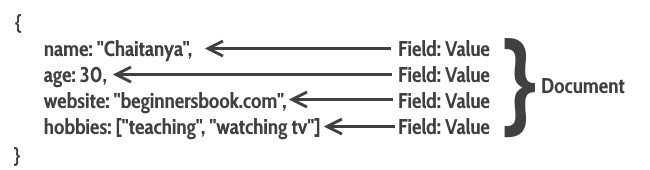
**MongoDB**

MongoDB is an open source, document oriented database that stores data in form of documents (key and value pairs).

**What is a document?**  
If you came from a relational database background then you can think of them as rows in RDBMS.



{

name: "Chaitanya",

age: 30,

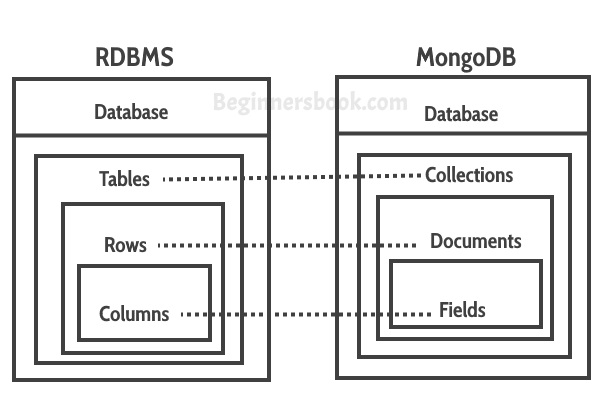
website: "beginnersbook.com",

hobbies: ["Teaching", "Watching TV"]

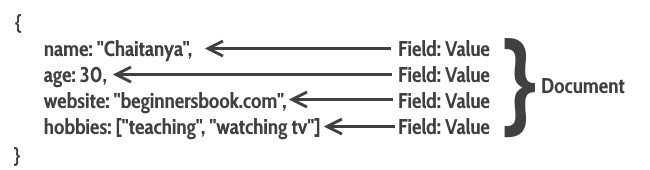
}

This is a [JSON](https://beginnersbook.com/2015/04/json-tutorial/) like structure. Where data is stored in form of key and value pairs.

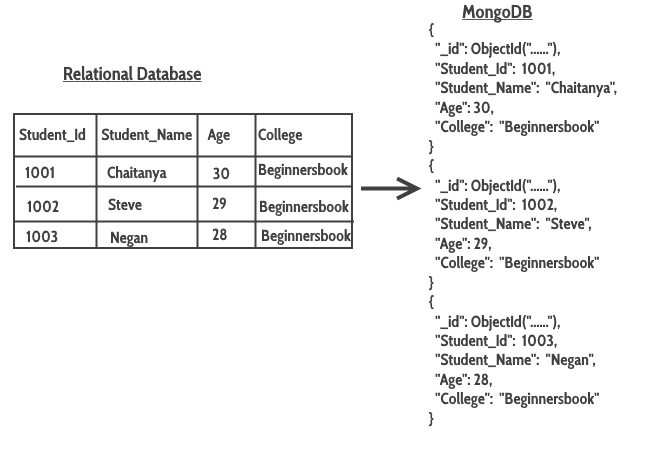
## Mapping relational database to MongoDB

  
**Collections** in MongoDB is equivalent to the tables in RDBMS.  
**Documents** in MongoDB is equivalent to the rows in RDBMS.  
**Fields** in MongoDB is equivalent to the columns in RDBMS.

Fields (key and value pairs) are stored in document, documents are stored in collection and collections are stored in database.

**This is how a document looks in MongoDB:** As you can see this is similar to the row in RDBMS. The only difference is that they are in JSON format.  


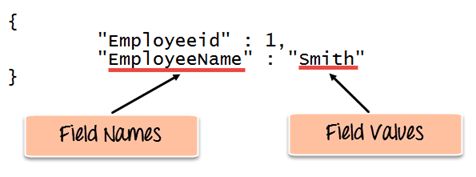
## Table vs Collection

Here we will see how a table in relational database looks in MongoDB. As you see columns are represented as key-value pairs([JSON](https://beginnersbook.com/2015/04/json-tutorial/) Format), rows are represented as documents. MongoDB automatically inserts a unique \_id(12-byte field) field in every document, this serves as primary key for each document.  
  
Another cool thing about MongoDB is that it supports dynamic schema which means one document of a collection can have 4 fields while the other document has only 3 fields. This is not possible in relational database.

**How to Create Database & Collection in MongoDB**

In MongoDB, the first basic step is to have a database and collection in place. The database is used to store all of the collections, and the collection in turn is used to store all of the documents. The documents in turn will contain the relevant Field Name and Field values.

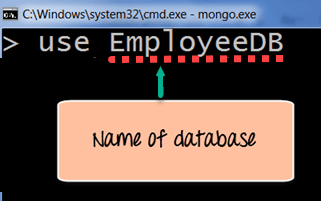
The snapshot below shows a basic example of how a document would look like.

[](https://www.guru99.com/images/MongoDB/112115_0607_Introductio1.png)

The Field Names of the document are "Employeeid" and "EmployeeName" and the Field values are "1" and "Smith' respectively. A bunch of documents would then make up a collection in MongoDB.

## Creating a database using “use” command

Creating a database in MongoDB is as simple as issuing the "**use**" command. The following example shows how this can be done.

[](https://www.guru99.com/images/MongoDB/112115_0607_Introductio2.png)

## Creating a Collection/Table using insert()

The easiest way to create a collection is to insert a record (which is nothing but a document consisting of Field names and Values) into a collection. If the collection does not exist a new one will be created.

The following example shows how this can be done.

db.Employee.insert

(

{

"Employeeid" : 1,

"EmployeeName" : "Martin"

}

)

**Code Explanation:**

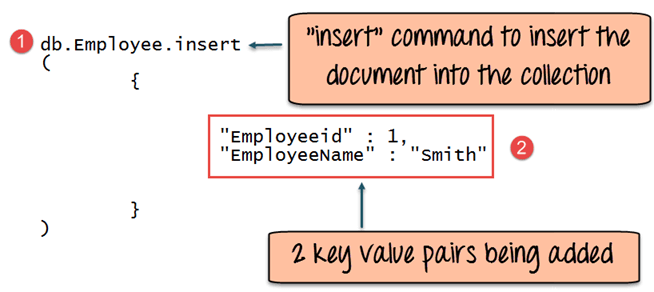
As seen above, by using the **"insert"** command the collection will be created.

