MEASI INSTITUTE OF INFORMATION TECHNOLOGY

(Approved by AICTE & Affiliated to University of Madras)

CHENNAI – 600 014

MEASI
Institute of
Information
Technology

MASTER OF COMPUTER APPLICATIONS

ACADEMIC YEAR 2025-2026

SEMESTER – I

Practical Record

Data Engineering and Management Lab
(435E1D)

REG. NO : _____

NAME : _____

BATCH : _____

MEASI INSTITUTE OF INFORMATION TECHNOLOGY

(Approved by AICTE & Affiliated to University of Madras)
CHENNAI- 600 014

MCA PRACTICAL

Data Engineering and Management Lab (435E1D)

Academic Year 2025-2026

Semester - I

NAME : **CLASS** :

REG-NO : **BATCH** :

This is to certify that this is the bonafide record of work done in the Computer Science Laboratory of **MEASI Institute of Information Technology**, submitted for the **University of Madras** Practical Examination held on ..... at **MEASI Institute of Information Technology**, **Chennai-600 014**.

STAFF IN-CHARGE **HEAD OF THE DEPARTMENT**

INTERNAL EXAMINER **EXTERNAL EXAMINER**

DATE: _____

INDEX

Ex.No : 1 **To customize your application using Zoho CRM**

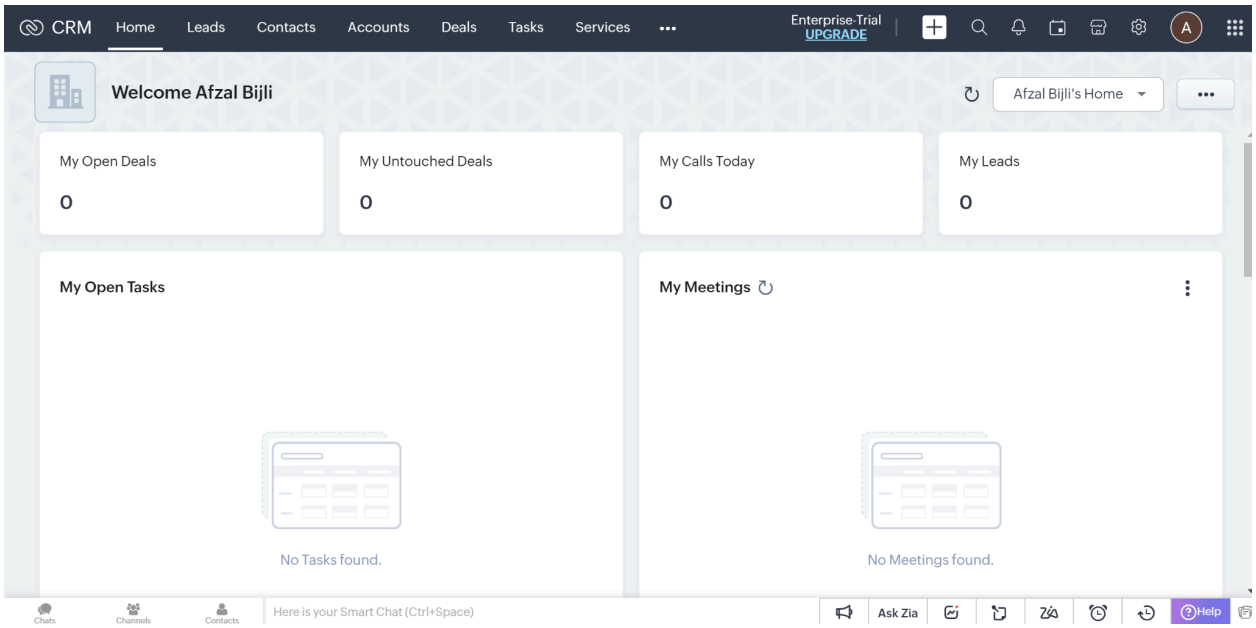
Date :

Aim

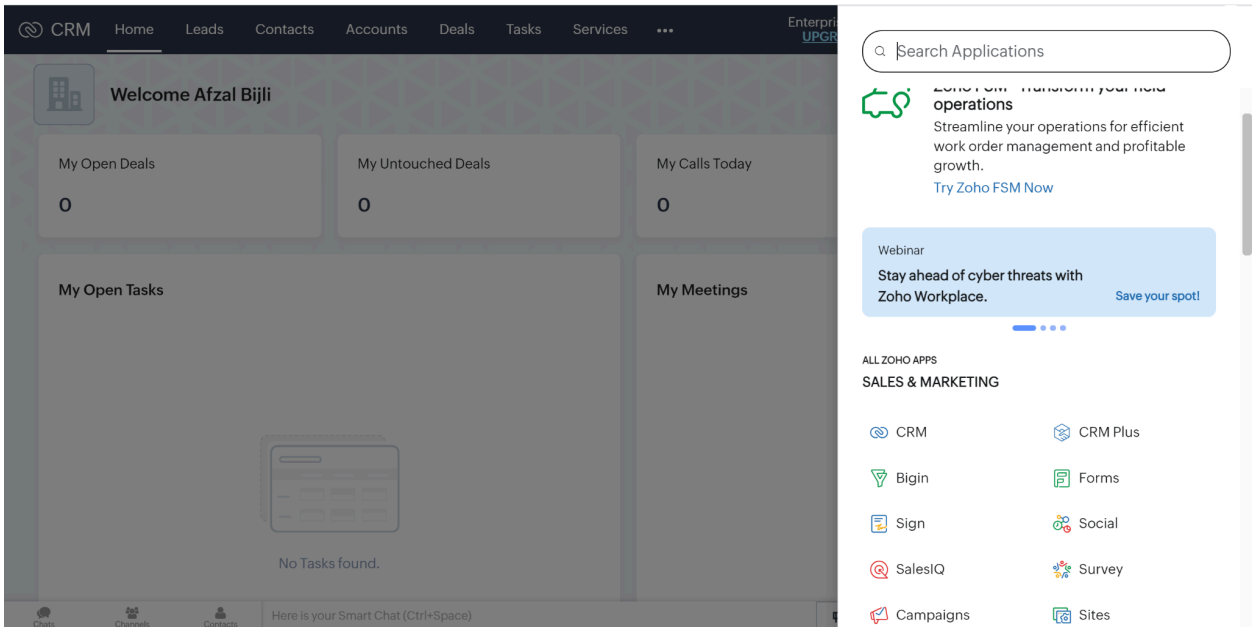
To customize your application using Zoho CRM

Algorithm

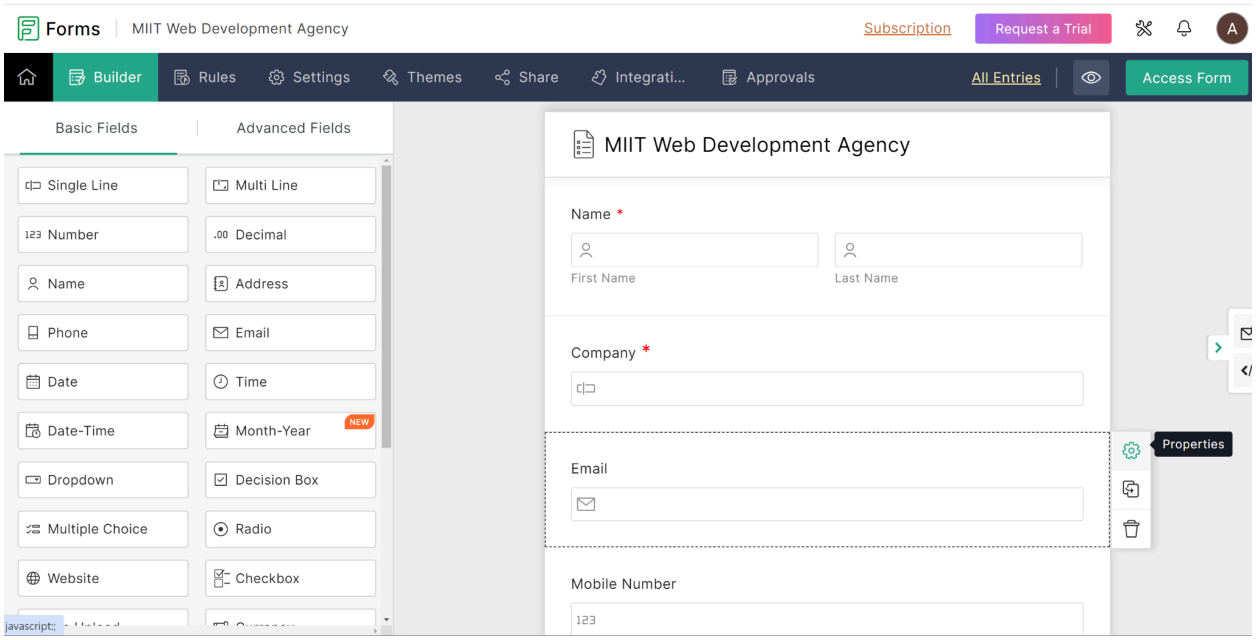
Step 1 : Goto **https://www.zoho.com/en-in/crm/** and signup for a new account



Step 2 : Select the Apps menu -> Sales & Marketing -> Forms -> New Form

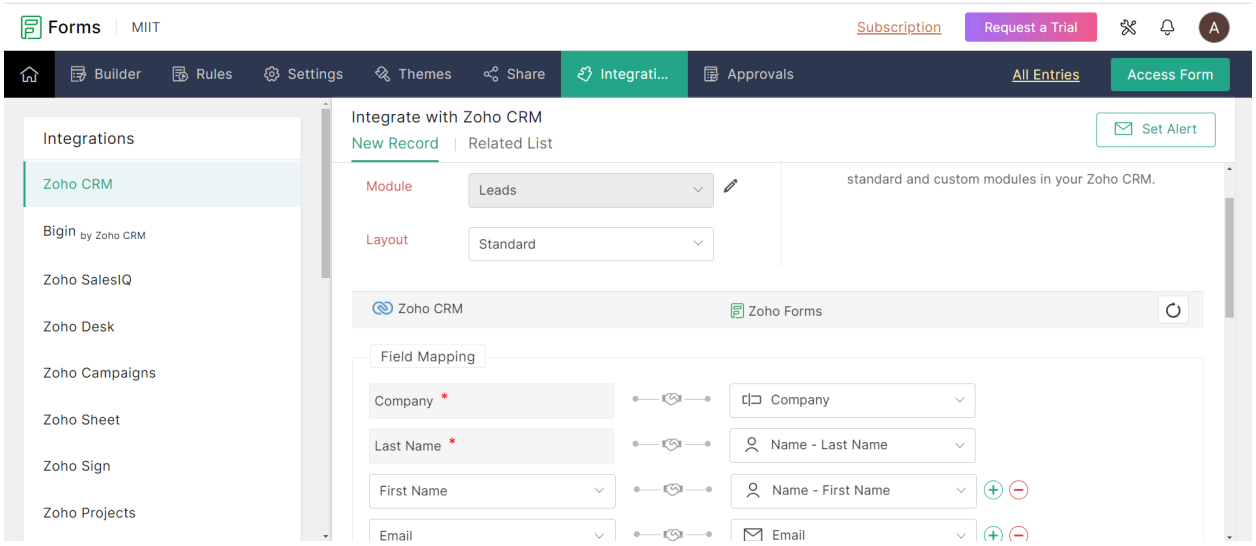


Step 3 : Create a form using drag and drop elements with various fields such as Name,Company, Email id, Mobile Number & Services offered using dropdown menu

