INTRODUCTION TO NOSQL

Engr. Laraib Siddiqui

Introduction

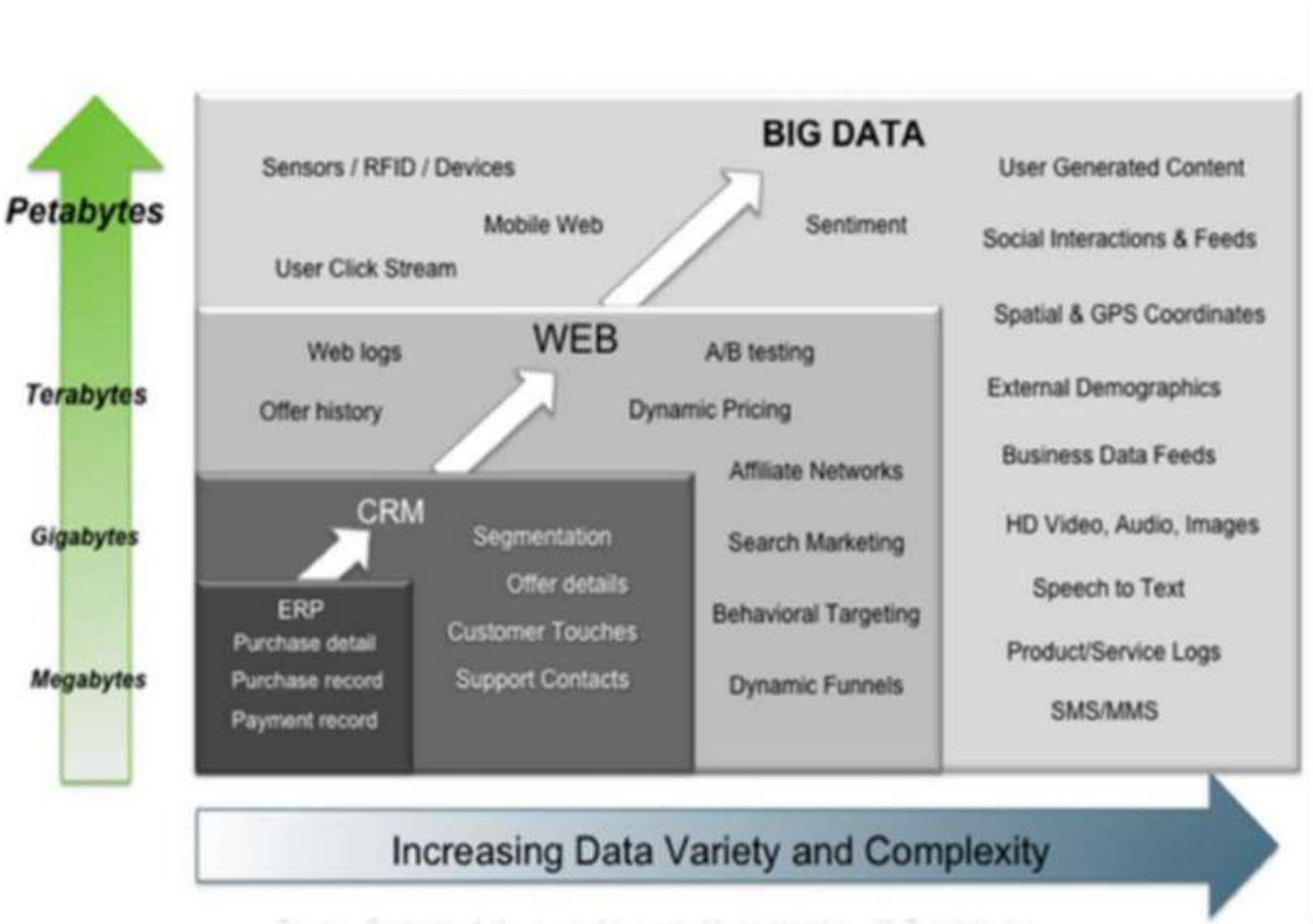
- NoSQL means ‘Not Only SQL’
- Non relational database.
- Flexible to used for big data & real time web applications.

Why use NOSQL

- Handles massive data.
- Schema less data model.
- Handles unstructured data.
- Horizontal Scaling.

Rise of Big data

Big data is a term for data sets that are so large that traditional methods of storage and processing are inadequate.



