IMEN891K / IDSC501 / AXGS-501 · DATA ENGINEERING · FALL 2026

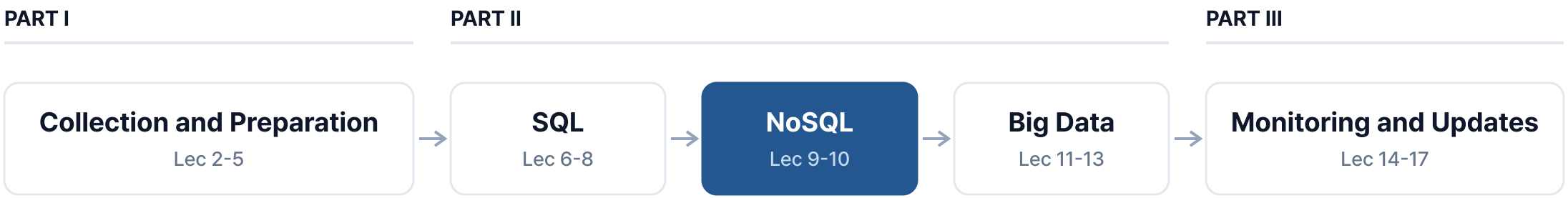
10.

NoSQL II: Document Stores

Sangwoo Mo · POSTECH

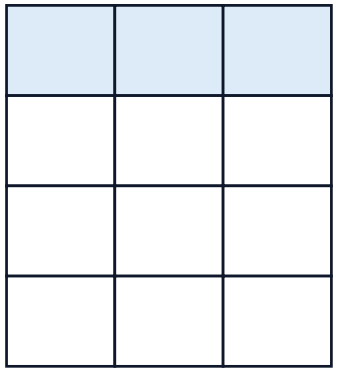
Industrial & Management Engineering

Where We Are


PART I
PART II
PART III
Collection and Preparation
Lec 2-5
SQL
Lec 6-8
NoSQL
Lec 9-10
Big Data
Lec 11-13
Monitoring and Updates
Lec 14-17

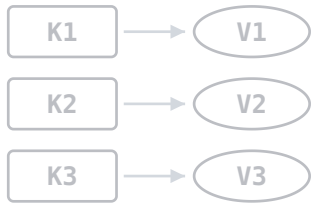
Recap: Types of NoSQL

SQL

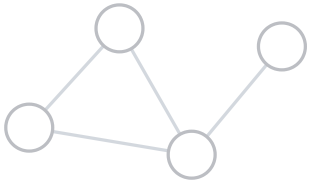


Relational

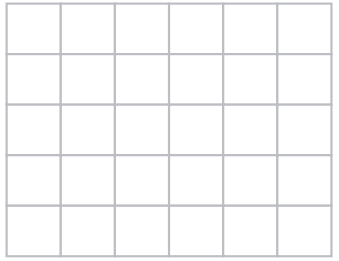
NoSQL



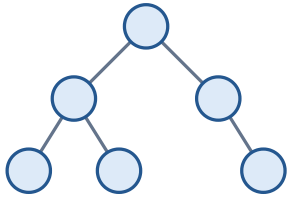
Key-value



Graph



Wide-column



Document

Last week: four data models. Today: the **document** model, in MongoDB

One Item, Several Locations

One item document holds an array of location-level stock entries

```
paper
  qty: 100
  status: "D"
  instock: [ {loc: "A", qty: 60},
             {loc: "B", qty: 15} ]
```

The item and its stock entries live at different levels

Outline: NoSQL II

01	**Documents and Queries**	read fields, nesting and paths
02	**Finding Documents**	choose documents and output fields
03	**Summarizing Documents**	transform documents through ordered stages
04	**Updating Documents**	select write targets and verify changes

01 · Documents and Queries

01 **Documents and Queries** — read fields, nesting and paths

02 **Finding Documents** — choose documents and output fields

03 **Summarizing Documents** — transform documents through ordered stages

04 **Updating Documents** — select write targets and verify changes

Collection and Document

Relational terms

Table	Row	Column
Product	one product	price

Document terms

Collection	Document	Field
inventory	one item	qty

A collection holds documents; each document has named fields

Fields, Objects and Arrays

One inventory document

```
{
  item: "paper",
  qty: 100,
  size: {h: 8.5, w: 11, uom: "in"},
  tags: ["red", "blank", "plain"],
  instock: [
    {loc: "A", qty: 60},
    {loc: "B", qty: 15}
  ]
}
```