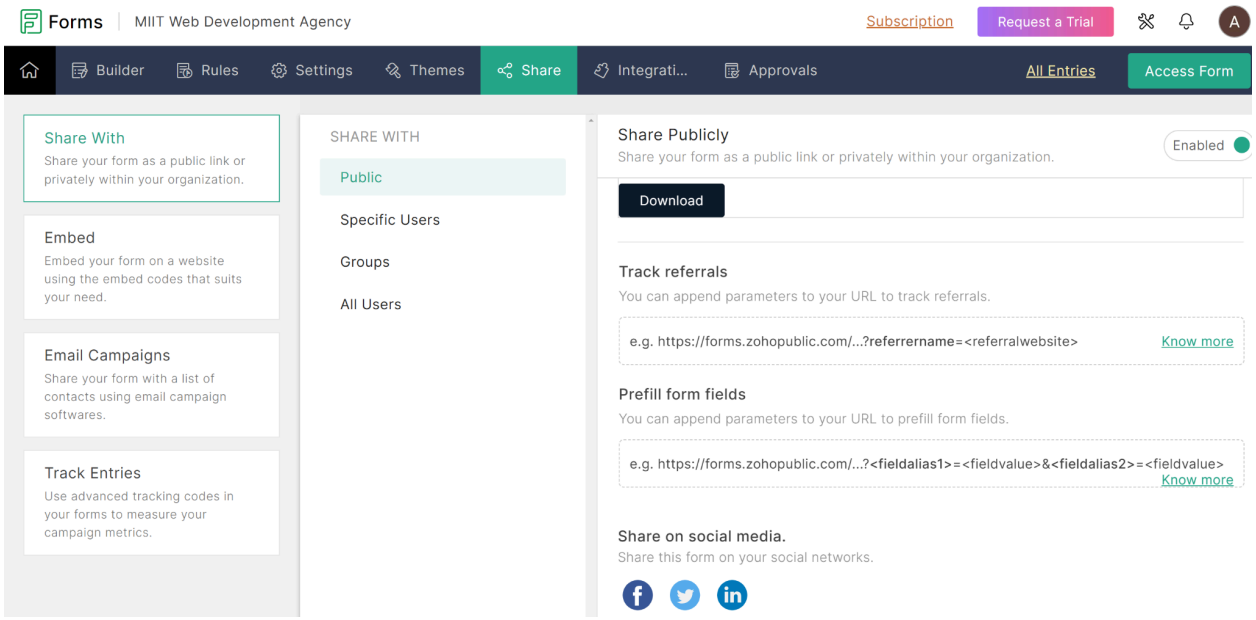


Step 4 : Click on preview form & select the theme

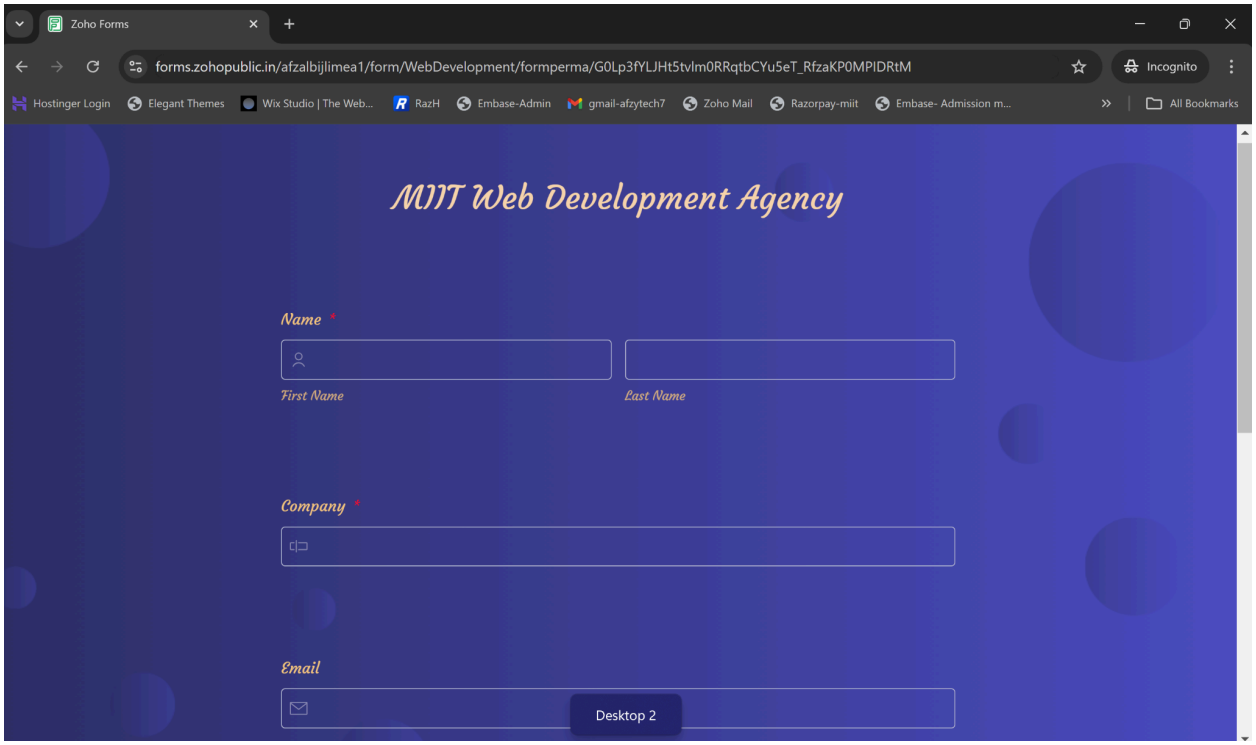
Step 5 : Click on integrate and do field mapping to successfully integrate the form into the lead module of Zoho CRM



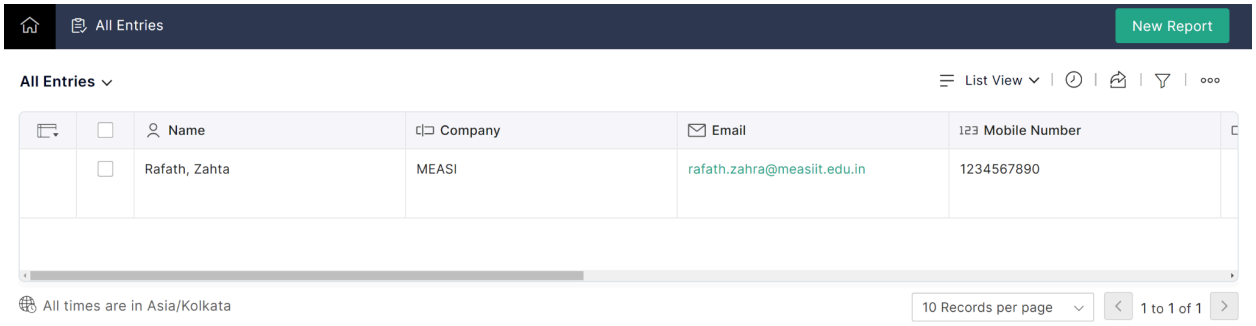
Step 6 : Click on the Share form & copy link



Step 7 : Open the link in another browser or in incognito mode and fill the form



Step 8 : Once form is filled by the client, you get the details into form entry



Result:

The customization of the application using Zoho CRM is successfully implemented.

Ex.No : 2 Write a script to create a MongoDB database and perform insert operation
Date :

Aim

To write a script to create a MongoDB database and perform insert operation

Algorithm

Step 1 : Open cmd prompt & type “mongosh” to connect with mongo db server

Step 2: Create a database & Collections

Step 3: Perform the insert operation

Step 4 : Display the output

Scripts Execution (Program)

1.To view databases

test> show dbs

```
admin> show dbs
admin     132.00 KiB
books      56.00 KiB
coll      296.00 KiB
```

2.Create New Database

test> use school

```
admin> use school
switched to db school
school>
```

3.Create Collection

```
school> db.createCollection("students")
{ ok: 1 }
school>
```

4.Show Collection Names

school> db.getCollectionNames()

```
school> db.getCollectionNames()
[ 'students' ]
school>
```

5.Get the details of content in collection

school> db.getCollectionInfos()

```
school> db.getCollectionInfos()
[
  {
    name: 'students',
    type: 'collection',
    options: {},
    info: {
      readOnly: false,
      uuid: UUID('9b0c2043-5fff-4fd4-92a9-c14eaa9de9ca')
    },
    idIndex: { v: 2, key: { _id: 1 }, name: '_id_' }
  }
]
school>
```

6.Insert the one document into the collection

school> db.students.insertOne({name:"afzal",age:20,gpa:8.9})

```
school> db.students.insertOne({name:"afzal",age:20,gpa:8.9})
{
  acknowledged: true,
  insertedId: ObjectId('67166cb7a48b3743a22710bf')
}
school>
```

7.Fetch the details from the collection

school> db.students.find()

```
school> db.students.find()
[
  {
    _id: ObjectId('67166cb7a48b3743a22710bf'),
    name: 'afzal',
    age: 20,
    gpa: 8.9
  }
]
school>
```

8.Insert more than one document into the collection

school>db.students.insertMany([{name:"karthik",age:20,gpa:3.4},{name:"rafaz",age:23,gpa:3.4,marks:77},{name:"kareem",age:24,gpa:3.4}])

```
school> db.students.insertMany([{name:"karthik",age:20,gpa:3.4},{name:"rafaz",age:23,gpa:3.4,marks:77},{name:"kareem",age:24,gpa:3.4}])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('67166d7fa48b3743a22710c0'),
    '1': ObjectId('67166d7fa48b3743a22710c1'),
    '2': ObjectId('67166d7fa48b3743a22710c2')
  }
}
school>
```

9.Fetch the details from the collection

school> db.students.find()

```
school> db.students.find()
[
  {
    _id: ObjectId('67166cb7a48b3743a22710bf'),
    name: 'afzal',
    age: 20,
    gpa: 8.9
  },
  {
    _id: ObjectId('67166d7fa48b3743a22710c0'),
    name: 'karthik',
    age: 20,
    gpa: 3.4
  },
  {
    _id: ObjectId('67166d7fa48b3743a22710c1'),
    name: 'rafaz',
    age: 23,
    gpa: 3.4,
    marks: 77
  },
  {
    _id: ObjectId('67166d7fa48b3743a22710c2'),
    name: 'kareem',
    age: 24,
    gpa: 3.4
  }
]
```

Result

The Script to create a MongoDB database and perform insert operation is successfully implemented.

