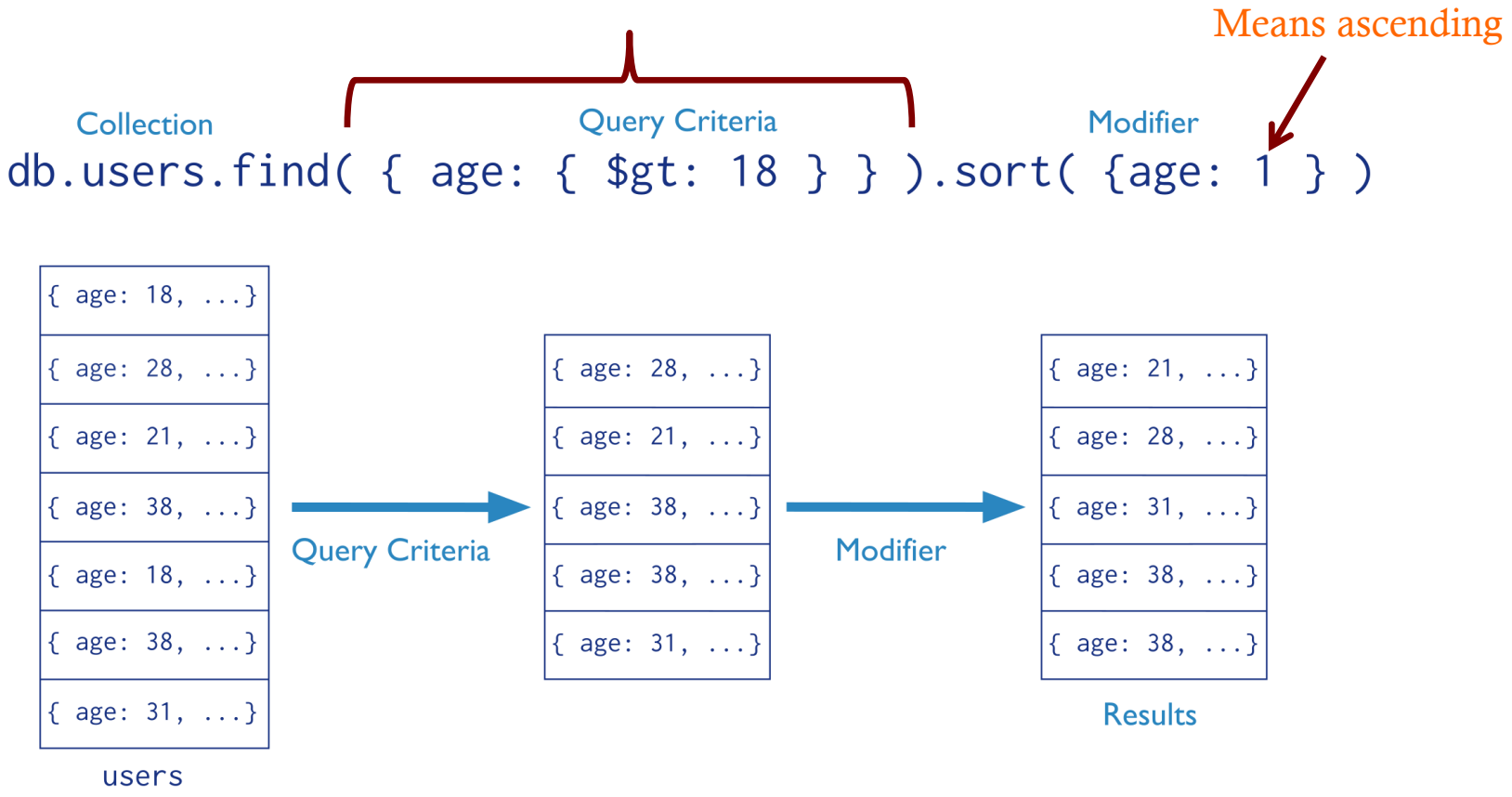
Query Language in MongoDB

Find() Operator



Find() + Projection

```
db.users.find(
    { age: { $gt: 18 } },
    { name: 1, address: 1 }
).limit(5)
```