Levels inside it

Field	Value
qty	number
size	nested object
tags	array of strings
instock	array of objects

A field can hold one value, a nested object or an array

Three Query Tasks

Task	Question	MQL entry point
Retrieve	Which items have status D?	find
Aggregate	How much stock is at each location?	aggregate
Update	Which widths must change?	updateMany

The operation follows the question asked of the documents

Dot Paths

The array to read

```
instock: [
  {loc: "A", qty: 60},  // index 0
  {loc: "B", qty: 15}   // index 1
]
```

A path reaches a value

Path	Meaning
instock.1	second stock entry
instock.1.qty	15
instock.qty	qty fields across the array

A numeric path step chooses one array position

Queries as Documents

SQL condition

```
WHERE qty > 30
```

MQL predicate

```
{ qty: { $gt: 30 } }
```

The same comparison can be written as a predicate document

02 · Finding Documents

01 Documents and Queries — read fields, nesting and paths

02 **Finding Documents** — choose documents and output fields

03 Summarizing Documents — transform documents through ordered stages

04 Updating Documents — select write targets and verify changes

Retrieving Documents

find(predicate, projection) chooses documents, then fields

```
db.inventory.find(
  { status: "D" },
  { _id: 0, item: 1, qty: 1 }
)
```

FIND RESULT

```
{ "item": "paper", "qty": 100 }
{ "item": "planner", "qty": 75 }
```

The condition chooses two items; projection shows name and quantity

Matching Nested Fields

```
db.inventory.find(
  { "size.uom": "cm", "size.h": { $gt: 14
} },
  { _id: 0, item: 1, size: 1 }
)
```

FIND RESULT

```
{ "item": "planner", "size": { "h": 22.85,
 "w": 30, "uom": "cm" } }
```

A dotted path tests a field inside the nested object

Exact Arrays or Contained Values

Exact array, including order

```
db.inventory.find(
  { tags: ["red", "blank"] },
  { _id: 0, item: 1, tags: 1 }
)
```

FIND RESULT

```
{ "item": "notebook", "tags": [ "red",
 "blank" ] }
```

Contains red and plain

```
db.inventory.find(
  { tags: { $all: ["red", "plain"] } },
  { _id: 0, item: 1, tags: 1 }
)
```

FIND RESULT

```
{ "item": "paper", "tags": [ "red", "blank",
 "plain" ] }
```

An array equality checks the whole sequence; $all checks membership

Preserving an Array in Projection

```
db.inventory.find(
  { item: "paper" },
  { _id: 0, item: 1, "instock.loc": 1 }
)
```

FIND RESULT

```
{ "item": "paper", "instock": [ { "loc":
 "A" }, { "loc": "B" } ] }
```

Projection removes fields while preserving the array structure

Sort, Then Limit

```
db.inventory.find(
  {}, { _id: 0, item: 1, dim: 1 }
).sort({ "dim.0": -1, item: 1 }).limit(2)
```

FIND RESULT

```
{ "item": "planner", "dim": [ 22.85, 30 ]
  }
{ "item": "journal", "dim": [ 14, 21 ] }
```

Sort determines which two documents the limit keeps

Recap: Inventory for Retrieval

item	qty	status	tags
journal	25	A	blank, red
notebook	50	A	red, blank
paper	100	D	red, blank, plain
planner	75	D	blank, red
postcard	45	A	blue

The same five items supply every retrieval exercise

Caveat: A Location and a Quantity

paper stock entries

loc	qty	Condition met
A	60	qty ≥ 30
B	15	loc = B

One entry must satisfy both

```
{ instock: {
    $elemMatch: {loc: "B", qty: {$gte:
30}}
} }
```

paper is excluded

A quantity at A cannot satisfy a stock requirement at B

Demo: Retrieval

Predict the matching entities, then check the condition scope

1. Find status A items with quantity at least 50; show item and qty
2. Require location B and qty at least 30 in the same stock entry. Does paper qualify?

```
// write a query here, then Run   e.g.
db.inventory.find({ status: "D" })
```

RESULT

```
Write a query on the left, then Run.
```

Answer: Retrieval

```
db.inventory.find(
  {status: "A", qty: {$gte: 50}},
  {_id: 0, item: 1, qty: 1}
)
```

