NoSQL databases

Introduction to MongoDB

DBMG

MongoDB: Introduction

- The leader in the NoSQL Document-based databases
- Full of features, beyond NoSQL
  - High performance
  - High availability
  - Native scalability
  - High flexibility
  - Open source

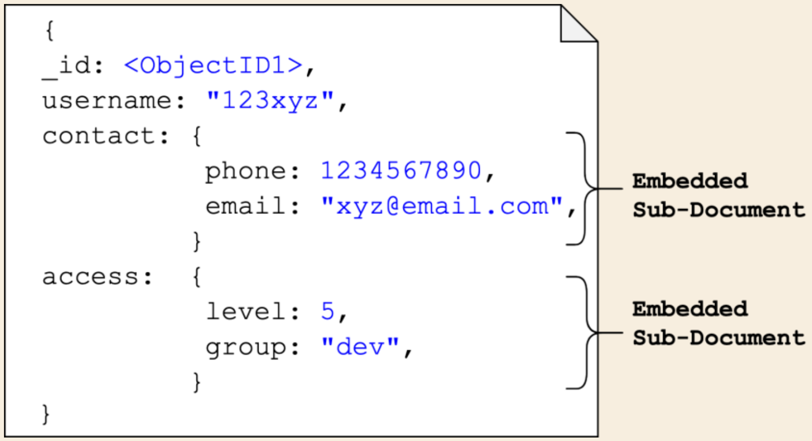
Terminology – Approximate mapping

Relational database	**MongoDB**
Table	Collection
Record	Document
Column	Field

MongoDB: Document Data Design

High-level, business-ready representation of the data

- Records are stored into Documents
  - field-value pairs
  - similar to JSON objects
  - may be nested



```
{
_id: <ObjectID1>,
username: "123xyz",
contact: {
          phone: 1234567890,
          email: "xyz@email.com",
         }
access:  {
          level: 5,
          group: "dev",
         }
}
```

MongoDB: Document Data Design

- High-level, business-ready representation of the data
- Flexible and rich syntax, adapting to most use cases
- Mapping into developer-language objects
  - year, month, day, timestamp,
  - lists, sub-documents, etc.
- Rich query language
  - Documents can be created, read, updated and deleted.
  - The **SQL language** is **not supported**
  - APIs available for many programming languages
    - JavaScript, PHP, Python, Java, C#, ..

MongoDB

Querying data using operators

MongoDB: query language

- Most of the operations available in SQL language can be expressend in MongoDB language

MySQL clause	MongoDB operator
`SELECT`	`find()`

`SELECT` `*`
`FROM people`
	`db.people.`**`find()`**

DBMG

MongoDB: Read data from documents

- Select documents
  - `db.<collection name>.find( {<conditions>}, {<fields of interest>} );`
- E.g.,

```
db.people.find();
```

  - Returns all documents contained in the people collection

MongoDB: Read data from documents

- Select documents
  - `db.<collection name>.find( {<conditions>}, {<fields of interest>} );`
- Select the documents satisfying the specified conditions and specifically only the fields specified in fields of interest
  - `<conditions>` are optional
    - conditions take a document with the form: `{field1 : <value>, field2 : <value> ... }`
    - Conditions may specify a value or a regular expression

MongoDB: Read data from documents

- Select documents
  - `db.<collection name>.find( {<conditions>}, {<fields of interest>} );`
- Select the documents satisfying the specified conditions and specifically only the fields specified in fields of interest
  - `<fields of interest>` are optional
    - projections take a document with the form:
      `{field1 : <value>, field2 : <value> ... }`
    - 1/true to include the field, 0/false to exclude the field

DBMG

MongoDB: Read data from documents

E.g.,

```
db.people.find().pretty();
```

- No conditions and no fields of interest
  - Returns all documents contained in the people collection
  - `pretty()` displays the results in an easy-to-read format

```
db.people.find({age:55})
```

- One condition on the value of age
  - Returns all documents having *age* equal to 55

MongoDB: Read data from documents

```
db.people.find({ }, { user_id: 1, status: 1 })
```

No conditions, but returns a specific set of fields of interest

- Returns only **user_id** and **status** of all documents contained in the people collection
- Default of fields is false, except for _id

```
db.people.find({ status: "A",  age: 55})
```

`status = "A"` and `age = 55`

- Returns all documents having **status = "A"** and **age = 55**

MongoDB: find() operator

MySQL clause	MongoDB operator
SELECT	find()

```
SELECT id,
       user_id,
       status
FROM people
```

```
db.people.find(
    { },
    { user_id: 1,
      status: 1
    }
)
```

