1. FIELD NAME AUR USKE AAGE EK VALUE .
   SLIDE NUMBER : 2

2. FIELD NAME AUR USKE AAGE EK DOCUMENT .
   SLIDE NUMBER : 4

3. FIELD NAME AUR USKE AAGE EK ARRAY .
   SLIDE NUMBER : 4

4. FIELD NAME AUR USKE AAGE EK ARRAY OF DOCUMENTS .
   SLIDE NUMBER : 3

==========

==========

# DISCUSSING EACH TYPE ONE BY ONE

# TYPE 1 ::
# EK FIELD NAME AUR USKE AAGE EK SIMPLE CHEEZ

# SLIDE NUMBER 2

| | | | |
|---|---|---|---|
| **Structure** | Contains multiple collections. | Contains multiple documents. | Contains key-value pairs (fields). |

## 1. Create a Database

## 2. Create a Collection

## 3. Insert a Document

**Create:**
Adding new documents to the collection.

**Read:**
Retrieving documents from collection.

**Update:**
Updating documents in collection.

**Delete:**
Removing documents from collection.

# 1. CREATE OPERATION

| Feature | Description |
| --- | --- |
| **Purpose** | Adds/Inserts a new document into a collection. |
| **Method Used** | `insertOne()`, `insertMany()`. |
| **Collection Handling** | If the collection does not exist, MongoDB **automatically creates it** when inserting a document. |
| **Bulk Insertion** | Supports inserting multiple documents at once. |

FIRST OF ALL .. UNDERSTAND ID WALI BAKCHODI ...

1. IF NOT ASSIGNED ::— THIS IS DEFAULT

```
_id: ObjectId("98232303df34948b4
```

2. IF ASSIGNED MANUALLY

```
_id: "user123"
```

```
_id: 1001
```

YAA TOH MANUALLY DE SAKTE HO YAA FIR AAPKI JO MONGO HAI WO KHUD HI ASSIGN KARDEGI …

CREATE HAS 2 THINGS :::—-

1. INSERTONE
2. INSERTMANY

==========

1. INSERT ONE

SYNTAX ::—

DATABASENAME.COLLECTIONNAME.insertOne({ name : “harry” , age : 34 })

JEHRA CONTENT HAI USS CH PEHLA TAH SIMPLE ROUND BRACKET … TE USS TOH BAAD

ANDER DOCUMENT LAGAN LAYI CURLY …. BAS KHATAM KYUNKI EK HI DOCUMENT INSERT KAR SAKTE HEIN

AGAR STRING HAI TOH DOUBLE QUOTES AND AGAR NUMBER HAI TOH NOTHING …

==========

2. INSERTMANY

CREATE MULTIPLE DOCUMENTS..
DB_NAME.COLL_NAME.
insertMany([{},{},{},{}])
CURLY BRACKETS KE ANDER DOCUMENTS BAN SAKTE HEIN
(END CH SEMICOLON)

==========

==========

# 2. UPDATE OPERATIONS

- updateOne()
- updateMany()
- replaceOne()

## A) UPDATEONE

| | |
|---|---|
| `db.collection` | Specifies the collection in which documents will be updated. |
| `<filter>` | A query that selects the documents to update. |
| `<update>` | The update operation using `$set`, `$currentDate`, `$inc`, etc. |

This method **updates the first matching document.**

```
db.users.updateOne(
  { name: "Bob" },
  {
    $set: { course: "CSE" },
    $currentDate: { lastModified: true }
  }
);
```

# B) UPDATEMANY

```
db.employees.updateMany(
  { salary: 70000 },
  {
    $set: { salary: 85000 }
  }
);
```

This method **updates all documents** that match the filter criteria.

==========

# UPDATE OPERATORS

## 1. Using `$currentDate` (Sets the field to the current date)

**Before Update:**

js Copy Edit

```
{
  _id: 1,
  name: "Alice",
  lastModified: "2024-02-01T12:00:00Z"
}
```

**Update Query:**

js Copy Edit

```
db.users.updateOne(
  { name: "Alice" },
  { $currentDate: { lastModified: true } }
);
```

**After Update:**

js Copy Edit

```
{
  _id: 1,
  name: "Alice",
  lastModified: "2025-02-06T15:25:00Z"  // Updated to current timestamp
}
```

