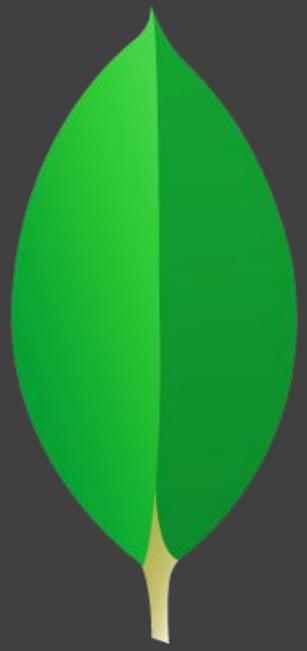


MangoDB Class

#MangoDB Notes

MongoDB Relationships

MongoDB Relationships

- The representation of how the number of multiple documents is connected logically to each other is known as MongoDB Relationships.

- **Methods to Create MongoDB Relationships :-**
  1. Embedded Relationships **(Denormalization)**
  2. Documented Reference Relationships **(Normalization)**

- Embed works great with One-to-One and One-to-Many relationships,
- The referenced is good for Many-to-Many relationships.

- These two types of relationships are also known as Denormalization, which is Embedded, while Reference relationships are known as Normalization.

- Establishing relationships between documents can help refine the database structure and work in favour to develop the performance and make execution time shorter.

Embedded Data Models (Denormalization)

```
{
  _id: "5db579f5faf1f8434098f7f5"
  title: "Tutorial #1",
  author: “akash"
  comments: [
                    {
                      username: "jack",
                      text: "This is a great tutorial.",
                      createdAt: 2019-10-27T11:05:39.898Z
                    },
                    {
                      username: "mary",
                      text: "Thank you, it helps me alot.",
                      createdAt: 2019-10-27T11:05:40.710Z
                    }
                  ]
}
```

EMBEDDED

```
{
  "_id": "61829c469145952548881d99c",
  "text": "This is the text of the post document",
  "comments": [
    {
      "text": "This is the text of the first comment",
      "author": "author"
    },
    {
      "text": "This is the text of the second comment",
      "author": "author2"
    }
  ]
}
```

LINKED

```
{
  "_id": "61829c469145952548881d99c",
  "text": "This is the text of the post document"
}

[
  {
    "_id": "61829c469145952548881d99c",
    "text": "This is the text of the first comment",
    "author": "author",
    "postId": "61829c469145952548881d99c"
  },
  {
    "_id": "61829c469145952548881d99c",
    "text": "This is the text of the second comment",
    "author": "author2",
    "postId": "61829c469145952548881d99c"
  }
]
```

Embedded Relationships

Embedded Relationship

- In this approach, one document will be embedded in another document (like a subset).
- If it will be embedded within the documents, queries will run faster than if we spread them on multiple documents.
- This will provide acceleration in the performance, especially with a large amount of data.
- Here, in embedded relationships, we will discuss two types of model:
  1. One-to-one relationship
  2. One-to-many relationship

One-to-one relationship:

- This is the simplest of all relationship. Here, we have a one-parent document and one child document for the parent, that's one to one.

- **Syntax:**

```
db.person.insertOne({
   name:"xyz",
   address:{
      city:"Ahmedabad",
      State:"Gujarat"
   }
})
```

Example

```
mydb> db.person.insertOne({name:"xyz", address:{city:"Ahmedabad",state:"Gujarat"}})
{
  acknowledged: true,
  insertedId: ObjectId("618ce0e0a82907e55e9cd6e8")
}
mydb> db.person.find()
[
  {
    _id: ObjectId("618ce0e0a82907e55e9cd6e8"),
    name: 'xyz',
    address: { city: 'Ahmedabad', state: 'Gujarat' }
  }
]
mydb>
```

One-to-many relationship:

- One to Many relationships consist of one parent with multiple child documents. Similar to one to one but with many child documents.

- **Syntax:-**

```
db.person.insertOne({
   name:"ABC",
   age:30,
   address:[{
      {city:"Surat"},
      {State:"Gujarat"}
   ]
})
```

Example

```
mydb> db.person.insertOne({name: "ABC",age: 30, address:[{city:"Surat"},{state:"Gujarat"}]})
{
  acknowledged: true,
  insertedId: ObjectId("618cea2fa82907e55e9cd6e9")
}
mydb> db.person.find({name:"ABC"})
[
  {
    _id: ObjectId("618cea2fa82907e55e9cd6e9"),
    name: 'ABC',
    age: 30,
    address: [ { city: 'Surat' }, { state: 'Gujarat' } ]
  }
]
mydb>
```