Ex.No : 3 **Write a MongoDB script to perform query operations**
Date :

Aim

To write a MongoDB script to perform query operations

Algorithm

Step 1 : Open cmd prompt & type “mongosh” to connect with mongo db server

Step 2: Create a database & Collections

Step 3: Perform the query operations

Step 4 : Display the output

Commands to remember

- **show dbs -** To view the available database
- **use databasename -** To go into a database
- **use test -** to come to the main
- **use newdatbasename -** To go into a database which was not created before
- **show collections -** To show the collections inside a Database
- **db.collectionname.find() -** to view documents
- **db.collectionname.insertOne({}) -** To insert a Document
- **db.collectionname.insertMany([{},{},{}]) -** To insert many documents in the collection

Scripts Execution (Program)

Note:

Here from the previous program let's assume we have already created a database named “school” , a collection named “students” & inserted four documents into it.

1.Filtering/Querying

- **db.collectionname.find({key:value, key:value })**

```
school> db.students.find()
[
  {
    _id: ObjectId('67166cb7a48b3743a22710bf'),
    name: 'afzal',
    age: 20,
    gpa: 8.9
  },
  {
    _id: ObjectId('67166d7fa48b3743a22710c0'),
    name: 'karthik',
    age: 20,
    gpa: 3.4
  },
  {
    _id: ObjectId('67166d7fa48b3743a22710c1'),
    name: 'rafaz',
    age: 23,
    gpa: 3.4,
    marks: 77
  },
  {
    _id: ObjectId('67166d7fa48b3743a22710c2'),
    name: 'kareem',
    age: 24,
    gpa: 3.4
  }
]
```

- **db.collectionname.find({key:value, key:value},{key:1})**

(Gives you only the details what you ask for in the second argument and not all the details after filtering based on the first argument)

```
school> db.students.find({age:20},{name:1})
[
  { _id: ObjectId('67166cb7a48b3743a22710bf'), name: 'afzal' },
  { _id: ObjectId('67166d7fa48b3743a22710c0'), name: 'karthik' }
]
```

- **db.collectionname.find({},{key:1})**
  (Gives you the only the details what you ask for and not all the details without any filter)

```
school> db.students.find({},{name:1})
[
  { _id: ObjectId('67166cb7a48b3743a22710bf'), name: 'afzal' },
  { _id: ObjectId('67166d7fa48b3743a22710c0'), name: 'karthik' },
  { _id: ObjectId('67166d7fa48b3743a22710c1'), name: 'rafaz' },
  { _id: ObjectId('67166d7fa48b3743a22710c2'), name: 'kareem' }
]
```

- **db.collectionname.findOne({key:value})**

```
school> db.students.findOne({name:"rafaz"})
{
  _id: ObjectId('67166d7fa48b3743a22710c1'),
  name: 'rafaz',
  age: 23,
  gpa: 3.4,
  marks: 77
}
```

2.Method Chaining

(Following one method after another)

- **db.collectionname.find().count()**

```
school> db.students.find().count()
4
```

3.Sorting (Ascending & Descending)

- **db.collectionname.find().sort({key:1})** - Ascending

```
school> db.students.find().sort({age:1})
[
  {
    _id: ObjectId('67166cb7a48b3743a22710bf'),
    name: 'afzal',
    age: 20,
    gpa: 8.9
  },
  {
    _id: ObjectId('67166d7fa48b3743a22710c0'),
    name: 'karthik',
    age: 20,
    gpa: 3.4
  },
  {
    _id: ObjectId('67166d7fa48b3743a22710c1'),
    name: 'rafaz',
    age: 23,
    gpa: 3.4,
    marks: 77
  },
  {
    _id: ObjectId('67166d7fa48b3743a22710c2'),
    name: 'kareem',
    age: 24,
    gpa: 3.4
  }
]
```

- **db.collectionname.find().sort({key:-1})** - Descending

```
school> db.students.find().sort({age:-1})
[
  {
    _id: ObjectId('67166d7fa48b3743a22710c2'),
    name: 'kareem',
    age: 24,
    gpa: 3.4
  },
  {
    _id: ObjectId('67166d7fa48b3743a22710c1'),
    name: 'rafaz',
    age: 23,
    gpa: 3.4,
    marks: 77
  },
  {
    _id: ObjectId('67166cb7a48b3743a22710bf'),
    name: 'afzal',
    age: 20,
    gpa: 8.9
  },
  {
    _id: ObjectId('67166d7fa48b3743a22710c0'),
    name: 'karthik',
    age: 20,
    gpa: 3.4
  }
]
```

4.Operators in MongoDB

4.1 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.

- **Example - Greater than operator ($gt):**

```
school> db.students.find({age:{$gt:20}})
[
  {
    _id: ObjectId('67166d7fa48b3743a22710c1'),
    name: 'rafaz',
    age: 23,
    gpa: 3.4,
    marks: 77
  },
  {
    _id: ObjectId('67166d7fa48b3743a22710c2'),
    name: 'kareem',
    age: 24,
    gpa: 3.4
  }
]
```

4.2 Logical Operators

Logical

Name	Description
`$and`	Joins query clauses with a logical `AND` returns all documents that match the conditions of both clauses.
`$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.

- **Example - OR operator ($or):**

```
school> db.students.find({$or: [{age:20},{age:23}]})
[
  {
    _id: ObjectId('67166cb7a48b3743a22710bf'),
    name: 'afzal',
    age: 20,
    gpa: 8.9
  },
  {
    _id: ObjectId('67166d7fa48b3743a22710c0'),
    name: 'karthik',
    age: 20,
    gpa: 3.4
  },
  {
    _id: ObjectId('67166d7fa48b3743a22710c1'),
    name: 'rafaz',
    age: 23,
    gpa: 3.4,
    marks: 77
  }
]
```

Result:

The MongoDB script to perform query operations is successfully executed

Ex.No : 4 **Write a MongoDB Script to perform update operations**
Date :

Aim

To Write a MongoDB Script to perform update operations

Algorithm

Step 1 : Open MongoDB Compass

Step 2: Create a database & Collections

Step 3: Insert the documents into it using “Add Data” option

Step 4: Open cmd prompt & type “mongosh” to connect with mongo db server

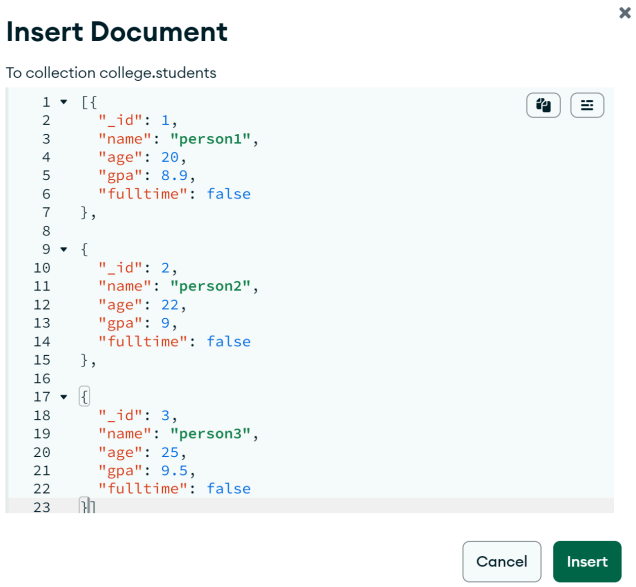
Step 5: Select the database which you created in compass

Step 6: Perform the update operations

Step 7 : Display the output

Scripts Execution (Program)

MongoDB Compass - Data Insertion



MongoDB Shell Commands

1.Select Database

admin > use college

```
admin> use college
switched to db college
```

college> show collections

```
college> show collections
students
college> db.students.find()
[
  { _id: 1, name: 'person1', age: 20, gpa: 8.9, fulltime: false },
  { _id: 2, name: 'person2', age: 22, gpa: 9, fulltime: false },
  { _id: 3, name: 'person3', age: 25, gpa: 9.5, fulltime: false }
]
```

college> db.students.find()

```
college> db.students.find()
[
  { _id: 1, name: 'person1', age: 20, gpa: 8.9, fulltime: false },
  { _id: 2, name: 'person2', age: 22, gpa: 9, fulltime: false },
  { _id: 3, name: 'person3', age: 25, gpa: 9.5, fulltime: false }
]
```

2.Update one value using $set operator, here we are setting fulltime to true

college> db.students.updateOne({name:"person1"},{$set:{fulltime:true}})

```
college> db.students.updateOne({name:"person1"},{$set:{fulltime:true}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

college> db.students.find({name:"person1"})

```
college> db.students.find({name:"person1"})
[ { _id: 1, name: 'person1', age: 20, gpa: 8.9, fulltime: true } ]
```

```
college> db.students.find()
[
  { _id: 1, name: 'person1', age: 20, gpa: 8.9, fulltime: true },
  { _id: 2, name: 'person2', age: 22, gpa: 9, fulltime: false },
  { _id: 3, name: 'person3', age: 25, gpa: 9.5, fulltime: false }
]
```

3.Update one value using $set operator, here we are setting fulltime to false again

db.students.updateOne({_id:1},{$set:{fulltime:false}})

