FerretDB

Building an Open-Source MongoDB-compatible Database on Top of PostgreSQL

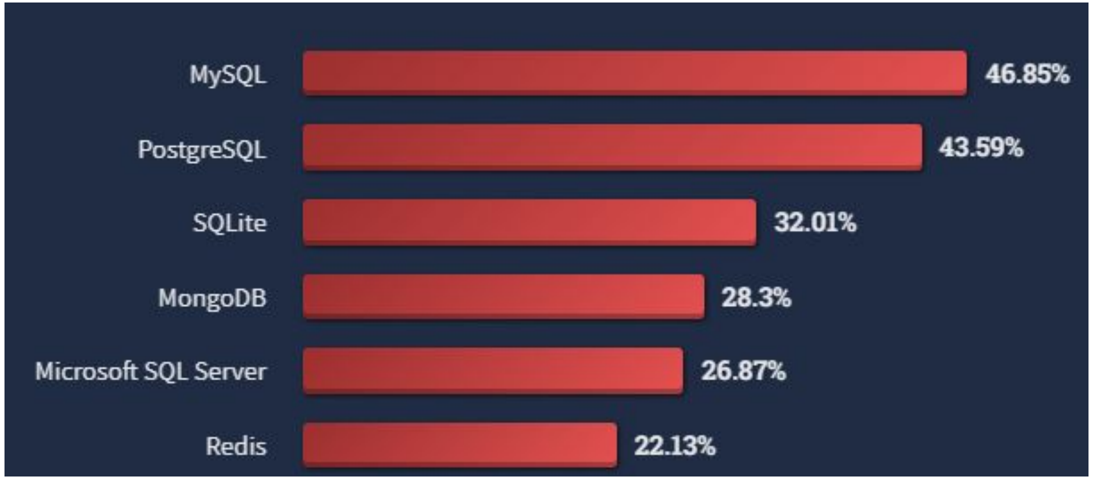
Elena Grahovac, Director of Engineering, FerretDB

PostgreSQL Berlin, May 2023

Agenda

1. MongoDB compatibility on PostgreSQL – why?
2. FerretDB’s concept and current state
3. How we store and query data
4. Challenges
5. What’s next?

MongoDB’s popularity



“Which database environments have you done extensive development work in over the past year, and which do you want to work in over the next year? “ 65k responses

Source: StackExchange Developer Survey, 2022 (excerpt)
“Which database environments have you done extensive development work in over the past year, and which do you want to work in over the next year?”

MongoDB’s license

MongoDB – since 2018, released under the Server Side Public License (SSPL).

If MongoDB is used as part of a Cloud Service …

… everything you use to provide that service needs to be open-sourced.

More info:

www.ssplisbad.com

We’ve talked to users - here’s what they said

“The SSPL license is vague - we are looking to replace MongoDB due to the legal risks and uncertainty.” - a FAANG company

“We are looking to find a MongoDB Atlas alternative, without vendor lock-in.” - Major travel search portal

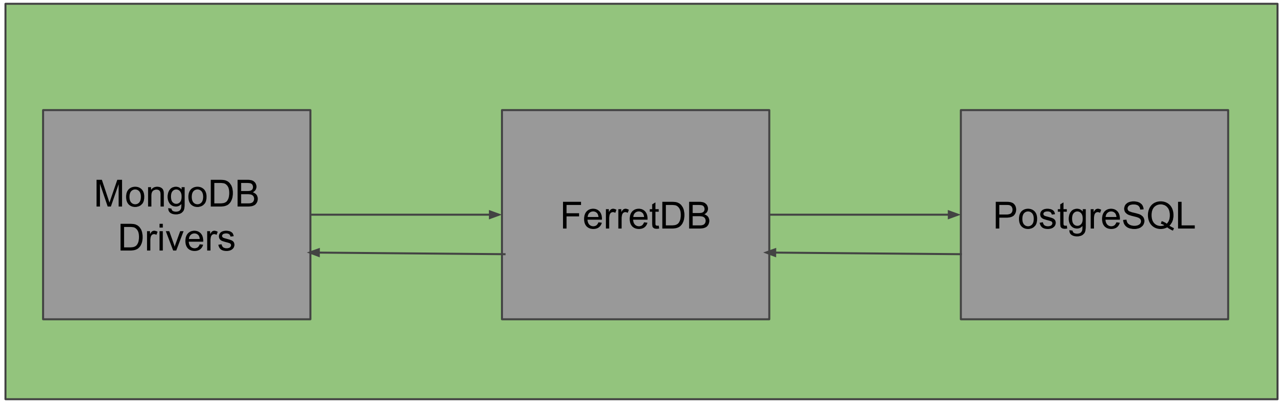
“Pricing of MongoDB Atlas is not suitable for our use case.” - Small SaaS business

