NO SQL DATABASE MANAGEMENT SYSTEM

LAB 10

SECTION 1

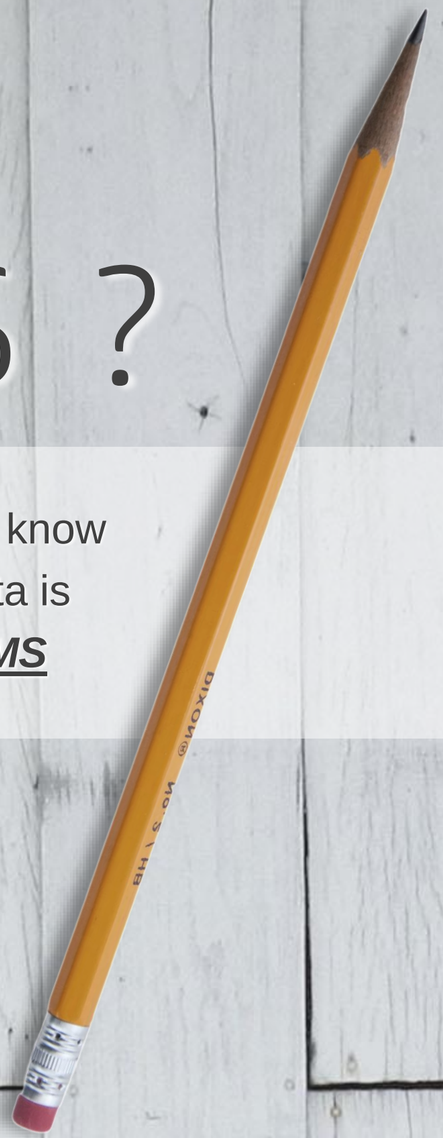
INTRODUCTION

Why we need databases?

Efficiently Storing & retrieving the data was a big challenge in FLAT FILE SYSTEM. Plus data security was also another challenge in flat file system. Means there was no standard implementation in FLAT FILE SYSTEM. In early 70’s database approach(particularly relational database approach came into being and still working effectively

What is No SQL DBMS ?

No Sql is one of the kind of Database Management system like Relational databases. We know that DBMS provide mechanism for retrieving and storing databases but as far as Big data is concern relational databases was not efficient for that purpose we go For ***No SQL DBMS***



NO SQL

The Word No SQL refers to NOT ONLY SQL

If data does not fall under relational database category it then goes for NO SQL

Basic idea is ***different data store architectures*** for specific problem set.

Generally we do not provide SQL to query data. Often it is more simple in NO SQL as APIs as interfaces to data are available

Tools are available that provides SQL dialects(standards as ORACLE or SQL) to NoSQL DBMS.

Which data we call as BIG Data

We know that now a days we have masses data like social media websites, Blogging databases and mobile application databases that have extensive data to be store

If size of data causes an organization to reconsider DB technology then this could be due to “**big data** volume”

Whose schema might potentially be unknown at design time

Why BIG DATA NEEDED

In past days we hardly have the data of terabytes but as far as present world is concern even the Petabytes, Zeta bytes, Exabyte's data seems smaller.

In past days we hardly have Minimal data networking structures but now a days universal high speed data networking has become the part of almost every system

In past days requirements can easily been full filled by using centralized computing but now a days distributed computing became the necessary part of efficient data computing

In past Memory was scarce and expensive but now a days you can find out it easily with cheaper prices

Types of NOSQL Databases

I

Document Stores

II

Graph Databases

III

Key Value Stores

IV

Columnar Databases

Types of NOSQL Databases

Key/value (examples are Dynamo, Voldemort):

key	value
123	123 Main St.
126	(805) 477-3900

Key value databases are basically hashtables as:

Hash table

What is hashing ? Which maintains hashtables

Hashing is one of the type of indexing where we perform transformation of a string of characters into a usually shorter fixed-length value or key that represents the original string. Hashing is used to index and retrieve items in a database because it is faster to find the item using the shorter hashed key than to find it using the original value.

Types of NOSQL Databases

Key — Value

Value is of BLOB data type → We can store whole user profile in value
Means against a single key

- Profile
- Order
- Address

Types of NOSQL Databases

Document (examples are Mongo DB, CouchDB):

JSON DOCUMENT

```
{
   '_id' : 1,
   'name' : { 'first' : 'John', 'last' : 'Backus' },
   'contribs' : [ 'Fortran', 'ALGOL', 'Backus-Naur Form', 'FP' ],
   'awards' : [
      {
         'award' : 'W.W. McDowell Award',
         'year' : 1967,
         'by' : 'IEEE Computer Society'
      }, {
         'award' : 'Draper Prize',
         'year' : 1993,
         'by' : 'National Academy of Engineering'
      }
   ]
}
```

FIRST PARMETER : ITS VALUE, SECOND PARMETER : ITS VALUE

FIRST PARMETER,SECOND PARAMETER

List of ‘awards’ ‘parameters’ : ‘their values’

A JSON database returns query results that can be easily parsed, with little or no transformation, directly by JavaScript and most popular programming languages – reducing the amount of logic you need to build into your application layer.

Types of NOSQL Databases

Document (examples are Mongo DB, CouchDB):

BSON DOCUMENT

MongoDB represents JSON documents in binary-encoded format called BSON behind the scenes. BSON extends the JSON model to provide additional data types and to be efficient for encoding and decoding within different languages.

MongoDB Goal of JSON to BSON

The MongoDB BSON implementation is lightweight, fast and highly traversable.

Like JSON, MongoDB's BSON implementation supports embedding objects and arrays within other objects and arrays – MongoDB can even 'reach inside' BSON objects to build indexes and match objects against query expressions on both top-level and nested BSON keys.

In short BSON implementation is Lightweight, traversable and efficient than JSON.

BSON also contains extensions that allow representation of data types that are not part of the JSON.

Types of NOSQL Databases

Document (examples are Mongo DB, CouchDB):

No SQL Databases donont fully support Relational database features like:
1. no join operations (except within partitions),
2. no referential integrity constraints across partitions.

BSON DOCUMENT

```
{
"_id" :        “XYZ"
"city" :       "ABC",
"pop" :        NNN,
"state" :      "TN",
“councilman” : {
                    name: “xyz”
                    address: “klm strret”
                }
}
```

_id field is a primary key of every Document

a partitioned collection is a collection that is broken into (or partitioned into) several individual collections, based on ranges of a “partition key”.
inserts, updates, and deletes just work with no syntactical changes, basically the data will be broken into several collections, with each collection responsible for all data for a range of the partition key.
We are not covering partitioning in depth as itself is a huge topic.

