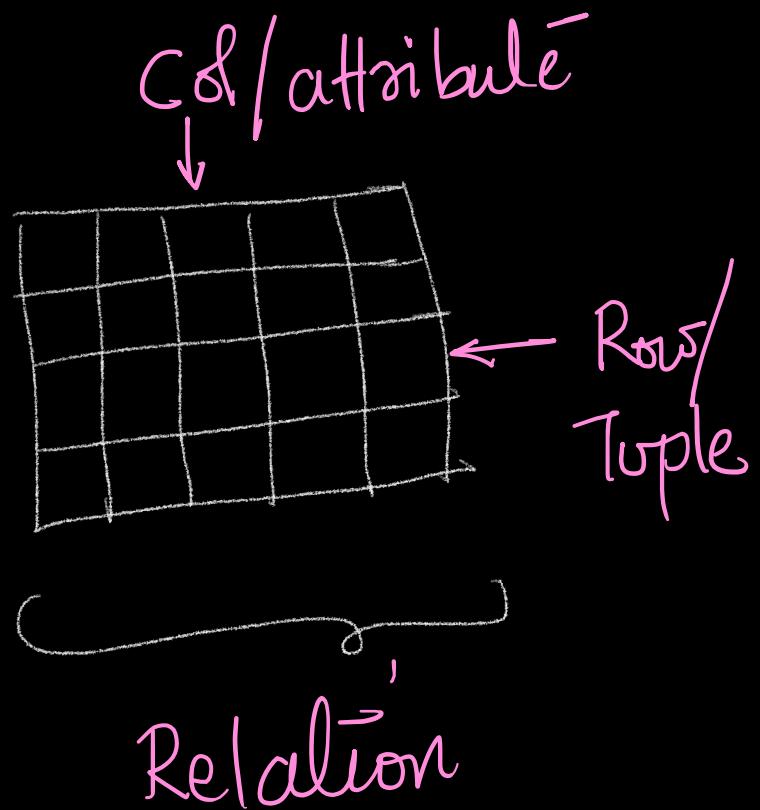


MongoDB

AppliedRoots.com

Relational Databases (1970's - Today)