collection
query criteria
projection
cursor modifier

Means inclusion +

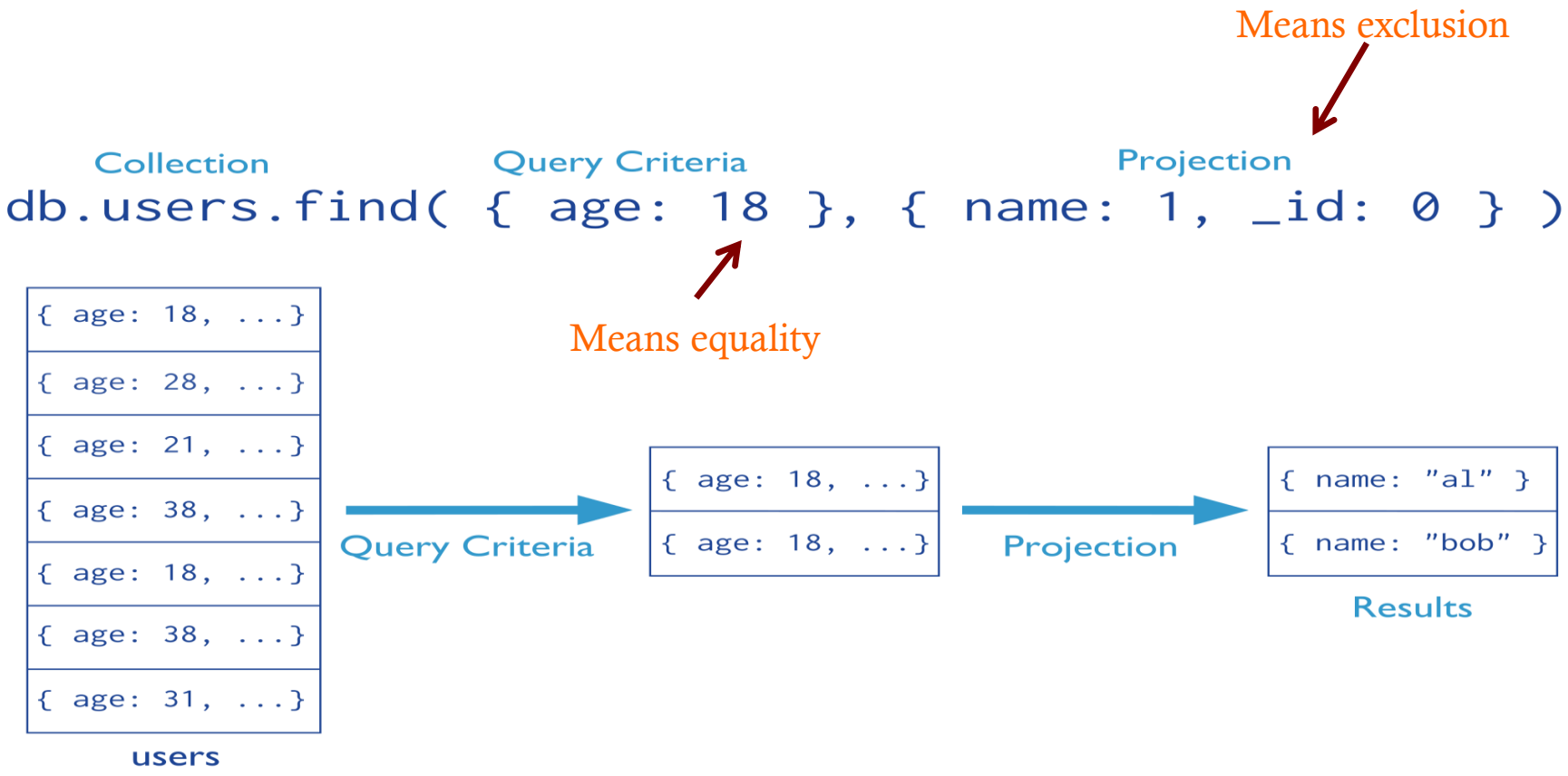
_id is always automatically included

Equivalent to in SQL:

```
SELECT _id, name, address
FROM   users
WHERE  age > 18
LIMIT  5
```

projection
table
select criteria
cursor modifier

Find(): Exclude Fields



Cannot mix "inclusion & exclusion" in the same operator except for *_id*

More Examples for Find()

