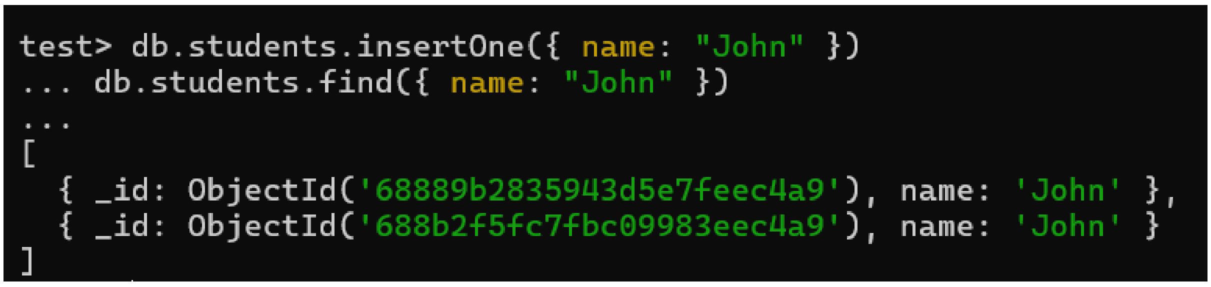
MongoDB Assignment – Data Types(24.07.25)

1. String

Query:

db.students.insertOne({ name: "John" })

db.students.find({ name: "John" })



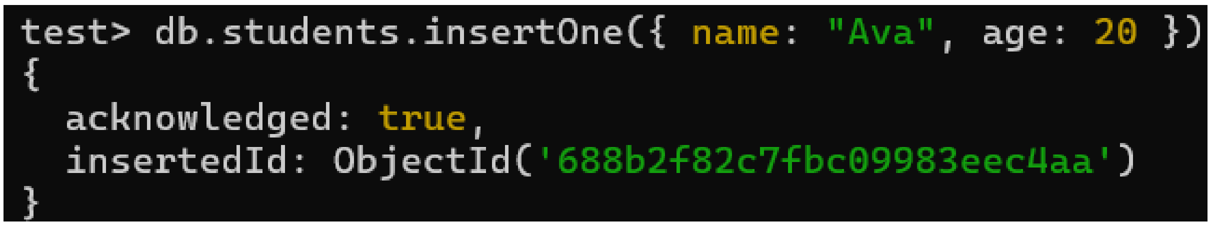
```
test> db.students.insertOne({ name: "John" })
... db.students.find({ name: "John" })
...
[
  { _id: ObjectId('68889b2835943d5e7feec4a9'), name: 'John' },
  { _id: ObjectId('688b2f5fc7fbc09983eec4a9'), name: 'John' }
]
```

Explanation:
Strings are used to store text values. MongoDB encodes them using UTF-8.

2. Integer

Query:

db.students.insertOne({ name: "Ava", age: 20 })



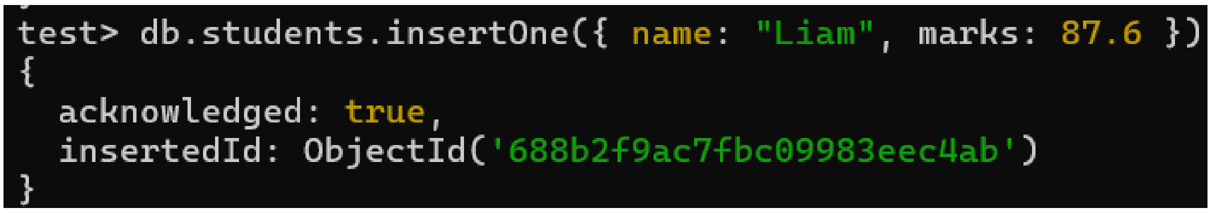
```
test> db.students.insertOne({ name: "Ava", age: 20 })
{
  acknowledged: true,
  insertedId: ObjectId('688b2f82c7fbc09983eec4aa')
}
```

Explanation:
Used to store whole numbers like age, count, etc. MongoDB supports both 32-bit and 64-bit integers.