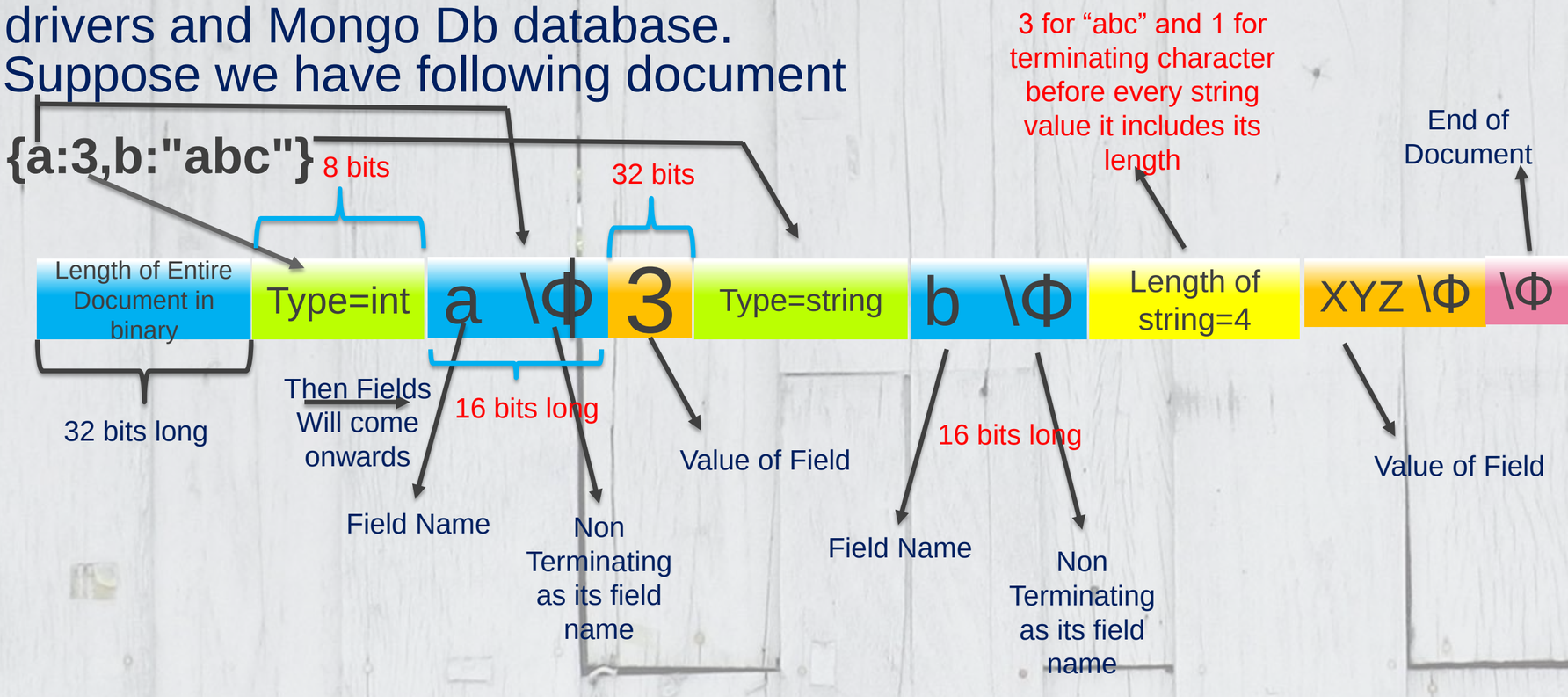
BSON Implementation

BSON Implementation

BSON TYPES with Implementation

Type	Number
Double	1
String	2
Object	3
Array	4
Binary data	5
Object id	7
Boolean	8
Date	9
Null	10
Regular Expression	11
JavaScript	13
Symbol	14
JavaScript (with scope)	15
32-bit integer	16
Timestamp	17
64-bit integer	18
Min key	255
Max key	127

Now lets just talk about actual BSON implementation.
In fact you shouldn't care about what the format is because it is some sort of internal implementation of BSON and is used by Mongo DV drivers and Mongo Db database.
Suppose we have following document



MONGO VS SQL

MONGO VS SQL

Mongo

Every record is basically ***Document***

Collection of Records(documents) is called as ***Collection***

Primary key is ***_id field***

We use ***Embedded Documents***

use ***Reference***

SQL(RDBMS)

Every record is basically ***Tuple***

Collection of Records(Tuple) is called as ***Table***

Primary key is ***Any attribute***

We use ***Joins***

use ***Foreign Key***

SECTION 2

Getting Started with mongoDB

WELCOME MESSAGE

To install mongoDB, go to this link and click on the appropriate OS and architecture:

http://www.mongodb.org/downloads

Introductory Video

Few basic Shell commands

In **Mongodb** check which db you're using write ***db*** command.

```
C:\Users\Ali Akhtar>mongo
MongoDB shell version: 3.2.10
connecting to: test
2016-10-02T04:37:44.649-0700 W NETWORK  [thread1] Failed to connect to 127.0.0.1:27017, reason: errno:10061 No connectio
n could be made because the target machine actively refused it.
2016-10-02T04:37:44.655-0700 E QUERY    [thread1] Error: couldn't connect to server 127.0.0.1:27017, connection attempt
failed :
connect@src/mongo/shell/mongo.js:229:14
@(connect):1:6

exception: connect failed

C:\Users\Ali Akhtar>mongo
MongoDB shell version: 3.2.10
connecting to: test
> db
test
```

In **oracle** what we do to check our database?

```
SQL> select ora_database_name from dual;

ORA_DATABASE_NAME
--------------------------------------------------------------------------------
ORCL
```

Few basic Shell commands

In **Mongodb** to see all databases names with spaces acquired by them is (we only have one db right now)

```
> show dbs
local  0.000GB
```

In **oracle** what we do to check our databases. We basically logon to SYSDBA and write following command

```
SQL> select * from dba_users;
rows will be truncated
```

USERNAME	USER_ID	PASSWORD	ACCOUNT_STATUS
MGMT_VIEW	74		OPEN
SYS	0		OPEN
SYSTEM	5		OPEN
DBSNMP	30		OPEN
SYSMAN	72		OPEN
SCOTT	84		OPEN
HR	85		OPEN
OUTLN	9		EXPIRED & LOCKED
FLOWS_FILES	75		EXPIRED & LOCKED
MDSYS	57		EXPIRED & LOCKED
ORDSYS	53		EXPIRED & LOCKED
EXFSYS	42		EXPIRED & LOCKED
WMSYS	32		EXPIRED & LOCKED
APPQOSSYS	31		EXPIRED & LOCKED
APEX_030200	78		EXPIRED & LOCKED
OWBSYS_AUDIT	83		EXPIRED & LOCKED
ORDDATA	54		EXPIRED & LOCKED
CTXSYS	43		EXPIRED & LOCKED
ANONYMOUS	46		EXPIRED & LOCKED
XDB	45		EXPIRED & LOCKED
ORDPLUGINS	55		EXPIRED & LOCKED
OWBSYS	79		EXPIRED & LOCKED
SI_INFORMTN_SCHEMA	56		EXPIRED & LOCKED
OLAPSYS	61		EXPIRED & LOCKED
ORACLE_OCM	21		EXPIRED & LOCKED
XS$NULL	2147483638		EXPIRED & LOCKED
BI	90		EXPIRED & LOCKED