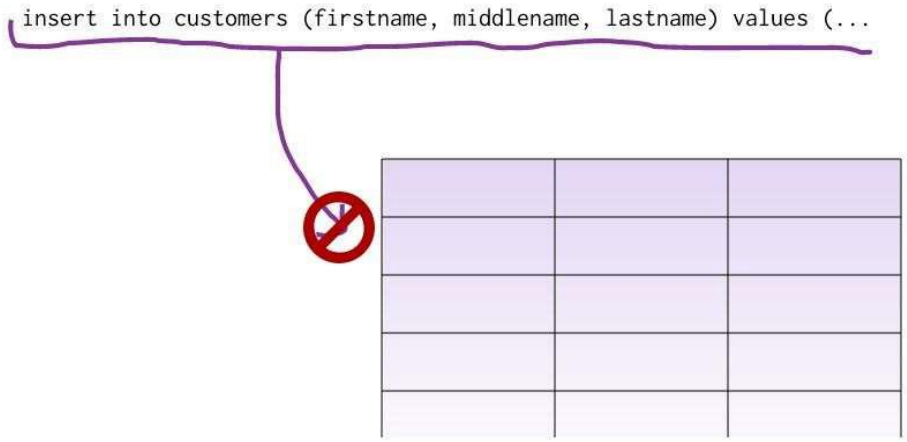
Schema-less Data model

In relational Databases:

- You can't add a record which does not fit the schema
- You need to add NULLs to unused items in a row
- You should consider the datatypes.
- You can't add multiple items in a field
- You should create another table: primary-key, foreign key, joins, normalization

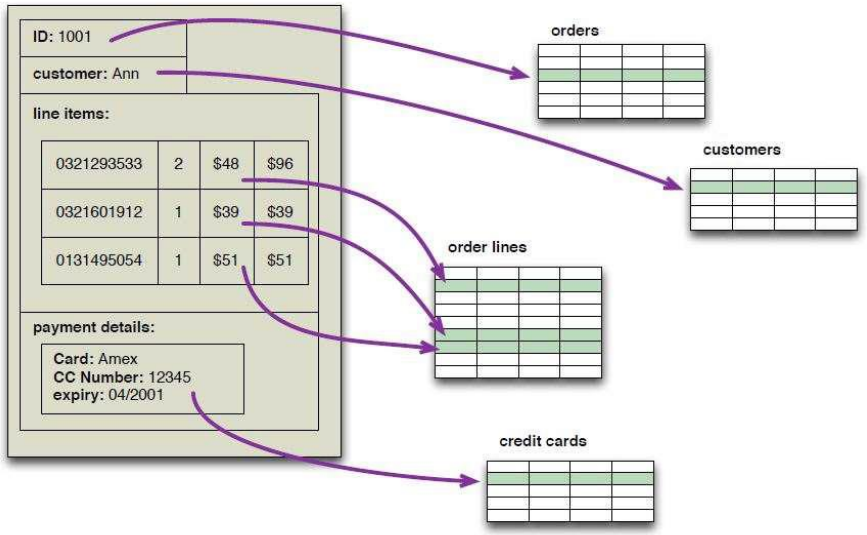
In NOSQL

- There is no schema to consider
- There is no unused cell
- There is no datatype (implicit)
- Most of considerations are done in application layer
- We gather all items in an aggregate (document)