MongoDB: find() operator

MySQL clause	MongoDB operator
`SELECT`	`find()`

`SELECT id,`
`       user_id,`
`       status`
`FROM people`
	`db.people.find(`
	`    { },`
	`    { user_id: 1,`
	`      status: 1`
	`    }`
	`)`

Where Condition

Select fields

MongoDB: find() operator

MySQL clause	MongoDB operator
SELECT	find()
WHERE	find({<WHERE CONDITIONS>})

SELECT *
 FROM people
 WHERE status = "A"
	db.people.find(
	 { status: "A" }
	)

```
SELECT *
FROM people
WHERE status = "A"
```

```
db.people.find(
    { status: "A" }
)
```

Where Condition

MongoDB: find() operator

MySQL clause	MongoDB operator
SELECT	find()
WHERE	find({<WHERE CONDITIONS>})

Where Condition

```
SELECT user_id, status
FROM people
WHERE status = "A"
```

```
db.people.find(
      { status: "A" },
      { user_id: 1,
        status: 1,
        _id: 0
      }
)
```

Selection fields

By default, the `_id` field is shown.
To remove it from visualization use: `_id: 0`

MongoDB: find() operator

MySQL clause	MongoDB operator
SELECT	find()
WHERE	find({<WHERE CONDITIONS>})

	db.people.find({"address.city":"Rome" })

```
{ _id: "A",
  address: {
          street: "Via Torino",
          number: "123/B",
          city: "Rome",
          code: "00184"
      }
}
```

nested document

MongoDB: Read data from documents

```
db.people.find({ age: { $gt: 25, $lte: 50 } })
```

Age greater than 25 and less than or equal to 50

- Returns all documents having **age > 25 and age <= 50**

```
db.people.find({$or:[{status: "A"},{age: 55}]})
```

Status = "A" or age = 55

- Returns all documents having **status="A" or age=55**

```
db.people.find({ status: {$in:["A", "B"]}})
```

Status = "A" or status = B

- Returns all documents where the **status** field value is **either "A" or "B"**

MongoDB: Read data from documents

- Select a single document
  - `db.<collection name>.findOne( {<conditions>}, {<fields of interest>} );`
- Select one document that satisfies the specified query criteria.
  - If multiple documents satisfy the query, it returns the first one according to the natural order which reflects the order of documents on the disk.

MongoDB: comparison operators

- In SQL language, comparison operators are essential to express conditions on data.
- In Mongo query language they are available with a different syntax.

MySQL	MongoDB	Description
`>`	`$gt`	`greater than`
`>=`	`$gte`	`greater equal then`
`<`	`$lt`	`less than`
`<=`	`$lte`	`less equal then`
`=`	`$eq`	`equal to`
`!=`	`$neq`	`not equal to`

MongoDB: Comparison query operators

Name	Description
`$eq or :`	Matches values that are equal to a specified value
`$gt`	Matches values that are greater than a specified value
`$gte`	Matches values that are greater than or equal to a specified value
`$in`	Matches any of the values specified in an array
`$lt`	Matches values that are less than a specified value
`$lte`	Matches values that are less than or equal to a specified value
`$ne`	Matches all values that are not equal to a specified value
`$nin`	Matches none of the values specified in an array

MongoDB: comparison operators (>)

MySQL	MongoDB	Description
>	`$gt`	`greater than`

`SELECT *`
`FROM people`
`WHERE age > 25`
	`db.people.find(`
	`    { age: { $gt: 25 } }`
	`)`

MongoDB: comparison operators (>=)

MySQL	MongoDB	Description
>	$gt	greater than
>=	**$gte**	**greater equal then**

SELECT *
FROM people
WHERE **age >= 25**
	db.people.find(
	    { **age: { $gte: 25 }** }
	)

MongoDB: comparison operators (<)

MySQL	MongoDB	Description
>	`$gt`	greater than
>=	`$gte`	greater equal then
<	**`$lt`**	**less than**

`SELECT *`
`FROM people`
`WHERE age < 25`
	`db.people.find(`
	`    { age: { $lt: 25 } }`
	`)`

MongoDB: comparison operators (<=)

MySQL	MongoDB	Description
>	`$gt`	greater than
>=	`$gte`	greater equal then
<	`$lt`	less than
<=	**`$lte`**	**less equal then**

SELECT *
FROM people
WHERE **age <= 25**
	db.people.find(
	    { **age: { $lte: 25 }** }
	)

MongoDB: comparison operators (=)

MySQL	MongoDB	Description
>	$gt	greater than
>=	$gte	greater equal then
<	$lt	less than
<=	$lte	less equal then
=	**$eq**	**equal to** The $eq expression is equivalent to { field: <value> }.