- MySQL, Oracle, SQL-Server, ...
- Tables / Relations

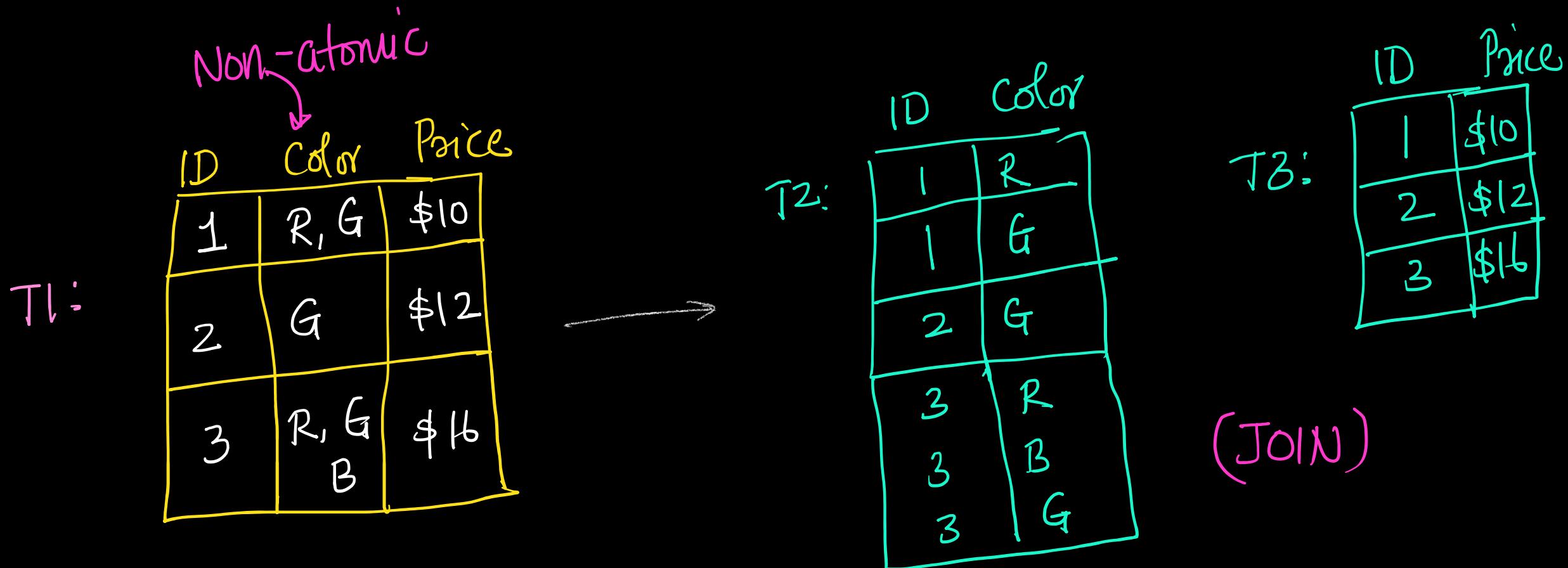


Normalization → avoid redundant data storage



atomic
attributes

Atomic - attributes :



Avoid redundant storage

| CustID | ProdID | Price | City |
|--------|--------|-------|------|
| 1 | 101 | 21 | HYD |
| 2 | 101 | 21 | ND |
| 1 | 102 | 10 | HYD |
| 2 | 103 | 12 | ND |

Normalization →

| CustID | City |
|--------|------|
| 1 | HYD |
| 2 | ND |

| ProdID | Price |
|--------|-------|
| 101 | 21 |
| 102 | 10 |
| 103 | 12 |

| CustID | ProdID |
|--------|--------|
| 1 | 101 |
| 2 | 101 |

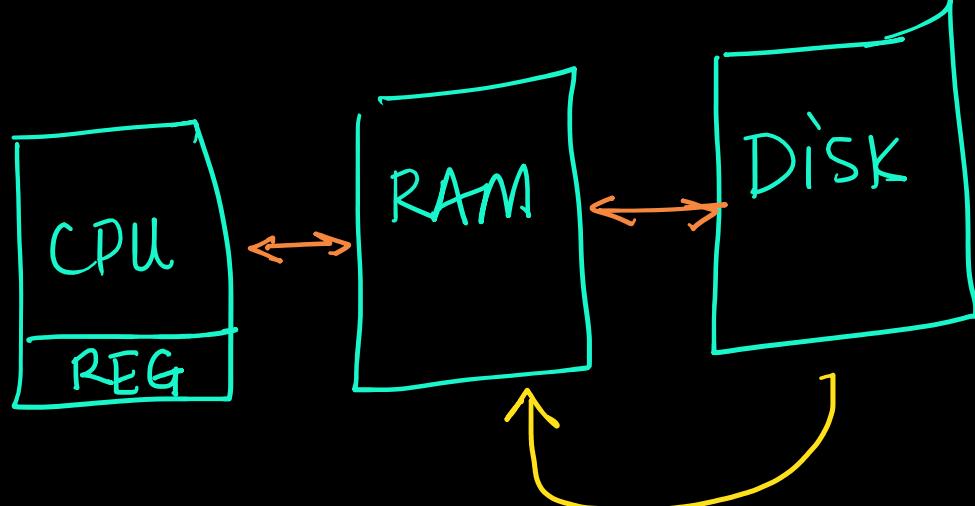
(JOINS)

Relational DB: fixed schema (Table structure)

Not flexible at all

Indexing (makes for faster searching)