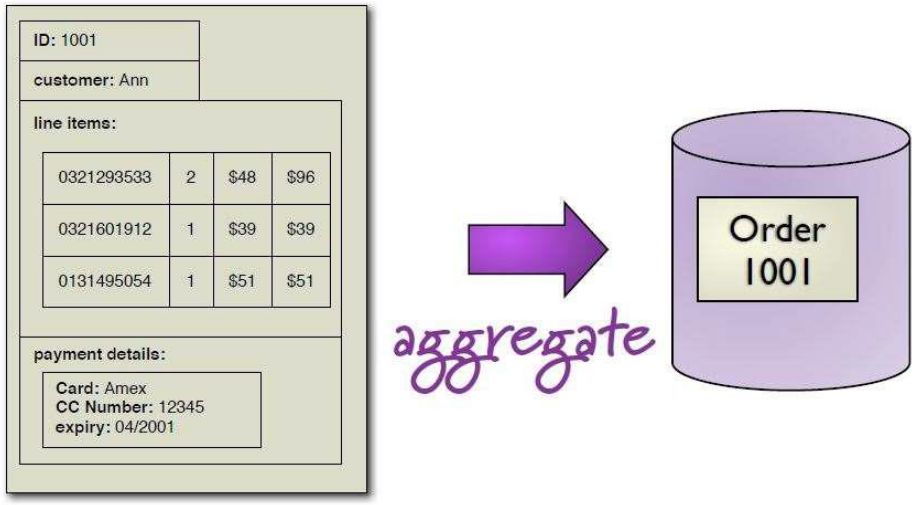


Aggregation

- An aggregate is a cluster of domain objects that can be treated as a single unit
- Aggregates are the basic element of transfer of data storage – you request to load or save whole aggregates




```
{
"id": "1001",
 "firstName": "Ann",
 "lastName": "Williams",
 "age": 55,
"purchasedItems":
  {
    0321290533 {qty, price... }
    0321601912 {qty, price... }
    0131495054 {qty, price... }
  }
"paymentDetails":
  { cc info... }
"address":
  {
   "street": "1234 Park",
   "city": "San Francisco",
   "state": "CA",
   "zip": "94102"
  }
}
```



Data structure

Unstructured data

The university has 5600 students.
John's ID is number 1, he is 18 years old and already holds a B.Sc. degree.
David's ID is number 2, he is 31 years old and holds a Ph.D. degree. Robert's ID is number 3, he is 51 years old and also holds the same degree as David, a Ph.D. degree.

Semi-structured data

```
<University>
  <Student ID="1">
    <Name>John</Name>
    <Age>18</Age>
    <Degree>B.Sc.</Degree>
  </Student>
  <Student ID="2">
    <Name>David</Name>
    <Age>31</Age>
    <Degree>Ph.D. </Degree>
  </Student>
....
</University>
```

Structured data

ID	Name	Age	Degree
1	John	18	B.Sc.
2	David	31	Ph.D.
3	Robert	51	Ph.D.
4	Rick	26	M.Sc.
5	Michael	19	B.Sc.

RDBMS are not designed to manage these type of data efficiently, and this is where NoSQL comes into the stage with the capability of handling huge amount of data properly.

Scaling

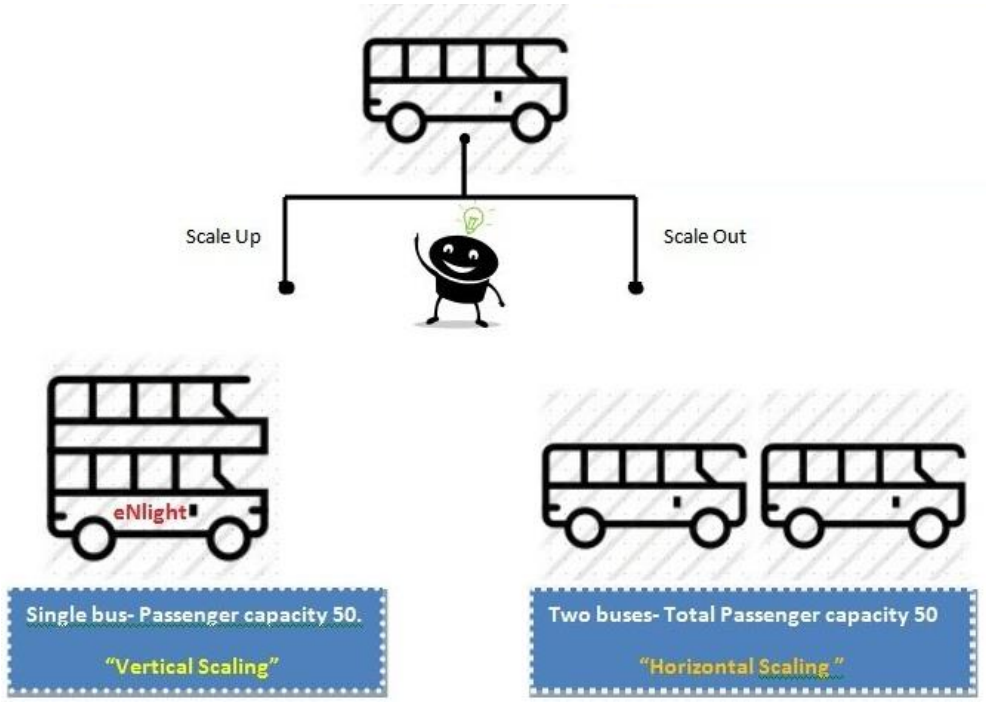
Horizontal Scalability:

adding more nodes for data storage and processing as the volume of data grows.

