**Introduction**

MongoDB is a document database. It stores data in a type of JSON format called BSON.

What is BSON?

BSON is a binary encoded Javascript Object Notation (JSON)—a textual object notation widely used to transmit and store data across web based applications. JSON is easier to understand as it is human-readable, but compared to BSON, it supports fewer data types. BSON encodes type and length information, too, making it easier for machines to parse.

{"hello": "world"} →

\x16\x00\x00\x00 // total document size

\x02 // 0x02 = type String

hello\x00 // field name

\x06\x00\x00\x00world\x00 // field value

\x00 // 0x00 = type EOO ('end of object')

Just like JSON, BSON supports various programming languages such as C, C++, C#, Java, JavaScript, PHP, Python, Ruby, and Swift.

What Does BSON Stand For?

BSON stands for Binary Javascript Object Notation. It is a binary encoded serialization of JSON documents. BSON has been extended to add some optional non-JSON-native data types, like dates and binary data.

BSON can be compared to other binary formats, like Protocol Buffers. The greater difference is that BSON is more "Schema-less" than Protocol Buffers, providing the advantage of flexibility and the slight disadvantage of space efficiency.

BSON Specifications and BSON Types

BSON specification version 1.1 is published at bsonspec.org. The topmost element in the structure must be of type BSON object and should contain one or more elements, where an element can be a field name (string), a type, or a value. The following are some of the types included in BSON.

|  |  |
| --- | --- |
| byte | 1 byte (8-bits) |
| int32 | 4 bytes (32-bit signed integer, two's complement) |
| int64 | 8 bytes (64-bit signed integer, two's complement) |
| uint64 | 8 bytes (64-bit unsigned integer) |
| double | 8 bytes (64-bit IEEE 754-2008 binary floating-point) |
| decimal128 | 16 bytes (128-bit IEEE 754-2008 decimal floating-point) |
| date | 8 bytes(64-bit integer) |
| objectId | 12 bytes(4-byte timestamp value, 5-byte random value, and 3-byte incrementing counter) |
| array | Storage is based on data (A byte array uses 1 byte, a short array uses 2 bytes, and an integer array uses 4 bytes) |

BSON vs JSON

|  |  |  |
| --- | --- | --- |
|  | JSON | BSON |
| Type | JSON files are written in text format. | BSON files are written in binary. |
| Speed | JSON is fast to read but slower to build. | BSON is slow to read but faster to build and scan. |
| Space | JSON data is slightly smaller in byte size. | BSON data is slightly larger in byte size. |
| Encode and Decode | We can send JSON through APIs without encoding and decoding. | BSON files are encoded before storing and decoded before displaying. |
| Parse | JSON is a human-readable format that doesn't require parsing. | BSON needs to be parsed as they are machine-generated and not human-readable. |
| Data Types | JSON has a specific set of data types—string, boolean, number for numeric data types, array, object, and null. | Unlike JSON, BSON offers additional data types such as bindata for binary data, decimal128 for numeric. |
| Usage | Used to send data through the network (mostly through APIs). | Databases use BSON to store data. |

Advantages of BSON

* Lightweight and Traversable

This makes it possible for a large amount of data to be stored in BSON file format. Additionally, BSON files are easily stored and sent through the network, making them perfect for storing and sending data.

* Efficient

BSON was made to efficiently store space and scan: In a BSON document, large elements are prefixed with a length field. Documents in BSON use more space than JSON due to the length prefixes and explicit array indices, but thanks to those prefixes, we are able to scan and query much faster. The prefixes make it easy to compare and calculate directly on data, simplifying application code consumption.

* Handles Additional Data Types

Unlike in JSON, you can find data types such as Bindata, Minkey, Maxkey, Binary Data, ObjectID, Regular Expression, JavaScript, Decimal128, and Date for datetime in BSON. These data types are crucial when working with specialty programs. Using BSON in such cases gives you high precision.

Convert JSON to BSON

You can use various converters between JSON and BSON formats. One such example is OnlineJSONTools. Various online converters that easily change the JSON format to BSON format and vice versa are available.

SQL vs Document Databases

SQL databases are considered relational databases. They store related data in separate tables.

MongoDB is a document database which is often referred to as a non-relational database. Relational data is stored differently. It is a non-tabular database.

MongoDB stores data in flexible documents. Instead of having multiple tables you can simply keep all of your related data together. This makes reading your data very fast.

You can still have multiple groups of data too. In MongoDB, instead of tables these are called collections.