Few basic Shell commands

In **Mongodb** we use command **USE DatabaseName either to switch to particular database or to create new database**

```
> use local
switched to db local
```

Lets create new database with name **test**

```
> use test
switched to db test
```

But if we will type **show dbs** this database will not come in our list until we insert atleast one document into it.

```
> show dbs
local  0.000GB
```

Lets insert a row in our database where we are standing write db to check database name first

```
> db
test
```

Few basic Shell commands

In **Mongodb** we use command **USE DatabaseName** either to switch to a particular database or to create new database

Lets insert a movie Collection(like table in Oracle or RDBMS we discussed above) in test database. This Is the beauty of MongoDB that direct insertion of record(document) create a collection.

```
> db.movie.insert({"name":"tutorials point"})
WriteResult({ "nInserted" : 1 })
```

Record inserted

Lets check databases by writing **show dbs**

```
> show dbs
local   0.000GB
test    0.000GB
```

Test database is now visible

In **Mongodb** you can see all collection of a particular database like we are in test by writing Command **Show Collections**

```
> show collections
movie
```

Few basic Shell commands

In **Mongodb** we have also seen how to insert a collection as:

db.<collection>.insert(<document>)

Same as table IN RDBMS

Same as Row in RDBMS

In **oracle** what we do for insertion ?

INSERT INTO <table>
VALUES(<attributevalues>);

In **Mongodb** multiple record can be inserted by using array. Let insert 20 more records so there will be total 21 records.

Few basic Shell commands

```
db.movie.insert(
  [
    { "name":"Titanic" },
    { "name":"Captain America" },
    { "name":"Inception" },
            { "name":"Dabang" },
            { "name":"Bol" },
            { "name":"The Avengers" },
            { "name":"Harry Potter" },
            { "name":"Raees" },
            { "name":"Star Wars" },
            { "name":"Avatar" },
            { "name":"The Jungle Book" },
            { "name":"Bin Roye" },
            { "name":"Wrong No" },
            { "name":"3 idiots" },
            { "name":"sholay" },
                    { "name":"PK" },
                    { "name":"DDLJ" },
                    { "name":“Undownloadable Lab Videos" },
                    { "name":"War" },
                    { "name":"Spider Man 2" },
    ]
)
```

Few basic Shell commands

In **Mongodb** we use db.<Collection>.find() command which retrieve first 20 records of a collection

```
> db.movie.find()
{ "_id" : ObjectId("57f103b4a3e7521c75e6829a"), "name" : "tutorials point" }
{ "_id" : ObjectId("57f1154ca3e7521c75e6829b"), "name" : "Titanic" }
{ "_id" : ObjectId("57f1154ca3e7521c75e6829c"), "name" : "Captain America" }
{ "_id" : ObjectId("57f1154ca3e7521c75e6829d"), "name" : "Inception" }
{ "_id" : ObjectId("57f1154ca3e7521c75e6829e"), "name" : "Dabang" }
{ "_id" : ObjectId("57f1154ca3e7521c75e6829f"), "name" : "Bol" }
{ "_id" : ObjectId("57f1154ca3e7521c75e682a0"), "name" : "The Avengers" }
{ "_id" : ObjectId("57f1154ca3e7521c75e682a1"), "name" : "Harry Potter" }
{ "_id" : ObjectId("57f1154ca3e7521c75e682a2"), "name" : "Raees" }
{ "_id" : ObjectId("57f1154ca3e7521c75e682a3"), "name" : "Star Wars" }
{ "_id" : ObjectId("57f1154ca3e7521c75e682a4"), "name" : "Avatar" }
{ "_id" : ObjectId("57f1154ca3e7521c75e682a5"), "name" : "The Jungle Book" }
{ "_id" : ObjectId("57f1154ca3e7521c75e682a6"), "name" : "Bin Roye" }
{ "_id" : ObjectId("57f1154ca3e7521c75e682a7"), "name" : "Wrong No" }
{ "_id" : ObjectId("57f1154ca3e7521c75e682a8"), "name" : "3 idiots" }
{ "_id" : ObjectId("57f1154ca3e7521c75e682a9"), "name" : "sholay" }
{ "_id" : ObjectId("57f1154ca3e7521c75e682aa"), "name" : "PK" }
{ "_id" : ObjectId("57f1154ca3e7521c75e682ab"), "name" : "DDLJ" }
{ "_id" : ObjectId("57f1154ca3e7521c75e682ac"), "name" : "Undownloadable Lab Videos" }
{ "_id" : ObjectId("57f1154ca3e7521c75e682ad"), "name" : "War" }
Type "it" for more
```

```
> it
{ "_id" : ObjectId("57f1188ba3e7521c75e682ae"), "name" : "Spider Man 2" }
```

You can’t view the last 21st record of “Spider Man 2” movie you have to type “it” to see more records

Few basic Shell commands

To print at a time 21 or n records we need to increase the batch size

```
> DBQuery.shellBatchSize = 21
21
```

```
> db.movie.find()
{ "_id" : ObjectId("57f103b4a3e7521c75e6829a"), "name" : "tutorials point" }
{ "_id" : ObjectId("57f1154ca3e7521c75e6829b"), "name" : "Titanic" }
{ "_id" : ObjectId("57f1154ca3e7521c75e6829c"), "name" : "Captain America" }
{ "_id" : ObjectId("57f1154ca3e7521c75e6829d"), "name" : "Inception" }
{ "_id" : ObjectId("57f1154ca3e7521c75e6829e"), "name" : "Dabang" }
{ "_id" : ObjectId("57f1154ca3e7521c75e6829f"), "name" : "Bol" }
{ "_id" : ObjectId("57f1154ca3e7521c75e682a0"), "name" : "The Avengers" }
{ "_id" : ObjectId("57f1154ca3e7521c75e682a1"), "name" : "Harry Potter" }
{ "_id" : ObjectId("57f1154ca3e7521c75e682a2"), "name" : "Raees" }
{ "_id" : ObjectId("57f1154ca3e7521c75e682a3"), "name" : "Star Wars" }
{ "_id" : ObjectId("57f1154ca3e7521c75e682a4"), "name" : "Avatar" }
{ "_id" : ObjectId("57f1154ca3e7521c75e682a5"), "name" : "The Jungle Book" }
{ "_id" : ObjectId("57f1154ca3e7521c75e682a6"), "name" : "Bin Roye" }
{ "_id" : ObjectId("57f1154ca3e7521c75e682a7"), "name" : "Wrong No" }
{ "_id" : ObjectId("57f1154ca3e7521c75e682a8"), "name" : "3 idiots" }
{ "_id" : ObjectId("57f1154ca3e7521c75e682a9"), "name" : "sholay" }
{ "_id" : ObjectId("57f1154ca3e7521c75e682aa"), "name" : "PK" }
{ "_id" : ObjectId("57f1154ca3e7521c75e682ab"), "name" : "DDLJ" }
{ "_id" : ObjectId("57f1154ca3e7521c75e682ac"), "name" : "Undownloadable Lab Videos" }
{ "_id" : ObjectId("57f1154ca3e7521c75e682ad"), "name" : "War" }
{ "_id" : ObjectId("57f1188ba3e7521c75e682ae"), "name" : "Spider Man 2" }
>
```

