DATA TYPES : MongoDB Server stores data using the BSON format which supports some additional data types that are not available using THE JSON FORMAT.

- **Date**
- **Int32**
- **Decimal**
- **Timestamp**

Date:

Stored as a 64-bit integer representing milliseconds since the Unix epoch, providing precise time information.

1. Supports various date and time operations, such as comparison, arithmetic, and formatting, making it versatile for date-related tasks.
2. Allows for efficient indexing and querying based on date ranges, facilitating fast retrieval of time-sensitive data.
3. Supports time zone adjustments, enabling accurate representation of date and time across different regions.
4. Handles leap years, leap seconds, and other calendar intricacies seamlessly, ensuring accurate date calculations.
5. Compatible with various programming languages and frameworks, simplifying integration into diverse software ecosystems.
6. Offers robust validation and error handling mechanisms to ensure data integrity and consistency.
7. Enables the storage of historical data, time-series data, and scheduling information with ease, making it suitable for a wide range of applications.

Int32:

1. Represents a 32-bit signed integer, accommodating whole numbers ranging from **-2,147,483,648 to 2,147,483,647.**
2. Provides efficient storage and computation for integer values within the specified range, optimizing performance and resource utilization.
3. Supports arithmetic operations like addition, subtraction, multiplication, and division, allowing for straightforward numeric calculations.
4. Enables indexing and sorting based on integer values, enhancing database performance for queries involving numeric fields.
5. Ensures data consistency and accuracy by enforcing type constraints and validation rules during data insertion and updates.
6. Facilitates seamless integration with programming languages and frameworks that utilize integer data types, promoting interoperability.

7. Offers built-in functions and operators for manipulating integer data, such as bitwise operations and mathematical functions.
8. Enables the representation of numerical identifiers, counts, quantities, and other integer-based attributes in database schemas

```
test> db.types.insertOne(
...     {
...         "_id": 1,
...         "value": Int32("1"),
...         "expectedType": "Int32"
...     }
... )
{ acknowledged: true, insertedId: 1 }
test>
```

3.Decimal:

1. Provides high precision for decimal numbers, ensuring

2.accurate.representation and computation of monetary values, scientific measurements, and other precise quantities

3.Supports variable precision and scale, allowing developers to specify the exact

4.Enables exact arithmetic operations on decimal numbers without loss of precision, crucial for financial calculations and scientific simulations

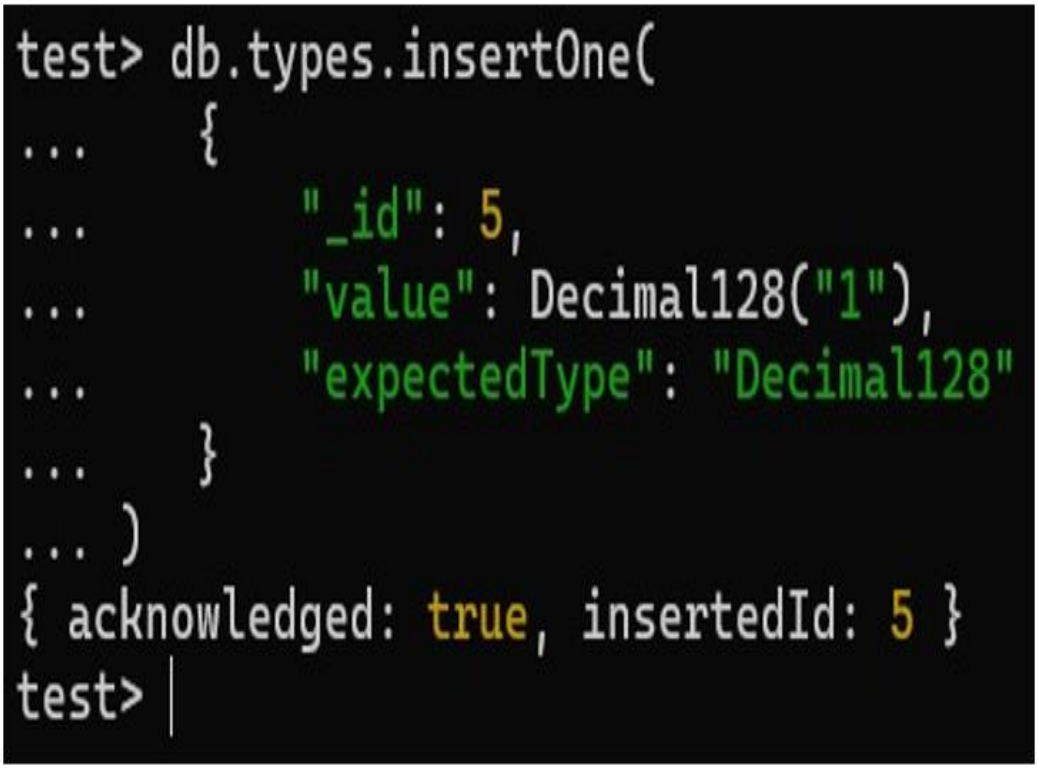
5.Facilitates rounding, truncation, and formatting of decimal values according to specific requirements or standards.