```
college> db.students.updateOne({_id:1},{$set:{fulltime:false}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

```
college> db.students.find()
[
  { _id: 1, name: 'person1', age: 20, gpa: 8.9, fulltime: false },
  { _id: 2, name: 'person2', age: 22, gpa: 9, fulltime: false },
  { _id: 3, name: 'person3', age: 25, gpa: 9.5, fulltime: false }
]
```

3.Update one value using $unset operator, here we are unsetting fulltime field from id:1

college> db.students.updateOne({_id:1},{$unset:{fulltime:""}})

```
college> db.students.updateOne({_id:1},{$unset:{fulltime:""}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

```
college> db.students.find()
[
  { _id: 1, name: 'person1', age: 20, gpa: 8.9 },
  { _id: 2, name: 'person2', age: 22, gpa: 9, fulltime: false },
  { _id: 3, name: 'person3', age: 25, gpa: 9.5, fulltime: false }
]
```

4.Update Many values using $set operator, here we are setting fulltime field as false for all the documents

college> db.students.updateMany({},{$set:{fulltime:false}})

```
college> db.students.updateMany({},{$set:{fulltime:false}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 3,
  modifiedCount: 1,
  upsertedCount: 0
}
```

```
college> db.students.find()
[
  { _id: 1, name: 'person1', age: 20, gpa: 8.9, fulltime: false },
  { _id: 2, name: 'person2', age: 22, gpa: 9, fulltime: false },
  { _id: 3, name: 'person3', age: 25, gpa: 9.5, fulltime: false }
]
```

5.Here in id: 3, we are unsetting the fulltime field

college> db.students.updateOne({_id:3},{$unset:{fulltime:""}})

```
college> db.students.updateOne({_id:3},{$unset:{fulltime:""}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

```
college> db.students.find()
[
  { _id: 1, name: 'person1', age: 20, gpa: 8.9, fulltime: false },
  { _id: 2, name: 'person2', age: 22, gpa: 9, fulltime: false },
  { _id: 3, name: 'person3', age: 25, gpa: 9.5 }
]
```

6.Here we use $exists:false operator to check if value exists or not, wherever the value is not found it sets fulltime to true using $set operator

college> db.students.updateMany({fulltime:{$exists:false}},{$set:{fulltime:true}})

```
college> db.students.updateMany({fulltime:{$exists:false}},{$set:{fulltime:true}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
college> db.students.find()
[
  { _id: 1, name: 'person1', age: 20, gpa: 8.9, fulltime: false },
  { _id: 2, name: 'person2', age: 22, gpa: 9, fulltime: false },
  { _id: 3, name: 'person3', age: 25, gpa: 9.5, fulltime: true }
]
```

Result:

The MongoDB Script to perform update operations is successfully implemented.

Ex.No : 5 Write a MongoDB Script to Update Documents with Aggregation Pipeline
Date :

Aim

To update documents with aggregation pipeline

Algorithm

Step 1: Use insertMany() to add multiple student documents

Step 2: Start an aggregation pipeline on the students collection.

Step 3: Use $addFields to create a new field called totalScore by adding the math and science scores for each student.

Step 4: Use $set to add a new field called averageScore.

Step 5: Use $project to include only the name, totalScore, and averageScore in the output documents.

Step 6: Execute the aggregation pipeline to update and view student documents with calculated scores.

Script to Execute:

```
db.students.insertMany([
{
_id: 1,
name: 'Alice',
maths: 85,
science: 90,
address: 'Kingfield'
},
```

```
{
_id: 2,
name: 'Bob',
maths: 75,
science: 80,
address: 'Levay'
},
{
_id: 3,
name: 'Charlie',
maths: 95,
science: 85,
address: 'Borovil'
}
])

db.students.aggregate([
{
$addFields: {
totalScore: { $add: ["$maths", "$science"] }
}
},
{
```

```
$set: {
averageScore: { $divide: ["$totalScore", 2] }
}
},
{
$project: {
name: 1,
totalScore: 1,
averageScore: 1
}
}
])
```

Output:

```
record> db.students.insertMany([
...   {
...     _id: 1,
...     name: 'Alice',
...     maths: 85,
...     science: 90,
...     address: 'Kingfield'
...   },
...   {
...     _id: 2,
...     name: 'Bob',
...     maths: 75,
...     science: 80,
...     address: 'Levay'
...   },
...   {
...     _id: 3,
...     name: 'Charlie',
...     maths: 95,
...     science: 85,
...     address: 'Borovil'
...   }
... ])
{ acknowledged: true, insertedIds: { '0': 1, '1': 2, '2': 3 } }
```

```
record> db.students.aggregate([
...   {
...     $addFields: {
...       totalScore: { $add: ["$maths", "$science"] }
...     }
...   },
...   {
...     $set: {
...       averageScore: { $divide: ["$totalScore", 2] }
...     }
...   },
...   {
...     $project: {
...       name: 1,
...       totalScore: 1,
...       averageScore: 1
...     }
...   }
... ])
[
  { _id: 1, name: 'Alice', totalScore: 175, averageScore: 87.5 },
  { _id: 2, name: 'Bob', totalScore: 155, averageScore: 77.5 },
  { _id: 3, name: 'Charlie', totalScore: 180, averageScore: 90 }
]
```

Result:

The MongoDB Script to perform updates using the aggregation pipeline is successfully executed.

Ex.No : 6 Write a MongoDB Script to Delete Single and Multiple Documents
Date :

Aim

To write a MongoDB Script to delete single and multiple documents.

Algorithm

Step1: Use insertMany to add six employee documents to the employees collection with fields like _id, name, age, and designation.

Step 2: Call deleteOne to remove one employee document with the designation "Manager".

Step 3: Execute find() to display the current state of the employees collection after the deletion.

Step 4: Use deleteMany to remove all employee documents with the designation "Programmer".

Step 5: Execute find() again to show the final state of the employees collection after the bulk deletion.

Script to Execute:

```
db.employees.insertMany([
{ _id: 1, name: "Kumar", age: 35, designation: "Manager" },
{ _id: 2, name: "Ram", age: 35, designation: "Manager" },
{ _id: 3, name: "Stephen", age: 25, designation: "Programmer" },
{ _id: 4, name: "Subbu", age: 35, designation: "Programmer" },
{ _id: 5, name: "Anbu", age: 22, designation: "Programmer" },
{ _id: 6, name: "Veena", age: 22, designation: "Manager" }
])
//Deletes one document with the designation as manager
```

db.employees.deleteOne({designation:"Manager"})

//To view the output after deleteone

db.employees.find()

//Deletes all the documents with designation as programmer

db.employees.deleteMany({designation:"Programmer"})

//To view the output after deleteMany

db.employees.find()

Output:

```
record> db.employees.insertMany([
... { _id: 1, name: "Kumar", age: 35, designation: "Manager" },
... { _id: 2, name: "Ram", age: 35, designation: "Manager" },
... { _id: 3, name: "Stephen", age: 25, designation: "Programmer" },
... { _id: 4, name: "Subbu", age: 35, designation: "Programmer" },
... { _id: 5, name: "Anbu", age: 22, designation: "Programmer" },
... { _id: 6, name: "Veena", age: 22, designation: "Manager" }
... ])
{
  acknowledged: true,
  insertedIds: { '0': 1, '1': 2, '2': 3, '3': 4, '4': 5, '5': 6 }
}
```

```
record> db.employees.deleteOne({designation:"Manager"})
{ acknowledged: true, deletedCount: 1 }
record> db.employees.find()
[
  { _id: 2, name: 'Ram', age: 35, designation: 'Manager' },
  { _id: 3, name: 'Stephen', age: 25, designation: 'Programmer' },
  { _id: 4, name: 'Subbu', age: 35, designation: 'Programmer' },
  { _id: 5, name: 'Anbu', age: 22, designation: 'Programmer' },
  { _id: 6, name: 'Veena', age: 22, designation: 'Manager' }
]
```

```
record> db.employees.deleteMany({designation:"Programmer"})
{ acknowledged: true, deletedCount: 3 }
record>

record> //To view the output after deleteMany

record> db.employees.find()
[
  { _id: 2, name: 'Ram', age: 35, designation: 'Manager' },
  { _id: 6, name: 'Veena', age: 22, designation: 'Manager' }
]
```

Result:

The MongoDB Script to delete single and multiple documents is successfully executed.

Ex.No : 7 Write a MongoDB Script to Perform String Aggregation Operations
Date :

Aim

To write a MongoDB Script to perform string aggregation operations.

Algorithm

Step1: Select the database.

Step 2: Create the collection (if it doesn't exist).

Step 3: Insert multiple documents into the collection using insertMany.

Step 4: Perform aggregation on the collection using the aggregate method.

Step 5: Use the $project stage to apply various transformations (e.g., $concat, $split, $strcasecmp, $substr, $toLower, $toUpper).

Step 6: View the aggregated results to verify the output.