Now you can see 21 records at a time

```
> db.movie.find().limit(2)
{ "_id" : ObjectId("57f103b4a3e7521c75e6829a"), "name" : "tutorials point" }
{ "_id" : ObjectId("57f1154ca3e7521c75e6829b"), "name" : "Titanic" }
```

To Restrict number of rows to let say 2 we use **db.<Collection>.find().limit(2)**

Few basic Shell commands

To find a particular record we use command

db.movie.findOne(
{ "name":"Spider Man 2" }
)

```
> db.movie.findOne(
...    { "name":"Spider Man 2" }
... )
{ "_id" : ObjectId("57f1188ba3e7521c75e682ae"), "name" : "Spider Man 2" }
```

In **Mongodb** to delete record **db.<collection>.remove({<field>:<value>})**

```
> db.movie.remove({ "name":"Undownloadable Lab Videos" })
WriteResult({ "nRemoved" : 1 })
```

Few basic Shell commands

```
> db.movie.find()
{ "_id" : ObjectId("57f103b4a3e7521c75e6829a"), "name" : "tutorials point" }
{ "_id" : ObjectId("57f1154ca3e7521c75e6829b"), "name" : "Titanic" }
{ "_id" : ObjectId("57f1154ca3e7521c75e6829c"), "name" : "Captain America" }
{ "_id" : ObjectId("57f1154ca3e7521c75e6829d"), "name" : "Inception" }
{ "_id" : ObjectId("57f1154ca3e7521c75e6829e"), "name" : "Dabang" }
{ "_id" : ObjectId("57f1154ca3e7521c75e6829f"), "name" : "Bol" }
{ "_id" : ObjectId("57f1154ca3e7521c75e682a0"), "name" : "The Avengers" }
{ "_id" : ObjectId("57f1154ca3e7521c75e682a1"), "name" : "Harry Potter" }
{ "_id" : ObjectId("57f1154ca3e7521c75e682a2"), "name" : "Raees" }
{ "_id" : ObjectId("57f1154ca3e7521c75e682a3"), "name" : "Star Wars" }
{ "_id" : ObjectId("57f1154ca3e7521c75e682a4"), "name" : "Avatar" }
{ "_id" : ObjectId("57f1154ca3e7521c75e682a5"), "name" : "The Jungle Book" }
{ "_id" : ObjectId("57f1154ca3e7521c75e682a6"), "name" : "Bin Roye" }
{ "_id" : ObjectId("57f1154ca3e7521c75e682a7"), "name" : "Wrong No" }
{ "_id" : ObjectId("57f1154ca3e7521c75e682a8"), "name" : "3 idiots" }
{ "_id" : ObjectId("57f1154ca3e7521c75e682a9"), "name" : "sholay" }
{ "_id" : ObjectId("57f1154ca3e7521c75e682aa"), "name" : "PK" }
{ "_id" : ObjectId("57f1154ca3e7521c75e682ab"), "name" : "DDLJ" }
{ "_id" : ObjectId("57f1154ca3e7521c75e682ad"), "name" : "War" }
{ "_id" : ObjectId("57f1188ba3e7521c75e682ae"), "name" : "Spider Man 2" }
```

You can see our one record has been removed

Few basic Shell commands

To remove the first document that match a deletion criteria we need to call the remove method with true parameter a part of it:

Like command equivalent in Mongodb is as:

Like "%ab%“ = /ab/ **&** Like "ab%“ = /^ab/

Like "%ab “ = /ab^/

To print Movies starting with B we will type:

```
> db.movie.find(
...     { name: /^B/ }
... )
{ "_id" : ObjectId("57f1154ca3e7521c75e6829f"), "name" : "Bol" }
{ "_id" : ObjectId("57f1154ca3e7521c75e682a6"), "name" : "Bin Roye" }
```

Now to delete first row of number of rows that contain B as their first letter we will

db.products.remove({ name: /^B/ }, true)

```
> db.movie.remove(     { name:/^B/ },true )
WriteResult({ "nRemoved" : 1 })
```

Few basic Shell commands

Now we will see from the respective condition only one record have been deleted. Others will be there as it is:

```
> db.movie.find(     { name: /^B/ } )
{ "_id" : ObjectId("57f1154ca3e7521c75e682a6"), "name" : "Bin Roye" }
```