Vertical Scalability

expanding the storage and computing power of existing nodes.

In NOSQL systems, horizontal scalability is employed while the system is operational, so techniques for distributing the existing data among new nodes without interrupting system operation are necessary.



Types of NOSQL

- Document database (Mongodb, Couchdb)
- Column databases (BigTable, Cassandra, SimpleDB)
- Key-Value store (Redis, Riak, Azure, DynamoDB)
- Cache systems (Redis, Memcache)
- Graph databases (Neo4J, OrientDB, Polyglot)

Each DB has its own query language

Column oriented data model

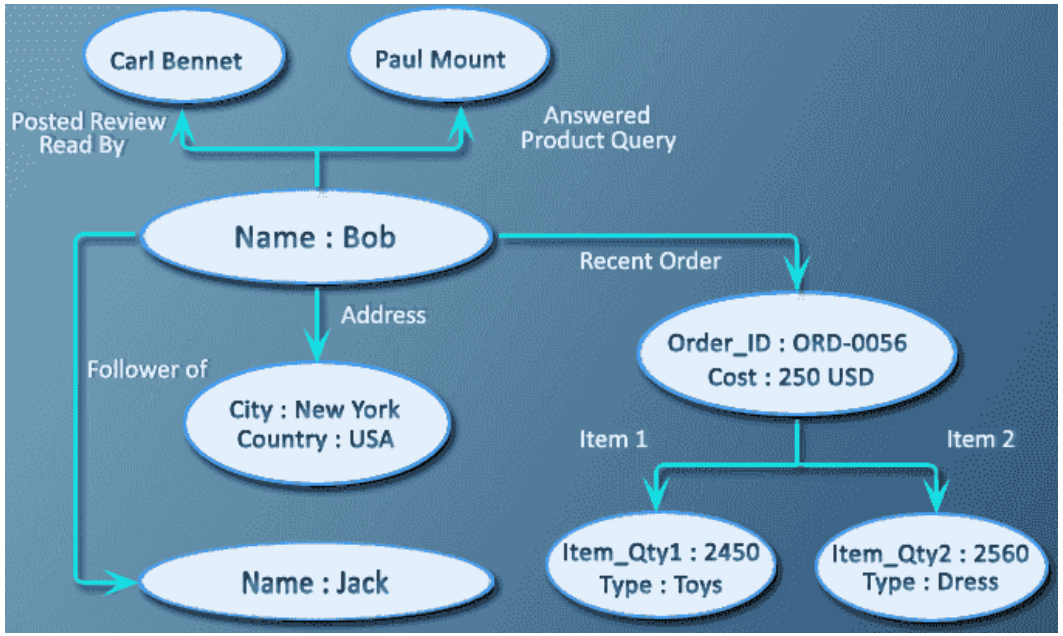
- Primarily work on columns and every column is treated individually.
- Column stores data in column specific files.
- In column stores, query processors work on columns too.

ColumnFamily: Authors

Key	Value
"Eric Long"	Columns Name: Value "email": "eric (at) long.com" "country": "United Kingdom" "registeredSince": "01/01/2002"
"John Steward"	Columns Name: Value "email": "john.steward (at) somedomain.com" "country": "Australia" "registeredSince": "01/01/2009"
"Ronald Mathies"	Columns Name: Value "email": "ronald (at) sodeso.nl" "country": "Netherlands, The" "registeredSince": "01/01/2010"

