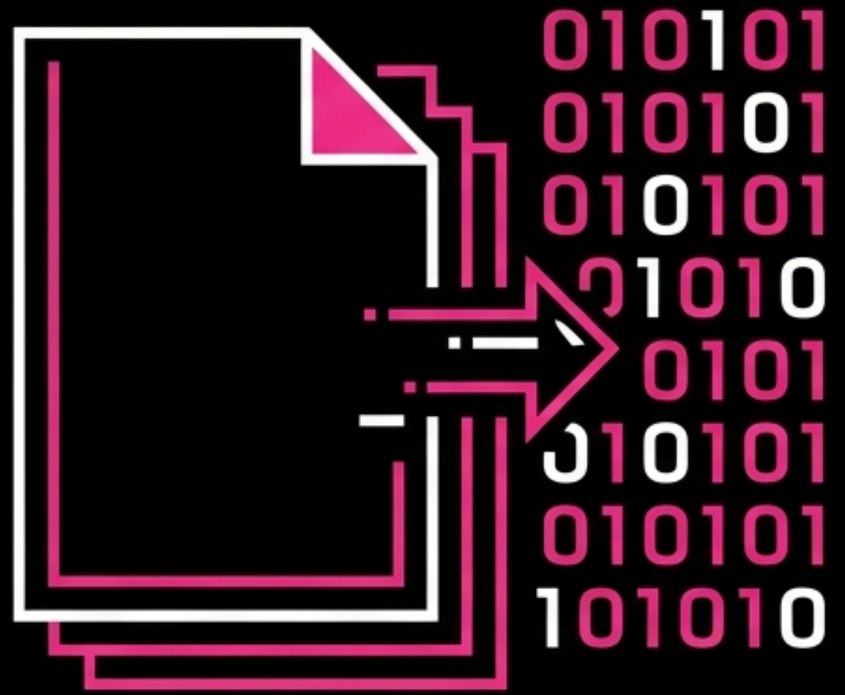
MongoDB Fundamentals

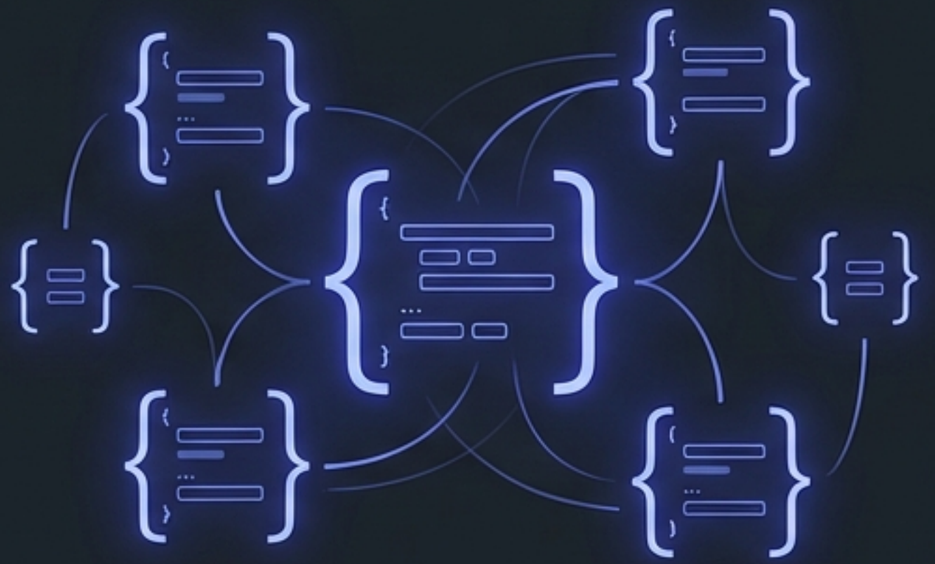
A First Look at NoSQL



Node2Know • Professor Solo

The Mental Shift: The Grid vs. The Vault

	Column A	Column B	Column C	...	...
1					
2					
3					
...					
...					
...					
...					
...					