In **Mongodb** to delete all records pass an empty document

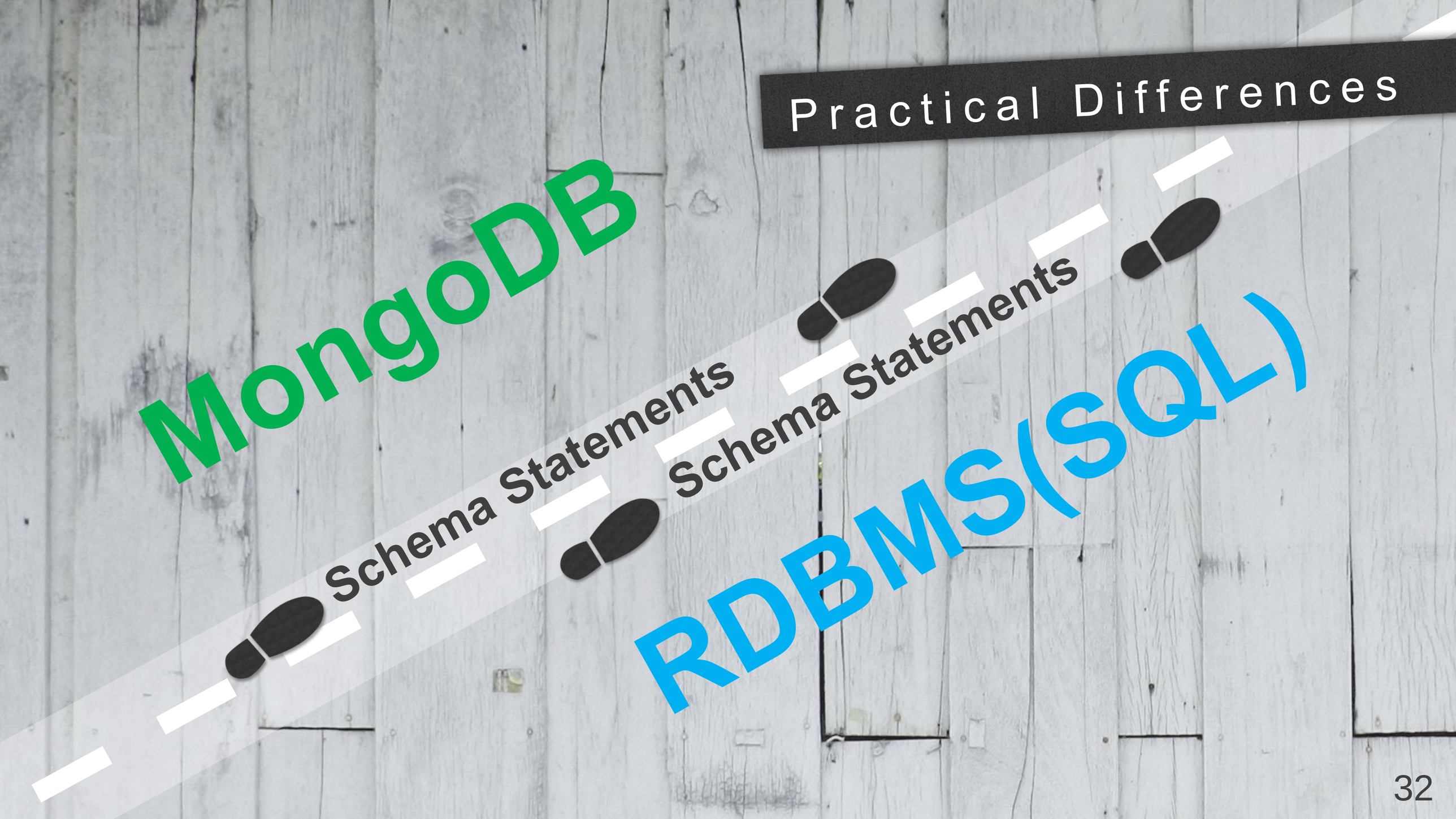
db.movie.remove(
{ })

```
> db.movie.remove(
...     { }   )
WriteResult({ "nRemoved" : 19 })
```

```
> db.movie.find()
>
```

Here you can see no records have been found

Practical Differences



SQL(RDBMS)

```
CREATE TABLE users (
    id number(4) not null,
    user_id Varchar(30),
    age Number(3),
    status char(1),
    PRIMARY KEY (id)
)
```

```
SQL> CREATE TABLE users (
  2      id number(4) not null,
  3      user_id Varchar(30),
  4      age Number(3),
  5      status char(1),
  6      PRIMARY KEY (id)
  7* )
SQL> /

Table created.
```

Mongo

Collection automatically created on first insert operation. The primary key _id is automatically added if _id field is not specified.

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

```
> db.users.insert( {
...     user_id: "abc123",
...     age: 55,
...     status: "A"
... } )
WriteResult({ "nInserted" : 1 })
```

However, you can also explicitly create a collection:

```
db.createCollection("users")
```

SQL(RDBMS)

ALTER TABLE users
ADD join_date DATETIME

Mongo

Collections do not describe or enforce the structure of its documents; i.e. there is no structural alteration at the collection level.

However, at the document level, update() operations can add fields to existing documents using the $set operator.

```
db.users.update(
    { },
    { $set: { join_date: new Date() } },
    { multi: true }
)
```

To update multiple documents, set the multi option to true.

new Date() returns the current date as a Date object

SQL(RDBMS)

ALTER TABLE users

DROP COLUMN join_date

Mongo

Collections do not describe or enforce the structure of its documents; i.e. there is no structural alteration at the collection level.

However, at the document level, update() operations can add fields to existing documents using the $set operator.

```
db.users.update(
    { },
    { $unset: { join_date: “” } },
    { multi: true }
)
```

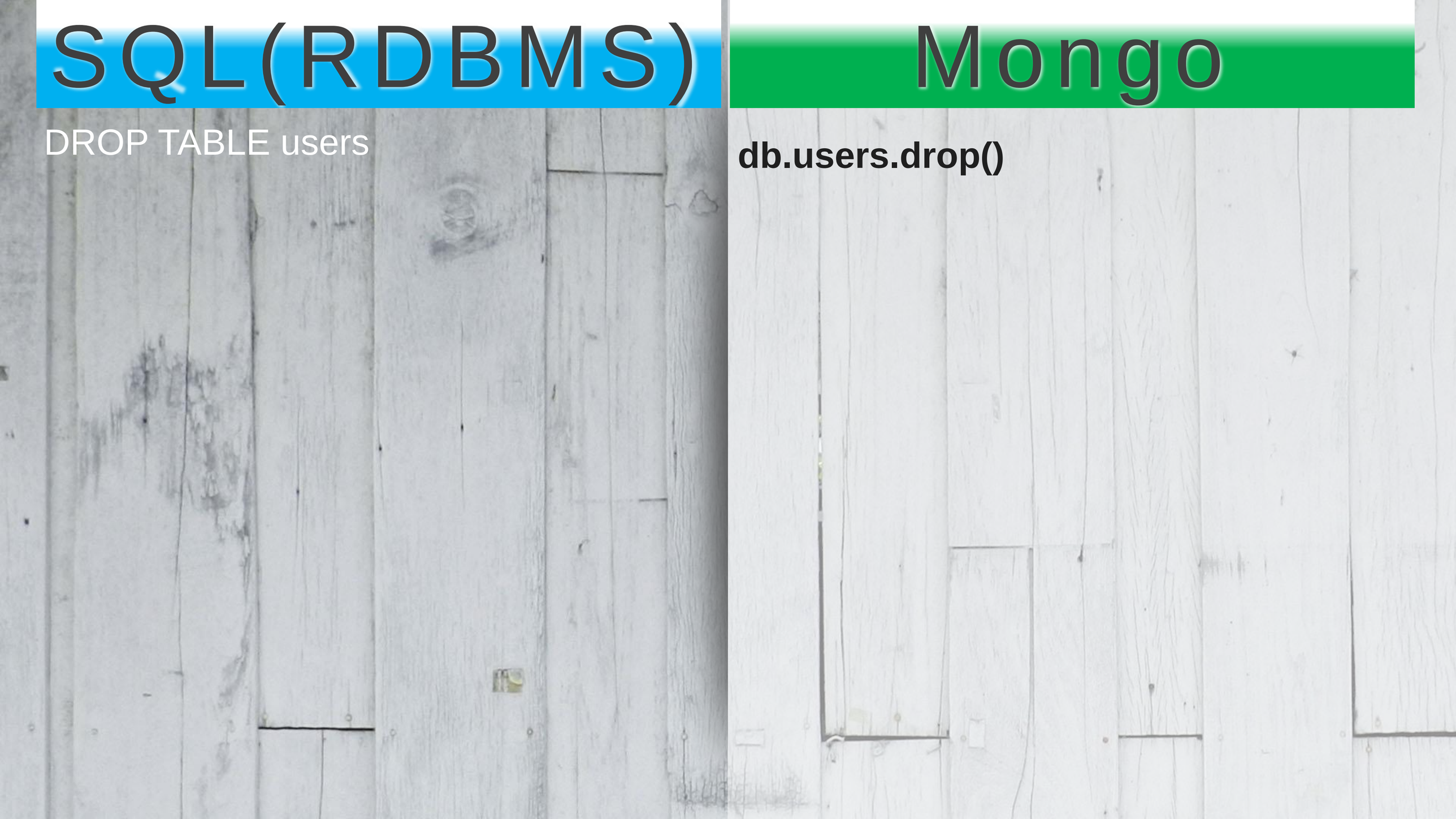
To update multiple documents, set the multi option to true.