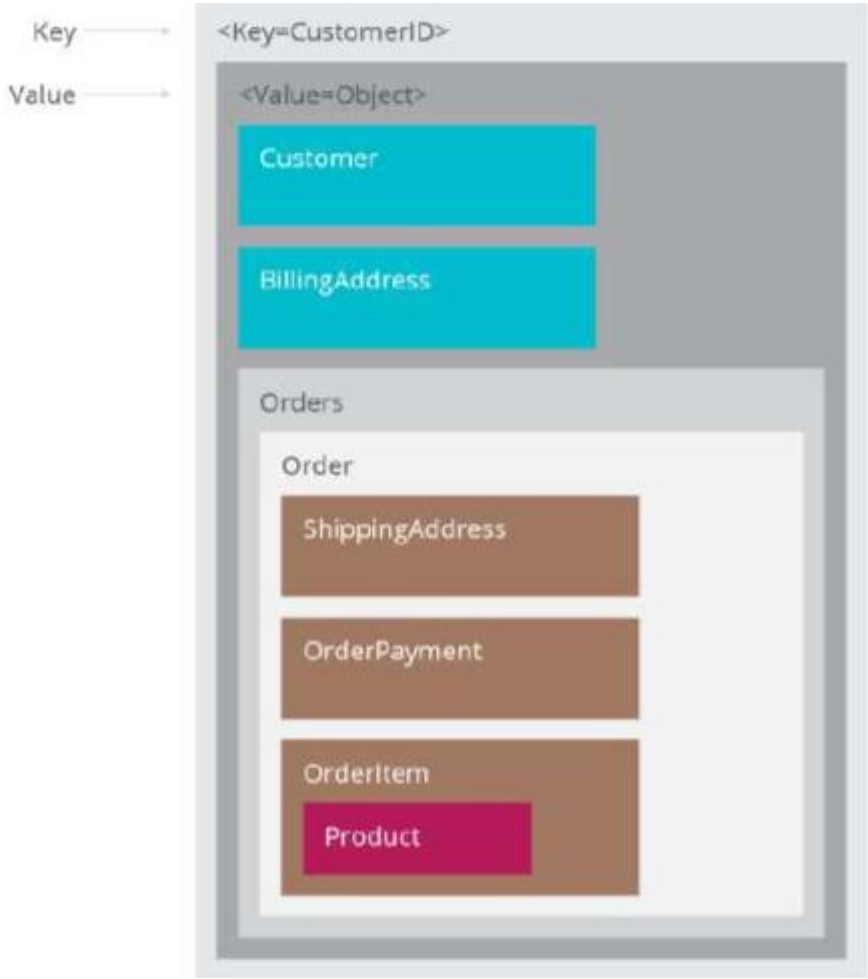
Graph data model

- Based on Graph Theory.
- stores data in a graph.
- Each node represents an entity (such as a student or business) and each edge represents a connection or relationship between two nodes.
- Every node and edge are defined by a unique identifier.
- Each node knows its adjacent nodes.



Key-value data model

- The main idea is the use of a hash table
- Access data (values) by strings called keys
- Data model: (key, value) pairs
- Basic Operations: Insert(key, value), Fetch(key), Update(key), Delete(key)
- "Value" is stored as a "Binary Large Object (blob)



Document-based data model

- A collection of documents
- Data in this model is stored inside documents.
- Usually JSON like interchange model.
- Documents are not typically forced to have a schema and therefore are flexible and easy to change.

```
{
  person: {
    first_name: "Peter",
    last_name: "Peterson",
    addresses: [
      {street: "123 Peter St"},
      {street: "504 Not Peter St"}
    ],
  }
}
```