## Adding documents using insert() command

MongoDB provides the **insert () command** to insert documents into a collection. The following example shows how this can be done.

**Step 1)** Write the "insert" command

**Step 2)** Within the "insert" command, add the required Field Name and Field Value for the document which needs to be created.

[](https://www.guru99.com/images/MongoDB/112115_0607_Introductio5.png)

**Code Explanation:**

1. The first part of the command is the "**insert statement"** which is the statement used to insert a document into the collection.
2. The second part of the statement is to add the Field name and the Field value, in other words, what is the document in the collection going to contain.

If the command is executed successfully, the following Output will be shown

**Output:**

The output shows that the operation performed was an insert operation and that one record was inserted into the collection.

# Add MongoDB Array using insert() with Example

The "insert" command can also be used to insert multiple documents into a collection at one time. The below code example can be used to insert multiple documents at a time.

The following example shows how this can be done,

**Step 1)** Create a[JavaScript](https://www.guru99.com/interactive-javascript-tutorials.html)variable called myEmployee to hold the array of documents

**Step 2)** Add the required documents with the Field Name and values to the variable

**Step 3)** Use the insert command to insert the array of documents into the collection

var myEmployee=

[

{

"Employeeid" : 1,

"EmployeeName" : "Smith"

},

{

"Employeeid" : 2,

"EmployeeName" : "Mohan"

},

{

"Employeeid" : 3,

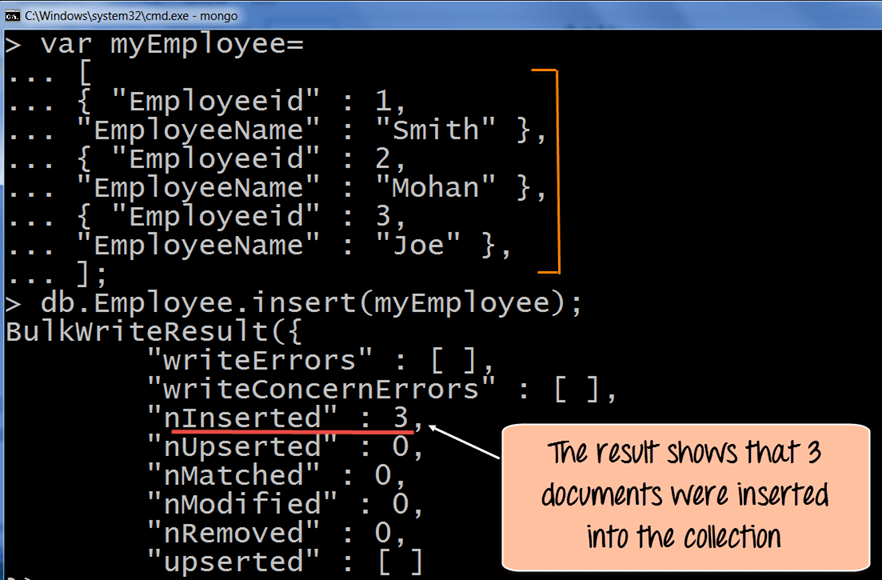
"EmployeeName" : "Joe"

},

];

db.Employee.insert(myEmployee);

If the command is executed successfully, the following Output will be shown

[](https://www.guru99.com/images/MongoDB/112115_0607_Introductio8.png)

The output shows that those 3 documents were added to the collection.

### Printing in JSON format

JSON is a format called **JavaScript Object Notation**, and is just a way to store information in an organized, easy-to-read manner. In our further examples, we are going to use the JSON print functionality to see the output in a better format.

Let's look at an example of printing in JSON format

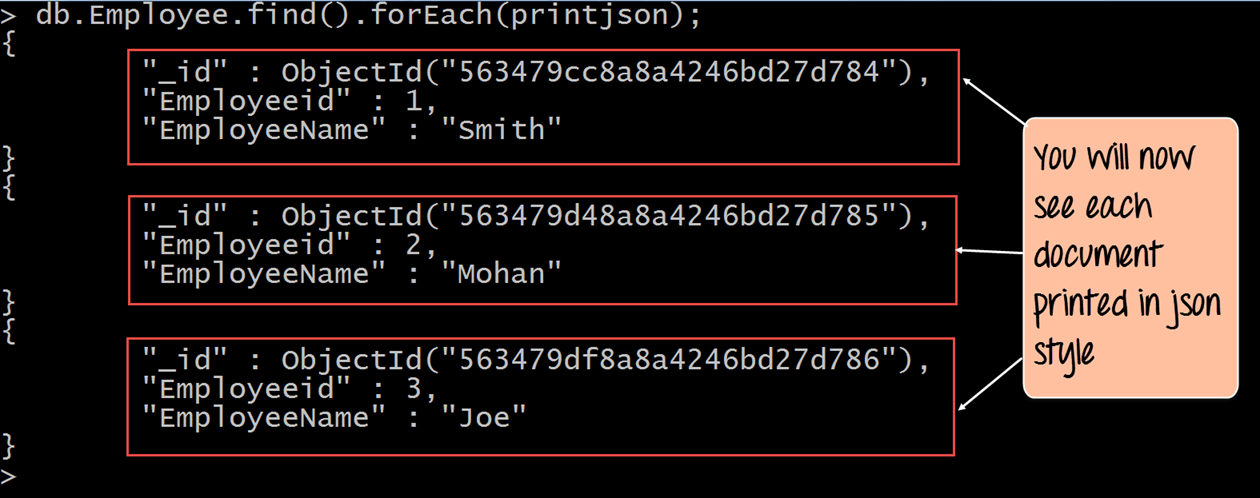
db.Employee.find().forEach(printjson)

**Code Explanation:**

1. The first change is to append the function called for Each() to the find() function. What this does is that it makes sure to explicitly go through each document in the collection. In this way, you have more control of what you can do with each of the documents in the collection.
2. The second change is to put the printjson command to the forEach statement. This will cause each document in the collection to be displayed in JSON format.