3. Double

Query:

db.students.insertOne({ name: "Liam", marks: 87.6 })



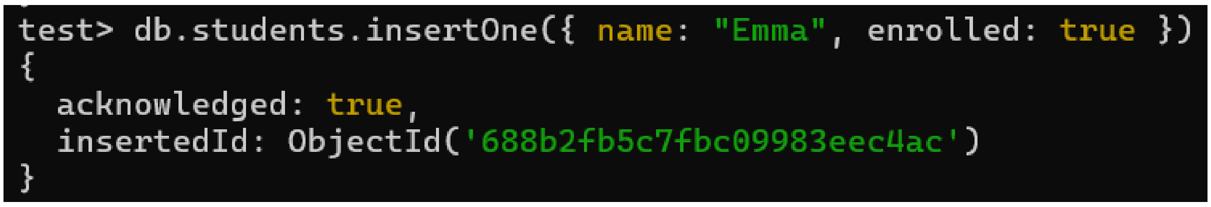
```
test> db.students.insertOne({ name: "Liam", marks: 87.6 })
{
  acknowledged: true,
  insertedId: ObjectId('688b2f9ac7fbc09983eec4ab')
}
```

Explanation:
Double is used for storing floating-point or decimal values such as percentages or prices.

4. Boolean

Query:

db.students.insertOne({ name: "Emma", enrolled: true })



```
test> db.students.insertOne({ name: "Emma", enrolled: true })
{
  acknowledged: true,
  insertedId: ObjectId('688b2fb5c7fbc09983eec4ac')
}
```

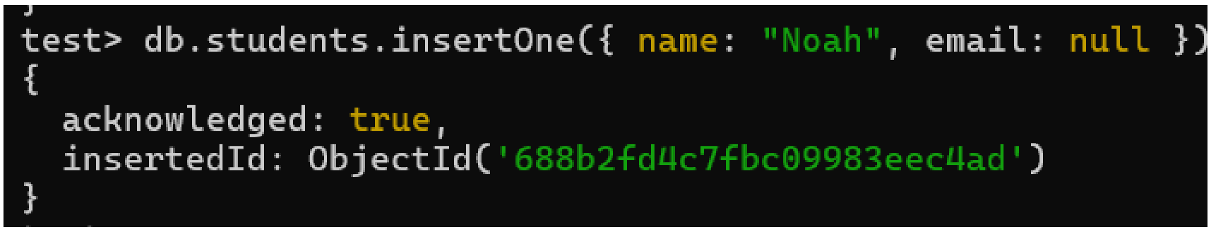
Explanation:

Represents true/false values — useful for storing logical flags.

5. Null

Query:

db.students.insertOne({ name: "Noah", email: null })



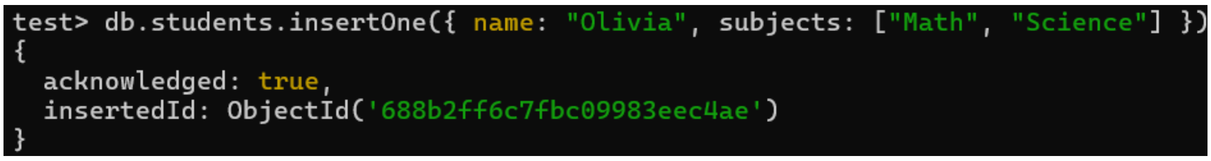
```
test> db.students.insertOne({ name: "Noah", email: null })
{
  acknowledged: true,
  insertedId: ObjectId('688b2fd4c7fbc09983eec4ad')
}
```

Explanation:
Null indicates an intentional absence of a value, such as when data is missing.