FIND RESULT

```
{ "item": "notebook", "qty": 50 }
```

Only notebook has status A and at least 50 units

Answer: One Stock Entry Must Qualify

Item	Location B stock	At least 30 at B?
journal	No B entry	No
notebook	No B entry	No
paper	15	No
planner	5	No
postcard	15	No

No item has one B stock entry with at least 30 units

Null and Missing Fields

Two different states

```
{ item: "x", color: null }
{ item: "y" }
```

Two different questions

Predicate	Matches
color: null	x and y
color: {$exists: false}	y only

A null-valued field and an absent field are different states

Caveat: A Mistyped Field

One document stores qty as the string "50"; the range filter skips it without an error

```
db.inventory.find(
  { qty: { $gte: 30 } },
  { _id: 0, item: 1, qty: 1 } )
```

QTY AT LEAST 30: NOTEBOOK (50) IS MISSING

```
{ "item": "paper", "qty": 100 }
{ "item": "planner", "qty": 75 }
{ "item": "postcard", "qty": 45 }
```

```
db.inventory.find(
  { qty: { $type: "string" } },
  { _id: 0, item: 1, qty: 1 } )
```

WHY: ITS QTY IS A STRING

```
{ "item": "notebook", "qty": "50" }
```

Remedy: validate qty as numeric before accepting the document

This numeric query predicate **skips string values**; numeric BSON types remain comparable

03 · Summarizing Documents

01 Documents and Queries — read fields, nesting and paths

02 Finding Documents — choose documents and output fields

03 **Summarizing Documents** — transform documents through ordered stages

04 Updating Documents — select write targets and verify changes

A Pipeline Changes Its Input

Input: CA ZIPs

ZIP	city	pop
90001	LOS ANGELES	57110
90002	LOS ANGELES	51223
94110	SAN FRANCISCO	74633
92101	SAN DIEGO	34560

→

Group by city

city	pop
LOS ANGELES	108333
SAN FRANCISCO	74633
SAN DIEGO	34560

→

Keep pop > 100,000

city	pop
LOS ANGELES	108333

Four ZIP records become three city totals; the filter keeps Los Angeles

The Zips Collection

One record represents a ZIP code; Los Angeles has two records

ZIP	city	pop
90001	LOS ANGELES	57110
90002	LOS ANGELES	51223
94110	SAN FRANCISCO	74633
92101	SAN DIEGO	34560

Four ZIP records represent only three cities

Group Totals by State

```
{$group: {_id: "$state", totalPop: {$sum: "$pop"}}}
```

state	totalPop
MA	34114
CA	217526
NY	331508
TX	260264
WI	87956
IL	192310

Twenty ZIP records produce six state totals; totalPop is a new field

Filter the Group Totals

After grouping

state	totalPop
MA	34114
CA	217526
NY	331508
TX	260264
WI	87956

totalPop ≥ 200,000

state	totalPop
NY	331508
TX	260264
CA	217526

```
{$match: {totalPop: {$gte: 200000}}}
```

The condition reads the new state totals and removes three groups

The Same Summary in SQL

```sql
SELECT state, SUM(pop) AS totalPop
FROM zips
GROUP BY state
HAVING SUM(pop) >= 200000
ORDER BY totalPop DESC, state;
```

GROUP BY, HAVING and ORDER BY express the same summary

First Group: City Totals

Before: one row per ZIP

ZIP	city	pop
90001	LOS ANGELES	57110
90002	LOS ANGELES	51223
94110	SAN FRANCISCO	74633
92101	SAN DIEGO	34560

After: one row per city

city	pop
LOS ANGELES	108333
SAN FRANCISCO	74633
SAN DIEGO	34560

```
{$group: {_id: {state: "$state", city: "$city"},
          pop: {$sum: "$pop"}}}
```

The two Los Angeles ZIP populations combine into one city total

The Intermediate City Documents

Los Angeles ZIPs

ZIP	pop
90001	57110
90002	51223

→

Add their populations

```
57,110
+ 51,223
= 108,333
```

→

One city total

city	pop
LOS ANGELES	108333

Los Angeles contributes one city total to the next average

Second Group: Average City Population

Three city totals

city	pop
LOS ANGELES	108333
SAN FRANCISCO	74633
SAN DIEGO	34560

Average the cities

```
(108,333 + 74,633 + 34,560) / 3
```

state	avgCityPop
CA	72508.667

```
{$group: {_id: "$_id.state", avgCityPop: {$avg: "$pop"}}}
```