Script to Execute:

```
//Insert Documents in Database
 db.people.insertMany([
 { "_id": 1, "fullName": "Alice Johnson", "age": 25 },
 { "_id": 2, "fullName": "Bob Smith", "age": 30 },
 { "_id": 3, "fullName": "Charlie Brown", "age": 35 }
 ])

//View the inserted documents
db.people.find()

//String Aggregation
db.people.aggregate([
  {
    $project: {
      _id: 0,
      fullName: 1,
```

```
//Concatenate a String
    greeting: { $concat: ["Hello, ", "$fullName", "!"] },

// Split Full Name
    nameParts: { $split: ["$fullName", " "] },

// Compare FullName with Alice Johnson
    nameComparison: { $strcasecmp: ["$fullName", "ALICE JOHNSON"] },

// Create a substring with first 5 letters
    nameSubstring: { $substr: ["$fullName", 0, 5] },

// Print Lower Case
    lowerCaseName: { $toLower: "$fullName" },

// Print Upper Case
    upperCaseName: { $toUpper: "$fullName" }
  }
 }
])
```

Output:

```
record> db.people.insertMany([{ "_id": 1, "fullName": "Alice Johnson", "age": 25 }, { "_id": 2, "fullName": "Bob
Smith",
 "age": 30 }, { "_id": 3, "fullName": "Charlie Brown", "age": 35 }] )
{ acknowledged: true, insertedIds: { '0': 1, '1': 2, '2': 3 } }
```

```
record> db.people.find()
[
  { _id: 1, fullName: 'Alice Johnson', age: 25 },
  { _id: 2, fullName: 'Bob Smith', age: 30 },
  { _id: 3, fullName: 'Charlie Brown', age: 35 }
]
```

```
record> db.people.aggregate([
...   {
...     $project: {
...       _id: 0,
...       fullName: 1,
...       greeting: {
...         $concat: ["Hello, ", "$fullName", "!"]
...       },
...       nameParts: {
...         $split: ["$fullName", " "]
...       },
...       nameComparison: {
...         $strcasecmp: ["$fullName", "ALICE JOHNSON"]
...       },
...       nameSubstring: {
...         $substr: ["$fullName", 0, 5]
...       },
...       lowerCaseName: {
...         $toLower: "$fullName"
...       },
...       upperCaseName: {
...         $toUpper: "$fullName"
...       }
...     }
...   }
... ])
```

```
[
  {
    fullName: 'Alice Johnson',
    greeting: 'Hello, Alice Johnson!',
    nameParts: [ 'Alice', 'Johnson' ],
    nameComparison: 0,
    nameSubstring: 'Alice',
    lowerCaseName: 'alice johnson',
    upperCaseName: 'ALICE JOHNSON'
  },
  {
    fullName: 'Bob Smith',
    greeting: 'Hello, Bob Smith!',
    nameParts: [ 'Bob', 'Smith' ],
    nameComparison: 1,
    nameSubstring: 'Bob S',
    lowerCaseName: 'bob smith',
    upperCaseName: 'BOB SMITH'
  },
  {
    fullName: 'Charlie Brown',
    greeting: 'Hello, Charlie Brown!',
    nameParts: [ 'Charlie', 'Brown' ],
    nameComparison: 1,
    nameSubstring: 'Charl',
    lowerCaseName: 'charlie brown',
    upperCaseName: 'CHARLIE BROWN'
  }
]
```

Result:

The MongoDB script to perform string aggregation operation is successfully executed.

Ex.No : 8 **Design a Data Model for MongoDB using DbVisualizer**
Date :

Aim

To design a data model for MongoDB using DbVisualizer

Algorithm

Step 1: Create a user account to connect with DBvisualizer

- Open mongodb shell
- Choose admin database
  - use admin
- Create user (Note: set any username and password of your choice)

```
db.createUser({
user: "myuser",
pwd: "mypassword",
roles: [{ role: "readWrite", db: "mydatabase" }]
})
```

Step 2 : Open the DBVisualizer & Click to add new database connection

Step 3 : Choose the mongodb driver from the dropdown menu

Step 4 : Enter the the user name & password that was created in **step 1** to connect with mongoDB server

Step 5 : Once the connection is established successfully, open the New Sql Commander from the top menu in the DBVisualizer & execute the mongoBD commands

Scripts Execution in DBVisualizer (Program)

1.Create Database

use shop

2.Create collections

db.createCollection("users")
db.createCollection("products")
db.createCollection("orders")

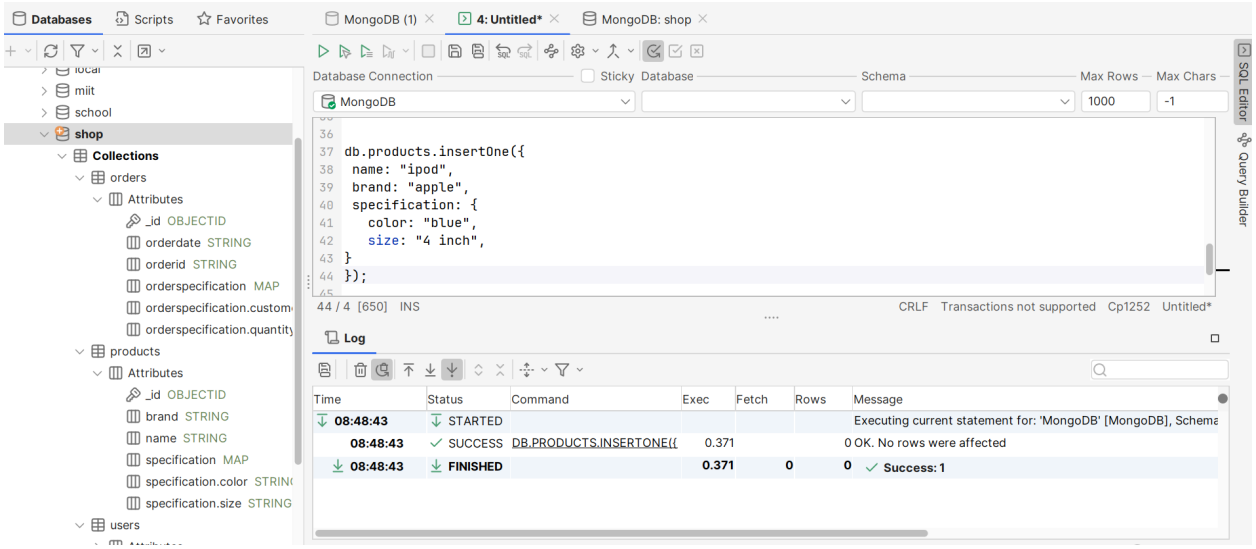
3.Insert Values into the Collections

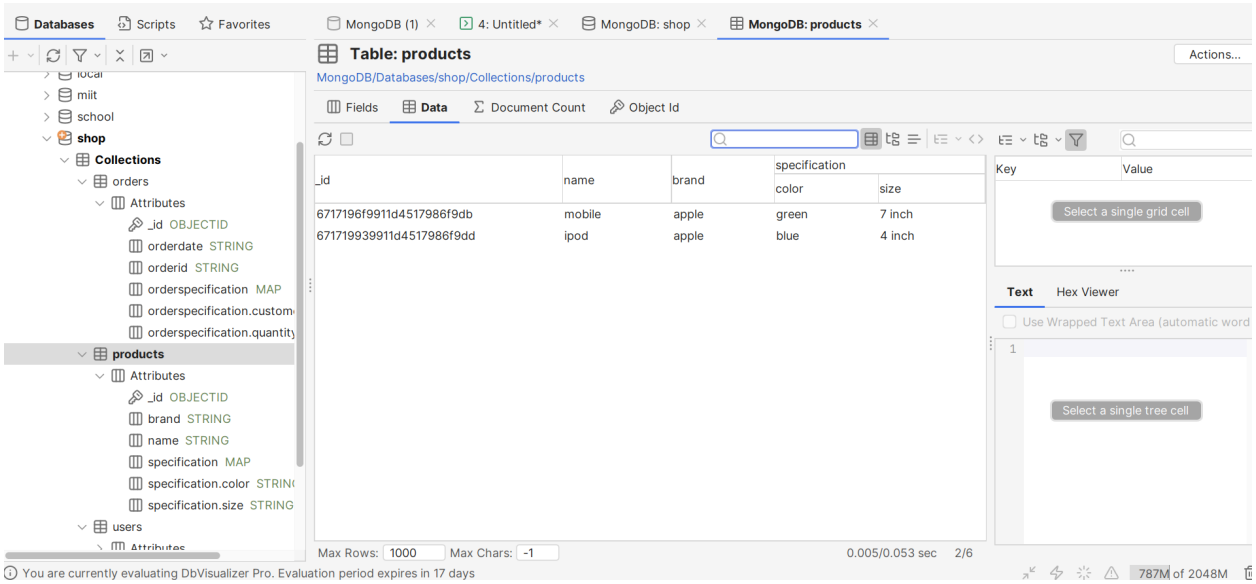
```
db.users.insertOne({
name: "John Doe",
email: "john.doe@example.com",
address: {
street: "123 Main St",
  city: "Anytown",
  state: "CA",
  zip: "12345"
}
});

db.products.insertOne({
name: "mobile",
brand: "apple",
specification: {
  color: "green",
  size: "7 inch",
}
});

db.orders.insertOne({
orderid: "123",
orderdate: "12 june",
orderspecification: {
  quantity: "10",
  customername: "karthick",
}
});


db.products.insertOne({
name: "ipod",
brand: "apple",
specification: {
  color: "blue",
  size: "4 inch",
}
});
```





Result:

The design of Data Model for MongoDB using DbVisualizer is successfully implemented.

Ex.No : 9 **Perform CRUD operations using DbVisualizer**
Date :

Aim

To perform CRUD operations using DbVisualizer

Algorithm

Step 1: Create a user account to connect with DBvisualizer

- Open mongodb shell
- Choose admin database
  - use admin
- Create user (Note: set any username and password of your choice)

```
db.createUser({
user: "myuser",
pwd: "mypassword",
roles: [{ role: "readWrite", db: "mydatabase" }]
})
```

Step 2 : Open the DBVisualizer & Click to add new database connection

Step 3 : Choose the mongodb driver from the dropdown menu

Step 4 : Enter the the user name & password that was created in **step 1** to connect with mongoDB server

Step 5 : Once the connection is established successfully, open the New Sql Commander from the top menu in the DBVisualizer & execute the mongoBD commands

Scripts Execution in DBVisualizer (Program)

1.Create Database

use shop

2.Create collections

db.createCollection("users")
db.createCollection("products")
db.createCollection("orders")

3.Insert Values into the Collections

```
db.users.insertOne({
name: "John Doe",
email: "john.doe@example.com",
address: {
street: "123 Main St",
  city: "Anytown",
  state: "CA",
  zip: "12345"
}
});

db.products.insertOne({
name: "mobile",
brand: "apple",
specification: {
  color: "green",
  size: "7 inch",
}
});

db.orders.insertOne({
orderid: "123",
orderdate: "12 june",
orderspecification: {
  quantity: "10",
  customername: "karthick",
}
});


db.products.insertOne({
name: "ipod",
brand: "apple",
specification: {
  color: "blue",
  size: "4 inch",
}
});
```

Table: products

MongoDB/Databases/shop/Collections/products

Fields | Data | Document Count | Object Id

_id	name	brand	specification	
			color	size
6717196f9911d4517986f9db	mobile	apple	green	7 inch
671719939911d4517986f9dd	ipod	apple	blue	4 inch

Table: orders

MongoDB/Databases/shop/Collections/orders

Fields | Data | Document Count | Object Id

_id	orderid	orderdate	orderspecification	
			quantity	customername
6717197c9911d4517986f9dc	123	12 june	10	karthick

4.Read Operation

db.products.find();

46 / 1 [652] INS CRLF Transactions not supp

Log | 1: db.products.find() [2]

_id	name	brand	specification	
			color	size
6717196f9911d4517986f9db	mobile	apple	green	7 inch
671719939911d4517986f9dd	ipod	apple	blue	4 inch

5.Update Operation

```
db.products.updateOne(
 { name: "ipod" },
 { $set: { age: 26 } }
)
```

db.products.find();

50 / 1 [675] INS CRLF Transactions

Log 1: db.products.updateOne({ name:... [2]

_id	name	brand	specification		age
			color	size	
6717196f9911d4517986f9db	mobile	apple	green	7 inch	
671719939911d4517986f9dd	ipod	apple	blue	4 inch	26

7.Delete Operation

db.products.deleteOne({name: "ipod"});

db.products.find();

59 / 13 [818] INS

Log 1: db.products.find() [1]

_id	name	brand	specification	
			color	size
6717196f9911d4517986f9db	mobile	apple	green	7 inch

Result:

The CRUD operations using DbVisualizer is successfully implemented.