6. Array

Query:

db.students.insertOne({ name: "Olivia", subjects: ["Math", "Science"] })

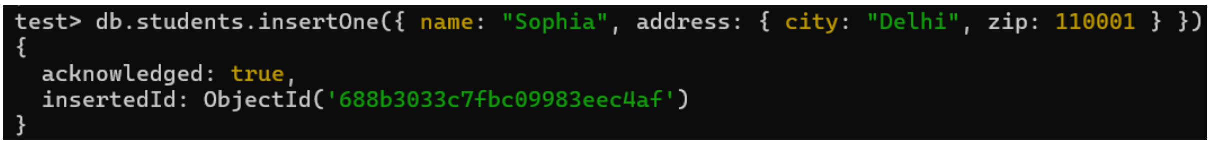

```
test> db.students.insertOne({ name: "Olivia", subjects: ["Math", "Science"] })
{
  acknowledged: true,
  insertedId: ObjectId('688b2ff6c7fbc09983eec4ae')
}
```

Explanation:
Arrays can store multiple values — strings, numbers, objects, or mixed types.

7. Embedded Document

Query:

db.students.insertOne({ name: "Sophia", address: { city: "Delhi", zip: 110001 } })


```
test> db.students.insertOne({ name: "Sophia", address: { city: "Delhi", zip: 110001 } })
{
  acknowledged: true,
  insertedId: ObjectId('688b3033c7fbc09983eec4af')
}
```

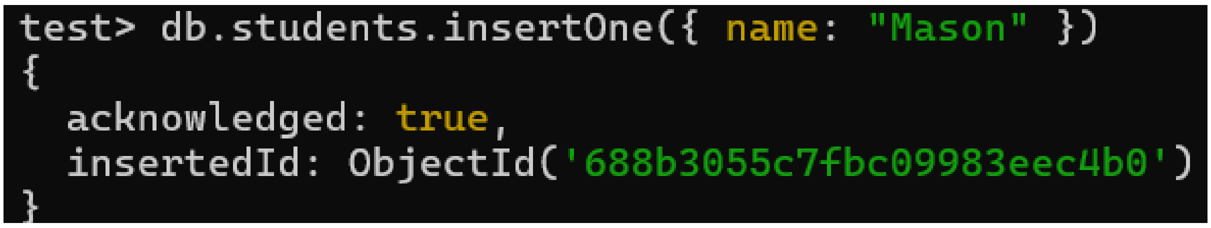
Explanation:

Allows nested objects inside a document, making related data easy to manage together.

8. ObjectId

Query:

db.students.insertOne({ name: "Mason" })

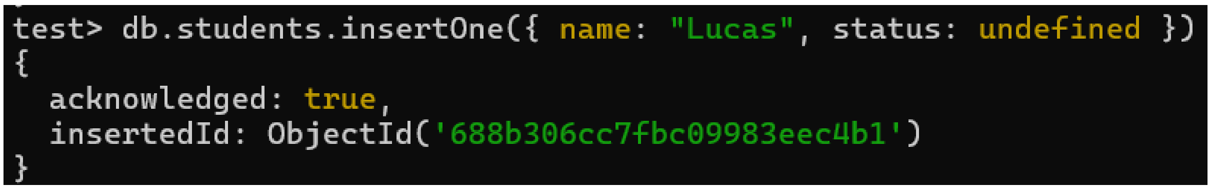

```
test> db.students.insertOne({ name: "Mason" })
{
  acknowledged: true,
  insertedId: ObjectId('688b3055c7fbc09983eec4b0')
}
```

Explanation:
MongoDB automatically generates a unique ObjectId for _id if not provided manually.