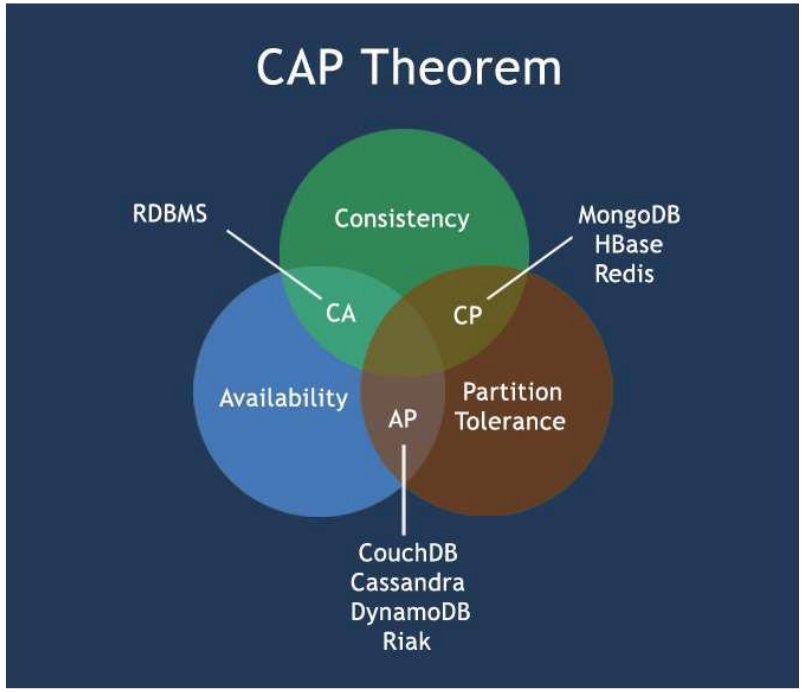
CAP theorem

- **Consistency:** Clients should read the same data.
- **Availability:** Data to be available all time.
- **Partial Tolerance:** Data to be partitioned across network segments due to network failures.

We can not achieve all the three items in distributed database systems.

Here is the brief description of three combinations CA, CP, AP :

- **CA** - Single site cluster, therefore all nodes are always in contact. When a partition occurs, the system blocks.
- **CP** -Some data may not be accessible, but the rest is still consistent/accurate.
- **AP** - System is still available under partitioning, but some of the data returned may be inaccurate.



INTRODUCTION TO MONGODB

Introduction

MongoDB is a cross-platform, document oriented database that provides, high performance, high availability, and easy scalability.

MongoDB works on concept of collection and document.

A sample MongoDB query

Mongodb

```
db.orders.find({"items.product.name":/Refactoring})
```

SQL

```
SELECT *
FROM customerOrder, orderItem, product
WHERE customerOrder.orderId = orderItem.customerOrderId AND orderItem.productId = product.productId AND product.name LIKE '%Refactoring%'
```

There is no join in MongoDB query because we are using an aggregate data

The use Command

Use command is used to create database. The command will create a new database if it doesn't exist, otherwise it will switch to mentioned database.

Syntax

USE *database_name*

Example

```
>use myfirstdb
switched to db myfirstdb
```

The Show command

If you want to check your databases list, use the command **show dbs**.

Syntax

show *dbs*

Example

```
>show dbs
local 0.78125GB
myfirstdb 0.23012GB
test 0.23012GB
```

There should be atleast one document in your db, otherwise its name will not be display with show dbs command.

The dropDatabase() method

MongoDB **db.dropDatabase()** command is used to drop an existing database.

Syntax

db.dropDatabase()

Example

```
>use myfirstdb
switched to db myfirstdb
>db.dropDatabase()
>{ "dropped" : “myfirstdb", "ok" : 1 }
```

The createCollection() method

MongoDB **db.createCollection(name, options)** is used to create collection.

Syntax

db.createCollection (name, options)

In the command, **name** is name of collection to be created. **Options** is a document and is used to specify configuration of collection.

Example

```
>use test
switched to db test
>db.createCollection("mycollection")
{ "ok" : 1 }
```

The drop() method

MongoDB's **db.collection.drop()** is used to drop a collection from the database.

Syntax

db.collection_name.drop()

Example

```
>db.mycollection.drop()
true
>
```

The insert() method

To insert data into collection, we use insert() method.

Syntax

db.collection_name.insert(document)

Example

```
> db.user.insert (
 { First_Name: "Harry", Last_Name: "Potter",
   Date_Of_Birth: "1989-09-26",
   e_mail: "harry.potter@gmail.com",
   phone: "9848022338" }
)
{
   "acknowledged" : true,
   "insertedId" : ObjectId("5dd62b4070fb13eec3963bea")
}
>
```

You can use insertOne( ) method if you want to add one document. For adding multiple documents, you can use insertMany( ) method.

The find() method

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

Syntax

db.collection_name.find()

Example

```
> db.person.find()
{ "_id" : ObjectId("5f87f1d693ba9be5a07bd7c1"), "name" : "Haider" }
{ "_id" : ObjectId("5f87fc4593ba9be5a07bd7c2"), "name" : "Ali", "age" : 25, "city" : "Karachi" }
{ "_id" : ObjectId("5f87fc4593ba9be5a07bd7c3"), "name" : "warda", "age" : 21, "city" : "Lahore" }
{ "_id" : ObjectId("5f87fc4593ba9be5a07bd7c4"), "name" : "Mahad", "age" : 23, "city" : "Islamabad" }
>
```

The find() method

To find filtered data or display specific information, we can add different comparison operators with find() method.