If the command is executed successfully, the following Output will be shown

**Output:**

[](https://www.guru99.com/images/MongoDB/112115_0607_Introductio10.png)

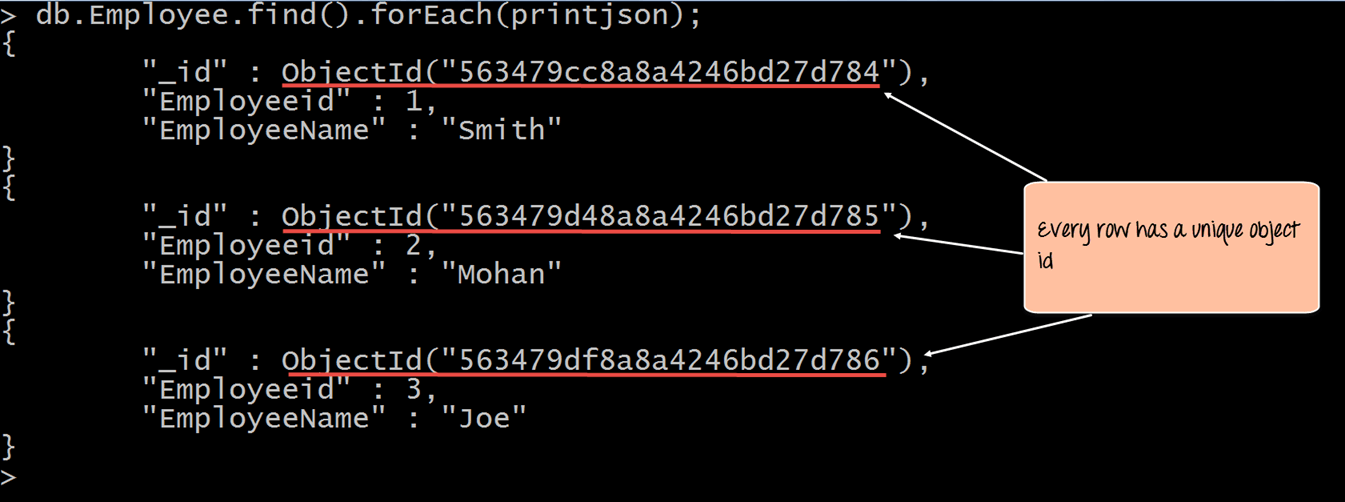
The output clearly shows that all of the documents are printed in JSON style

# Mongodb Primary Key: Example to set \_id field with ObjectId()

### What is Primary Key in MongoDB?

In MongoDB, \_id field as the primary key for the collection so that each document can be uniquely identified in the collection. The \_id field contains a unique ObjectID value.

By default when inserting documents in the collection, if you don't add a field name with the \_id in the field name, then MongoDB will automatically add an Object id field as shown below

[](https://www.guru99.com/images/MongoDB/112115_0607_Introductio11.png)

When you query the documents in a collection, you can see the ObjectId for each document in the collection.

If you want to ensure that MongoDB does not create the \_id Field when the collection is created and if you want to specify your own id as the \_id of the collection, then you need to explicitly define this while creating the collection.

When explicitly creating an id field, it needs to be created with \_id in its name.

Let's look at an example on how we can achieve this.

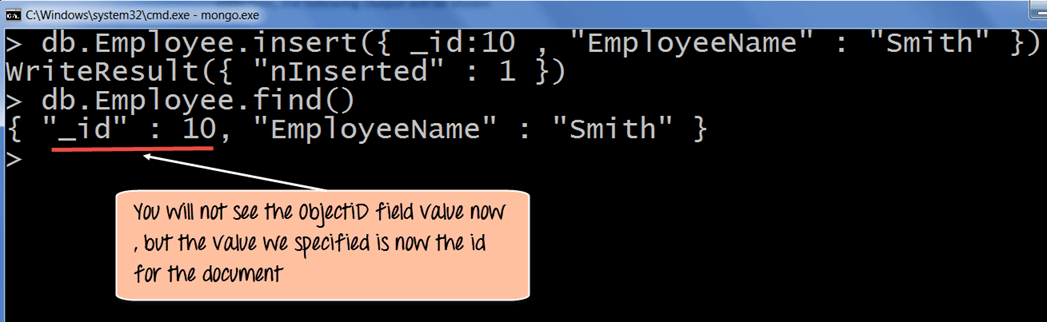
db.Employee.insert({\_id:10, "EmployeeName" : "Smith"})

**Code Explanation:**

1. We are assuming that we are creating the first document in the collection and hence in the above statement while creating the collection, we explicitly define the field \_id and define a value for it.

If the command is executed successfully and now use the find command to display the documents in the collection, the following Output will be shown

**Output:**

[](https://www.guru99.com/images/MongoDB/112115_0607_Introductio13.png)

The output clearly shows that the \_id field we defined while creating the collection is now used as the primary key for the collection.

# MongoDB Query Document using find() with Example

The method of fetching or getting data from a MongoDB database is carried out by using queries. While performing a query operation, one can also use criteria’s or conditions which can be used to retrieve specific data from the database.

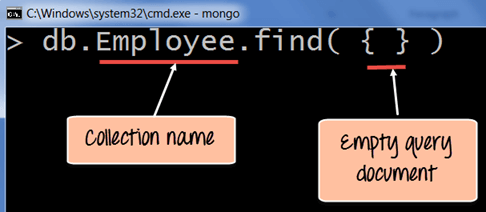
MongoDB provides a function called **db.collection.find ()** which is used for retrieval of documents from a MongoDB database.

During the course of this tutorial, you will see how this function is used in various ways to achieve the purpose of document retrieval.

## Basic query operations

The basic query operations cover the simple operations such as getting all of the documents in a MongoDB collection. Let’s look at an example of how we can accomplish this.

All of our code will be run in the MongoDB[JavaScript](https://www.guru99.com/interactive-javascript-tutorials.html)command shell. Consider that we have a collection named ‘Employee’ in our MongoDB database and we execute the below command.

