Northwestern
INFORMATION TECHNOLOGY

NoSQL Databases: MongoDB

Scott Coughlin, Computational Specialist

Some slides adapted from Professor Michael Coughlin and Matthew Graham

July 19th 2022

Databases, SQL, and Django

Topics Covered

1. What drove the development of NoSQL? - Pros and cons of traditional SQL systems
2. NoSQL: motivation and promise
3. What are the different flavors of NoSQL?
4. Some SQL to MongoDB Command Mapping

What drove the development of NoSQL?

Pros of traditional SQL systems

- Have been around for quite a while
- SQL: mature and powerful
- Transactions + ACID-compliance built-in
- Data normalization + joins
- Schema: good when data can be represented in appropriate way
- Many open-source solutions

Reminder of ACID

By definition, a database transaction must be:

- **A**tomic: all or nothing
- **C**onsistent: no integrity constraints violated
- **I**solated: does not interfere with any other transaction
- **D**urable: committed transaction effects persist

Cons of traditional SQL systems

- Horizontal scale-out | scaling issues
- Joins + transactions across multiple databases quickly become costly for complicated objects and big data -> performance and availability are affected
- Object-relational impedance mismatch
- Schema: bad for unstructured/evolving data

NoSQL: motivation and promise

NoSQL: motivation and promise

The design and development of NoSQL databases has been largely driven by the RDBMS cons from the previous slide.

- Deliver performance for big, potentially unstructured or evolving data that may come in in real time
- Simple design
- Horizontal scaling
- NoSQL ~= not only SQL, i.e. some systems support SQL-like query languages

NoSQL: motivation and promise

Always comes with a cost!

- Most such systems lack ACID transactions and offer BASE (Basic Availability, Soft state and Eventual consistency) instead
- Some systems exhibit potential lost writes and other forms of data loss but not MongoDB
- No schema means data integrity might become an issue
- Some systems do allow defining schemas and perform validation/enforcement (e.g. MongoDB)
- Many query languages vs single SQL (albeit with different flavors)

NoSQL: relational data

In practice, you almost always still need to deal with relational data! There are three main techniques to do that:

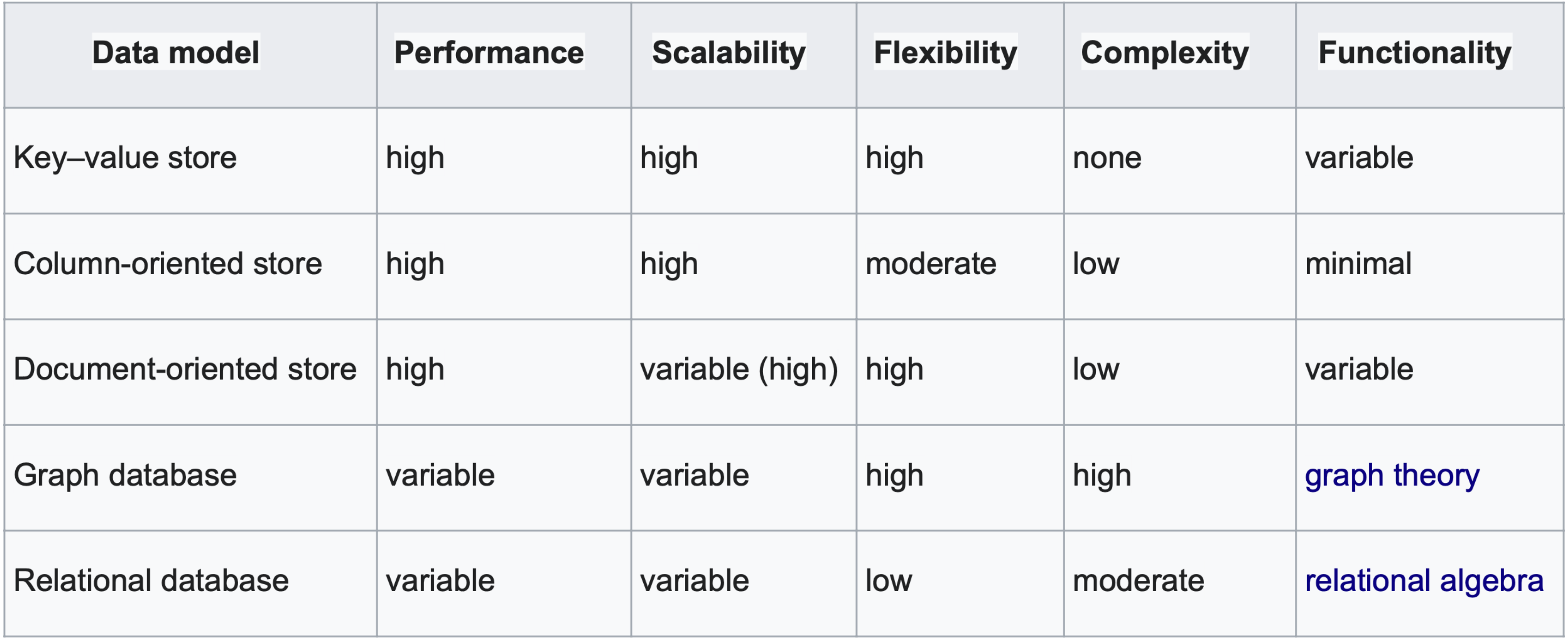
- Nesting/embedding data
- Store all data needed for a specific task in one place (e.g. in a single document)
- Linking + Multiple queries
- Store a foreign key and fetch data in multiple queries.
- Caching and replication
- Instead of storing a foreign key, store the actual values