SELECT *
FROM people
WHERE **age = 25**
	db.people.find(
	    { **age: { $eq: 25 }** }
	)

MongoDB: comparison operators (!=)

MySQL	MongoDB	Description
>	$gt	greater than
>=	$gte	greater equal then
<	$lt	less than
<=	$lte	less equal then
=	$eq	equal to
!=	**$neq**	**Not equal to**

SELECT *
FROM people
WHERE **age != 25**
	db.people.find(
	    { **age: { $neq: 25 }** }
	)

MongoDB: conditional operators

- To specify multiple conditions, **conditional operators** are used
- MongoDB offers the same functionalities of MySQL with a different syntax.

MySQL	**MongoDB**	**Description**
AND	,	Both verified
OR	$or	At least one verified

MongoDB: conditional operators (AND)

MySQL	MongoDB	Description
AND	,	Both verified

```
SELECT *
FROM people
WHERE status = "A"
AND age = 50
```

```
db.people.find(
     { status: "A",
       age: 50 }
)
```

MongoDB: conditional operators (OR)

MySQL	MongoDB	Description
AND	,	Both verified
OR	**$or**	**At least one verified**

SELECT *
 FROM people
 WHERE status = "A"
 OR age = 50
	db.people.find(
	 { **$or**:
	 [{ status: "A" } **,**
	 { age: 50 }
	]
	 }
	)

MongoDB: Cursor

- `db.collection.find()` gives back a cursor. It can be used to iterate over the result or as input for next operations.
- E.g.,
  - `cursor.sort()`
  - `cursor.count()`
  - `cursor.forEach() //shell method`
  - `cursor.limit()`
  - `cursor.max()`
  - `cursor.min()`
  - `cursor.pretty()`

MongoDB: Cursor

Cursor examples:

```
db.people.find({ status: "A"}).count()
```

- Select documents with status="A" and count them.

```
db.people.find({ status: "A"}).forEach(
   function(myDoc) { print( "user: ”+myDoc.name );
   })
```

- `forEach` applies a JavaScript function to apply to each document from the cursor.
  - Select documents with status="A" and print the document name.

MongoDB: sorting data

- Sort is a cursor method
- Sort documents
  - `sort( {<list of field:value pairs>} );`
  - field specifies which filed is used to sort the returned documents
  - value = -1 descending order
  - Value = 1 ascending order
- Multiple field: value pairs can be specified
  - Documents are sort based on the first field
  - In case of ties, the second specified field is considered

MongoDB: sorting data

E.g.,

```
db.people.find({ status: "A"}).sort({age:1})
```

- Select documents with status="A" and sort them in ascending order based on the age value
  - Returns all documents having status="A". The result is sorted in ascending age order

MongoDB: sorting data

Sorting data with respect to a given field in MongoDB: sort() operator

MySQL clause	MongoDB operator
`ORDER BY`	`sort()`

`SELECT *`
`FROM people`
`WHERE status = "A"`
`ORDER BY user_id ASC`
	`db.people.find(`
	`  { status: "A" }`
	`).`**`sort( { user_id: 1 } )`**

MongoDB: sorting data

Sorting data with respect to a given field in MongoDB: sort() operator

MySQL clause	MongoDB operator
`ORDER BY`	`sort()`

`SELECT *`
`FROM people`
`WHERE status = "A"`
`ORDER BY user_id ASC`
	`db.people.find(`
	`  { status: "A" }`
	`).`**`sort( { user_id: 1 } )`**

`SELECT *`
`FROM people`
`WHERE status = "A"`
`ORDER BY user_id DESC`
	`db.people.find(`
	`  { status: "A" }`
	`).`**`sort( { user_id: -1 } )`**

MongoDB: counting

MySQL clause	MongoDB operator
`COUNT`	`count()` or `find().count()`

`SELECT COUNT(*)`
`FROM people`
	`db.people.count()`
	or
	`db.people.find().count()`

MongoDB: counting

MySQL clause	MongoDB operator
`COUNT`	`count()` or `find().count()`

- Similar to the find() operator, count() can embed conditional statements.

`SELECT COUNT(*)`
`FROM people`
`WHERE age > 30`
	`db.people.count(`
	`{ age: { $gt: 30 } }`
	`)`