manual/reference/operator/query/all/#op._S_all) operator:  db.inventory.find( { tags: { $all: ["red", "blank"] } } )  Query an Array for an Element  The following example queries for all documents where tags is an array that contains the string "red" as one of its elements:  db.inventory.find( { tags: "red" } ) |  |
|  |  |
|  |  |
|  |  |
|  |  |
| Select odd/even rows |  |
| We can use for loop to insert data here | Cannot use |
| Db.students.ensureindex({“name”:1 or -1})  //create index for name in student table 1//asc -1//desc | CREATE INDEX idx1 ON t1 ((col1 + col2));  CREATE INDEX idx2 ON t1 ((col1 + col2), (col1 - col2), col1);  ALTER TABLE t1 ADD INDEX ((col1 \* 40) DESC); |
| Db.students.dropindex({“name”:1}) | DROP INDEX index\_name ON tbl\_name |
| Db.students.findone() | Select \* from students Limit 1; |
| Db.students.save()🡺update but replace old one | Not any in mysql |
| Db.collection\_\_name.count() // to count  db.manu.find( { a: 5, b: 5 } ).count() | Select count(\*) from manu where a=? and b=? |
| countDocument({query},options)// to count document on a base condition  db.Student.countDocuments( { ord\_dt: { $gt: **new** Date('01/01/2012') } }, { limit: 100 } ) | select distinct item.sku from students where dept=”A” |
| db.Student.distinct( "dept" )  db.Student.distinct( "item.sku", { dept: "A" } )// nested documents | select distinct item.sku from students where dept=”A” |
| * Another inserts method * [db.collection.update()](https://docs.mongodb.com/manual/reference/method/db.collection.update/#db.collection.update) when used with the upsert: true option. * [db.collection.updateOne()](https://docs.mongodb.com/manual/reference/method/db.collection.updateOne/#db.collection.updateOne) when used with the upsert: true option. * [db.collection.updateMany()](https://docs.mongodb.com/manual/reference/method/db.collection.updateMany/#db.collection.updateMany) when used with the upsert: true option. * [db.collection.findAndModify()](https://docs.mongodb.com/manual/reference/method/db.collection.findAndModify/#db.collection.findAndModify) when used with the upsert: true option. * [db.collection.findOneAndUpdate()](https://docs.mongodb.com/manual/reference/method/db.collection.findOneAndUpdate/#db.collection.findOneAndUpdate) when used with the upsert: true option. * [db.collection.findOneAndReplace()](https://docs.mongodb.com/manual/reference/method/db.collection.findOneAndReplace/#db.collection.findOneAndReplace) when used with the upsert: true option. * [db.collection.save()](https://docs.mongodb.com/manual/reference/method/db.collection.save/#db.collection.save). * [db.collection.bulkWrite()](https://docs.mongodb.com/manual/reference/method/db.collection.bulkWrite/#db.collection.bulkWrite). |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| **DATA MODELLING** |  |
| Array and sub types in mongoDb |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| **AGGREGRATION** |  |
|  |  |
| db.COLLECTION\_NAME.aggregate([<stage1>,<stage2>,<stagen>])  Now aggregrate function divided in more  db.Students.aggregate([{$group : {\_id : "$name", num\_tutorial : {$sum : 1}}}]) // count  else {$sum : “$ref”}}}]) // on which you want operation  first{🡪$group by which we have to group //Condition  Second{=>\_id it will group with this name,  ’’$name’’=>done group by on this  ,=>num\_tutorial the name of the column of result of aggregrate function  $sum:1🡺aggregrate function work as count  **$group : {**  **\_id : { month: { $month: "$date" }, day: { $dayOfMonth: "$date" }, year: { $year: "$date" } },**  //to group by multiple  Different functions=> $avg, $min, $max,$count,$first,$last  “$name”=>dollar inside “” treated as refrence  Calculating sum ,average and count example | Same as group by with (+) aggregate functions  E.g=>select name,count(\*) from students  Group by name having name=”manu” |
| $match🡺Match specified document using query  Syntax🡺{$match:{<query}}} //work as having when used by group by  Db.Manu.aggregrate([{$match:{age:{$gt:25}}}])  Db.Manu.aggregrate([{$match:{age:{$gt:25}}},{$group : {\_id : "$name", num\_tutorial : {$sum : 1}}}]) | Equivalent 2  Select name,sum(\*) from students groupby name having age>25; |
| Always remember preference of staging match>group  Else you have to call it through the refrence of group one🡺  Db.Manu.aggregrate([{$group : {\_id : "$name", num\_tutorial : {$sum : 1}}},{$match:{\_id.age:{$gt:25}}}]) | No such things |
| $count=> to count in group use as a stage in last  {$count:”field\_name”}  Db.Manu.aggregrate([{$match:{age:{$gt:25}}},{$group : {\_id : "$name", num\_tutorial : {$count : “name of new column”}}}]) |  |
| Remember ,🡺pipelining  And num\_tutorial:{$count:”name of new column”} work on same line is not pipelining |  |
| $sort:{“column\_name”: 1 or -1},🡸🡺.sort({name:1})  Match>group>sort |  |
| $project excludes,includeand add new columns  {project:{c\_name:1,c\_name:0},🡸🡺//like we done in projections |  |
| $limit same as limit()..can be first and can belast in staging  Db.persons.aggregrate([{$limit:100},{$match:{age:{$gt:27}}},$group etc]) |  |
|  |  |
|  |  |
|  |  |
|  |  |
| $unwind🡺use when we want to some aggregaration on array fields  Create one document on per array element……continue |  |
|  |  |