→ B^+ trees



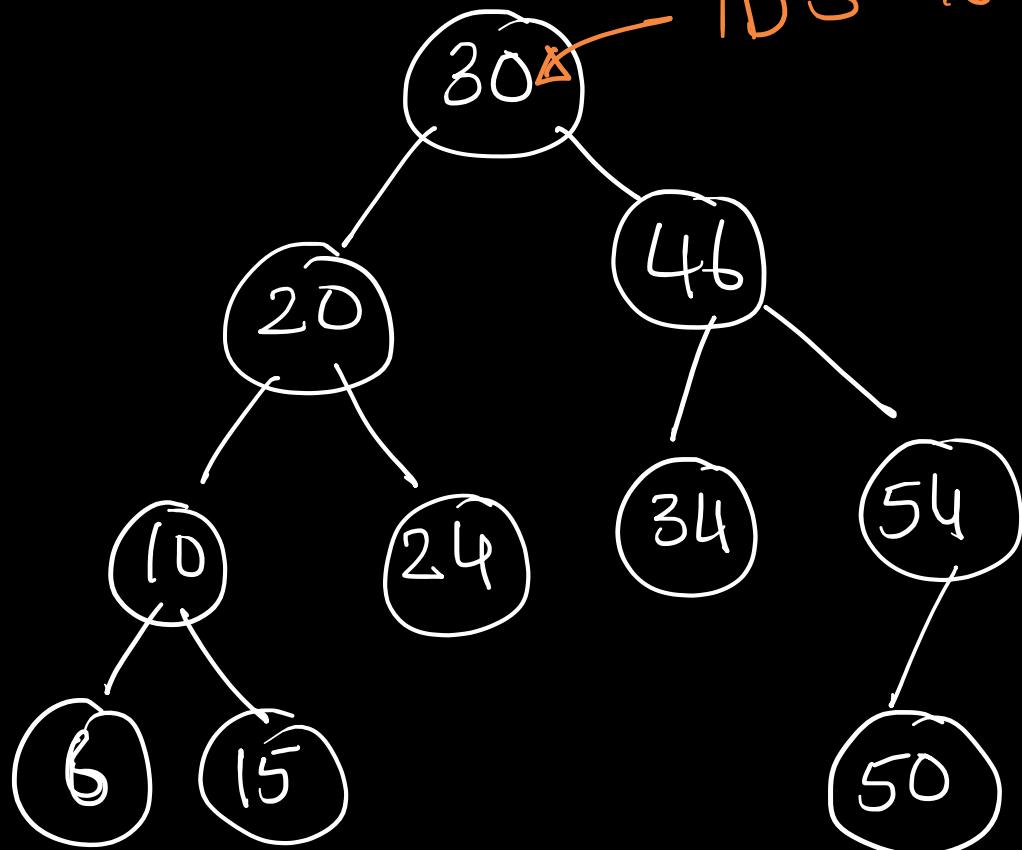
BLOCK ~ 512B (Linux)

} Computer
Architecture
(simplified)