Ex.No : 10 Create a Zoho CRM account and organize your Tasks, Meetings and Deals
Date :

Aim

To create a Zoho CRM account and organize Tasks, Meetings and Deals

Algorithm

Step 1: Visit Zoho CRM Website

Step 2: Create a Zoho CRM Account:
- Click on the "Sign Up" button.
- Fill in the required details (name, email, etc.) and complete the registration Process.

Step 3: Login to Zoho CRM:
-Once registered, log in to your Zoho CRM account using your credentials.

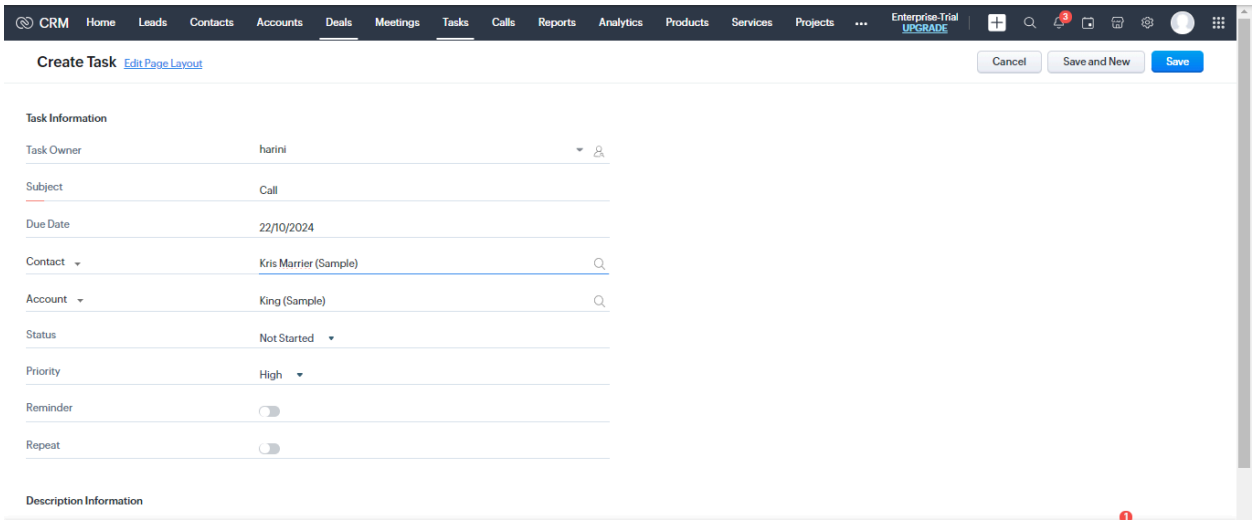
Step 4: Set Up Your Organization:
-Define your organization's details in the CRM, such as the company name and other relevant information.

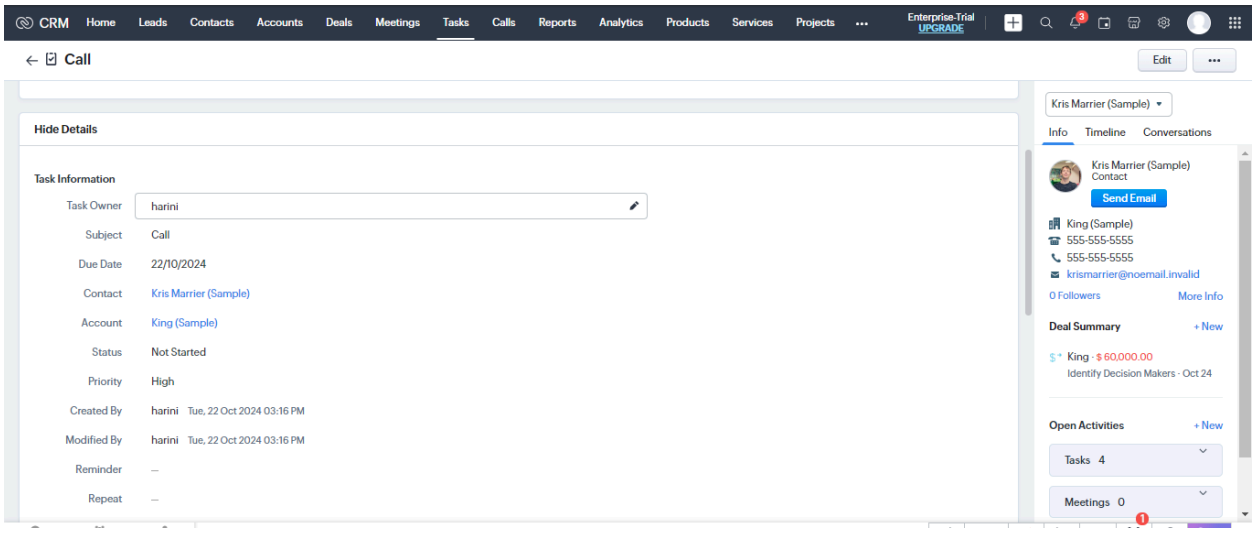
Step 5: Organize Tasks:
- From the dashboard, go to the "Tasks" module.
- Click on "Create Task" and enter the details (title, description, due date, etc.).
- Assign the task to team members or yourself and set reminders if needed.

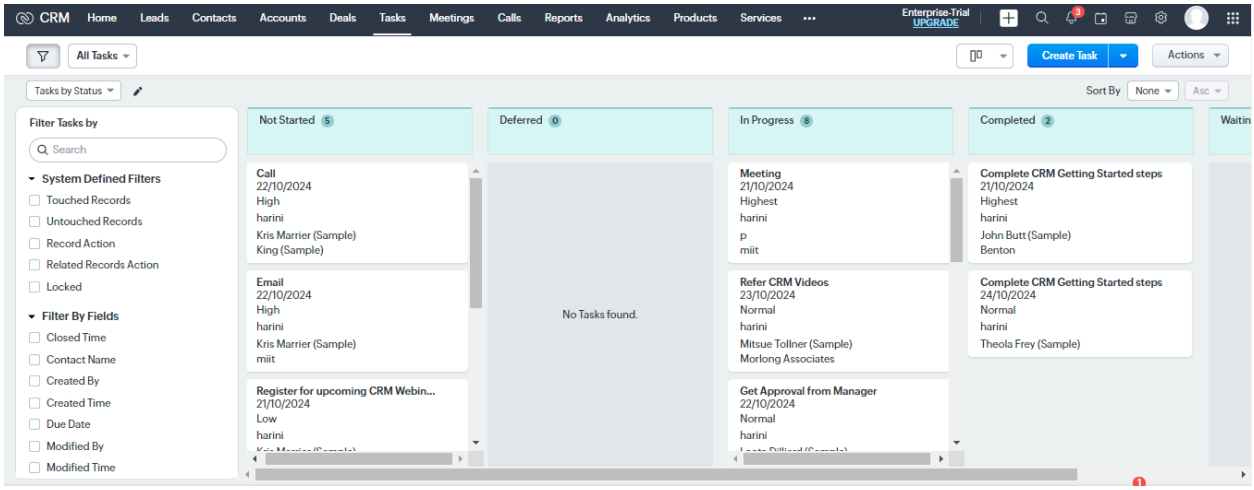
Step 6: Organize Meetings:
- In the CRM dashboard, go to the "Meetings" module (or "Events").
-Schedule a new meeting by entering the meeting title, time, location, and invitees.
-Add reminders and attach relevant documents or links if necessary.

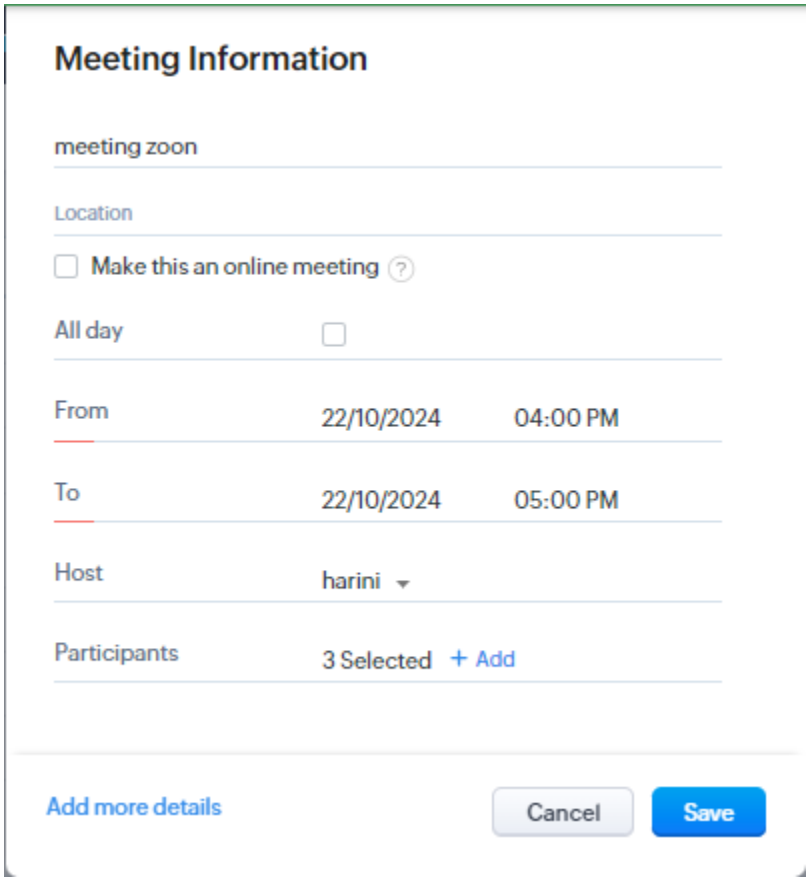
Step 7: Organize Deals:
- Go to the "Deals" module.
- Click "Add New Deal" and fill in the deal details (deal name, amount, expected closing date, etc.).
- Assign the deal to a team member and associate it with the relevant contacts or accounts.