Commands

Show dbs

db

use <dbname>

db.createCollection("collectionname")

db. collectionname.insertOne({

name: "value"

})

db. collectionname.insertMany({

name: "value",

},

{

name: "value",

})

db. collectionname.find({match condn}, {project elements })

// 1 display 0 no display

// Note: Cannot use both 0 and 1 in the same object. Either specify the fields to include or the fields to exclude.

db. collectionname.find()

db. collectionname.find({category: "XYZ"})

db. collectionname.find({}, {title: 1, date: 1}) // title & date projected

db. collectionname.find({}, {category: 0}) // no category

db. collectionname.updateOne( { name: "XYZ" }, { $set: { pin: 546562 } } )

db. collectionname.updateOne(

{ name: "XYZ" }, { $set: { pin: 546562 } } ,

{ upsert: true }) // Update the document, but if not found insert

db. collectionname.updateMany({}, { $inc: { bonus: 500 } })

db. collectionname.deleteOne( { name: "XYZ" } )

db. collectionname.deleteMany( { name: "XYZ" } )

QUERIES

db.users.insertOne({

name: "John Doe",

age: 30,

email: "john.doe@example.com",

isActive: true

});

// Find all users where the age is greater than 25

db.users.find({ age: { $gt: 25 } });

// Find all active users

db.users.find({ isActive: true });

// Find users whose name is "John Doe" and age is greater than 25

db.users.find({ name: "John Doe", age: { $gt: 25 } });

// Update the age of a user named "John Doe"

db.users.updateOne(

{ name: "John Doe" },

{ $set: { age: 31 } }

);

// Increment the age of all users by 1

db.users.updateMany(

{},

{ $inc: { age: 1 } }

);

// Delete the first user whose name is "John Doe"

db.users.deleteOne({ name: "John Doe" });

// Delete all users whose age is greater than 40

db.users.deleteMany({ age: { $gt: 40 } });

// Find users with age either 25 or 30

db.users.find({ age: { $in: [25, 30] } });

// Find users with age greater than 20 and less than 40

db.users.find({ age: { $gt: 20, $lt: 40 } });

// Find users with name "John Doe" or "Jane Doe"

db.users.find({ $or: [{ name: "John Doe" }, { name: "Jane Doe" }] });

// Find documents where the "age" field is equal to 30

db.users.find({ age: { $eq: 30 } });

// Find documents where the "age" field is not equal to 30

db.users.find({ age: { $ne: 30 } });

// Find documents where the "age" field is greater than 30

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

// Find documents where the "age" field is greater than or equal to 30

db.users.find({ age: { $gte: 30 } });

// Find documents where the "age" field is less than 30

db.users.find({ age: { $lt: 30 } });

// Find documents where the "age" field is less than or equal to 30

db.users.find({ age: { $lte: 30 } });

// Find documents where the "age" field is either 25 or 30

db.users.find({ age: { $in: [25, 30] } });

// Find documents where the "age" field is not 25 or 30

db.users.find({ age: { $nin: [25, 30] } });

AGGREGATE PIPELINES

[

{

\_id: 1,

name: "John Doe",

age: 30,

email: "john.doe@example.com",

isActive: true,

languages: ["JavaScript", "Python"],

scores: [80, 85, 90]

},

{

\_id: 2,

name: "Jane Smith",

age: 28,

email: "jane.smith@example.com",

isActive: false,

languages: ["Java", "C++"],

scores: [70, 75, 80]

},

{

\_id: 3,

name: "Bob Johnson",

age: 35,

email: "bob.johnson@example.com",

isActive: true,

languages: ["Python", "Java"],

scores: [90, 95, 100]

}

]

//Calculating Average Age:

db.users.aggregate([

{

$group: {

\_id: null,

averageAge: { $avg: "$age" }

}

}

]);

// Number of Active Users:

db.users.aggregate([

{

$match: { isActive: true }

},

{

$count: "activeUsersCount"

}

]);

//Grouping Users by Language

db.users.aggregate([

{

$unwind: "$languages"

},

{

$group: {

\_id: "$languages",

users: { $push: "$name" }

}

}

]);

// Calculating Maximum Score

db.users.aggregate([

{

$project: {

name: 1,

email: 1,

maxScore: { $max: "$scores" }

}

}

]);

//Sorting Users by Age in Descending Order

db.users.aggregate([

{

$sort: { age: -1 }

}

]);

//Grouping Users by Age Range and Counting Users in Each Range:

db.users.aggregate([

{

$group: {

\_id: {

$switch: {

branches: [

{ case: { $lte: ["$age", 30] }, then: "Under 30" },

{ case: { $lte: ["$age", 40] }, then: "31-40" },

{ case: { $lte: ["$age", 50] }, then: "41-50" },

{ case: { $gt: ["$age", 50] }, then: "Above 50" }

],

default: "Unknown"

}

},

count: { $sum: 1 }

}

}

]);