Document Referenced Relationships

- Rather than implanting a child document into the parent document, we separate the child and parent document respectively.
- When data needs to be repeated across many documents, it is helpful to have them in their own separate document.
- This reduces error and keeps data consistent.
- **Note:- define _id in each document.**

a. Parent Document

- Syntax :-

```
db.teacher.insertOne({
  _id:1,
  tName:"ABC”
})
```

Parent Document /Teacher Collection

```
mydb> db.createCollection("teacher")
{ ok: 1 }
mydb> db.teacher.insertOne({_id:1,tName:"ABC"})
{ acknowledged: true, insertedId: 1 }
mydb> db.teacher.find()
[ { _id: 1, tName: 'ABC' } ]
mydb>
```

b. Child Documents

- Syntax :-

```
db.class.insertOne({
   _id:1,
   cName:"FY",
   subject:"HTML",
   tID:1
})
```

Child Documents/Class Collection

```
mydb> db.createCollection("class")
{ ok: 1 }
mydb> db.class.insertOne({_id:1,cName:"FY",subject:"HTML",tId:1})
{ acknowledged: true, insertedId: 1 }
mydb> db.class.insertOne({_id:2,cName:"SY",subject:"PHP",tId:1})
{ acknowledged: true, insertedId: 2 }
mydb> db.class.find()
[
  { _id: 1, cName: 'FY', subject: 'HTML', tId: 1 },
  { _id: 2, cName: 'SY', subject: 'PHP', tId: 1 }
]
mydb>
```

Aggregation Pipeline/Lookup

- It is simply a collection of commands to be executed one by one in a sequence.
- We will create a pipeline to return documents from class collection, where _id will be local, and tId will be foreign fields.
- Finally, we will add a $match operator, which will only return documents with a teacher name as ABC in the teacher collections.

$lookup

- $lookup allows you to perform joins on collections in the same database.
- $lookup works by returning documents from a "joined" collection as a sub-array of the original collection.
- $lookup supports both basic equality matches as well as uncorrelated sub-queries.

Syntax

```
db.teacher.aggregate([
  {
    "$lookup":{
      "from":"class",
      "localField":"_id",
      "foreignField":"tId",
      "as":"class name"
    }
  },
  {
    "$match":{
      "tName":"ABC"
    }
  }
])
```

- Here,
  - from: the collection we want to join with
  - localField: the field we want to join by in the local collection (the collection we are running the query on)
  - foreignField: the field we want to join by in the foreign collection (the collection we want to join with)
  - as: the name of the output array for the results

Example

```
mydb> db.teacher.aggregate([{$lookup: {from: "class",localField: "_id",foreignField: "tId",as:"class name" }},{$match: {tName:
 "ABC"}}])
[
  {
    _id: 1,
    tName: 'ABC',
    'class name': [
      { _id: 1, cName: 'FY', subject: 'HTML', tId: 1 },
      { _id: 2, cName: 'SY', subject: 'PHP', tId: 1 }
    ]
  }
]
mydb>
```

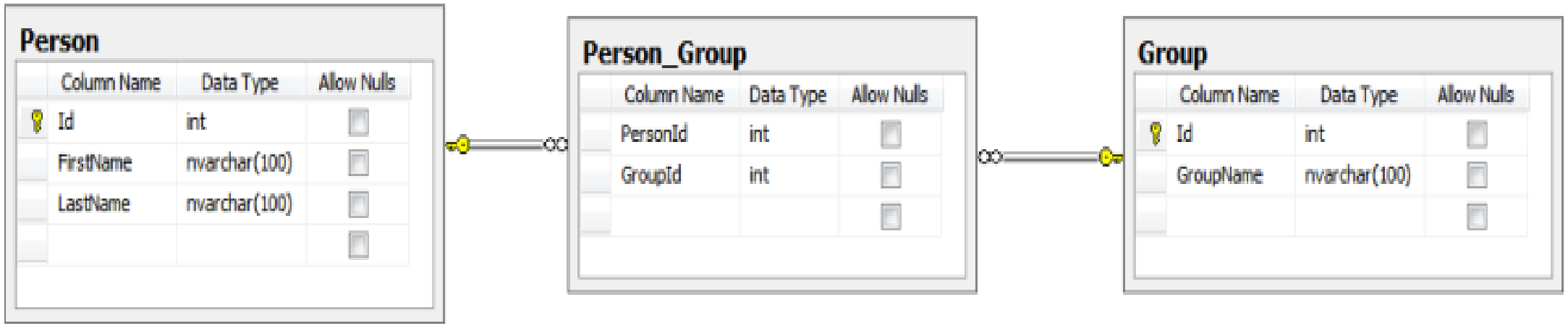
Documented Reference Relationships

many-to-many relationship

- Implementing a many-to-many relationship in a relational database is not as straight forward as a one-to-many relationship because there is no single command to accomplish it.