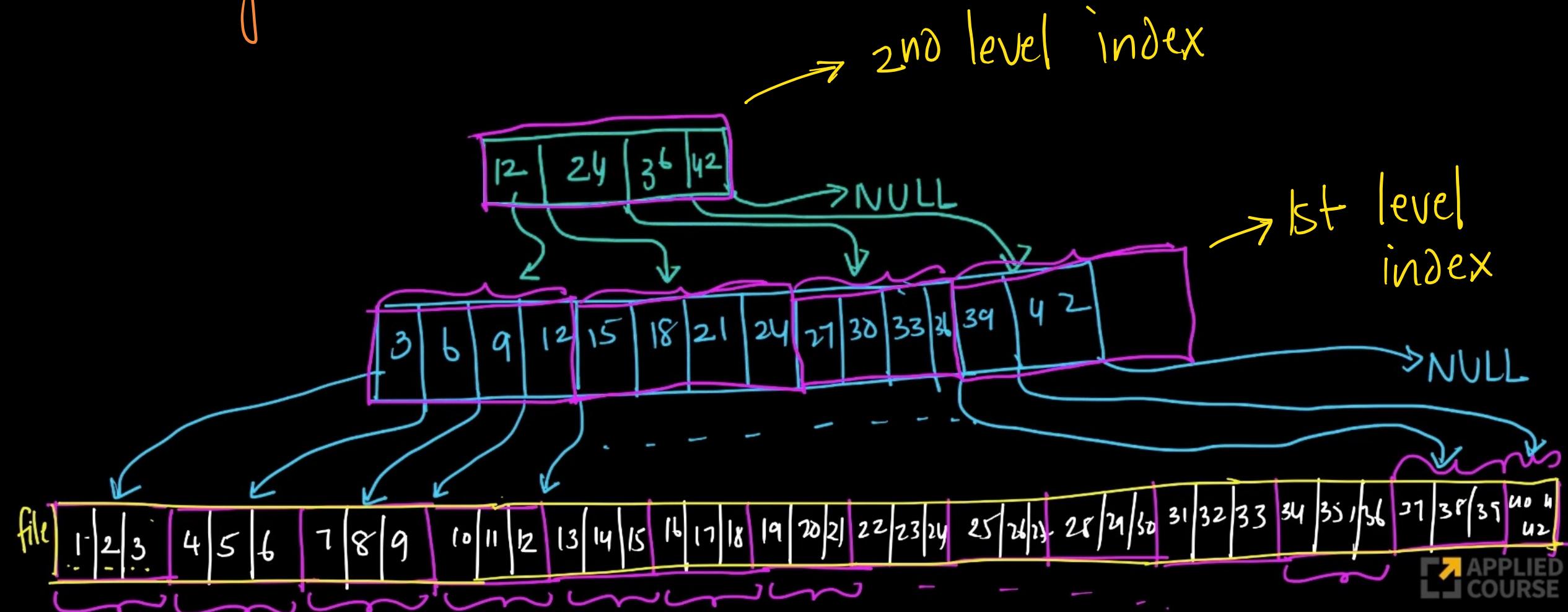
④ minimize block-reads

binary-Search trees:

ID's to search on



M-array Search Tree



B^+ tree \rightarrow block oriented

\downarrow balanced trees

\rightarrow m-way tree / multilevel index



B⁺ tree vs hash index



multi level
index

Key = ID
Value = block number

Equality & Range Queries



$ID \geq 10$ and $ID \leq 50$

$ID = 26$

B⁺ tree indexing

Hash indexing



Relational

NoSQL (e.g.: MongoDB)

1970 - 2000 :

- disk space was precious \Rightarrow avoid redundancy
- latency was not critical \Rightarrow it's ok to join tables
- scale of data was small

2000 - 2010:

- Explosion of internet & devices \Rightarrow more data
- Speed / Low-latency is key
- Compute; RAM, disk space became cheaper
- Hadoop (2006); MongoDB (2009); Redis (2009);

NO-SQL databases

⇒ Non-relational → Document store (MongoDB)

→ Key-value DB (Redis)

→ Graph DB

→ Columnar RDBMS (Vertica)
2005

→ HIVE (Hadoop based)
2010

columnar storage → PostgreSQL (2014)

in-memory
(RAM) → MySQL - MEMORY engine

* MySQL → @ Petabyte scale (FB)
(modified)

↓
HIVE & others

Late 1990's:

- OOP (objects, classes) has become popular

object oriented storage

Mongo DB:

- Most popular document-oriented DB

| Rank | | | | DBMS | Database Model | Score | | |
|------|-------------|-------------|---|------|---|-------------|-------------|-------------|
| | Oct 2021 | Sep 2021 | Oct 2020 | | | Oct 2021 | Sep 2021 | Oct 2020 |
| 1. | 1. | 1. | MongoDB  | | Document, Multi-model  | 493.55 | -2.95 | +45.53 |
| 2. | 2. | 2. | Amazon DynamoDB  | | Multi-model  | 76.55 | -0.38 | +8.14 |
| 3. | 3. | 3. | Microsoft Azure Cosmos DB  | | Multi-model  | 40.29 | +1.76 | +8.28 |
| 4. | 4. | 4. | Couchbase  | | Document, Multi-model  | 27.91 | +0.23 | -2.41 |
| 5. | 5. | ↑ 6. | Firebase Realtime Database | | Document | 19.02 | +1.00 | +2.76 |
| 6. | 6. | ↓ 5. | CouchDB | | Document, Multi-model  | 15.79 | +0.31 | -1.62 |
| 7. | 7. | 7. | MarkLogic  | | Multi-model  | 9.43 | -0.16 | -2.30 |
| 8. | 8. | 8. | Realm  | | Document | 9.29 | -0.22 | +0.55 |
| 9. | 9. | 9. | Google Cloud Firestore | | Document | 8.37 | +0.27 | -0.24 |
| 10. | ↑ 11. | ↑ 20. | Virtuoso  | | Multi-model  | 4.69 | +0.27 | +2.12 |
| 11. | ↓ 10. | 11. | ArangoDB  | | Multi-model  | 4.45 | -0.34 | -1.10 |
| 12. | 12. | ↓ 10. | Google Cloud Datastore | | Document | 4.43 | +0.05 | -1.50 |
| 13. | 13. | ↓ 12. | OrientDB | | Multi-model  | 4.05 | -0.20 | -1.43 |
| 14. | 14. | ↑ 15. | Oracle NoSQL | | Multi-model  | 4.03 | -0.20 | -0.21 |
| 15. | 15. | ↑ 16. | IBM Cloudant | | Document | 3.71 | +0.22 | -0.34 |
| 16. | 16. | ↓ 13. | RavenDB  | | Document, Multi-model  | 3.21 | -0.25 | -1.13 |
| 17. | 17. | ↓ 14. | RethinkDB | | Document, Multi-model  | 3.09 | -0.06 | -1.19 |
| 18. | 18. | ↓ 17. | PouchDB | | Document | 2.86 | +0.00 | -0.61 |

<https://db-engines.com/en/ranking/document+store>

Data organization:

data record = document (~ row / Tuple)

many docs = collections (~ Table)

multiple collections = Data base (~ DB)

BSON : Binary JSON (JavaScript Object Notation)

JSON:

```
{  
    name: "sue",  
    age: 26,  
    status: "A",  
    groups: [ "news", "sports" ]  
}
```

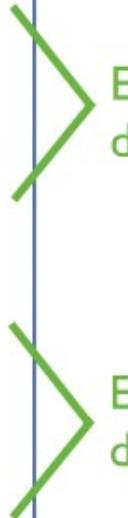
The diagram illustrates a JSON object with four fields: 'name', 'age', 'status', and 'groups'. Each field is preceded by a green arrow pointing to the text 'field: value'.

<https://docs.mongodb.com/manual/core/document/>

More about BSON: <https://www.mongodb.com/basics/bson> (**) examples

Embedded Sub-doc

```
{  
  _id: <ObjectId1>,  
  username: "123xyz",  
  contact: {  
    phone: "123-456-7890",  
    email: "xyz@example.com"  
  },  
  access: {  
    level: 5,  
    group: "dev"  
  }  
}
```