SQL (The Grid)

- Rigid Blueprints
- Pre-defined Tables
- Migrations required for change

NoSQL (The Vault)

- Fluid Documents
- Code-First Structure
- Built for speed & scale

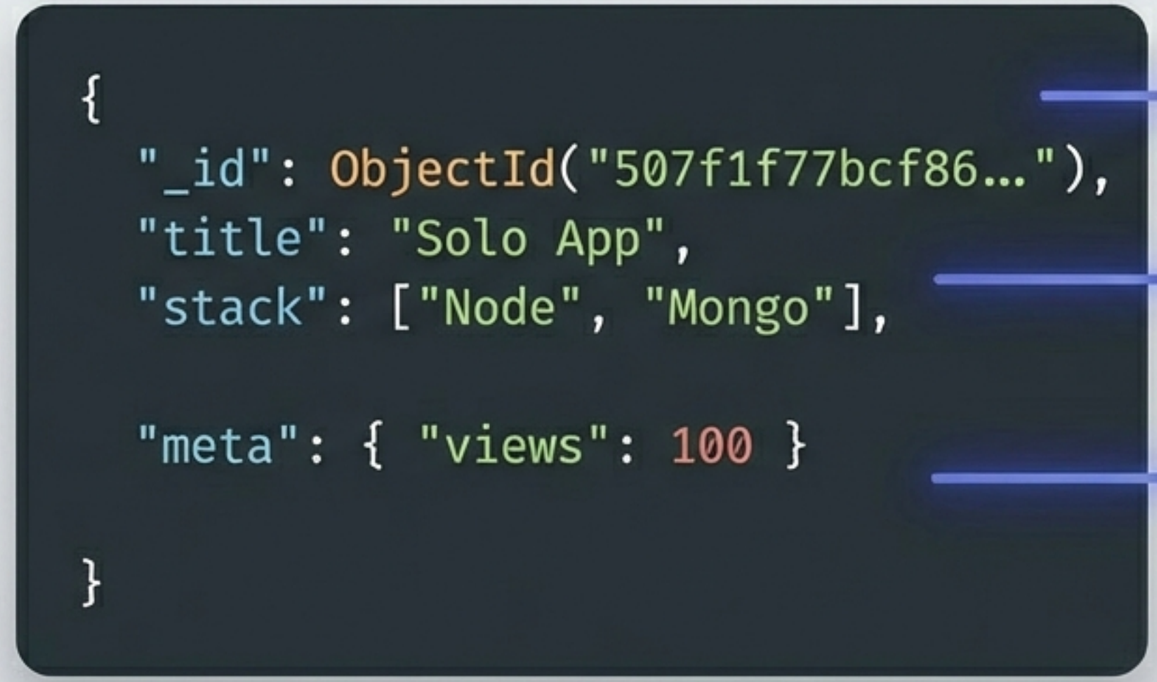
The Vocabulary Map

Translating the blueprints to the new protocol.

Database	Database (The Facility)
Table	Collection (The Room)
Row	Document (The Safety Deposit Box)
Column	Field (The Item Inside)

Professor Solo Note: We don’t join tables here. We embed data.

Anatomy of a Document



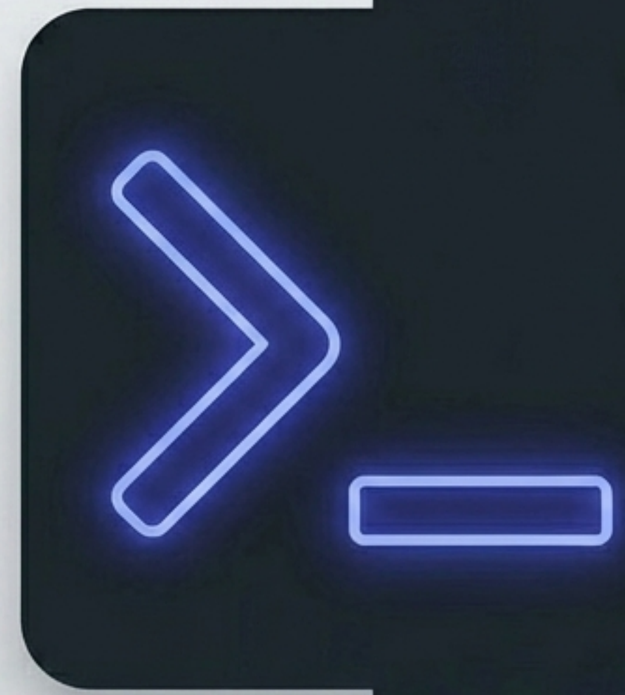
The Immutable Tracking Number (Auto-generated)