Example

- Here,there is a many-to-may relationship between the person table and the group table. The person_group table is the junction table.
- A junction table is just the table that stores the keys from the two parent tables to form the relationship.



Person Collection

- Syntax :-

```
db.person.insert({
  _id: 1,
  firstName: "ABC",
  lastName "ABC",
  groups: [1,2]
})
```

Person Collection Example

```
mydb> db.person.insertOne({_id:1,firstName:"ABC",lastName:"ABC",groups: [1,2]})
{ acknowledged: true, insertedId: 1 }
mydb> db.person.insertOne({_id:2,firstName:"XYZ",lastName:"XYZ",groups: [1]})
{ acknowledged: true, insertedId: 2 }
mydb> db.person.find()
[
  { _id: 1, firstName: 'ABC', lastName: 'ABC', groups: [ 1, 2 ] },
  { _id: 2, firstName: 'XYZ', lastName: 'XYZ', groups: [ 1 ] }
]
mydb>
```

Groups Collection

- Syntax :-

```
db.groups.insert({
  _id: 1,
  groupName: "mongoDB Training",
  "person": [1,2]
})
```

Groups Collection Example

```
mydb> db.createCollection("groups")
{ ok: 1 }
mydb> db.groups.insertOne({_id:1,groupName:"mongoDB Training",person:[1,2]})
{ acknowledged: true, insertedId: 1 }
mydb> db.groups.insertOne({_id:2,groupName:"angular Training",person:[1]})
{ acknowledged: true, insertedId: 2 }
mydb> db.groups.find()
[
  { _id: 1, groupName: 'mongoDB Training', person: [ 1, 2 ] },
  { _id: 2, groupName: 'angular Training', person: [ 1 ] }
]
mydb>
```

Explanation

- Basically, you can store an array of the Id's from the group collection in person collection to identify what groups a person belongs to.
- Likewise, you can store an array of the Id's from the person collection in the group document to identify what persons belong to a group.
- The documents above show that "ABC ABC" belongs to the "mongoDB Training" and "angular Training" groups.
- Similarly, "XYZ XYZ" only belongs to the "mongoDB Training" group.

Getting the data

- The following queries will show you how you can query the data without having to use joins as you would in a relational database.

```
mydb> db.person.find({groups:1})
[
  { _id: 1, firstName: 'ABC', lastName: 'ABC', groups: [ 1, 2 ] },
  { _id: 2, firstName: 'XYZ', lastName: 'XYZ', groups: [ 1 ] }
]
mydb> db.person.find({groups:2})
[ { _id: 1, firstName: 'ABC', lastName: 'ABC', groups: [ 1, 2 ] } ]
mydb> db.groups.find({person:1})
[
  { _id: 1, groupName: 'mongoDB Training', person: [ 1, 2 ] },
  { _id: 2, groupName: 'angular Training', person: [ 1 ] }
]
mydb> db.groups.find({person:2})
[ { _id: 1, groupName: 'mongoDB Training', person: [ 1, 2 ] } ]
mydb>
```

Example

```
db.product.{
_id:1,
product_name:'iphone12',
category:['iphone','mobile']
}
db.product.{
_id:1,
product_name:'iphone13',
category:['iphone','mobile']
}


db.category.{
_id:1,
category_name:'mobile',
product:['iphone11','iphone12','iphone13']
}

db.category.{
_id:1,
category_name:'iphone',
product:['iphone11','iphone12','iphone13']
}
```

many-to-one relationship

- Create CLASS collection and student collection.
- Where documents in student contain a reference to the CLASS document.

Class Collection

```
mydb> db.createCollection("class")
{ ok: 1 }
mydb> db.class.insertOne({_id:1,className:"First Class"})
{ acknowledged: true, insertedId: 1 }
mydb> db.class.insertOne({_id:2,className:"Second Class"})
{ acknowledged: true, insertedId: 2 }
mydb> db.class.find()
[
  { _id: 1, className: 'First Class' },
  { _id: 2, className: 'Second Class' }
]
mydb>
```