9. Undefined

Query:

db.students.insertOne({ name: "Lucas", status: undefined })



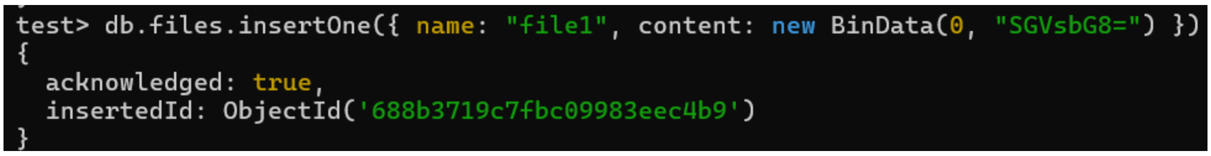
```
test> db.students.insertOne({ name: "Lucas", status: undefined })
{
  acknowledged: true,
  insertedId: ObjectId('688b306cc7fbc09983eec4b1')
}
```

Explanation:
Deprecated in modern apps. Field usually gets omitted; null is preferred.

10. Binary

Query:

db.files.insertOne({ name: "file1", content: new BinData(0, "SGVsbG8=") })



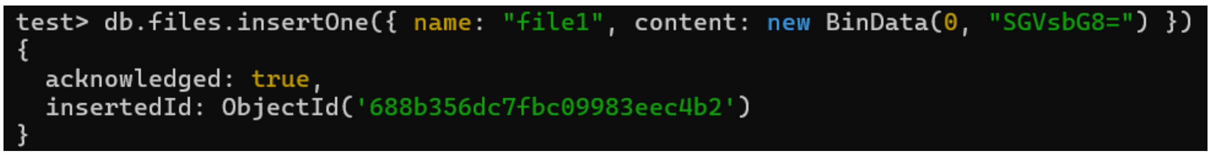
```
test> db.files.insertOne({ name: "file1", content: new BinData(0, "SGVsbG8=") })
{
  acknowledged: true,
  insertedId: ObjectId('688b3719c7fbc09983eec4b9')
}
```

Explanation:
Binary data type is used for storing non-text files like images, audio, or documents.

11. Date

Query:

db.students.insertOne({ name: "Ella", registeredOn: new Date() })



```
test> db.files.insertOne({ name: "file1", content: new BinData(0, "SGVsbG8=") })
{
  acknowledged: true,
  insertedId: ObjectId('688b356dc7fbc09983eec4b2')
}
```

Explanation:

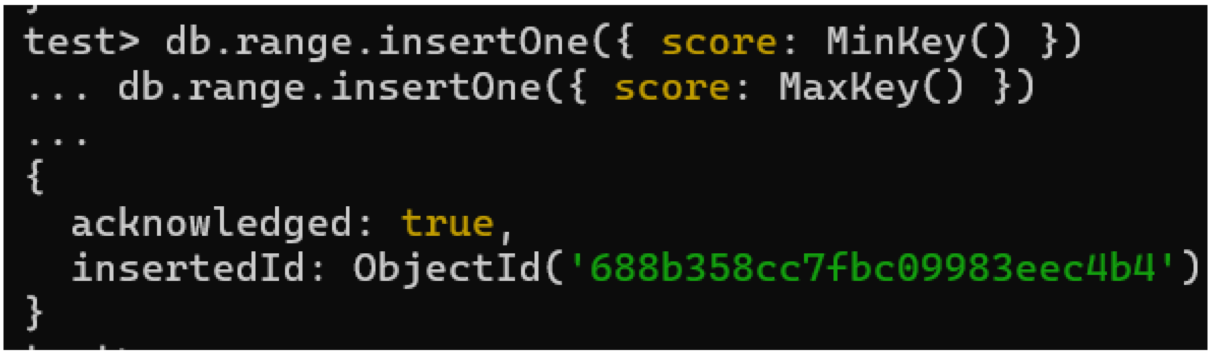
Stores dates and times in UTC format. Commonly used for logs, timestamps, and events.