Main types of NoSQL databases

NoSQL: What are the different flavors of NoSQL?

- Key-value store
  - Uses maps/dictionaries/associated lists/hash tables with corresponding operation complexities
  - Examples: Redis, ArangoDB, ZooKeeper, Couchbase, Cassandra, Amazon DynamoDB
- Wide column store
  - Essentially, a two-dimensional semi-structured key-value store
  - Examples: Cassandra, HBase
- Document store
  - Semi-structured; data are encapsulated in some standard form (XML, JSON, BSON) in "documents" with unique keys/identifiers
  - Often uses B-trees with corresponding operation complexities
  - Examples: Couchbase, MongoDB, Amazon DynamoDB
- Graph store
  - Uses graphs to represent data + relationship between them (the latter can be queried, too)
  - Examples: Neo4J, ArangoDB

NoSQL: What are the performance differences for these flavors of NoSQL?



Data model	Performance	Scalability	Flexibility	Complexity	Functionality
Key–value store	high	high	high	none	variable
Column-oriented store	high	high	moderate	low	minimal
Document-oriented store	high	variable (high)	high	low	variable
Graph database	variable	variable	high	high	graph theory
Relational database	variable	variable	low	moderate	relational algebra

MongoDB: Document-Based NoSQL

MongoDB

- Uses BSON (serialized binary **python-dictionary-like** structures) **documents** to store the data in **collections** (a rough analog of a **table** in the relational DBMS world).
- Giant B-tree (Binary Tree): O(log(N)) guaranteed for search, insert, and delete operations, where N is the number of documents in a collection.
- MongoDB Query Language (MQL): cone and general searches, aggregation pipelines
- Horizontal scale-out
- ACID-compliant transactions

MongoDB Commands Compared to SQL

In the following slides we compare Mongo commands to SQL

CREATE

CREATE TABLE *tableName* (name1 type1, name2 type2, …);

```
CREATE TABLE star (name varchar(20), ra float, dec float,
vmag float);
```

MongoDB: Create a Collection

```
db.people.insertOne( {
user_id: "abc123",
age: 55,
status: "A"
} )
```

ALTER

ALTER TABLE *table …;*

```
ALTER TABLE star ADD COLUMN bmag double AFTER vmag;

ALTER TABLE star DROP COLUMN bmag;
```

MongoDB

```
db.people.updateMany(
{ },
{ $set: { join_date: new Date() } }
)
```

```
db.people.updateMany(
{ },
{ $unset: { "join_date": "" } }
)
```

DROP

DROP TABLE *table;*

```
DROP TABLE star;
```

MongoDB: Drop a Collection

```
db.people.drop()
```

INSERT

INSERT INTO *table* VALUES(val1, val2, ..);

```
INSERT INTO star VALUES('Sirius', 101.287, -16.716, -1.47);
```

Mongo: insertOne

```
db.people.insertOne(

{ user_id: "bcd001", age: 45, status: "A" }

)
```

SELECT

SELECT *column1, column2* FROM *table* WHERE *condition* (LIMIT *#ofrows)*
ORDER BY *sort_expression [ASC | DESC];*

```
SELECT name, constellation FROM star WHERE dec > 0 ORDER

BY vmag;
```