The denominator is three cities, even though the original input contained four ZIPs

Recap: Zips for Aggregation

ZIP	city	pop
90001	LOS ANGELES	57110
90002	LOS ANGELES	51223
94110	SAN FRANCISCO	74633
92101	SAN DIEGO	34560

Use these same four CA records when calculating the average city population

Demo: Aggregation

Inspect each stage’s output and the population unit before adding another stage

1. Total population per state; keep totals above 100,000 and sort largest first
2. For CA, sum ZIP populations per city, then average those city totals

```
// write a query here, then Run   e.g.
db.inventory.find({ status: "D" })
```

RESULT

```
Write a query on the left, then Run.
```

Answer: State Totals

```
db.zips.aggregate([
  {$group: {_id: "$state", totalPop:
{$sum: "$pop"}}},
  {$match: {totalPop: {$gt: 100000}}},
  {$sort: {totalPop: -1, _id: 1}}
])
```

AGGREGATE RESULT

```
{ "_id": "NY", "totalPop": 331508 }
{ "_id": "TX", "totalPop": 260264 }
{ "_id": "CA", "totalPop": 217526 }
{ "_id": "IL", "totalPop": 192310 }
```

Filter the new state totals, then order the remaining groups

Answer: Average City Population

```
db.zips.aggregate([
  {$match: {state: "CA"}},
  {$group: {_id: {state: "$state", city:
"$city"},
            pop: {$sum: "$pop"}}},
  {$group: {_id: "$_id.state", avgCityPop:
{$avg: "$pop"}}}
])
```

AGGREGATE RESULT


```
{ "_id": "CA", "avgCityPop": 72508.667 }
```


Average three city totals, rather than four ZIP populations

Unwinding One Array

One item document

```
{item: "paper",
 instock: [
  {loc:"A", qty:60},
  {loc:"B", qty:15}
 ]}
```

One row per stock entry

item	loc	stock qty
paper	A	60
paper	B	15

One paper document becomes two records, one for A and one for B

Caveat: An Item Appears Twice

item	loc	stock qty
paper	A	60
paper	B	15

Stock entries

```
2 rows
```

Distinct items

```
1 item: paper
```

Stock quantity

```
60 + 15 = 75
```

Two stock records describe one item; choose the entity before counting or summing

Total Stock per Item

```
db.inventory.aggregate([
  {$unwind: "$instock"},
  {$group: {_id: "$item", totalQty: {$sum:
"$instock.qty"}}}
])
```

AGGREGATE RESULT

```
{ "_id": "journal", "totalQty": 20 }
{ "_id": "notebook", "totalQty": 5 }
{ "_id": "paper", "totalQty": 75 }
{ "_id": "planner", "totalQty": 45 }
{ "_id": "postcard", "totalQty": 50 }
```

Unwind exposes the numbers; group recombines them by item

Recap: Stock by Location

item	A	B	C
journal	5	—	15
notebook	—	—	5
paper	60	15	—
planner	40	5	—
postcard	—	15	35

Location-level quantities come from the instock entries

Demo: Stock by Location

Inspect the changed grain before summing any quantities

1. Unwind instock and inspect the result: what does one document now represent?
2. Group those entries by location, sum their quantities and sort largest first

```
// write a query here, then Run   e.g.
db.inventory.find({ status: "D" })
```

RESULT

```
Write a query on the left, then Run.
```

Answer: Location Totals

```
db.inventory.aggregate([
  {$unwind: "$instock"},
  {$group: {_id: "$instock.loc", qty:
{$sum: "$instock.qty"}}},
  {$sort: {qty: -1, _id: 1}}
])
```

AGGREGATE RESULT

```
{ "_id": "A", "qty": 105 }
{ "_id": "C", "qty": 55 }
{ "_id": "B", "qty": 35 }
```

The group key is now a location, not an item

Answer: One Document per Stock Entry

Stage	Documents	Meaning of one document
Inventory	5	One item with a stock array
Unwind instock	9	One item at one location
Group by location	3	Total stock at one location

Unwinding repeats item fields; sum the quantity at the grain your question requires

Lookup Adds an Array

items

item	supplier
journal	S1
paper	S1
postcard	S9

suppliers

supplier	country
S1	US

After lookup

item	details
journal	["US"]
paper	["US"]
postcard	[]