12. MinKey / MaxKey

Query:

db.range.insertOne({ score: MinKey() })

db.range.insertOne({ score: MaxKey() })

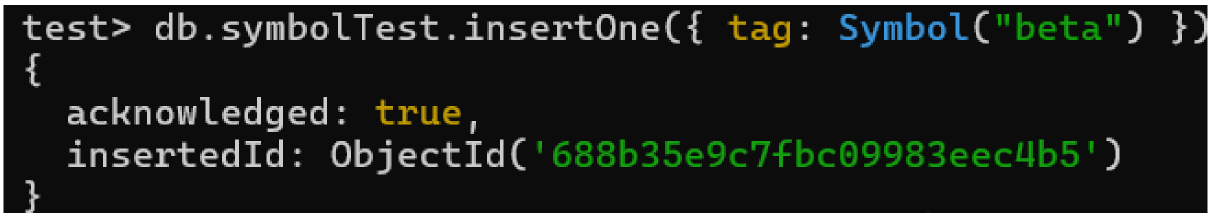

```
test> db.range.insertOne({ score: MinKey() })
... db.range.insertOne({ score: MaxKey() })
...
{
  acknowledged: true,
  insertedId: ObjectId('688b358cc7fbc09983eec4b4')
}
```

Explanation:
Used in advanced querying. MinKey represents the lowest possible value; MaxKey the highest.

13. Symbol

Query:

db.symbolTest.insertOne({ tag: new Symbol("beta") })

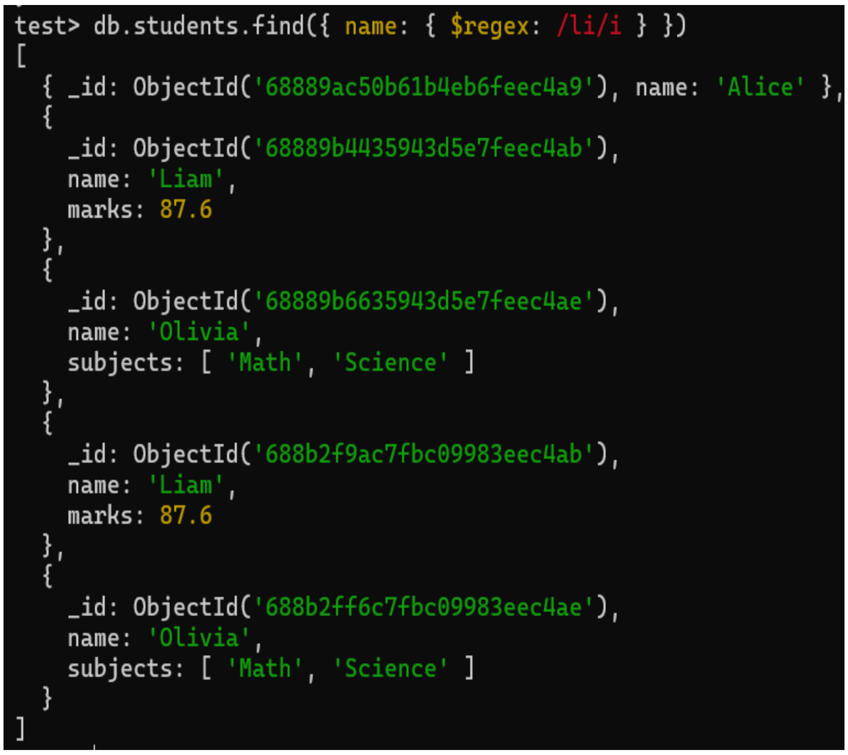

```
test> db.symbolTest.insertOne({ tag: Symbol("beta") })
{
  acknowledged: true,
  insertedId: ObjectId('688b35e9c7fbc09983eec4b5')
}
```

Explanation:
Rarely used now. Acts similar to strings but originally intended for special system usage.

