To connect to mongo shell

open the command prompt type the

mongosh command

A computer screen shot of a black screen

AI-generated content may be incorrect.

To view the database  
A black screen with white text

AI-generated content may be incorrect.

To create a database

use the command

**use bookDb**

book DB database created and its acting as a current database

Now try with the **show dbs;**  
 the bookDB will not be displayed as it do not have any collection created in it

A screen shot of a computer program

AI-generated content may be incorrect.

Now Let us see how to create a collection in MongoDB

we can perform the CRUD operation same like other database

Now we are going to create a collection called books

Within the book collection we are going to insert a new document

**db.books.insertOne({**

**title: "MongoDB Tutorial",**

**published\_year: 2020**

**})**

A screenshot of a computer screen

AI-generated content may be incorrect.

**Read**

To select all the documents in a collection, you use the find method:

**bookdb> db.books.find();**

A screenshot of a computer

AI-generated content may be incorrect.

To select one the documents in a collection, you use the findOne() method:

**db.books.findOne();**

**A screenshot of a computer

AI-generated content may be incorrect.**

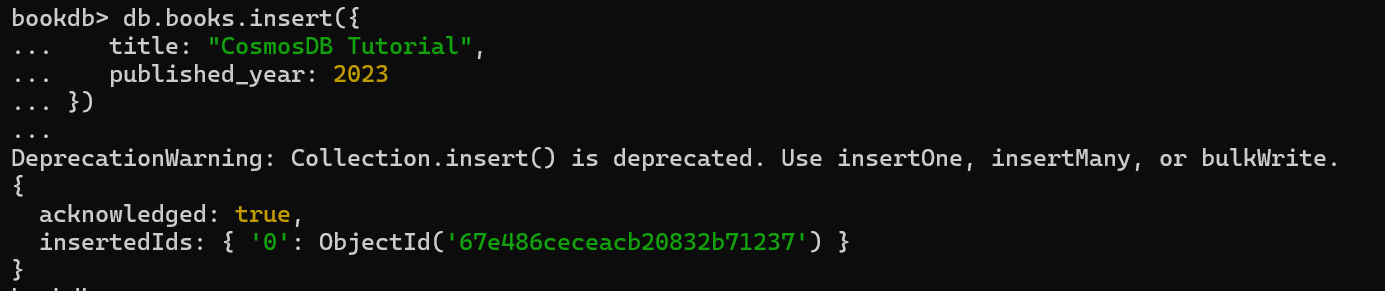
**Insert the new document**db.books.insertOne({

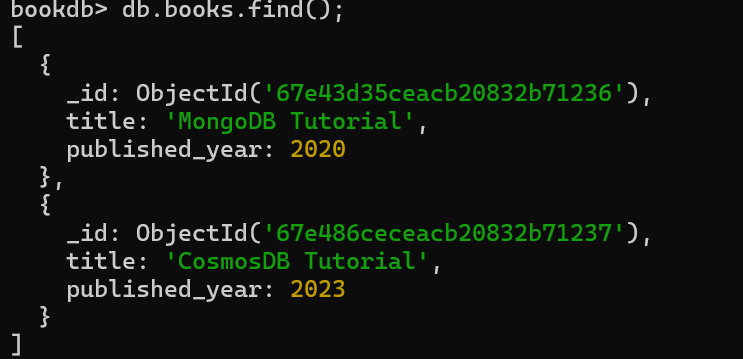
title: "CosmosDB Tutorial",

published\_year: 2023

})

**Note : DeprecationWarning: Collection.insert() is deprecated.**

****

**To find   
**

**To find the first record  
bookdb> db.books.findOne();  
  
A screenshot of a computer

AI-generated content may be incorrect.**

**Pretty:** in older version when we try to find the record it was displaying the key value as a single row  
 to display the key values in the document in a different row we use pretty() .But in new version its not required **db.books.find().pretty();  
  
A computer screen with text and numbers

AI-generated content may be incorrect.**

**Column 🡪 Field  
  
Update:  
  
To update the single field  
  
db.books.updateOne(**

**... { title: "MongoDB Tutorial"},**

**... { $set: { published\_year: 2019 }}**

**... )**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**Check the data is updated using find()  
db.books.find();**

**Delete:  
bookdb> db.books.deleteOne({title: "MongoDB Tutorial"});  
**

**Inserting Multiple records:  
  
InsertMany  
  
bookdb>** db.employees.insertMany([{

... "\_id": 1,

... "name": "Alice",

... "age": 30,

... "department": "HR",

... "salary": 5000,

... "skills": ["communication", "management"],

... "active": true

... },

... {

... "\_id": 2,

... "name": "Bob",

... "age": 25,

... "department": "IT",

... "salary": 7000,

... "skills": ["javascript", "nodejs", "mongodb"],

... "active": false

... },

... {

... "\_id": 3,

... "name": "Charlie",

... "age": 35,

... "department": "Finance",

... "salary": 6500,

... "skills": ["excel", "accounting"],

... "active": true

... },

... {

... "\_id": 4,

... "name": "Diana",

... "age": 28,

... "department": "IT",

... "salary": 7200,

... "skills": ["python", "mongodb"],

... "active": true

... }])  
**A screenshot of a computer program

AI-generated content may be incorrect.**  
  
**## 🔍 MongoDB Query Operators with Examples**

---

**\*\*Explanation\*\***: Finds employees whose `department` is exactly "IT".

**### 1. `$eq` (equal to)**

db.employees.find({ department: { $eq: "IT" } })

A screen shot of a computer

AI-generated content may be incorrect.