Mongo: find

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

```
db.people.distinct( "status" )
```

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

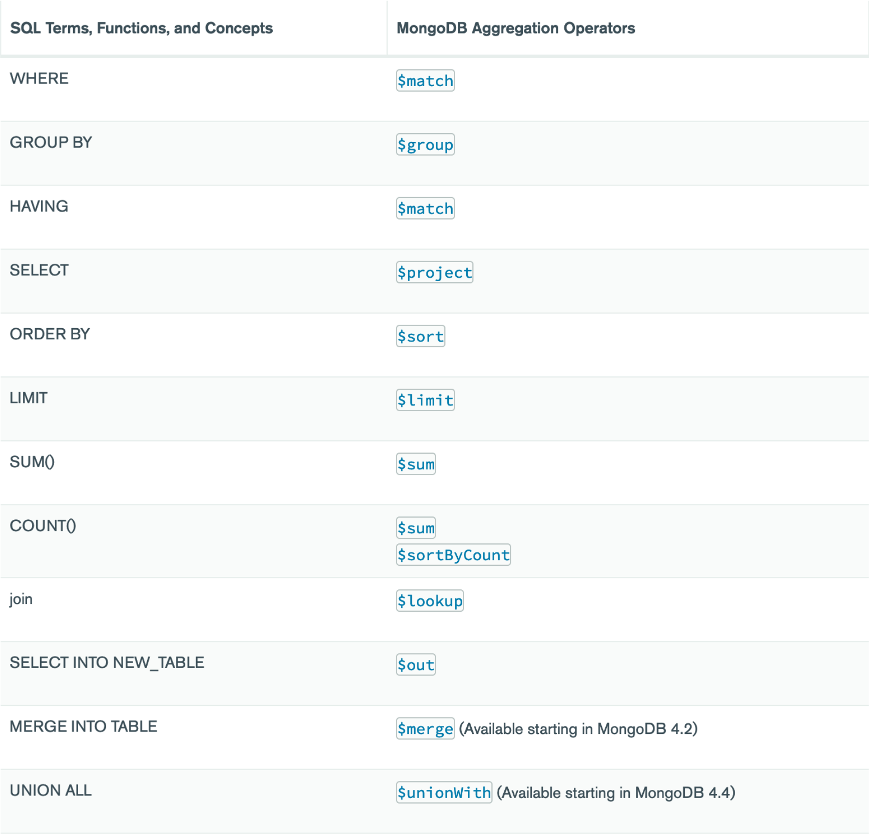
```
db.people.find(
{ status: "A" },
{ user_id: 1, status: 1, _id: 0 }
)
```

Aggregate Functions

```
SELECT cust_id,
ord_date,
SUM(price) AS total
FROM orders
GROUP BY cust_id,
ord_date
HAVING total > 250
```

```
db.orders.aggregate( [
{
$group: {
_id: {
cust_id: "$cust_id",
ord_date: { $dateToString: {
format: "%Y-%m-%d",
date: "$ord_date"
}}
},
total: { $sum: "$price" }
}
},
{ $match: { total: { $gt: 250 } } }
] )
```

Aggregate Functions



SQL Terms, Functions, and Concepts	MongoDB Aggregation Operators
WHERE	`$match`
GROUP BY	`$group`
HAVING	`$match`
SELECT	`$project`
ORDER BY	`$sort`
LIMIT	`$limit`
SUM()	`$sum`
COUNT()	`$sum` `$sortByCount`
join	`$lookup`
SELECT INTO NEW_TABLE	`$out`
MERGE INTO TABLE	`$merge` (Available starting in MongoDB 4.2)
UNION ALL	`$unionWith` (Available starting in MongoDB 4.4)

UPDATE

UPDATE *table* SET *column* = val1 WHERE condition;

```
UPDATE star SET vmag = vmag + 0.5;

UPDATE star SET vmag = -1.47 WHERE name LIKE 'Sirius';

UPDATE star INNER JOIN temp on star.id = temp.id SET star.vmag = temp.mag;
```

Mongo: updateMany

```
db.people.updateMany(
{ status: "A" } ,
{ $inc: { age: 3 } }
)
```

```
db.people.updateMany(
{ age: { $gt: 25 } },
{ $set: { status: "C" } }
)
```

Thank You