14. Regular Expression

Query:

db.students.find({ name: { $regex: /li/i } })



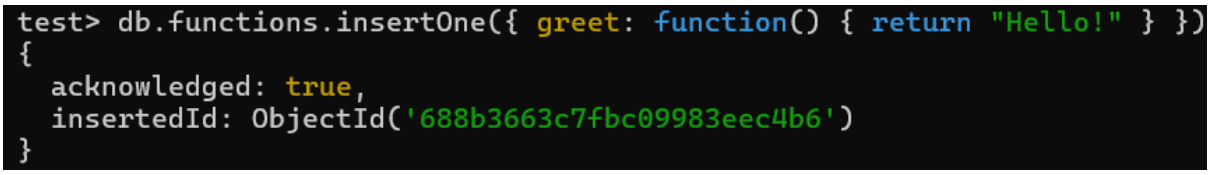
```
test> db.students.find({ name: { $regex: /li/i } })
[
  { _id: ObjectId('68889ac50b61b4eb6feec4a9'), name: 'Alice' },
  {
    _id: ObjectId('68889b44435943d5e7feec4ab'),
    name: 'Liam',
    marks: 87.6
  },
  {
    _id: ObjectId('68889b66335943d5e7feec4ae'),
    name: 'Olivia',
    subjects: [ 'Math', 'Science' ]
  },
  {
    _id: ObjectId('688b2f9ac7fbc09983eec4ab'),
    name: 'Liam',
    marks: 87.6
  },
  {
    _id: ObjectId('688b2ff6c7fbc09983eec4ae'),
    name: 'Olivia',
    subjects: [ 'Math', 'Science' ]
  }
]
```

Explanation:
Used for pattern matching — e.g., to find names containing “li” (case-insensitive).

15. JavaScript

Query:

db.functions.insertOne({ greet: function() { return "Hello!" } })



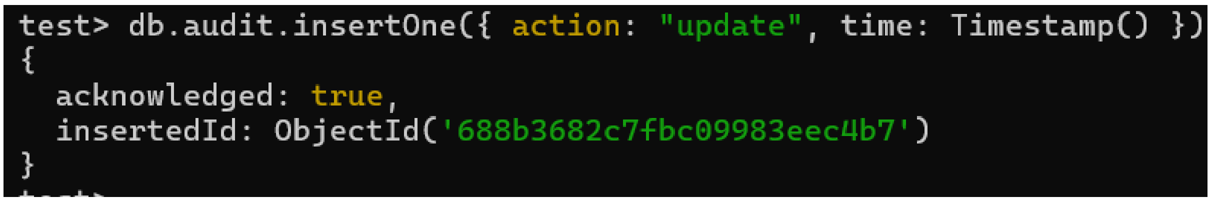
```
test> db.functions.insertOne({ greet: function() { return "Hello!" } })
{
  acknowledged: true,
  insertedId: ObjectId('688b3663c7fbc09983eec4b6')
}
```

Explanation:
Allows storing JavaScript code. Mostly used in system.js collection or eval functions.

16. Timestamp

Query:

db.audit.insertOne({ action: "update", time: Timestamp() })



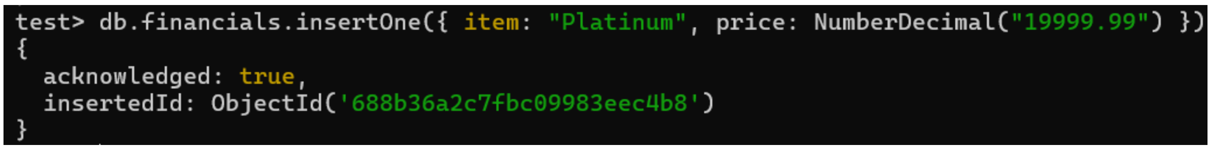
```
test> db.audit.insertOne({ action: "update", time: Timestamp() })
{
  acknowledged: true,
  insertedId: ObjectId('688b3682c7fbc09983eec4b7')
}
```

Explanation:
Timestamps are used for versioning and replication events — more precise than Date.

17. Decimal128

Query:

db.financials.insertOne({ item: "Platinum", price: NumberDecimal("19999.99") })



```
test> db.financials.insertOne({ item: "Platinum", price: NumberDecimal("19999.99") })
{
  acknowledged: true,
  insertedId: ObjectId('688b36a2c7fbc09983eec4b8')
}
```

Explanation:
Decimal128 supports high-precision decimal numbers — ideal for banking and scientific use.