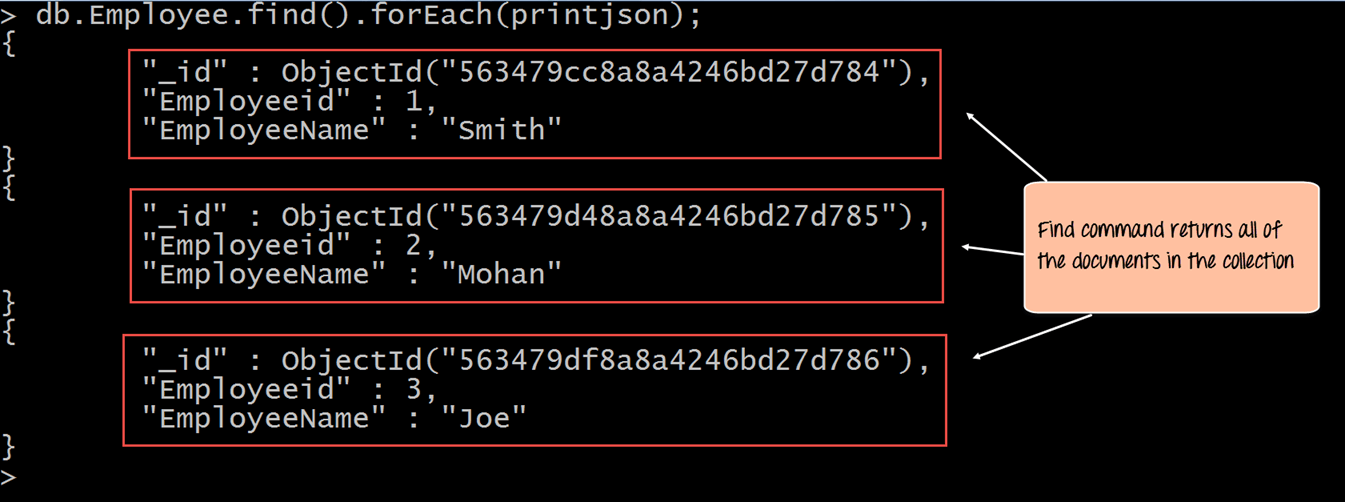
[](https://www.guru99.com/images/MongoDB/112115_0613_Introductio1.png)

**Code Explanation:**

1. Employee is the collection name in the MongoDB database
2. The find command is an in-built function which is used to retrieve the documents in the collection.

If the command is executed successfully, the following Output will be shown

**Output:**

[](https://www.guru99.com/images/MongoDB/112115_0613_Introductio2.png)

The output shows all the documents which are present in the collection.

We can also add criteria to our queries so that we can fetch documents based on certain conditions.

### Example 1

Let's look at a couple of examples of how we can accomplish this.

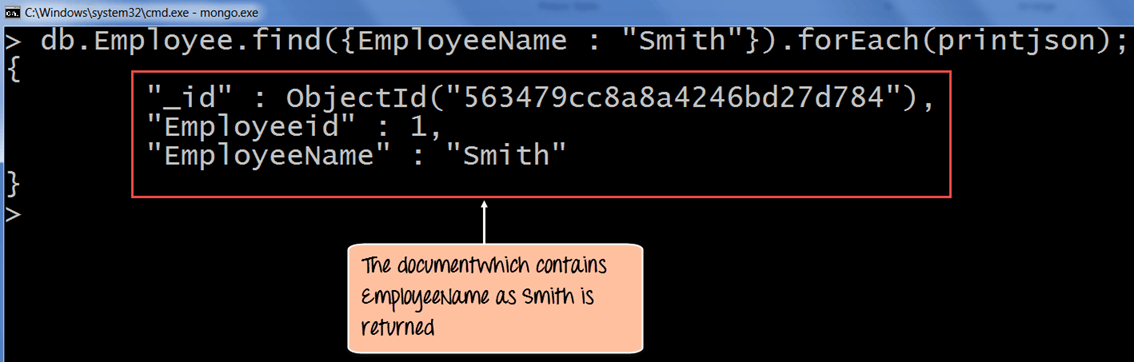
db.Employee.find({EmployeeName : "Smith"}).forEach(printjson);

**Code Explanation:**

1. Here we want to find for an Employee whose name is "Smith" in the collection , hence we enter the filter criteria as EmployeeName : "Smith"

If the command is executed successfully, the following Output will be shown

**Output:**

[](https://www.guru99.com/images/MongoDB/112115_0613_Introductio4.png)

The output shows that only the document which contains "Smith" as the Employee Name is returned.

### Example 2

Now, let's take a look at another code example which makes use of the greater than search criteria. When this criteria is included, it actually searches those documents where the value of the field is greater than the specified value.

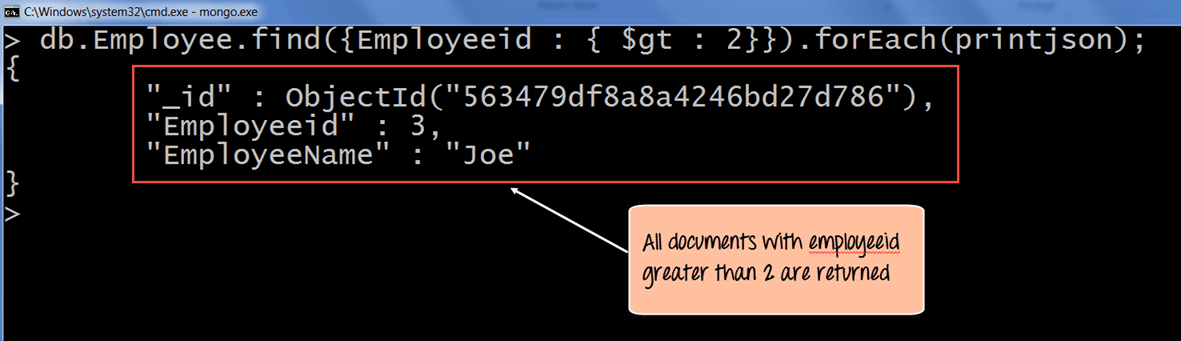
db.Employee.find({Employeeid : {$gt:2}}).forEach(printjson);

**Code Explanation:**

1. Here we want to find for all Employee's whose id is greater than 2. The $gt is called a query selection operator, and what is just means is to use the greater than expression.

If the command is executed successfully, the following Output will be shown

**Output:**

[](https://www.guru99.com/images/MongoDB/112115_0613_Introductio6.png)

All of the documents wherein the Employee id is greater than 2 is returned.