Arrays: Storing lists directly

Embedded Objects: 3D Depth

Stored as BSON (Binary JSON) for speed

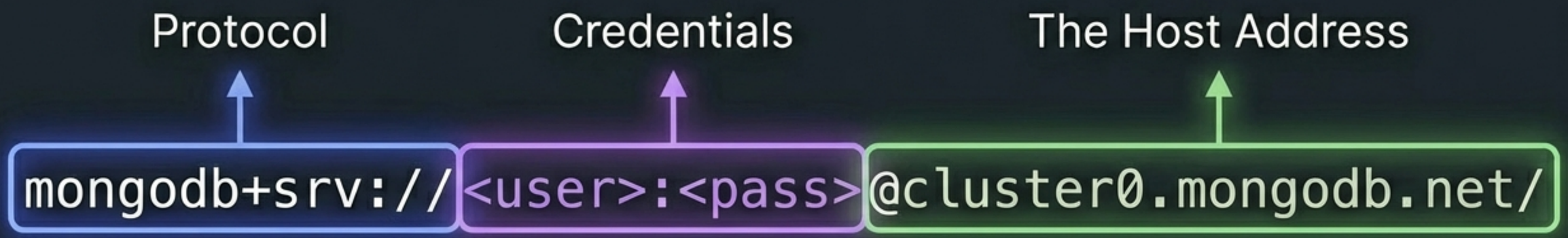
The Tool: mongosh



- The “bare metal” interface for the cluster.
- A JavaScript runtime environment.
- Used for admin tasks and direct data manipulation.

```
$ mongosh --version
```

Protocol: Connection



We connect to a Cluster (a team of servers), not a single machine.

⚠️ Security Warning: Requires Whitelisted IP & Database User.

```
$ mongosh 'mongodb+srv://student:password@cluster0.mongodb.net/'
```

Context & Lazy Creation

Navigation

```
> show dbs
> use node2know


switched to db node2know
```

Concept: Lazy Creation

The database does not physically exist until the first document is inserted.

Collections: The Container

The Cowboy Method (Implicit)

```
db.projects.insertOne({ ... })
```

Just insert data. Mongo builds the room automatically.

The Architect Method (Explicit)

```
db.createCollection('projects')
```

Define the room intentionally.

```
> show collections
```

Operation: insertOne()

- The precision tool for adding a single asset.
- Returns an acknowledged receipt.

```
db.projects.insertOne({
  title: 'Solo App',
  tagline: 'Terminal aesthetics',
  stack: ['Node', 'Mongo'],
  stars: 120,
  createdAt: new Date()
})
```

```
{ acknowledged: true, insertedId: ObjectId('...') }
```