SQL(RDBMS)

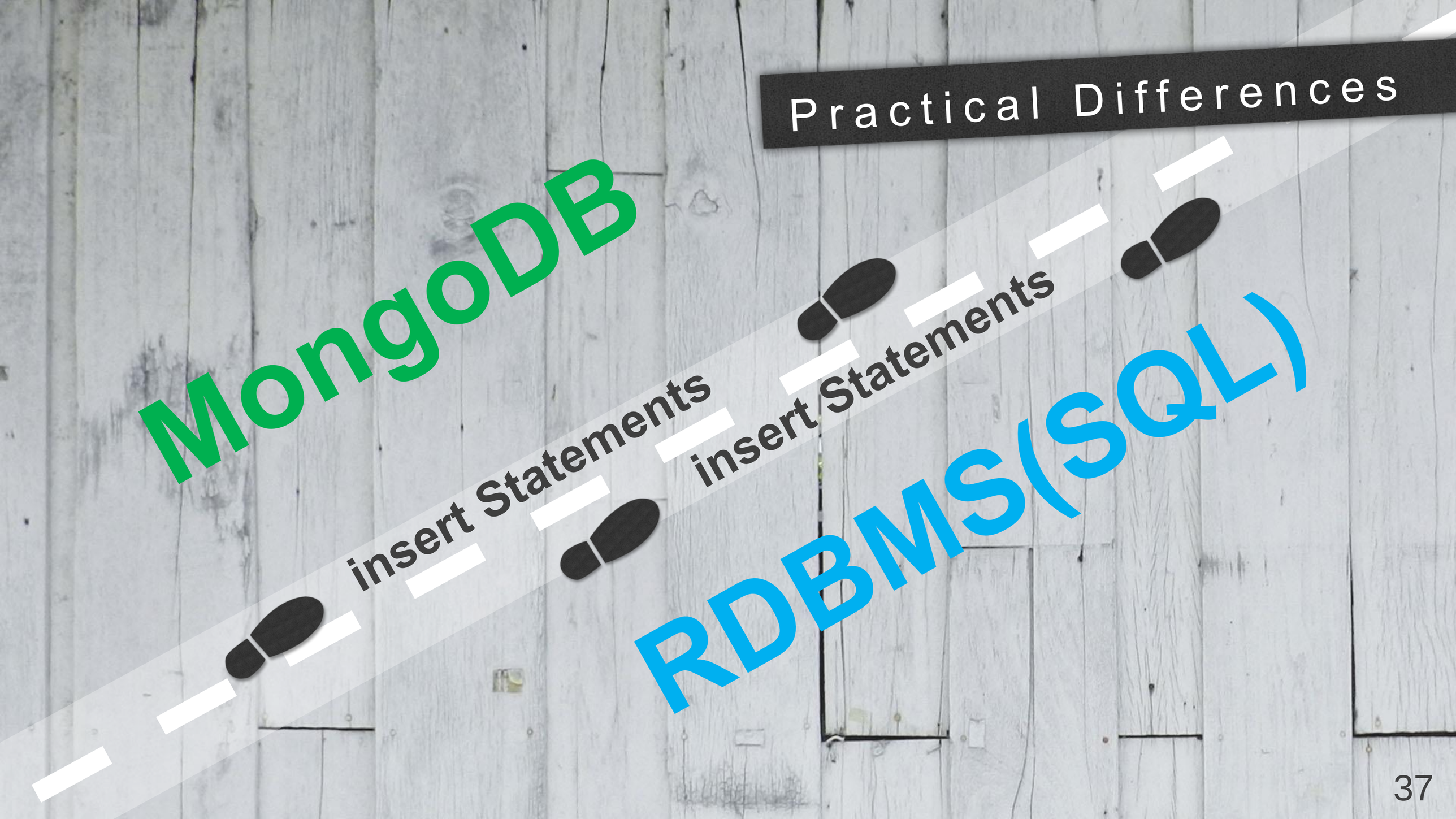
DROP TABLE users

Mongo

db.users.drop()