db.person.find ({ age :{ $lt :25}})

```
> db.person.find({age:{$lt:25}})
{ "_id" : ObjectId("5f87fc4593ba9be5a07bd7c3"), "name" : "warda", "age" : 21, "city" : "Lahore" }
{ "_id" : ObjectId("5f87fc4593ba9be5a07bd7c4"), "name" : "Mahad", "age" : 23, "city" : "Islamabad" }
```

The pretty() method

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

Syntax

db.collection_name.find().pretty()

Example

```
> db.person.find().pretty()
{ "_id" : ObjectId("5f87f1d693ba9be5a07bd7c1"), "name" : "Haider" }
{
        "_id" : ObjectId("5f87fc4593ba9be5a07bd7c2"),
        "name" : "Ali",
        "age" : 25,
        "city" : "Karachi"
}
{
        "_id" : ObjectId("5f87fc4593ba9be5a07bd7c3"),
        "name" : "warda",
        "age" : 21,
        "city" : "Lahore"
}
{
        "_id" : ObjectId("5f87fc4593ba9be5a07bd7c4"),
        "name" : "Mahad",
        "age" : 23,
        "city" : "Islamabad"
}
>
```

The Update() method

The update() method updates the values in the existing document.

Syntax

Db.collection_name.update(SELECTION_CRITERIA, UPDATED_DATA)

Example

```
> db.person.update({"age":25},{$set:{"age":31}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
```

Comparison Operators

Name	Description.
$eq	Matches values that are equal to a specified value.
$gt	Matches values that are greater than a specified value.
$gte	Matches values that are greater than or equal to a specified value.
$in	Matches any of the values specified in an array.
$lt	Matches values that are less than a specified value.
$lte	Matches values that are less than or equal to a specified value.
$ne	Matches all values that are not equal to a specified value.
$nin	Matches none of the values specified in an array.
$and	Joins query clauses with a logical AND returns all documents that match the conditions of both clauses.
$not	Inverts the effect of a query expression and returns documents that do not match the query expression.
$nor	Joins query clauses with a logical NOR returns all documents that fail to match both clauses.
$or	Joins query clauses with a logical OR returns all documents that match the conditions of either clause.

The remove() method

The **remove()** method is used to remove a document from the collection.

Syntax

Db.collection_name.remove(DELETION_CRITERIA)

Example

```
> db.person.remove({"name":"Haider"})
WriteResult({ "nRemoved" : 1 })
>
```

Other methods

To limit the records, we use limit() method.

Syntax

db.collection_name.find().limit(NUMBER)

To sort the document sort() method is used. 1 is used for ascending order and -1 is use to sort in decending order

Syntax

db.collection_name.find().sort({KEY:1})

Connection to PHP

To use MongoDB with PHP, you need to use MongoDB PHP driver.

To make a connection, you need to specify the database name, if the database doesn't exist then MongoDB creates it automatically.

```
<?php
 // connect to mongodb
  $m = new MongoClient();
  echo "Connection to database successfully";
 // select a database
   $db = $m->mydb;
   echo "Database mydb selected";
?>
```

Practice

1. Write a MongoDB query to display all the documents in the collection restaurants.
2. Write a MongoDB query to display all the restaurant which is in the borough Bronx.
3. Write a MongoDB query to display the fields restaurant_id, name, borough and cuisine for all the documents in the collection restaurant.
4. Write a MongoDB query to find the restaurants that achieved a score is more than 80 but less than 100.
5. Write a MongoDB query to arrange the name of the restaurants in ascending order along with all the columns.


```
{
  "address": {
     "building": "1007",
     "coord": [ -73.856077, 40.848447 ],
     "street": "Morris Park Ave",
     "zipcode": "10462"
  },
  "borough": "Bronx",
  "cuisine": "Bakery",
  "grades": [
     { "date": { "$date": 1393804800000 }, "grade": "A", "score": 2 },
     { "date": { "$date": 1378857600000 }, "grade": "A", "score": 6 },
     { "date": { "$date": 1358985600000 }, "grade": "A", "score": 10 },
     { "date": { "$date": 1322006400000 }, "grade": "A", "score": 9 },
     { "date": { "$date": 1299715200000 }, "grade": "B", "score": 14 }
  ],
  "name": "Morris Park Bake Shop",
  "restaurant_id": "30075445"
}
```