**Greater Than Criteria:**  
Syntax:

db.collection\_name.find({"field\_name":{$gt:criteria\_value}}).pretty()

For example: I would like to fetch the details of students having age > 32 then the query should be:

db.students.find({"age":{$gt:32}}).pretty()

**Less than Criteria:**  
Syntax:

db.collection\_name.find({"field\_name":{$lt:criteria\_value}}).pretty()

Example: Find all the students having id less than 3000. The command for this criteria would be:

db.students.find({"StudentId":{$lt:3000}}).pretty()

**Not Equals Criteria:**  
Syntax:

db.collection\_name.find({"field\_name":{$ne:criteria\_value}}).pretty()

Example: Find all the students where id is not equal to 1002. The command for this criteria would be:

db.students.find({"StudentId":{$ne:1002}}).pretty()

## Querying all the documents in JSON format

To print the data in JSON format run the command **db.collection\_name.find().forEach(printjson)**

Lets say we have a collection students in a database named beginnersbookdb. To get all the documents we use this command:

db.students.find()

However the output we get is not in any format and less-readable. To improve the readability, we can format the output in JSON format with this command:

db.students.find().forEach(printjson);

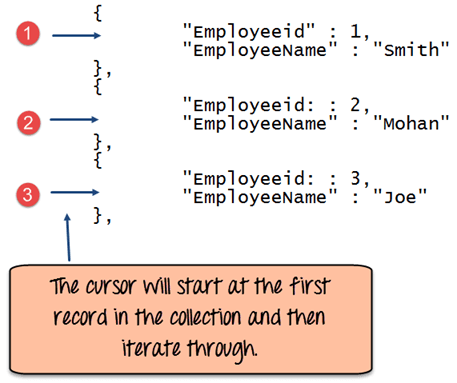
OR simply use pretty() – It does the same thing.

db.students.find().pretty()

## What is Cursor in MongoDB?

When the **db.collection.find ()** function is used to search for documents in the collection, the result returns a pointer to the collection of documents returned which is called a cursor.

By default, the cursor will be iterated automatically when the result of the query is returned. But one can also explicitly go through the items returned in the cursor one by one. If you see the below example, if we have 3 documents in our collection, the cursor object will point to the first document and then iterate through all of the documents of the collection.

[](https://www.guru99.com/images/MongoDB/112115_0613_Introductio7.png)

The following example shows how this can be done.

var myEmployee = db.Employee.find( { Employeeid : { $gt:2 }});

while(myEmployee.hasNext())

{

print(tojson(myEmployee.next()));

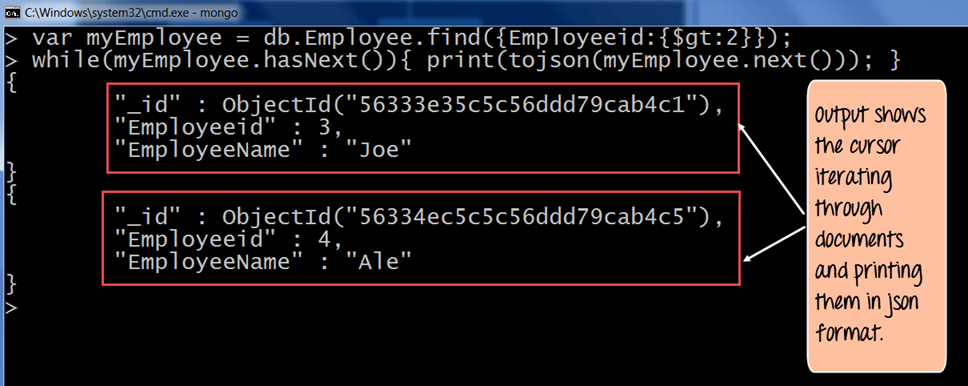
}

**Code Explanation:**

1. First we take the result set of the query which finds the Employee's whose id is greater than 2 and assign it to the[JavaScript](https://www.guru99.com/interactive-javascript-tutorials.html)variable 'myEmployee'
2. Next we use the while loop to iterate through all of the documents which are returned as part of the query.
3. Finally for each document, we print the details of that document in JSON readable format.

If the command is executed successfully, the following Output will be shown

**Output:**

[](https://www.guru99.com/images/MongoDB/112115_0613_Introductio9.png)

# MongoDB order with Sort() & Limit() Query with Examples

## What is Query Modifications?

Mongo DB provides query modifiers such as the 'limit' and 'Orders' clause to provide more flexibility when executing queries. We will take a look at the following query modifiers

## MongoDB Limit Query Results

This modifier is used to limit the number of documents which are returned in the result set for a query. The following example shows how this can be done.

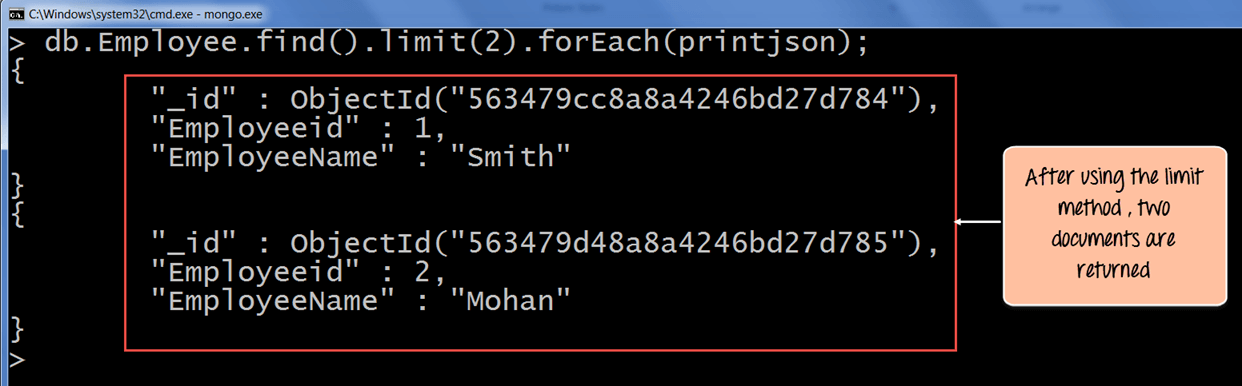
db.Employee.find().limit(2).forEach(printjson);

**Code Explanation:**

1. The above code takes the find function which returns all of the documents in the collection but then uses the limit clause to limit the number of documents being returned to just 2.

**Output:**

If the command is executed successfully, the following Output will be shown

[](https://www.guru99.com/images/MongoDB/112115_0613_Introductio11.png)

The output clearly shows that since there is a limit modifier, so at most just 2 records are returned as part of the result set based on the ObjectId in ascending order.

## MongoDB Sort by Descending Order

One can specify the order of documents to be returned based on ascending or descending order of any key in the collection. The following example shows how this can be done.

