

PATTERNS

▶ Attribute: subset of fields that share characteristics

+: fewer indexes, simpler queries

ex:

```

releases: [
  { "location": <String>, "date": <Date> }, ...
]
    
```

▶ Bucket: instead of one document per measurement per ID bucket the data under the ID's

+: reduces number of documents, improve index performance and simplify data access

ex:

```

{
  sensor_id: 1234
  measure: ...
  date: ...
}
    
```

→

```

{
  sensor_id: <...>
  start: <date>
  end: <date>
  measurements: [ { sensor:
                   date:
                   }, ... ]
}
    
```

... ~ SAME PATTERN & ID

+: reduced CPU workload for frequent computations
-: difficult to identify the need

▶ Computed: leave to the program the operation (like avg/sum)

▶ Document Versioning: add a field for "revision" (incrementing)

+: easy to implement with no performance impact on queries
-: double the writes and need to target the correct collection

▶ Extended Reference: copy frequently accessed fields from other referenced documents

+: improve performance on join operations with faster reads and less data fetching
-: data is duplicated so could be bad for rapidly changing attributes

▶ Outlier: if few documents in a collection require a certain field → add a boolean field to check for extras (Y/N)

+: prevents a few documents or queries from determining application solution, queries are the same only outliers change
-: ad hoc queries may not perform well, much of this pattern is application side

(→ add new collection with space for overflow info and <object ID> linking)

▶ Pre-allocation

+: simplify design when the document structure is known in advance -: simplicity VS performance

▶ Polymorphic: keep fields in common across all documents and add others (and others) depending

► **POLYMORPHIC**: keep fields in common across all documents and change only specific attributes (add reviewers) depending on the category they belong to

- +: easy to implement, with queries that run across multiple collections
- : different code path required in the application based on the document

► **SCHEMA VERSIONING**

- +: no downtime in schema updating, control the migration with less future debt
- : might need two indexes

► **SUBSET** → store the collections in 2 more subcollections (product + reviews → product & reviews)

- +: reduction in the overall size of the working set and shorter disk access time for the most used data
- : the subset must be managed and pulling extra data requires extra time

► **TREE**: store the full path from a node to the top of the hierarchy as a list of the parents

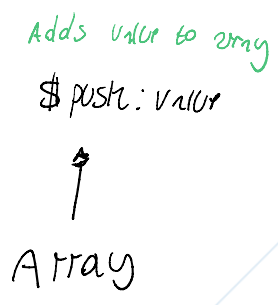
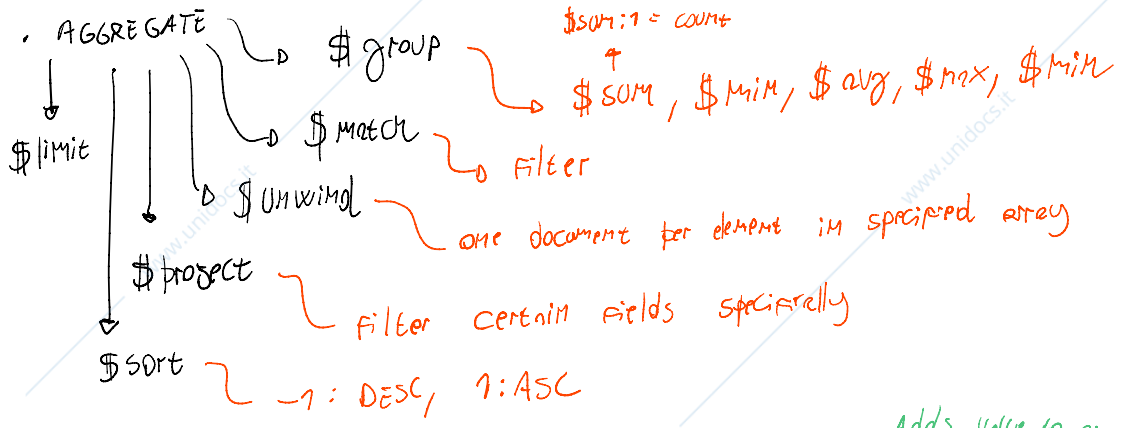
- +: increased performance by avoiding multiple JOINS
- : updates to the graph must be managed in the application

QUERIES

- OR → $\$or [\{ "x": ... \}, ...]$
- NOT → $\{ \$ne: "x" \}$

- db. xyz.find ({ }, { })
- .aggregate ([])
- .findOne ({ }, { })
- .updateOne ({ }, { })
- .updateMany ({ }, { })

same for $\$or, \and



- UPDATE
 - \$inc: amount increment
 - \$currentDate
 - \$set → set field x value to y
 - \$unset → remove specific value

\$mul \rightarrow \$unset \sim remove specific value

Multiply

► the \$ before a field is the value of said field ✓

\$in: [] - match any value of the array

\$gt / \$gte \$eq
\$lt / \$lte