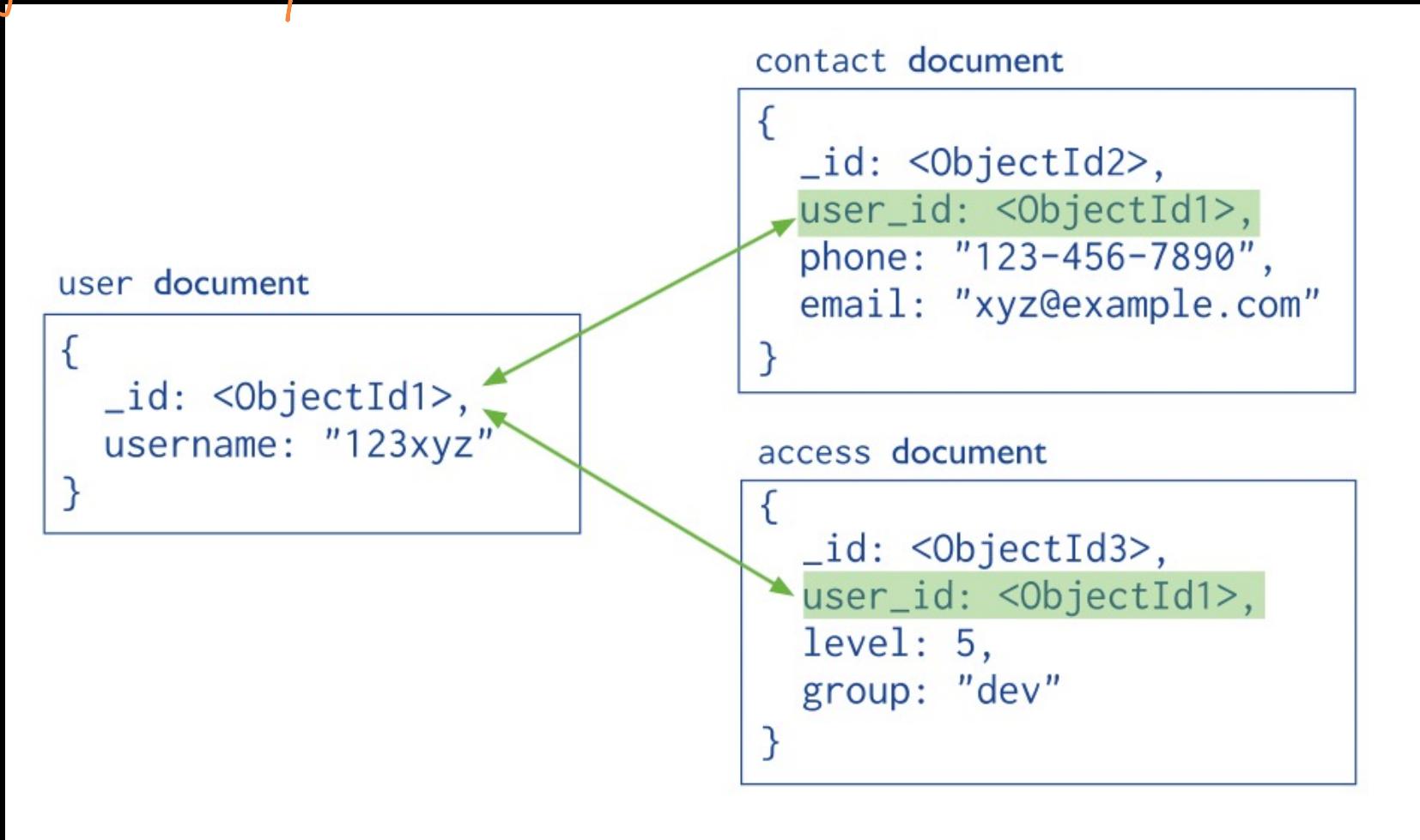


Embedded sub-document

Embedded sub-document

<https://docs.mongodb.com/manual/core/data-modeling-introduction/>

References/Links:-



JOINS

- possible but not recommended
- Data Modeling based on application needs:
 - e.g: prefer sub-doc embedding if we query for user's phone number using username a lot =

flexible schema: all docs dont need to have same fields.

```
{  
  name: "sue",  
  age: 26,  
  status: "A",  
  groups: [ "news", "sports" ]  
}  
  }  
  ↑ field:value  
  ↑ field:value  
  ↑ field:value  
  ↑ field:value
```

gender: "Male"

More advanced concepts (later)

- indexing
- scaling (horizontal & vertical)
- shading
- Replication
- :

Installation

- MongoDB Community
<https://docs.mongodb.com/manual/administration/install-community/>
- MongoDB Enterprise
- MongoDB Atlas (on cloud)

<https://www.mongodb.com/cloud/atlas>

Accessing:

- Mongo shell (Mongosh)
- Python / any programming language
- GUI: Compass
 - (connect → URI:
mongodb://localhost:27017)

CRUD : very simple

Create

Read

Update

Delete

<https://www.mongodb.com/basics/create-database>

(Shell)

① show dbs

```
[test]> show dbs
admin      41 kB
config     111 kB
local      41 kB
test       8.19 kB
```