“We are unable to offer our customers an open-source, MongoDB-compatible database.” - Cloud Infrastructure provider

About FerretDB

- A MongoDB compatible interface
- Set out to become the de facto MongoDB Alternative
- Can be used on-prem or in the cloud
- Can utilize various RDBMS as backend (mainly PgSQL)
- Released under Apache 2.0

How it works



Why PostgreSQL?

Probably no need to explain here :)

- FOSS
- Huge, supportive community
- High number of PostgreSQL users run MongoDB
- Existing JSON compatibility
- Strong interest from users with extensive operational experience

FerretDB

Current state

FerretDB
It's here.
FerretDB 1.0 GA
a truly Open Source MongoDB database alternative
www.ferretdb.io

Current state

- Concentrating on compatibility, no effort on performance
- Support for CRUD, basic MongoDB, MeteorJS apps
- 200+ active instances with telemetry on

FerretDB

Map BSON documents to PostgreSQL JSONB

MongoDB: BSON serialization

- JSON-like
- Binary-encoded
- Documents with **order-preserving** key-value pairs
- Supports some special types (e.g. date)

https://bsonspec.org/

MongoDB: BSON serialization

```
{"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')
```

PostgreSQL: JSONB

- Order in objects is not preserved
- Same data types as in JSON

BSON -> JSONB

- Store order
- Store data types information

BSON -> JSONB

```
{"hello": "world"}
```

```
{
  "$s": {
    "p": {
      "hello": {"t": "string"}
    },
    "$k": ["hello"]
  },
  "hello": "world"
}
```

BSON -> JSONB

```
{
  "_id": ObjectID("645a510e4717b4a771e206b0"),
  "hello": "world"
}
```

```
{
  "$s": {
    "p": {
      "_id": {"t": "objectId"},
      "foo": {"t": "string"}
    },
    "$k": ["_id", "foo"]
  },
  "_id": "645a510e4717b4a771e206b0",
  "foo": "bar"
}
```

FerretDB

Queries

MongoDB query example

```
db.coll.find({v: "foo"}).sort({v: 1})

db.coll.find({v: {$ne: "foo"}}).sort({v: 1})
```

Compatibility

- Step 1. Fetch everything and implement the logic in Go.
- Step 2. Test! Test! Test!
- Step 3. Exclude edge cases that are hard to support.
- Step 4. Pushdown to PostgreSQL:
    - WHERE
    - ORDER BY (experimental)

Comparing BSON values

- Filtering and sorting (work differently)
- Values of different types
- Special values (infinity, NaN, null)

Order example

```
db.coll.insertMany(
  [
    {_id: 1,v: null},
    {_id: 2,v: [null]},
    {_id: 3,v: []}
  ]
)
```

Order example

```
db.coll.find().sort({v:1, _id:1})

[
  { _id: 3, v: [] },
  { _id: 1, v: null },
  { _id: 2, v: [ null ] }
]
```

Order example

```
db.coll.find().sort({v:1, _id:1})

[
  { _id: 3, v: [] },
  { _id: 1, v: null },
  { _id: 2, v: [ null ] }
]


db.coll.find().sort({v:-1, _id:1})
```

What should be the result?

Order example

```
db.coll.find().sort({v:1, _id:1})

[
  { _id: 3, v: [] },
  { _id: 1, v: null },
  { _id: 2, v: [ null ] }
]


db.coll.find().sort({v:-1, _id:1})

[
  { _id: 1, v: null },
  { _id: 2, v: [ null ] },
  { _id: 3, v: [] }
]
```

Order example

```
db.coll.find().sort({v:-1, _id:-1})

[
  { _id: 2, v: [ null ] },
  { _id: 1, v: null },
  { _id: 3, v: [] }
]
```

Pushdown $eq

```
_jsonb->v @> "foo"
```

Pushdown $ne

```
sql := `NOT ( ` +
    // does document contain the key,
    // it is necessary, as NOT won't work correctly if the key does not exist.
    `_jsonb ? %[1]s AND ` +

    // does the value under the key is equal to filter value
    `_jsonb->%[1]s @> %[2]s AND ` +

    // does the value type is equal to the filter's one
    `_jsonb->'$s'->'p'->%[1]s->'t' = '"%[3]s"' )`
```

At least it’s easy for $eq, right?..