6.Offers support for handling currency conversions, tax calculations, and other financial operations with precision and reliability.

7.Ensures consistency and accuracy in calculations across different computing environments and platforms, regardless of hardware or software limitations.

8.Allows for efficient storage and retrieval of decimal data in database systems, minimizing storage space while preserving precision

9.Enables seamless integration with programming languages and libraries that offer native support for decimal arithmetic and formatting.



Timestamp:

1. Represents elapsed time since the Unix epoch in seconds, providing a standardized way to track time intervals and events
2. Offers granularity down to the second, enabling precise measurement and comparison of time-based data.
3. Supports efficient indexing and querying based on timestamp values, facilitating fast retrieval of temporal data.
4. Enables accurate synchronization of distributed systems and concurrent processes by establishing a common reference point for time.
5. Allows for the calculation of time differences, durations, and intervals between timestamp values, aiding in performance monitoring and analysis.
6. Facilitates the generation of time-based unique identifiers or sequence numbers, useful for creating primary keys or tracking data modifications.
7. Supports time zone conversions and daylight saving adjustments, ensuring consistent interpretation of timestamp values across different regions and time zones.
8. Integrates seamlessly with programming languages and frameworks that utilize timestamp data types, simplifying data exchange and interoperability.

1.WHERE Clause in MongoDB:

- MongoDB doesn't use traditional SQL WHERE clause.
- Instead, it employs the `$where` operator.
- This operator executes JavaScript functions for complex queries.
- However, its use is discouraged due to performance issues.
- MongoDB query example:
  `db.students.find({gpa:{$gt:3.5}});.`

```
db> db.students.find({gpa:{$gt:3.5}});
[
  {
    _id: ObjectId('6649bb89b51b15a423b44ad1'),
    name: 'Student 930',
    age: 25,
    courses: "['English', 'Computer Science', 'Mathematics', 'History']",
    gpa: 3.63,
    home_city: 'City 3',
    blood_group: 'A-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('6649bb89b51b15a423b44ad3'),
    name: 'Student 268',
    age: 21,
    courses: "['Mathematics', 'History', 'Physics']",
    gpa: 3.98,
    blood_group: 'A+',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('6649bb89b51b15a423b44add'),
    name: 'Student 368',
    age: 20,
    courses: "['English', 'History', 'Physics', 'Computer Science']",
    gpa: 3.91,
    home_city: 'City 9',
    blood_group: 'O-',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('6649bb89b51b15a423b44ae4'),
    name: 'Student 468',
    age: 21,
    courses: "['Computer Science', 'Physics', 'Mathematics', 'History']",
    gpa: 3.97,
    blood_group: 'A-',
    is_hotel_resident: true
  },
```

2.AND Operator in MongoDB:

- `$and` operator combines multiple conditions, requiring all to be met.
- Useful for specifying multiple criteria in a query.

- MongoDB query example: `db.students.find({ $and: [ { home_city: "City 5" }, { blood_group: "A+" } ] }).`

```
db> db.students.find({
... $and:[
... {home_city:"City 5"},
... {blood_group:"A+"}
... ]
... });
[
  {
    _id: ObjectId('6649bb89b51b15a423b44b04'),
    name: 'Student 142',
    age: 24,
    courses: "['History', 'English', 'Physics', 'Computer Science']",
    gpa: 3.41,
    home_city: 'City 5',
    blood_group: 'A+',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('6649bb89b51b15a423b44c24'),
    name: 'Student 947',
    age: 20,
    courses: "['Physics', 'History', 'English', 'Computer Science']",
    gpa: 2.86,
    home_city: 'City 5',
    blood_group: 'A+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('6649bb89b51b15a423b44c96'),
    name: 'Student 567',
    age: 22,
    courses: "['Computer Science', 'History', 'English', 'Mathematics']",
    gpa: 2.01,
    home_city: 'City 5',
    blood_group: 'A+',
    is_hotel_resident: true
  }
]
db>
```

3.OR Operator in MongoDB:

- `$or` operator performs logical OR operation between expressions.
- Matches documents satisfying at least one condition.
- Allows for constructing queries with multiple criteria.
- MongoDB query example: `db.students.find({ $or: [ { is_hotel_resident: true }, { gpa: { $lt: 3.0 } } ]`

```
db> db.students.find({
... $or:[
... {is_hotel_resident:true},
... {gpa:{$lt:3.0}}
... ]
... });
[
  {
    _id: ObjectId('6649bb89b51b15a423b44acd'),
    name: 'Student 948',
    age: 19,
    courses: "['English', 'Computer Science', 'Physics', 'Mathematics']",
    gpa: 3.44,
    home_city: 'City 2',
    blood_group: 'O+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('6649bb89b51b15a423b44ace'),
    name: 'Student 157',
    age: 20,
    courses: "['Physics', 'English']",
    gpa: 2.27,
    home_city: 'City 4',
    blood_group: 'O-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('6649bb89b51b15a423b44acf'),
    name: 'Student 316',
    age: 20,
    courses: "['Physics', 'Computer Science', 'Mathematics', 'History']",
    gpa: 2.32,
    blood_group: 'B+',
    is_hotel_resident: true
  },
```

4.CRUD Operations in MongoDB:

- **Create/Insert (C):** Utilizes `insertOne` method to add single documents.
- **Remove (R):** Involves `deleteOne` for deleting a single document.
- **Update (U):** Employs `updateOne` for updating a single document.
- **Delete (D):** Uses `deleteOne` for deleting a single document.

5.InsertOne Method in MongoDB:

- Inserts a single document into a collection.
- Returns an object with insertion details.
- Example involves inserting student data into the `students` collection.

```
db> const studentData={
... "name":"Alice Smith",
... "age":22,
... "courses":["Mathematics","Computer Science","English"],
... "gpa":3.8,
... "home_city":"New York",
... "blood_group":"A+",
... "is_hotel_resident":false
... };

db> db.students.insertOne(studentData);
{
  acknowledged: true,
  insertedId: ObjectId('6661da38b0d232162dcdcdf6')
}
db>
```

6.UpdateOne Method in MongoDB:

- Updates a single document matching a filter.
- Returns details about the update operation.
- Example includes updating a student's GPA.

```
db> db.students.insertOne(studentData);
{
  acknowledged: true,
  insertedId: ObjectId('6661da38b0d232162dcdcdf6')
}
db>  db.students.updateOne({name:"Alice Smith"},{$set:{gpa:3.8}});
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 0,
  upsertedCount: 0
}
db>
```

7.DeleteOne Method in MongoDB:

- Deletes a single document matching a filter.
- Returns details about the delete operation.
- Example involves attempting to delete a document by name.

```
db> db.students.deleteOne({name:"John Doe"});
{ acknowledged: true, deletedCount: 0 }
db>
```

8. UpdateMany and DeleteMany Methods in MongoDB:

- **UpdateMany:** Updates multiple documents matching a filter.
- **DeleteMany:** Deletes multiple documents matching a filter.

- Examples include updating GPAs and deleting hotel residents.

8a.UpdateMany:

```
db> db.students.updateMany({gpa:{$lt:3.0}},{$inc:{gpa:0.5}});
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 261,
  modifiedCount: 261,
  upsertedCount: 0
}
```

8b.DeleteMany:

```
db> db.students.deleteMany({is_hotel_resident:false});
{ acknowledged: true, deletedCount: 255 }
db>
```

9.Projection in MongoDB:

- Specifies fields to be returned in query results.
- Enhances performance by retrieving only necessary data.
- Example query retrieves only name and GPA fields while excluding `_id`.

```
db> db.students.find({},{name:1,gpa:1,_id:0});
[
  { name: 'Student 948', gpa: 3.44 },
  { name: 'Student 157', gpa: 2.77 },
  { name: 'Student 316', gpa: 2.82 },
  { name: 'Student 346', gpa: 3.31 },
  { name: 'Student 930', gpa: 3.63 },
  { name: 'Student 305', gpa: 3.4 },
  { name: 'Student 440', gpa: 2.56 },
  { name: 'Student 256', gpa: 3.44 },
  { name: 'Student 177', gpa: 3.02 },
  { name: 'Student 487', gpa: 2.6 },
  { name: 'Student 213', gpa: 2.89 },
  { name: 'Student 690', gpa: 2.75 },
  { name: 'Student 647', gpa: 3.43 },
  { name: 'Student 232', gpa: 3.04 },
  { name: 'Student 328', gpa: 3.42 },
  { name: 'Student 468', gpa: 3.97 },
  { name: 'Student 504', gpa: 2.92 },
  { name: 'Student 915', gpa: 3.37 },
  { name: 'Student 367', gpa: 3.11 },
  { name: 'Student 969', gpa: 3.71 }
]
```