② use mydb

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

③ db.user.insert({name: "Ada Lovelace", age: 205})

```
mydb> db.user.insert({name: "Ada Lovelace", age: 205})
DeprecationWarning: Collection.insert() is deprecated. Use insertOne, insertMany, or bulkWrite.
{
  acknowledged: true,
  insertedIds: { '0': ObjectId("6162deb24674410b36f580b2") }
}
```

```
mydb> db.user.insertOne({name: "Alan Turing", age: 100})
{
  acknowledged: true,
  insertedId: ObjectId("6162e0be4674410b36f580b3")
}
```

```
_id: ObjectId("6162deb24674410b36f580b2")
name: "Ada Lovelace"
age: 205
```

default primary key ; immutable ; index ;

default
BSON-`id` ;
useful for references

Python 3:

\$ pip install pymongo

```
import pymongo
```

Code :

```
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
db = myclient["mydb"]
collect = db["user"]

d = {"name": "John Von Neuman", "age": 90} #dict

x = collect.insert_one(d)

print(x)
```

```
[varma@Srikanths-MacBook-Air ~ % python3 mongo1.py
<pymongo.results.InsertOneResult object at 0x101526a00>
```

Insert Documentation :

<https://docs.mongodb.com/manual/tutorial/insert-documents/>

SQL - MongoDB mapping :

<https://docs.mongodb.com/manual/reference/sql-comparison/>

SQL - Querying [partially Supported]

↳ MongoDB Atlas:

<https://docs.mongodb.com/datalake/admin/query-with-sql/>

(beta)

→ Studio 3T

<https://studio3t.com/knowledge-base/articles/sql-query/>

(some SQL operations)

Next Session:

→ CRUD

- indexing
- scaling (horizontal & vertical)
- Sharding
- Replication

Create Empty collection :

```
[test> use mydb
switched to db mydb
[mydb> db.createCollection("coll")
{ ok: 1 }
[mydb> show collections
coll
user
mydb>
```

Insert Command:

CRUD

<https://www.mongodb.com/developer/quickstart/cheat-sheet/>

Create

```
1 db coll insertOne({name: "Max"})
2 db coll insert([{name: "Max"}, {name:"Alex"}]) // ordered bulk insert
3 db coll insert([{name: "Max"}, {name:"Alex"}], {ordered: false}) // unordered bulk inse
4 db coll insert({date: ISODate()})
5 db coll insert({name: "Max"}, {"writeConcern": {"w": "majority", "wtimeout": 5000}})
```

copy code

<https://docs.mongodb.com/manual/reference/write-concern/>

CRUD

Read

 copy code

```
1 db.coll.findOne() // returns a single document
2 db.coll.find()    // returns a cursor – show 20 results – "it" to display more
3 db.coll.find().pretty()
4 db.coll.find({name: "Max", age: 32}) // implicit logical "AND".
5 db.coll.find({date: ISODate("2020-09-25T13:57:17.180Z")})
6 db.coll.find({name: "Max", age: 32}).explain("executionStats") // or "queryPlanner" or "allPlans"
7 db.coll.distinct("name")
8
```

<https://docs.mongodb.com/manual/reference/explain-results/>

```
9   // Count
10  db.coll.count({age: 32})           // estimation based on collection metadata
11  db.coll.estimatedDocumentCount() // estimation based on collection metadata
12  db.coll.countDocuments({age: 32}) // alias for an aggregation pipeline – accurate coun
t
```

```
14 // Comparison
15 db.coll.find({"year": {$gt: 1970}})
16 db.coll.find({"year": {$gte: 1970}})
17 db.coll.find({"year": {$lt: 1970}})
18 db.coll.find({"year": {$lte: 1970}})
19 db.coll.find({"year": {$ne: 1970}})
20 db.coll.find({"year": {$in: [1958, 1959]}}})
21 db.coll.find({"year": {$nin: [1958, 1959]}})
```

```
23 // Logical  
24 db.coll.find({name:{$not: {$eq: "Max"}}})  
25 db.coll.find({$or: [{"year" : 1958}, {"year" : 1959}]})  
26 db.coll.find({$nor: [{price: 1.99}, {sale: true}]})  
27 db.coll.find({  
28     $and: [  
29         {$or: [{qty: {$lt :10}}, {qty :{$gt: 50}}]},  
30         {$or: [{sale: true}, {price: {$lt: 5 }}]}  
31     ]  
32 })
```

```
34 // Element  
35 db.coll.find({name: {$exists: true}})  
36 db.coll.find({"zipCode": {$type: 2 }})  
37 db.coll.find({"zipCode": {$type: "string"}})
```

```
39 // Aggregation Pipeline
40 db.coll.aggregate([
41   {$match: {status: "A"}},
42   {$group: {_id: "$cust_id", total: {$sum: "$amount}}},
43   {$sort: {total: -1}}
44 ])
```