No, it’s not. Different data types might have different rules.

Big floats

MongoDB rounds values when compares big numbers

```
db.coll.insert({"v": Double(2305843009213693952)})
```

Big floats

MongoDB rounds values when compares big numbers

```
db.coll.insert({"v": Double(2305843009213693952)})

db.coll.find()
[
  { _id: ObjectId("645b77396b8886a0e968b07d"), v: 2305843009213694000 }
]
```

Big floats

MongoDB rounds values when compares big numbers

```
db.coll.insert({"v": Double(2305843009213693952)})

db.coll.find()
[
  { _id: ObjectId("645b77396b8886a0e968b07d"), v: 2305843009213694000 }
]

db.coll.find({"v": Double(2305843009213693952)})
[
  { _id: ObjectId("645b77396b8886a0e968b07d"), v: 2305843009213694000 }
]
```

Big floats

MongoDB rounds values when compares big numbers

```
db.coll.insert({"v": Double(2305843009213693952))})

db.coll.find()
[
  { _id: ObjectId("645b77396b8886a0e968b07d"), v: 2305843009213694000 }
]

db.coll.find({"v": Double(2305843009213693952)})
[
  { _id: ObjectId("645b77396b8886a0e968b07d"), v: 2305843009213694000 }
]

db.coll.find({"v": Double(2305843009213694001)})
[
  { _id: ObjectId("645b77396b8886a0e968b07d"), v: 2305843009213694000 }
]
```

Big floats

MongoDB rounds values when compares big numbers

```
db.coll.insert({"v": Double(2305843009213693952)})

db.coll.find()
[
  { _id: ObjectId("645b77396b8886a0e968b07d"), v: 2305843009213694000 }
]

db.coll.find({"v": Double(2305843009213693952)})
[
  { _id: ObjectId("645b77396b8886a0e968b07d"), v: 2305843009213694000 }
]

db.coll.find({"v": Double(2305843009213694001)})
[
  { _id: ObjectId("645b77396b8886a0e968b07d"), v: 2305843009213694000 }
]
```

Big floats

Pushdown implementation

```
    case v > types.MaxSafeDouble:
        sql = `_jsonb->%[1]s > %[2]s`
        v = types.MaxSafeDouble

    case v < -types.MaxSafeDouble:
        sql = `_jsonb->%[1]s < %[2]s`
        v = -types.MaxSafeDouble
```

Dot notation

Access elements of arrays and embedded documents

```
{
  people: {
    name: Anna, age: 20
  }
}
```

```
db.coll.find({"people.name": "Anna"})
```

Dot notation

Access elements of arrays and embedded documents

```
{
  people: {
    name: Anna, age: 20
  }
}

{
  people: [
    {name: Anna, age: 20},
    {name: Bob, age: 18}
  ]
}
```

```
db.coll.find({"people.name": "Anna"})
db.coll.find({"people.0.name": "Anna"})
```

Dot notation

```
a.b.c
```

```
a.b.c
a.*.b.c
a.b.*.c
a.*.b.*.c
```

Indexes

- Unique index for _id field:

```
CREATE UNIQUE INDEX coll__id__f787baf7_idx ON
test.coll_6c6216e1 USING btree (((_jsonb -> '_id')))
```

- Compound indexes:

```
db.c.createIndex({"foo": 1, "bar": -1})
CREATE INDEX IF NOT EXISTS c_foo___bar___8e413c86_idx ON
test.c_e60c2c52 (((_jsonb->'foo')) ASC, ((_jsonb->'bar'))
DESC)
```

FerretDB

What’s next?

Community

- Try FerretDB!
- Roadmap: https://github.com/orgs/FerretDB/projects/2
- Star us: https://github.com/ferretdb/ferretdb
- Feedback and questions: https://github.com/ferretdb/ferretdb#community

https://www.ferretdb.io/

https://github.com/FerretDB/FerretDB

https://twitter.com/ferret_db

https://techhub.social/@ferretdb

FerretDB