Practical Differences



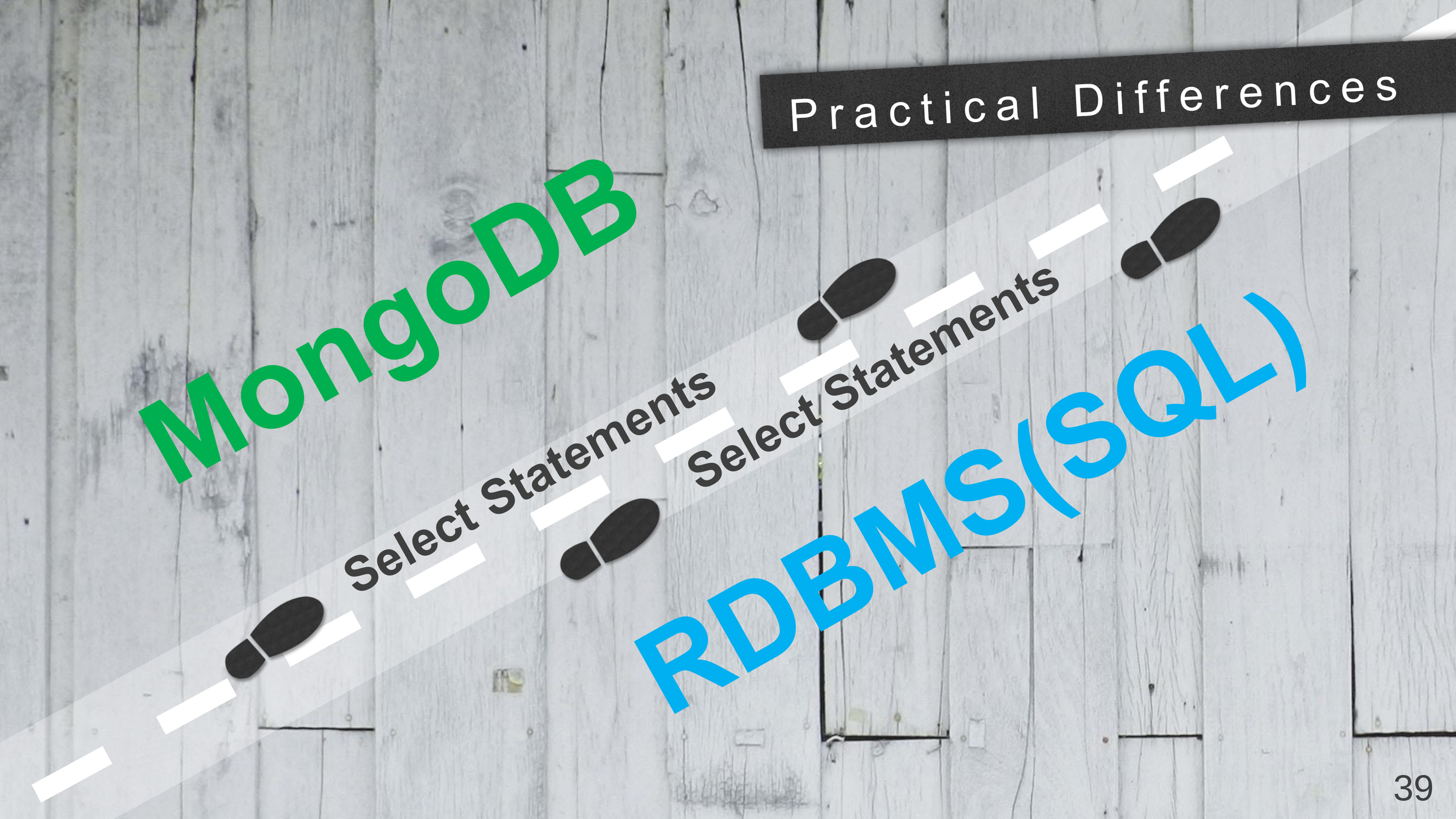
SQL(RDBMS)

```
INSERT INTO users(user_id,
                  age,
                  status)
VALUES ("bcd001",
        45,
        "A")
```

Mongo

```
db.users.insert( {
      user_id: "bcd001",
      age: 45,
      status: "A"
} )
```

Practical Differences



SQL(RDBMS)	Mongo
SELECT * FROM users	db.users.find()
SELECT id, user_id, status FROM users	db.users.find({ }, { user_id: 1, status: 1 }) Column value set to zero means we don't want to Print that value similarly 1 for printing that value Default value of id is always 1 so dnt need to mention.
SELECT user_id, status FROM users	db.users.find({ }, { user_id: 1, status: 1, _id: 0 })
SELECT * FROM users WHERE status = "A"	db.users.find({ status: "A" })
SELECT user_id, status FROM users WHERE status = "A"	db.users.find({ status: "A" }, { user_id: 1, status: 1, _id: 0 })

SQL(RDBMS)

```
SELECT *
FROM users
WHERE status != "A"
```

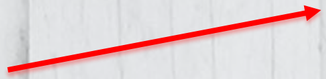
```
SELECT *
FROM users
WHERE status = "A"
AND age = 50
```

```
SELECT *
FROM users
WHERE status = "A"
OR age = 50
```

```
SELECT *
FROM users
WHERE age > 25
```

```
SELECT *
FROM users
WHERE age < 25
```

Mongo



Use only Comma for And Operation

```
db.users.find(
   { $or: [ { status: "A" } ,
            { age: 50 } ] }
)
```

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

```
db.users.find(
  { age: { $lt: 25 } }
)
```

SQL(RDBMS)

```
SELECT *
FROM users
WHERE age > 25
AND   age <= 50
```

```
SELECT *
FROM users
WHERE user_id like "%bc%"
```

```
SELECT *
FROM users
WHERE user_id like "bc%"
```

```
SELECT *
FROM users
WHERE status = "A"
ORDER BY user_id ASC
```

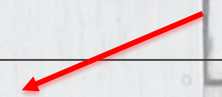
Mongo

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

```
db.users.find(
   { user_id: /bc/ }
)
```

```
db.users.find(
   { user_id: /^bc/ }
)
```

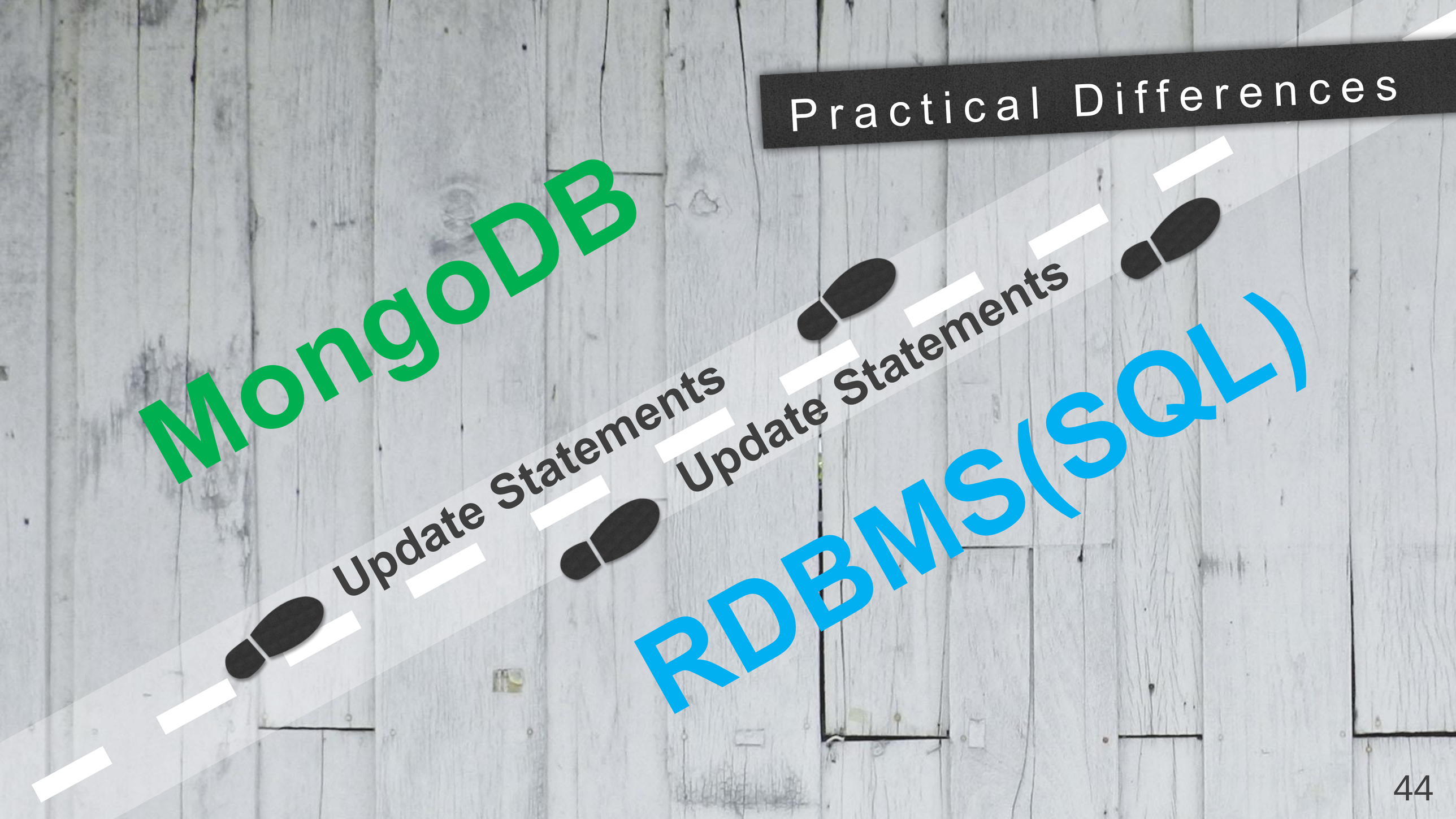
```
db.users.find( { status: "A" } ).sort( { user_id: 1 } )
```