## 3. Using `$min` (Updates a field only if the new value is smaller than the existing one)

**Before Update:**

```js
{
  _id: 3,
  name: "Charlie",
  salary: 60000
}
```

**Update Query:**

```js
db.users.updateOne(
  { name: "Charlie" },
  { $min: { salary: 55000 } }
);
```

**After Update (Salary Updated Since 55000 < 60000):**

```js
{
  _id: 3,
  name: "Charlie",
  salary: 55000  // Updated because 55000 < 60000
}
```

### 4. Using `$max` (Updates a field only if the new value is greater than the existing one)

**Before Update:**

```js
{
  _id: 4,
  name: "David",
  salary: 70000
}
```

**Update Query:**

```js
db.users.updateOne(
  { name: "David" },
  { $max: { salary: 80000 } }
);
```

**After Update (Salary Updated Since 80000 > 70000):**

```js
{
  _id: 4,
  name: "David",
  salary: 80000  // Updated because 80000 > 70000
}
```

## 2. Using `$inc` (Increments a field by a specified value)

**Before Update:**

```js
{
  _id: 2,
  name: "Bob",
  age: 25
}
```

**Update Query:**

```js
db.users.updateOne(
  { name: "Bob" },
  { $inc: { age: 5 } }
);
```

**After Update:**

```js
{
  _id: 2,
  name: "Bob",
  age: 30  // Age increased by 5
}
```

## 5. Using `$mul` (Multiplies the field value by a specified number)

**Before Update:**

```js
{
  _id: 5,
  name: "Eve",
  bonus: 2000
}
```

**Update Query:**

```js
db.users.updateOne(
  { name: "Eve" },
  { $mul: { bonus: 2 } }
);
```

**After Update (Bonus Multiplied by 2):**

```js
{
  _id: 5,
  name: "Eve",
  bonus: 4000  // Bonus doubled
}
```

### 7. Using `$set` (Sets a new field value or updates an existing field)

**Before Update:**

js Copy Edit

```
{
  _id: 7,
  name: "Grace",
  role: "Intern"
}
```

**Update Query:**

js Copy Edit

```
db.users.updateOne(
  { name: "Grace" },
  { $set: { role: "Manager" } }
);
```

**After Update (Role Updated):**

js Copy Edit

```
{
  _id: 7,
  name: "Grace",
  role: "Manager"  // Updated role
}
```

New field vi add kar salsa if not present .

## 6. Using `$rename` (Renames a field)

**Before Update:**

```js
{
  _id: 6,
  name: "Frank",
  department: "HR"
}
```

**Update Query:**

```js
db.users.updateOne(
  { name: "Frank" },
  { $rename: { department: "team" } }
);
```

**After Update (Field `department` Renamed to `team`):**

```js
{
  _id: 6,
  name: "Frank",
  team: "HR"  // Field renamed
}
```

## Difference Between `updateOne()` and `replaceOne()`

| Feature | `updateOne()` | `replaceOne()` |
| --- | --- | --- |
| Updates specific fields | ✅ Yes, using `$set` | ❌ No, replaces the entire document |
| Keeps existing fields | ✅ Yes | ❌ No, removes old fields |

C)

**Before Update:**

js Copy Edit

```
{
  _id: 101,
  module: "MongoDB",
  difficulty: "Intermediate",
  duration: "10 days"
}
```

**Update Query:**

js Copy Edit

```
db.course.replaceOne(
  { module: "MongoDB" },
  {
    module: "NoSQL MongoDB",
    time_days: 5,
    tags: "BD",
    description: "Basic database design"
  }
);
```

**After Update (Entire Document Replaced):**

js Copy Edit

```
{
  _id: 101,  // _id remains unchanged
  module: "NoSQL MongoDB",
  time_days: 5,
  tags: "BD",
  description: "Basic database design"
}
```

# REPLACE ONE

(Id wali bakchodi yaar rakhi hai)

POORA DA POORA DOC HI NAVA BAN JANDA … PICCHE AALE DA KOI ASTITV NI REHNDA ..

# 3. DELETE OPERATION

- The deleteOne() method removes a single document from a collection. It has a single required parameter which is a filter criteria to match a specific document to delete.
- db.courses.deleteOne({module: “java”})
- The .deleteMany() method removes all documents that match a given filter criteria. It takes in a single required parameter, the filter criteria to match multiple documents.
- db.courses.deleteMany({module: “java”})

==========

==========

# 4. FIND OPERATIONS

```
db.Rockers.find({ gender: "F", age: 19 })
```

# IT FINDS ALL

==========

==========

```
db.Rockers.findOne({ gender: "F" })
```

Result:

ONLY FIRST IN DOCUMENT

==========

==========