Output:


CRM
Home
Leads
Contacts
Accounts
Deals
Meetings
Tasks
Calls
Reports
Analytics
Products
Services
Projects
Enterprise-Trial
UPGRADE
Create Task
Edit Page Layout
Cancel
Save and New
Save
Task Information
Task Owner
harini
Subject
Call
Due Date
22/10/2024
Contact
Kris Marrier (Sample)
Account
King (Sample)
Status
Not Started
Priority
High
Reminder
Repeat
Description Information


CRM
Home
Leads
Contacts
Accounts
Deals
Meetings
Tasks
Calls
Reports
Analytics
Products
Services
Projects
Enterprise-Trial
UPGRADE
Call
Edit
Hide Details
Task Information
Task Owner
harini
Subject
Call
Due Date
22/10/2024
Contact
Kris Marrier (Sample)
Account
King (Sample)
Status
Not Started
Priority
High
Created By
harini Tue, 22 Oct 2024 03:16 PM
Modified By
harini Tue, 22 Oct 2024 03:16 PM
Reminder
Repeat
Kris Marrier (Sample)
Info
Timeline
Conversations
Kris Marrier (Sample)
Contact
Send Email
King (Sample)
555-555-5555
555-555-5555
krismarrier@noemail.invalid
0 Followers
More Info
Deal Summary
+ New
King · $ 60,000.00
Identify Decision Makers · Oct 24
Open Activities
+ New
Tasks 4
Meetings 0


CRM
Home
Leads
Contacts
Accounts
Deals
Tasks
Meetings
Calls
Reports
Analytics
Products
Services
Enterprise-Trial
UPGRADE
All Tasks
Create Task
Actions
Tasks by Status
Sort By
None
Asc
Filter Tasks by
Search
System Defined Filters
Touched Records
Untouched Records
Record Action
Related Records Action
Locked
Filter By Fields
Closed Time
Contact Name
Created By
Created Time
Due Date
Modified By
Modified Time
Not Started 5
Deferred 0
In Progress 8
Completed 2
Call
22/10/2024
High
harini
Kris Marrier (Sample)
King (Sample)
Email
22/10/2024
High
harini
Kris Marrier (Sample)
miit
Register for upcoming CRM Webin...
21/10/2024
Low
harini
No Tasks found.
Meeting
21/10/2024
Highest
harini
p
miit
Refer CRM Videos
23/10/2024
Normal
harini
Mitsue Tollner (Sample)
Morlong Associates
Get Approval from Manager
22/10/2024
Normal
harini
Complete CRM Getting Started steps
21/10/2024
Highest
harini
John Butt (Sample)
Benton
Complete CRM Getting Started steps
24/10/2024
Normal
harini
Theola Frey (Sample)


Meeting Information
meeting zoon
Location
Make this an online meeting
All day
From
22/10/2024
04:00 PM
To
22/10/2024
05:00 PM
Host
harini
Participants
3 Selected
+ Add
Add more details
Cancel
Save

CRM Home Leads Contacts Accounts Deals Meetings Tasks Calls Reports Analytics Products Services Projects ... Enterprise-Trial UPGRADE

All Meetings

Meeting created successfully. View details

Create Meeting | Actions

Total Records 12

10 Records Per Page • 1 - 10

Filter Meetings by

Search

System Defined Filters

- Touched Records
- Untouched Records
- Record Action
- Related Records Action

Filter By Fields

- All day
- Check-In Address
- Check-In By
- Check-In City
- Check-In Country
- Check-In State
- Check-In Sub-Locality
- Check-In Time
- Checked In Status

Title All	From	To	Related To	Contact Name	Host
meeting zoon	22/10/2024 04:00 PM	22/10/2024 05:00 PM			harini
harinimeeting	22/10/2024 04:00 PM	22/10/2024 05:00 PM			harini
New Meeting	21/10/2024 01:00 PM	21/10/2024 02:00 PM			harini
Demo	21/10/2024 03:00 PM	21/10/2024 04:00 PM	Printing Dimensions	Donette Foller (Sample)	harini
Webinar	21/10/2024 05:00 PM	21/10/2024 06:00 PM	Commercial Press (Sample)	Leota Dilliard (Sample)	harini
TradeShow	21/10/2024	21/10/2024	Chemel	James Venere (Sample)	harini
Webinar	21/10/2024 04:00 PM	21/10/2024 07:00 PM	Chanay (Sample)	Josephine Darakjy (Sample)	harini
Seminar	21/10/2024 03:00 PM	21/10/2024 05:00 PM	Carissa Kidman (Sample)		harini
Attend Customer conference	21/10/2024	21/10/2024	Feltz Printing Service	Capla Paprocki (Sample)	harini
CRM Webinar	21/10/2024 02:00 PM	21/10/2024 04:00 PM	Morlong Associates	Mitsue Tollner (Sample)	harini

CRM Home Leads Contacts Accounts Deals Meetings Tasks Calls Reports Analytics Products Services Projects ... Enterprise-Trial UPGRADE

Create Deal

Edit Page Layout

Cancel | Save and New | Save

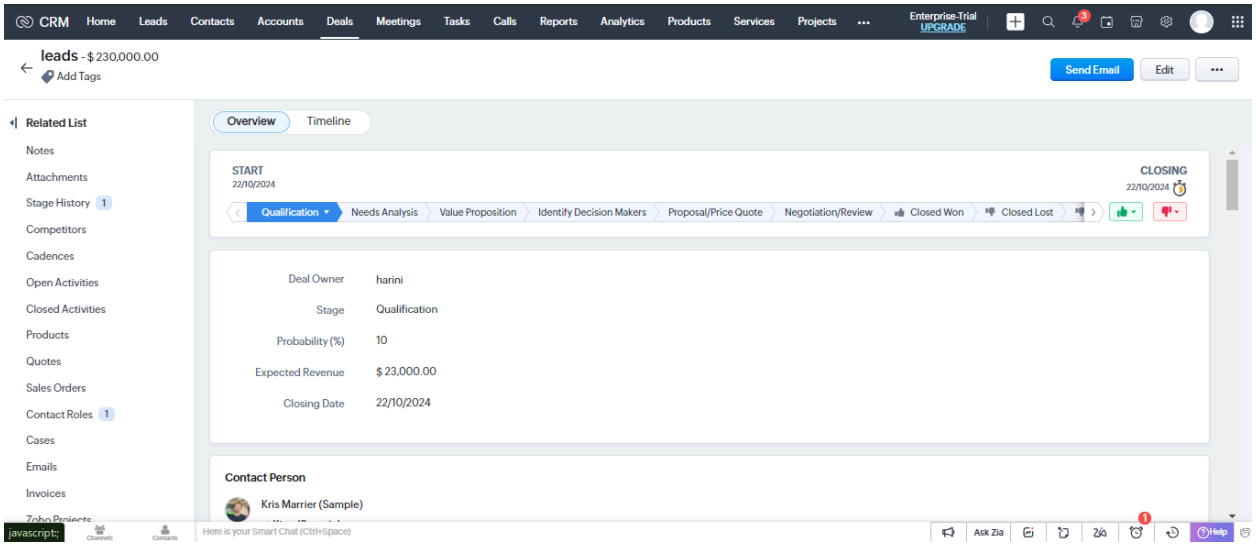
Deal Information

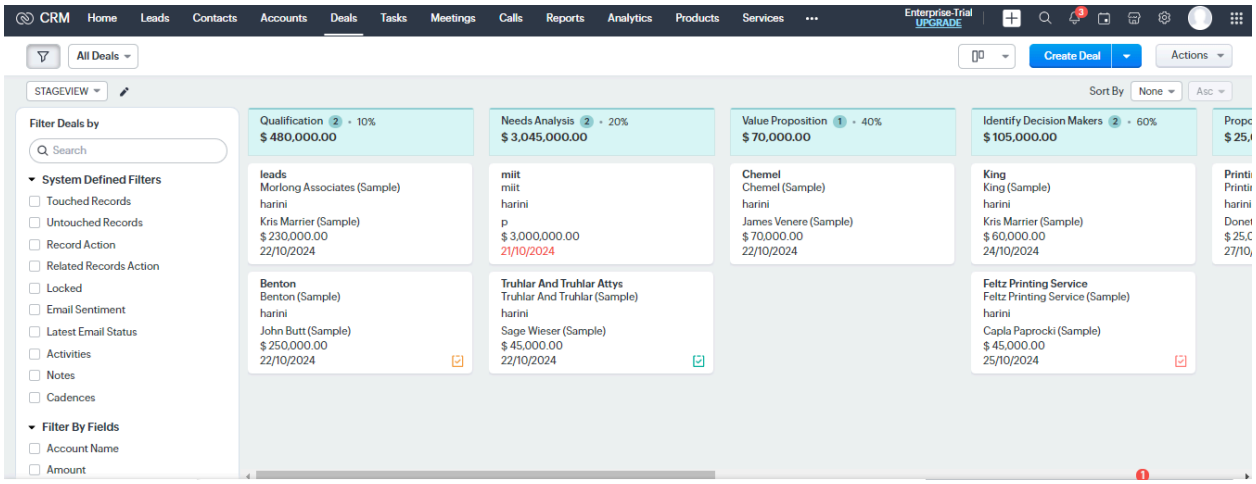
Deal Owner	harini	Amount	$ 230000
Deal Name	leads	Closing Date	22/10/2024
Account Name	Morlong Associates (Sample)	Stage	Qualification
Type	-None-	Probability (%)	10
Next Step		Expected Revenue	$ 23000.00
Lead Source	-None-	Campaign Source	
Contact Name	Kris Marrier (Sample)		

Description Information

Description

Client Script





Result:

The Zoho CRM account was created and the organization of Tasks, Meeting and Deals is successfully implemented.