1 for ascending (inside a sort Function) and -1 for descending

SQL(RDBMS)	Mongo
SELECT * FROM users WHERE status = "A" ORDER BY user_id DESC	db.users.find({ status: "A" }).sort({ user_id: -1 })
SELECT COUNT(*) FROM users	db.users.count() OR db.users.find().count()
SELECT COUNT(user_id) FROM users	db.users.count({ user_id: { $exists: true } }) OR db.users.find({ user_id: { $exists: true } }).count()
SELECT COUNT(*) FROM users WHERE age > 30	db.users.count({ age: { $gt: 30 } }) OR db.users.find({ age: { $gt: 30 } }).count()

Practical Differences



SQL(RDBMS)

```
UPDATE users
SET status = "C"
WHERE age > 25
```

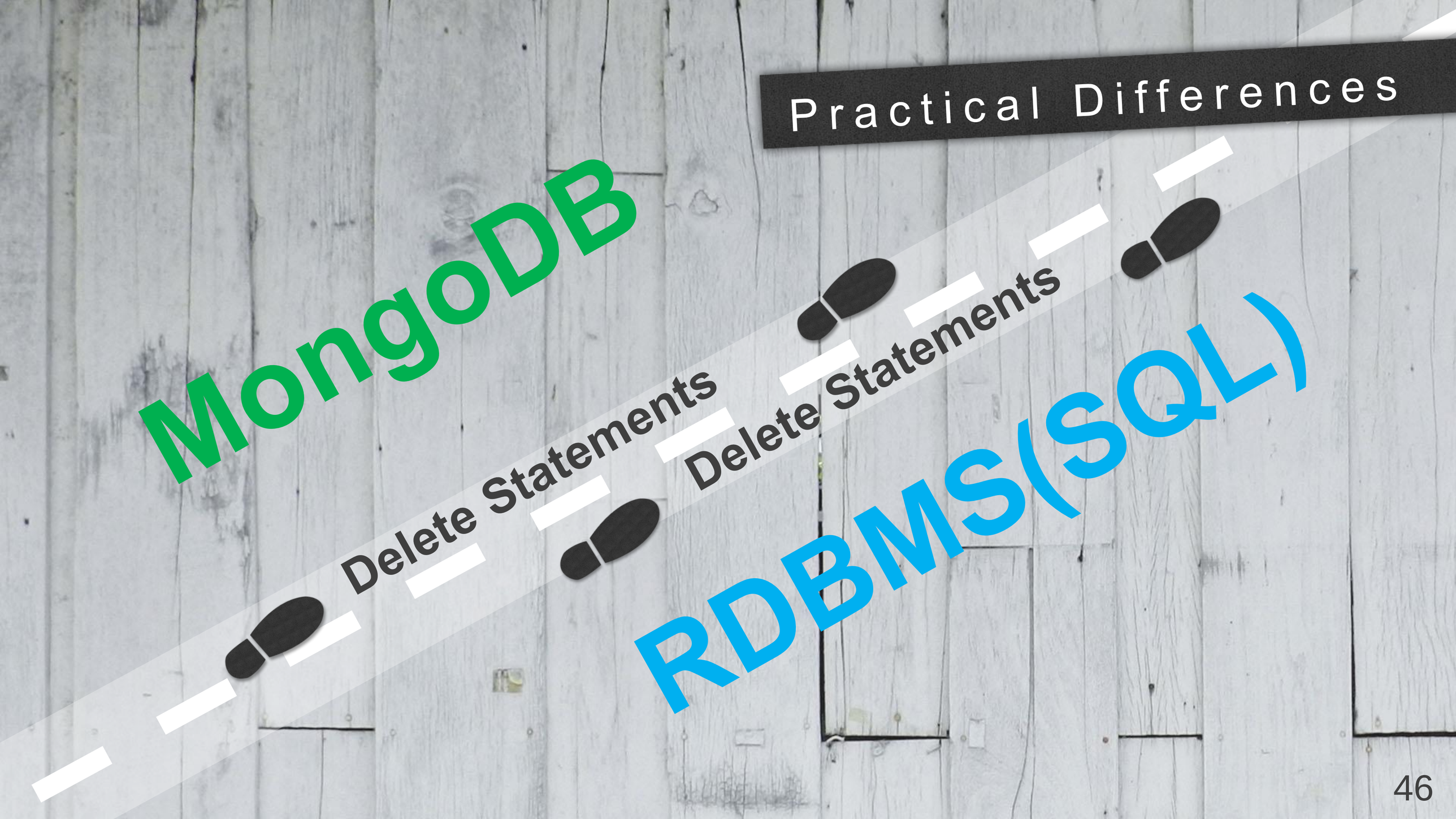
```
UPDATE users
SET age = age + 3
WHERE status = "A"
```

Mongo

```
db.users.update(
   { age: { $gt: 25 } },
   { $set: { status: "C" } },
   { multi: true }
)
```

```
db.users.update(
   { status: "A" } ,
   { $inc: { age: 3 } },
   { multi: true }
)
```

Practical Differences



SQL(RDBMS)	Mongo
DELETE FROM users WHERE status = "D"	db.users.remove({ status: "D" })
DELETE FROM users	db.users.remove( )

NO SQL FINISHED!

Practice all the scenarios that I have discussed