Matching suppliers enter an array; postcard stays with an empty array

Lookup on a Shared Key

```
db.items.aggregate([
  {$lookup: {from: "suppliers",
             localField: "supplier",
foreignField: "supplier",
             as: "details"}},
  {$project: {_id: 0, item: 1, details:
1}}
])
```

AGGREGATE RESULT

```
{ "item": "journal", "details": [ {
  "supplier": "S1", "country": "US" } ] }
{ "item": "paper", "details": [ {
  "supplier": "S1", "country": "US" } ] }
{ "item": "postcard", "details": [ ] }
```

An unmatched supplier produces an empty array, not a missing item

04 · Updating Documents

01 Documents and Queries — read fields, nesting and paths

02 Finding Documents — choose documents and output fields

03 Summarizing Documents — transform documents through ordered stages

04 **Updating Documents** — select write targets and verify changes

Inserting Documents

```
db.inventory.insertMany([
  {item: "journal", qty: 25},
  {item: "notebook", qty: 50}
])
```

Missing collection: created

Missing `_id`: generated

Insert supplies documents; MongoDB supplies an absent _id

Selecting the Update Targets

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

FIND RESULT

```
{ "item": "journal", "status": "A" }
{ "item": "notebook", "status": "A" }
{ "item": "postcard", "status": "A" }
```

Preview the three selected documents before changing them

Changing the Selected Documents

Before

item	status
journal	A
notebook	A
paper	D
planner	D
postcard	A

After updating A

item	status
journal	Review
notebook	Review
paper	D
planner	D
postcard	Review

```
db.inventory.updateMany({status: "A"},
                        {$set: {status: "Review"}});
```

Only the three A statuses change; paper and planner keep D

Caveat: One Document or Many

Single document update

Its field changes are
atomic together

updateMany

Several document updates
are not one atomic unit

Use a transaction when all selected documents must change together

Recap: Writes Start from a Fresh Copy

Check	Expected result
Read status A before writing	journal, notebook, postcard
Change A to Review	Three documents change
Read status A after writing	No documents remain

Preview, write and verify in the same script; each Run starts from the original inventory

Demo: Writes

Preview the same records, change one field, then check the result

1. Preview status A items, then change their status to Review
2. Verify that status A has zero items and status Review has three

```
db.inventory.find({status: "A"}, {_id: 0,
item: 1, status: 1});
```

FIND RESULT

```
{ "item": "journal", "status": "A" }
{ "item": "notebook", "status": "A" }
{ "item": "postcard", "status": "A" }
```

Answer: Verify the Writes

```
db.inventory.updateMany(
  {status: "A"},
  {$set: {status: "Review"}}
);
db.inventory.countDocuments({status:
"Review"});
```

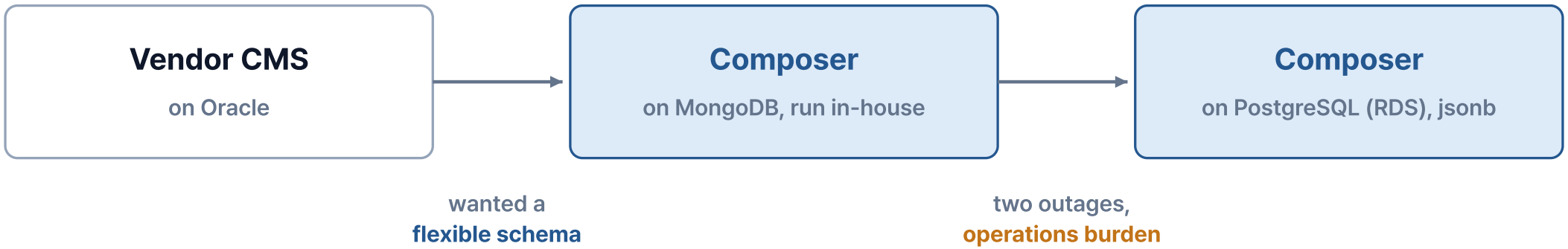
COUNTDOCUMENTS RESULT

```
3
```

Three items are marked Review; the two original D items remain unchanged

Case study: The Guardian’s Composer

A newsroom CMS moved off MongoDB in 2018 but kept its documents, as jsonb in PostgreSQL



The documents stayed documents; **the engine** under them changed

Takeaway: Document Queries

1. A condition must name the right **entity and array element**
2. Each pipeline stage changes the input; track **structure, grain and repeated values**
3. Verify **write targets and atomicity boundaries**; choose the model and engine separately