---

**### 2. `$ne` (not equal to)**

```js

db.employees.find({ department: { $ne: "HR" } })

```

**\*\*Explanation\*\***: Finds all employees not in the "HR" department.

A screen shot of a computer

AI-generated content may be incorrect.

---

**### 3. `$gt` (greater than)**

```js

db.employees.find({ age: { $gt: 30 } })

```

**\*\*Explanation\*\***: Finds employees older than 30.

A computer screen with text and images

AI-generated content may be incorrect.

---

**### 4. `$gte` (greater than or equal to)**

```js

db.employees.find({ salary: { $gte: 7000 } })

```

**\*\*Explanation\*\***: Finds employees whose salary is 7000 or more.

A computer screen shot of a black screen with white text

AI-generated content may be incorrect.

---

**### 5. `$lt` (less than)**

```js

db.employees.find({ salary: { $lt: 6000 } })

```

**\*\*Explanation\*\***: Finds employees with salary less than 6000.

A screen shot of a computer

AI-generated content may be incorrect.

---

**### 6. `$lte` (less than or equal to)**

```js

db.employees.find({ age: { $lte: 28 } })

```

**\*\*Explanation\*\***: Finds employees aged 28 or less.

A computer screen shot of a code

AI-generated content may be incorrect.

---

**### 7. `$in` (match any value from an array)**

```js

db.employees.find({ department: { $in: ["IT", "Finance"] } })

```

**\*\*Explanation\*\***: Finds employees in either IT or Finance departments.

A computer screen shot of a black screen

AI-generated content may be incorrect.

---

**### 8. `$nin` (not in array)**

```js

db.employees.find({ department: { $nin: ["HR"] } })

```

**\*\*Explanation\*\***: Finds employees whose department is not HR.

A computer screen shot of a program code

AI-generated content may be incorrect.

---

**### 9. `$and` (combine multiple conditions)**

```js

db.employees.find({ $and: [ { age: { $gt: 25 } }, { department: "IT" } ] })

```

**\*\*Explanation\*\***: Finds IT employees older than 25.

A black screen with white text

AI-generated content may be incorrect.

---

**### 10. `$or` (either condition)**

```js

db.employees.find({ $or: [ { age: { $lt: 28 } }, { salary: { $gt: 7000 } } ] })

```

**\*\*Explanation\*\***: Finds employees younger than 28 OR with salary over 7000.

A computer screen with green and yellow text

AI-generated content may be incorrect.

---

**### 11. `$not` (negate condition)**

```js

db.employees.find({ age: { $not: { $gt: 30 } } })

```

**\*\*Explanation\*\***: Finds employees whose age is **\*\*not greater than\*\*** 30.

A screen shot of a computer

AI-generated content may be incorrect.

---

**### 12. `$exists` (check field existence)**

```js

db.employees.find({ skills: { $exists: true } })

```

**\*\*Explanation\*\***: Finds documents where the `skills` field exists.

A screenshot of a computer program

AI-generated content may be incorrect.

---

**### 13. `$type` (check field type)**

```js

db.employees.find({ age: { $type: "int" } })

```

**\*\*Explanation\*\***: Finds documents where `age` is of integer type.

A screen shot of a computer

AI-generated content may be incorrect.

---

**### 14. `$all` (match all elements in array)**

```js

db.employees.find({ skills: { $all: ["mongodb", "javascript"] } })

```

**\*\*Explanation\*\***: Finds employees who have **\*\*both\*\*** `mongodb` and `javascript` skills.

A screen shot of a computer

AI-generated content may be incorrect.

---

**### 15. `$size` (exact array length)**

```js

db.employees.find({ skills: { $size: 2 } })

```

**\*\*Explanation\*\***: Finds employees who have exactly 2 skills.

A screenshot of a computer program

AI-generated content may be incorrect.

---

**### 16. `$regex` (pattern matching)**

```js

db.employees.find({ name: { $regex: "^A", $options: "i" } })

```

**\*\*Explanation\*\***: Finds employees whose name starts with "A" (case-insensitive).

A screen shot of a computer

AI-generated content may be incorrect.

---

**### 17. `$elemMatch` (condition on array elements)**

```js

db.employees.find({ skills: { $elemMatch: { $eq: "mongodb" } } })

```

**\*\*Explanation\*\***: Finds employees whose skills include `mongodb`.

A screen shot of a computer

AI-generated content may be incorrect.

---