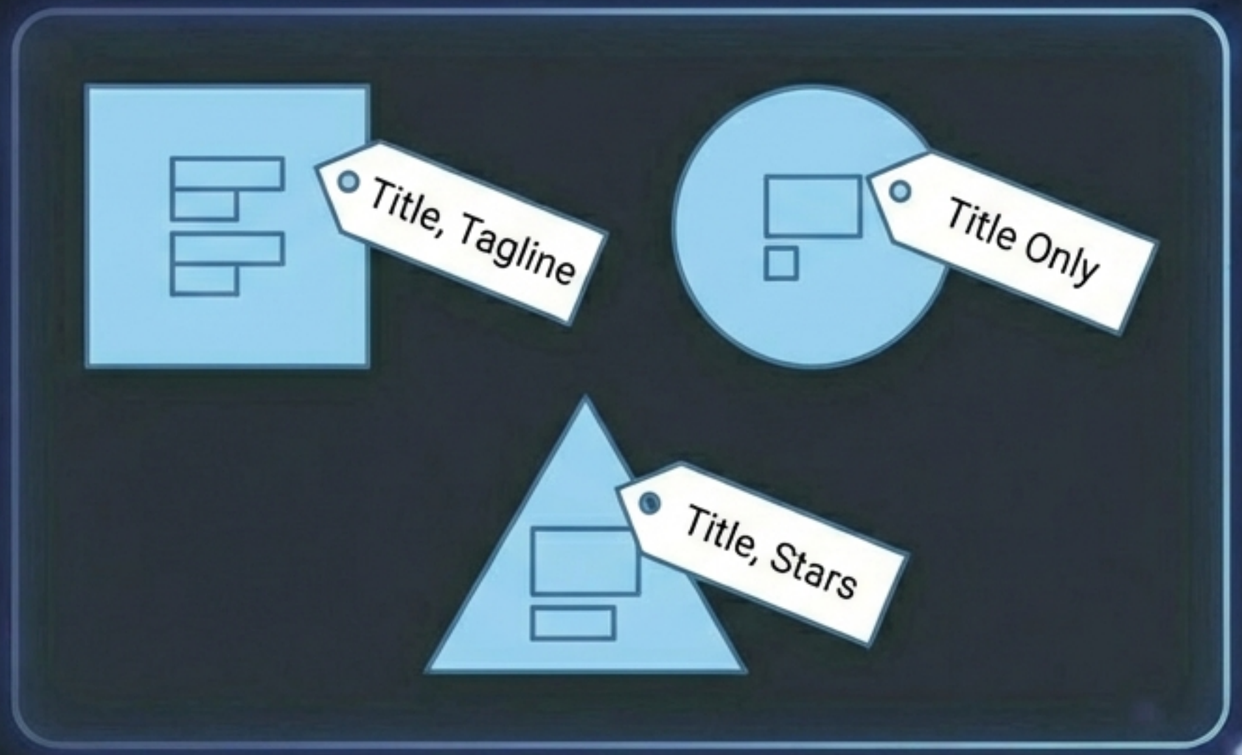
Operation: insertMany()

- Bulk ingestion for seeding data.
- Requires an Array [ ] of objects.

```
db.projects.insertMany([
  { title:'Project A',
  stars: 10,
  stack: ['React'] },
  { title: 'Project B',
  stars: 50,
  stack: ['Vue'] }
])
```

The Flexibility Power

- We just inserted documents with different shapes.
- Document 1: Had a tagline.
- Document 2 & 3: Did not.
- Result: Success. No downtime migrations required.
- Risk: Messy Room Syndrome (We are responsible for structure).



Retrieval: find()

- The "Select All" Command.
- Empty brackets {} imply "Show me everything".
- Returns a Cursor (pointer) to results.

```
db.projects.find()
  { title:'Project A',
  stars: 10,
  stack: ['React'] },
  { title: 'Project B',
  stars: 50,
  stack: ['Vue'] }
  { title: 'Project A',
  { title: 'Project C',
  { title: 'Project C',
    stack: ['Vue'] }
```

Precision: findOne()

- Returns the FIRST matching document.
- Returns the actual object, not a cursor.
- Ideal for ‘Get by ID’.

```
db.projects.findOne({
  title: 'Solo App'
})
```

Query Filters

- Pass a ‘Query Object’ to narrow the search.

Exact Match

```
db.projects.find({ stars: 120 })
```

Implicit AND

```
db.projects.find({ stars: 50, stack: 'Vue' })
```

Array Magic (Finds if value exists in array)

```
db.projects.find({ stack: 'React' })
```

Mission Debrief

Context

```
use <db_name>
```

Deposit

```
insertOne( { } )
insertMany( [ ] )
```

Verify

```
show dbs
show collections
```

Retrieve

```
find( { filter } )
findOne( { filter } )
```

We have breached the vault. Next time, we bring order to the chaos using Mongoose.

p.s., keep learning!