<https://docs.mongodb.com/manual/aggregation/>

<https://docs.mongodb.com/manual/reference/operator/aggregation/sort/>

MongoDB uses Perl compatible regular expressions

```
49 // Regex
50 db.coll.find({name: /^Max/})    // regex: starts by letter "M"
51 db.coll.find({name: /^Max$/i}) // regex case insensitive
```

<https://docs.mongodb.com/manual/reference/operator/query/regex/#examples>

Indexed Text Search

<https://docs.mongodb.com/manual/reference/operator/query/text/#examples>

```
58 // Projections
59 db.coll.find({"x": 1}, {"actors": 1})           // actors + _id
60 db.coll.find({"x": 1}, {"actors": 1, "_id": 0}) // actors
61 db.coll.find({"x": 1}, {"actors": 0, "summary": 0}) // all but "actors" and "summary"
```

<https://docs.mongodb.com/manual/reference/method/db.collection.find/#projections>

```
63 // Sort, skip, limit
64 db.coll.find({}).sort({"year": 1, "rating": -1}).skip(10).limit(3)
65
66 // Read Concern
67 db.coll.find().readConcern("majority")
```

Array Querying :

<https://docs.mongodb.com/manual/tutorial/query-arrays/>

Update :

<https://docs.mongodb.com/manual/tutorial/update-documents/>

Delete

 copy code

```
1 db.coll.remove({name: "Max"})
2 db.coll.remove({name: "Max"}, {justOne: true})
3 db.coll.remove({}) // WARNING! Deletes all the docs but not the collection itself and its index definitions
4 db.coll.remove({name: "Max"}, {"writeConcern": {"w": "majority", "wtimeout": 5000}})
5 db.coll.findOneAndDelete({"name": "Max"})
```

Drop

 copy code

```
1 db.coll.drop()    // removes the collection and its index definitions  
2 db.dropDatabase() // double check that you are *NOT* on the PROD cluster... :-)
```

JOIN:

<https://docs.mongodb.com/manual/reference/operator/aggregation/lookup/#examples>

Indexing

Create Indexes

 copy code

```
1 // Index Types
2 db.coll.createIndex({"name": 1})           // single field index
3 db.coll.createIndex({"name": 1, "date": 1})    // compound index
4 db.coll.createIndex({foo: "text", bar: "text"}) // text index
5 db.coll.createIndex({"$**": "text"})          // wildcard text index
6 db.coll.createIndex({"userMetadata.$**": 1})    // wildcard index
7 db.coll.createIndex({"loc": "2d"})            // 2d index
8 db.coll.createIndex({"loc": "2dsphere"})       // 2dsphere index
9 db.coll.createIndex({"_id": "hashed"})         // hashed index
10
```

equality and NOT for ranges

Horizontal & Vertical Scaling

<https://www.mongodb.com/databases/scaling>

→ Sharding / Partitioning

→ Replica Sets

→ Replication + sharding

How does sharding work internally:

<https://docs.mongodb.com/manual/sharding/#sharding-strategy>

Transactions:

- Single vs multi-doc atomicity
- Replication: write & read concerns

<https://docs.mongodb.com/manual/core/transactions/#transactions-and-atomicity>

Q & A