Ex.No : 11 **Create and Maintain a Project using Zoho CRM Features**
Date :

Aim

To create and maintain a project using Zoho CRM feature.

Algorithm

Step 1: Log in to Zoho CRM

Step 2: Navigate to Projects Module

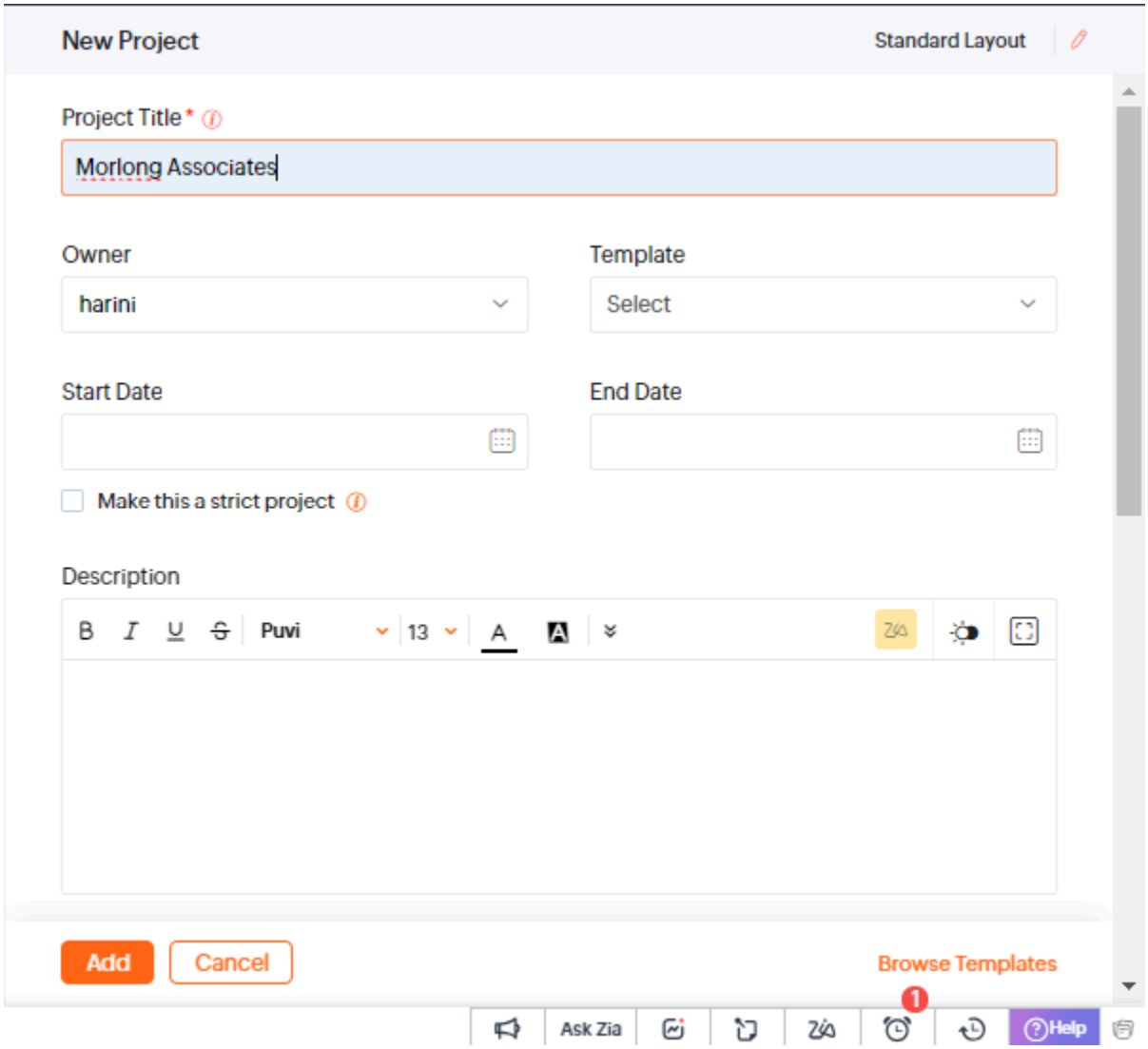
Step 3: Create a New Project
- Click on the Create New button .
- Fill in the required details such as:
  Project Name
  Project Description
  Start Date
  End Date
  Owner.

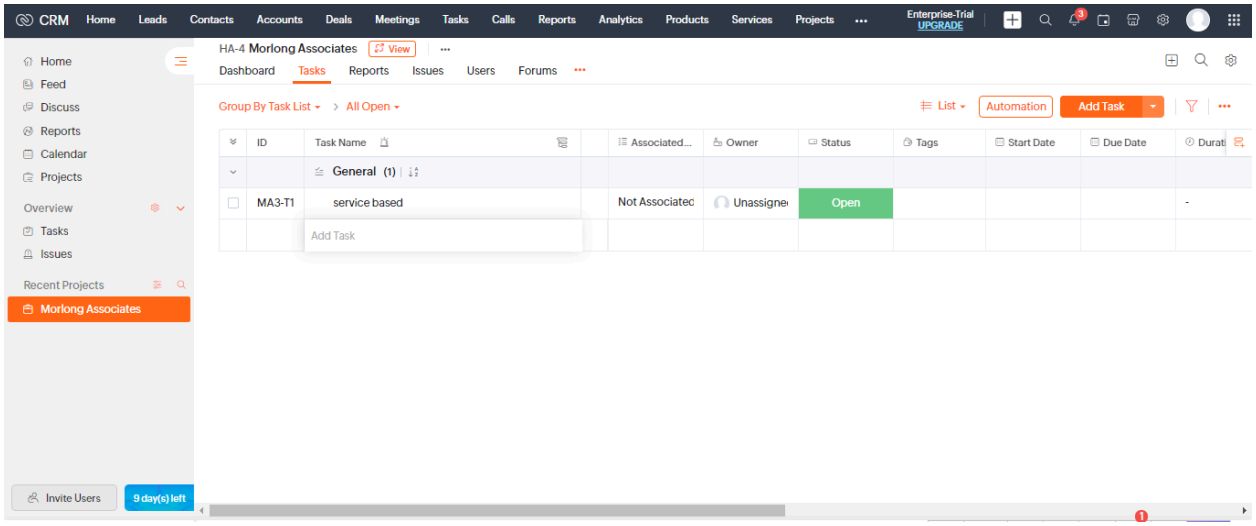
Step 4: Save the Project

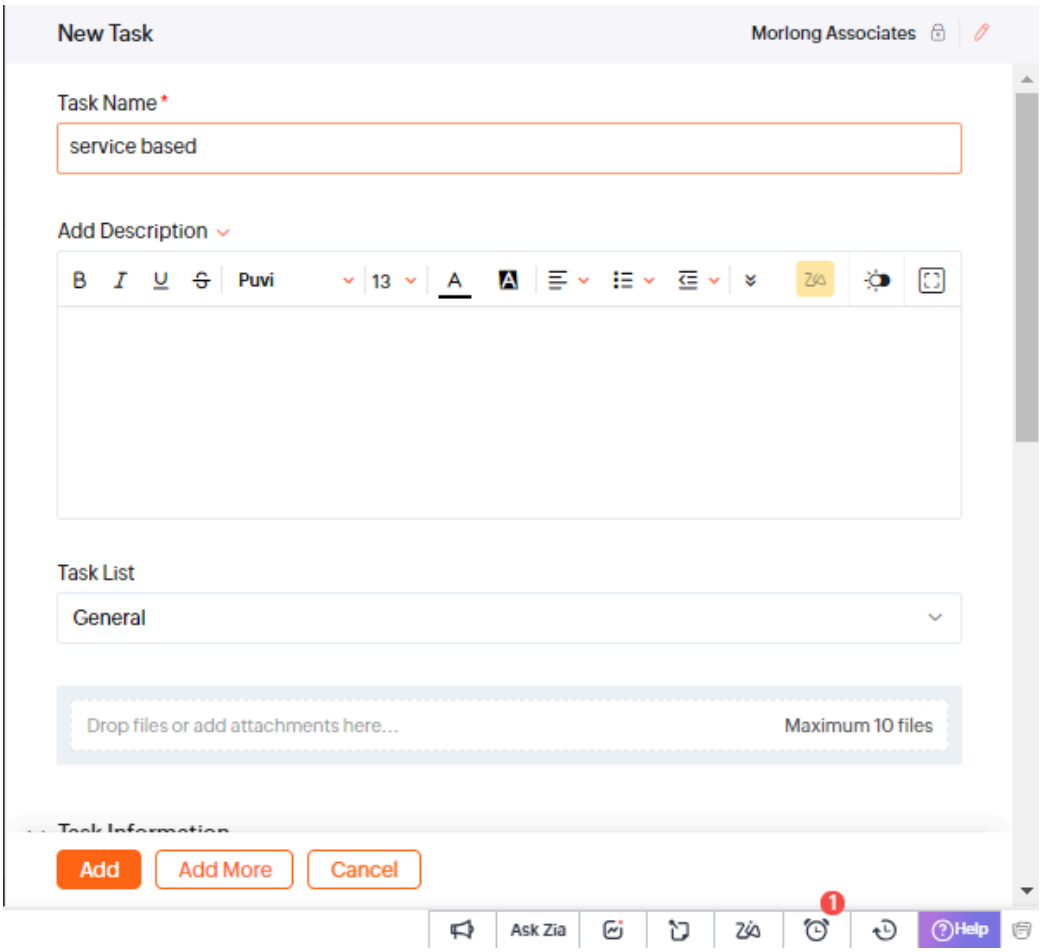
Step 5: Add Tasks to the Project
- Navigate to the Tasks section within the project you just created.
- Click on Add Task or New Task.
- Fill in the task details:
- Task Name.
- Task Description.
- Due Date.
- Priority.
- Assign To.

Step 6: Save the task for maintaining the project.

Output


New Project
Standard Layout
Project Title *
Morlong Associates
Owner
harini
Template
Select
Start Date
End Date
Make this a strict project
Description
Puvi
13
Add
Cancel
Browse Templates
Ask Zia
Help





Result

The creation of project and maintenance using Zoho CRM features is successfully implemented.