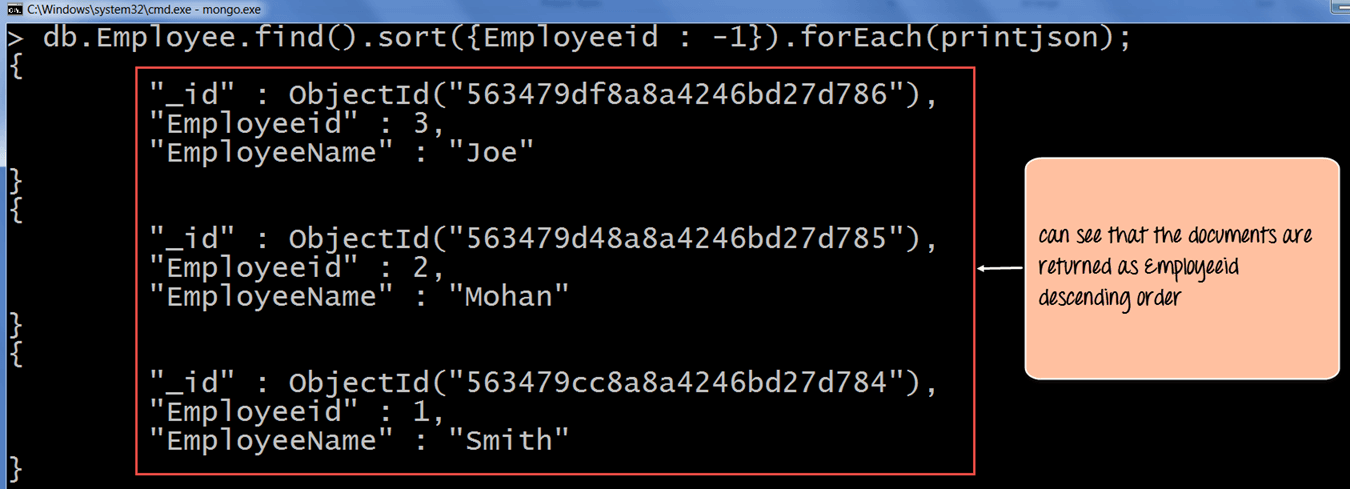
db.Employee.find().sort({Employeeid:-1}).forEach(printjson)

**Code Explanation:**

1. The above code takes the sort function which returns all of the documents in the collection but then uses the modifier to change the order in which the records are returned. Here the -1 indicates that we want to return the documents based on the descending order of Employee id.

If the command is executed successfully, the following Output will be shown

**Output:**

[](https://www.guru99.com/images/MongoDB/112115_0613_Introductio13.png)

The output clearly shows the documents being returned in descending order of the Employeeid.

Ascending order is defined by value 1.

# MongoDB Count() & Remove() Functions with Examples

The concept of aggregation is to carry out a computation on the results which are returned in a query. For example, suppose you wanted to know what is the count of documents in a collection as per the query fired, then MongoDB provides the count() function.

Let's look at an example of this.

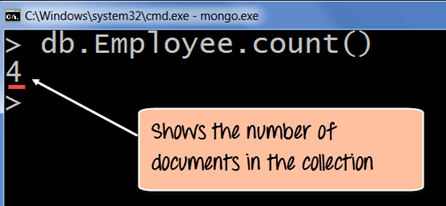
db.Employee.count()

**Code Explanation:**

1. The above code executes the count function.

If the command is executed successfully, the following Output will be shown

**Output:**

[](https://www.guru99.com/images/MongoDB/112115_0613_Introductio15.png)

The output clearly shows that 4 documents are there in the collection.

**Performing Modifications**

The other two classes of operations in MongoDB are the update and remove statements.

The update operations allow one to modify existing data, and the remove operations allow the deletion of data from a collection.

## Deleting Documents

In MongoDB, the **db.collection.remove ()** method is used to remove documents from a collection. Either all of the documents can be removed from a collection or only those which matches a specific condition.

If you just issue the remove command, all of the documents will be removed from the collection.

The following code example demonstrate how to remove a specific document from the collection.

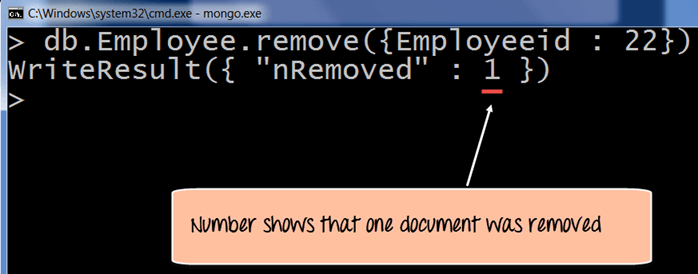
db.Employee.remove({Employeeid:22})

**Code Explanation:**

1. The above code use the remove function and specifies the criteria which in this case is to remove the documents which have the Employee id as 22.

If the command is executed successfully, the following Output will be shown

**Output:**

[](https://www.guru99.com/images/MongoDB/112115_0613_Introductio21.png)

The output will show that 1 document was modified.

# MongoDB Update() Document with Example

## Basic document updates

MongoDB provides the update() command to update the documents of a collection. To update only the documents you want to update, you can add a criteria to the update statement so that only selected documents are updated.

The basic parameters in the command is a condition for which document needs to be updated, and the next is the modification which needs to be performed.

The following example shows how this can be done.

**Step 1)**Issue the update command

**Step 2)**Choose the condition which you want to use to decide which document needs to be updated. In our example, we want to update the document which has the Employee id 22.

**Step 3)** Use the set command to modify the Field Name

**Step 4)** Choose which Field Name you want to modify and enter the new value accordingly.

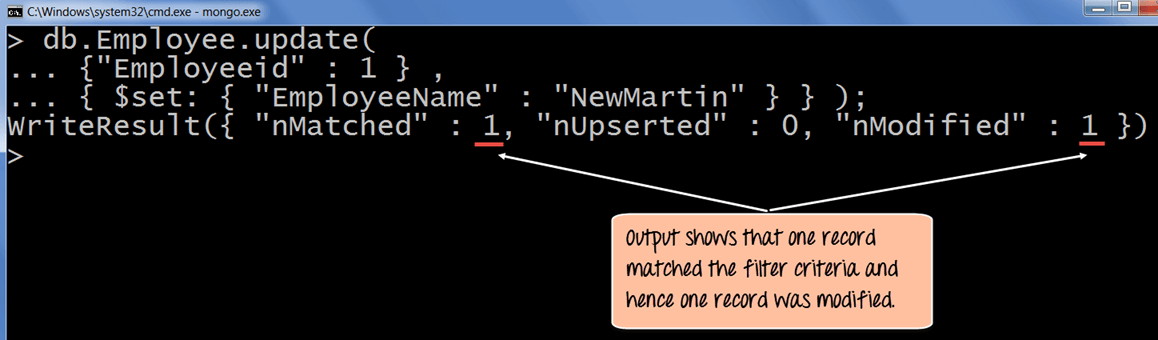
db.Employee.update(

{"Employeeid" : 1},

{$set: { "EmployeeName" : "NewMartin"}});

If the command is executed successfully, the following Output will be shown

**Output:**

[](https://www.guru99.com/images/MongoDB/112115_0613_Introductio17.png)

The output clearly shows that one record matched the condition and hence the relevant field value was modified.