Student Collection

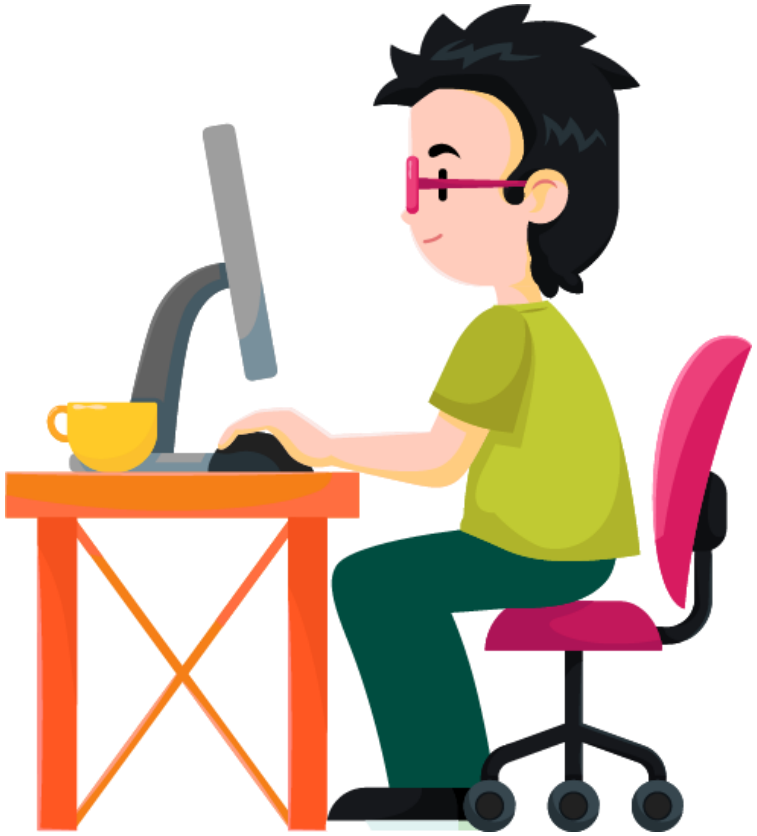
```
mydb> db.createCollection("student")
{ ok: 1 }
mydb> db.student.insertOne({_id:1,name:"ABC",classId:1})
{ acknowledged: true, insertedId: 1 }
mydb> db.student.insertOne({_id:2,name:"XYZ",classId:1})
{ acknowledged: true, insertedId: 2 }
mydb> db.student.insertOne({_id:3,name:"JKL",classId:2})
{ acknowledged: true, insertedId: 3 }
mydb> db.student.find()
[
  { _id: 1, name: 'ABC', classId: 1 },
  { _id: 2, name: 'XYZ', classId: 1 },
  { _id: 3, name: 'JKL', classId: 2 }
]
mydb>
```

Aggregation/Lookup

- This example list the students details who are from First class…

```
mydb> db.class.aggregate([{$lookup: {from: "student",localField:"_id",foreignField:"classId",
as:"Details"}},{$match:{className:"First Class"}}])
[
  {
    _id: 1,
    className: 'First Class',
    Details: [
      { _id: 1, name: 'ABC', classId: 1 },
      { _id: 2, name: 'XYZ', classId: 1 }
    ]
  }
]
mydb>
```

Get Exclusive Video Tutorials



www.aptutorials.com

https://www.youtube.com/user/Akashtips

Get More Details

www.akashsir.com

If You Liked It !

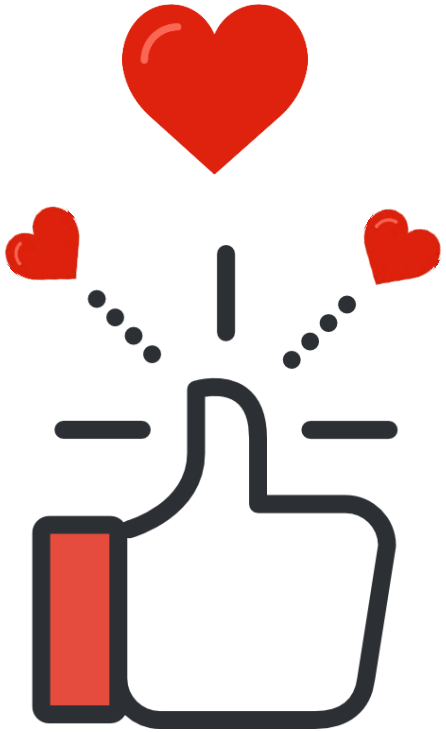
Rating Us Now

Just Dial

https://www.justdial.com/Ahmedabad/Akash-Technolabs-Navrangpura-Bus-Stop-Navrangpura/079PXX79-XX79-170615221520-S5C4_BZDET

Sulekha

https://www.sulekha.com/akash-technolabs-navrangpura-ahmedabad-contact-address/ahmedabad



Connect With Me

Akash Padhiyar

#AkashSir

www.akashsir.com

www.akashtechnolabs.com

www.akashpadhiyar.com

www.aptutorials.com

Social Info

Akash.padhiyar

Akashpadhiyar

Akash_padhiyar

+91 99786-21654

#Akashpadhiyar

#aptutorials