## Updating Multiple Values

To ensure that multiple/bulk documents are updated at the same time in MongoDB you need to use the multi option because otherwise by default only one document is modified at a time.

The following example shows how to update many documents.

In this example, we are going to first find the document which has the Employee id as "1" and change the Employee name from "Martin" to "NewMartin"

**Step 1)** Issue the update command

**Step 2)** Choose the condition which you want to use to decide which document needs to be updated. In our example, we want the document which has the Employee id of "1" to be updated.

**Step 3)** Choose which Field Name's you want to modify and enter their new value accordingly.

db.Employee.update

(

{

Employeeid : 1

},

{

$set :

{

"EmployeeName" : "NewMartin"

"Employeeid" : 22

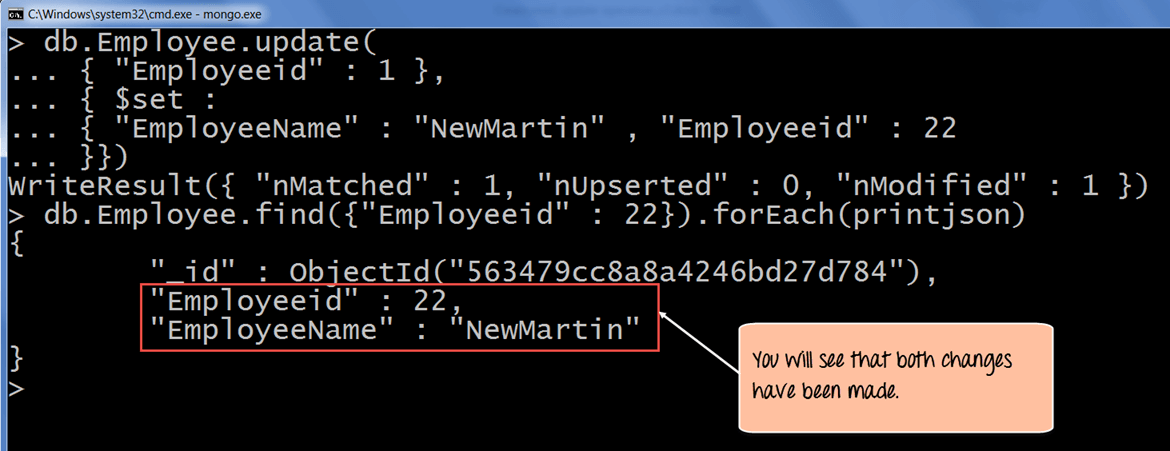
}

}

)

If the command is executed successfully and if you run the **"find"** command to search for the document with Employee id as 22 you will see the following Output will be shown

**Output:**

[](https://www.guru99.com/images/MongoDB/112115_0613_Introductio19.png)

The output clearly shows that one record matched the condition and hence the relevant field value was modified.

# MongoDB - Query Document

In this chapter, we will learn how to query document from MongoDB collection.

## **The find() Method**

To query data from MongoDB collection, you need to use MongoDB's **find()** method.

### **Syntax**

The basic syntax of **find()** method is as follows −

>db.COLLECTION\_NAME.find()

**find()** method will display all the documents in a non-structured way.

## **The pretty() Method**

To display the results in a formatted way, you can use **pretty()** method.

### **Syntax**

>db.mycol.find().pretty()

## **Example**

>db.mycol.find().pretty()

{

"\_id": ObjectId(7df78ad8902c),

"title": "MongoDB Overview",

"description": "MongoDB is no sql database",

"by": "tutorials point",

"url": "http://www.tutorialspoint.com",

"tags": ["mongodb", "database", "NoSQL"],

"likes": "100"

}

>

Apart from find() method, there is **findOne()** method, that returns only one document.

## **RDBMS Where Clause Equivalents in MongoDB**

To query the document on the basis of some condition, you can use following operations.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operation** | **Syntax** | **Example** | **RDBMS Equivalent** |
| Equality | {<key>:<value>} | db.mycol.find({"by":"tutorials point"}).pretty() | where by = 'tutorials point' |
| Less Than | {<key>:{$lt:<value>}} | db.mycol.find({"likes":{$lt:50}}).pretty() | where likes < 50 |
| Less Than Equals | {<key>:{$lte:<value>}} | db.mycol.find({"likes":{$lte:50}}).pretty() | where likes <= 50 |
| Greater Than | {<key>:{$gt:<value>}} | db.mycol.find({"likes":{$gt:50}}).pretty() | where likes > 50 |
| Greater Than Equals | {<key>:{$gte:<value>}} | db.mycol.find({"likes":{$gte:50}}).pretty() | where likes >= 50 |
| Not Equals | {<key>:{$ne:<value>}} | db.mycol.find({"likes":{$ne:50}}).pretty() | where likes != 50 |

## **AND in MongoDB**

### **Syntax**

In the **find()** method, if you pass multiple keys by separating them by ',' then MongoDB treats it as **AND**condition. Following is the basic syntax of **AND** −

>db.mycol.find(

{

$and: [

{key1: value1}, {key2:value2}

]

}

).pretty()

### **Example**

Following example will show all the tutorials written by 'tutorials point' and whose title is 'MongoDB Overview'.

>db.mycol.find({$and:[{"by":"tutorials point"},{"title": "MongoDB Overview"}]}).pretty() {

"\_id": ObjectId(7df78ad8902c),

"title": "MongoDB Overview",

"description": "MongoDB is no sql database",

"by": "tutorials point",

"url": "http://www.tutorialspoint.com",

"tags": ["mongodb", "database", "NoSQL"],

"likes": "100"

}

For the above given example, equivalent where clause will be **' where by = 'tutorials point' AND title = 'MongoDB Overview' '**. You can pass any number of key, value pairs in find clause.

## **OR in MongoDB**

### **Syntax**

To query documents based on the OR condition, you need to use **$or** keyword. Following is the basic syntax of **OR** −

>db.mycol.find(

{

$or: [

{key1: value1}, {key2:value2}

]

}

).pretty()

### **Example**

Following example will show all the tutorials written by 'tutorials point' or whose title is 'MongoDB Overview'.

>db.mycol.find({$or:[{"by":"tutorials point"},{"title": "MongoDB Overview"}]}).pretty()

{

"\_id": ObjectId(7df78ad8902c),

"title": "MongoDB Overview",

"description": "MongoDB is no sql database",

"by": "tutorials point",

"url": "http://www.tutorialspoint.com",

"tags": ["mongodb", "database", "NoSQL"],

"likes": "100"

}

>

## **Using AND and OR Together**

### **Example**

The following example will show the documents that have likes greater than 10 and whose title is either 'MongoDB Overview' or by is 'tutorials point'. Equivalent SQL where clause is **'where likes>10 AND (by = 'tutorials point' OR title = 'MongoDB Overview')'**

>db.mycol.find({"likes": {$gt:10}, $or: [{"by": "tutorials point"},

{"title": "MongoDB Overview"}]}).pretty()

{

"\_id": ObjectId(7df78ad8902c),

"title": "MongoDB Overview",

"description": "MongoDB is no sql database",

"by": "tutorials point",

"url": "http://www.tutorialspoint.com",

"tags": ["mongodb", "database", "NoSQL"],

"likes": "100"

}

>

1. See the list of databases using **show dbs** command.

> show dbs

admin 0.000GB

beginnersbook 0.000GB

local 0.000GB

It is showing two default databases and one database “beginnersbook” that I have created.

2. Switch to the database that needs to be dropped by typing this command.

use database\_name

For example I want to delete the database “beginnersbook”.

> use beginnersbook

switched to db beginnersbook

Note: Change the database name in the above command, from beginnersbook to the database that needs to be deleted.

3. Now, the currently selected database is beginnersbook so the command db.dropDatabase() would delete this database.

> db.dropDatabase()

{ "dropped" : "beginnersbook", "ok" : 1 }

The command executed successfully and showing the operation “dropped” and status “ok” which means that the database is dropped.

# Introduction to NoSQL Databases

A database Management System provides the mechanism to store and retrieve the data. There are different kinds of database Management Systems:  
1. RDBMS (Relational Database Management Systems)  
2. OLAP (Online Analytical Processing)  
3. NoSQL (Not only SQL)

In this guide, We will discuss NoSQL. **NoSQL databases** were created to overcome the limitations of relational databases.

## What is a NoSQL database?

NoSQL databases are different than relational databases like MQSql. In relational database you need to create the table, define schema, set the data types of fields etc before you can actually insert the data. In NoSQL you don’t have to worry about that, you can insert, update data on the fly.

One of the advantage of NoSQL database is that they are really easy to scale and they are much faster in most types of operations that we perform on database. There are certain situations where you would prefer relational database over NoSQL, however when you are dealing with huge amount of data then NoSQL database is your best choice.

## Limitations of Relational databases

1. In relational database we need to define structure and schema of data first and then only we can process the data.

2. Relational database systems provides consistency and integrity of data by enforcing [ACID properties](https://beginnersbook.com/2015/04/acid-properties-in-dbms/) (Atomicity, Consistency, Isolation and Durability ). There are some scenarios where this is useful like banking system. However in most of the other cases these properties are significant performance overhead and can make your database response very slow.

3. Most of the applications store their data in [JSON](https://beginnersbook.com/2015/04/json-tutorial/) format and RDBMS don’t provide you a better way of performing operations such as create, insert, update, delete etc on this data. On the other hand NoSQL store their data in JSON format, which is compatible with most of the today’s world application.

## What are the advantages of NoSQL

There are several advantages of working with NoSQL databases such as MongoDB and Cassandra. The main advantages are high scalability and high availability.

**High scalability:** NoSQL database such as MongoDB uses sharding for horizontal scaling. Sharding is partitioning of data and placing it on multiple machines in such a way that the order of the data is preserved. Vertical scaling means adding more resources to the existing machine while horizontal scaling means adding more machines to handle the data. Vertical scaling is not that easy to implement, on the other hand horizontal scaling is easy to implement. Horizontal scaling database examples: MongoDB, Cassandra etc. Because of this feature NoSQL can handle huge amount of data, as the data grows NoSQL scale itself to handle that data in efficient manner.

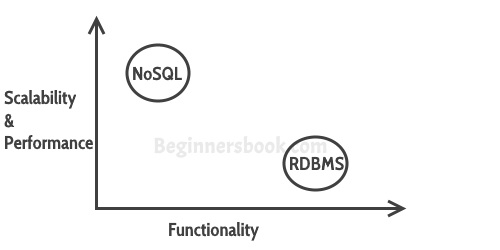
**High Availability:** Auto replication feature in MongoDB makes it highly available because in case of any failure data replicates itself to the previous consistent state.

## Types of NoSQL database

Here are the types of NoSQL databases and the name of the databases system that falls in that category. MongoDB falls in the category of NoSQL document based database.  
**Key Value Store:** Memcached, Redis, Coherence  
**Tabular:** Hbase, Big Table, Accumulo  
**Document based:** MongoDB, CouchDB, Cloudant

## RDBMS Vs NoSQL

[**RDBMS:**](https://beginnersbook.com/2015/04/rdbms-concepts/) It is a structured data that provides more functionality but gives less performance.  
**NoSQL:** Structured or semi structured data, less functionality and high performance.



### So when I say less functionality in NoSQL what’s missing:

1. You can’t have constraints in NoSQL  
2. Joins are not supported in NoSQL  
These supports actually hinders the scalability of a database, so while using NoSQL database like MongoDB, you can implements these functionalities at the application level.

## When to go for NoSQL

When you would want to choose NoSQL over relational database:

1. When you want to store and retrieve huge amount of data.
2. The relationship between the data you store is not that important
3. The data is not structured and changing over time
4. Constraints and Joins support is not required at database level
5. The data is growing continuously and you need to scale the database regular to handle the data.