Report all documents in the "inventory" collection

```
db.inventory.find( )
```

Equivalent to in SQL:

```
Select *
From inventory;
```

Report all documents in the "inventory" collection
Where **type = 'food' or 'snacks'**

```
db.inventory.find(
        { type: { $in: [ 'food', 'snacks' ] } }
)
```

Equivalent to in SQL:

```
Select *
From inventory
Where type in
        ('food', 'snacks');
```

Find(): AND & OR Connectors

AND Semantics (default and thus implicit connector)

```
db.inventory.find( { type: 'food', price: { $lt: 9.95 } } )
```

OR Semantics

```
db.inventory.find(
    {
      $or: [ { qty: { $gt: 100 } }, { price: { $lt: 9.95 } } ]
    }
)
```

AND + OR Semantics

```
db.inventory.find(
    {
      type: 'food',
      $or: [ { qty: { $gt: 100 } }, { price: { $lt: 9.95 } } ]
    }
)
```

Type = ‘food’ and (qty > 100 or price < 9.95)

$AND: Logical AND Operation

{ $and: [{ <expression1> }, { <expression2> } , ... , {<expressionN> }] }

```
db.inventory.find(
{ $and :
[
{ $or : [ { price : 0.99 }, { price : 1.99 }] },
{ $or : [ { sale : true }, { qty : { $lt : 20 }}] }
]
} )
```

Queries Return Cursors

- All queries return the results in a cursor
- If not assigned to a variable ➔ Printed to screen
  - Results are stored in a cursor
  - Operators on results can manipulate the cursor

```
var myCursor = db.users.find( { type: 2 } );

while (myCursor.hasNext())

{ print(  tojson(myCursor.next())); }
```

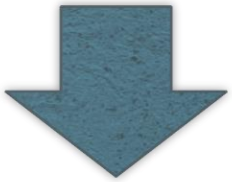
Cursor's Methods: http://docs.mongodb.org/manual/reference/method/js-cursor/

Querying Complex Types

Querying Complex Types

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 25,
  "height_cm": 167.6,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [],
  "spouse": null
}
```

Documents can be complex, e.g., arrays, embedded documents, any nesting of these, many levels



Queries can query these complex types

Array Manipulation (Exact Match)

```
{ _id: 5, type: "food", item: "aaa", ratings: [ 5, 8, 9 ] }
{ _id: 6, type: "food", item: "bbb", ratings: [ 5, 9 ] }
{ _id: 7, type: "food", item: "ccc", ratings: [ 9, 5, 8 ] }
```

```
db.inventory.find( { ratings: [ 5, 8, 9 ] } )
```

Array Manipulation (Exact Match)

```
{ _id: 5, type: "food", item: "aaa", ratings: [ 5, 8, 9 ] }
{ _id: 6, type: "food", item: "bbb", ratings: [ 5, 9 ] }
{ _id: 7, type: "food", item: "ccc", ratings: [ 9, 5, 8 ] }
```

```
db.inventory.find( { ratings: [ 5, 8, 9 ] } )
```

The operation returns the following document:

```
{ "_id" : 5, "type" : "food", "item" : "aaa", "ratings" : [ 5, 8, 9 ] }
```

Array Manipulation (Search By Element)

```
{ _id: 5, type: "food", item: "aaa", ratings: [ 5, 8, 9 ] }
{ _id: 6, type: "food", item: "bbb", ratings: [ 5, 9 ] }
{ _id: 7, type: "food", item: "ccc", ratings: [ 9, 5, 8 ] }
```

```
db.inventory.find( { ratings: 5 } )
```

Array Manipulation (Search By Element)

```
{ _id: 5, type: "food", item: "aaa", ratings: [ 5, 8, 9 ] }
{ _id: 6, type: "food", item: "bbb", ratings: [ 5, 9 ] }
{ _id: 7, type: "food", item: "ccc", ratings: [ 9, 5, 8 ] }
```

```
db.inventory.find( { ratings: 5 } )
```

The operation returns the following documents:

```
{ "_id" : 5, "type" : "food", "item" : "aaa", "ratings" : [ 5, 8, 9 ] }
{ "_id" : 6, "type" : "food", "item" : "bbb", "ratings" : [ 5, 9 ] }
{ "_id" : 7, "type" : "food", "item" : "ccc", "ratings" : [ 9, 5, 8 ] }
```

Notice: if a document has "ratings" as an Integer field = 5, it will be returned

Array Manipulation (Search By Position)

```
{ _id: 5, type: "food", item: "aaa", ratings: [ 5, 8, 9 ] }
{ _id: 6, type: "food", item: "bbb", ratings: [ 5, 9 ] }
{ _id: 7, type: "food", item: "ccc", ratings: [ 9, 5, 8 ] }
```

```
db.inventory.find( { 'ratings.0': 5 } )
```

Array Manipulation (Search By Position)

```
{ _id: 5, type: "food", item: "aaa", ratings: [ 5, 8, 9 ] }
{ _id: 6, type: "food", item: "bbb", ratings: [ 5, 9 ] }
{ _id: 7, type: "food", item: "ccc", ratings: [ 9, 5, 8 ] }
```

```
db.inventory.find( { 'ratings.0': 5 } )
```

The operation returns the following documents:

```
{ "_id" : 5, "type" : "food", "item" : "aaa", "ratings" : [ 5, 8, 9 ] }
{ "_id" : 6, "type" : "food", "item" : "bbb", "ratings" : [ 5, 9 ] }
```

Array Manipulation ($elemMatch)

```
{ <field>: { $elemMatch: { <query1>, <query2>, ... } } }
```

$elemMatch operator matches documents
that contain an **array** field
with at least **one** element
that matches **all** specified query criteria.

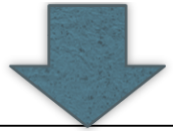
Another Example

```
{ _id: 5, type: "food", item: "aaa", ratings: [ 5, 8, 9 ] }
{ _id: 6, type: "food", item: "bbb", ratings: [ 5, 9 ] }
{ _id: 7, type: "food", item: "ccc", ratings: [ 9, 5, 8 ] }
```

```
db.inventory.find( { ratings: { $elemMatch: { $gt: 5, $lt: 9 } } } )
```

```
{ "_id" : 5, "type" : "food", "item" : "aaa", "ratings" : [ 5, 8, 9 ] }
{ "_id" : 7, "type" : "food", "item" : "ccc", "ratings" : [ 9, 5, 8 ] }
```

```
db.inventory.find( { ratings: { $gt: 5, $lt: 9 } } )
```



```
{ "_id" : 5, "type" : "food", "item" : "aaa", "ratings" : [ 5, 8, 9 ] }
{ "_id" : 6, "type" : "food", "item" : "bbb", "ratings" : [ 5, 9 ] }
{ "_id" : 7, "type" : "food", "item" : "ccc", "ratings" : [ 9, 5, 8 ] }
```

Embedded Object Matching (**Exact document Matching**)

```
{
        name: "Joe",
        address: {
                city: "San Francisco",
                state: "CA" } ,
        likes: [ 'scuba', 'math', 'literature' ]
}
```

```
db.persons.find( { "address" : { state: "CA" }} )  //don't match

db.persons.find( { "address" : {city: "San Francisco", state: "CA" }} )  // match

db.persons.find( { "address" : {state: "CA" , city: "San Francisco"}} ) //don't match
```

Exact-match (entire object)

Embedded Object Matching (**Field Matching**)

```
{
        name: "Joe",
        address: {
                city: "San Francisco",
                state: "CA" } ,
        likes: [ 'scuba', 'math', 'literature' ]
}
```

Find the user documents where the address's state = 'CA'

db.persons.find({"address.state" : "CA"})  Using dot notation

Matching Arrays of Embedded Documents

```
{
  _id: 100,
  type: "food",
  item: "xyz",
  qty: 25,
  price: 2.5,
  ratings: [ 5, 8, 9 ],
  memos: [ { memo: "on time", by: "shipping" }, { memo: "approved", by: "billing" } ]
}

{
  _id: 101,
  type: "fruit",
  item: "jkl",
  qty: 10,
  price: 4.25,
  ratings: [ 5, 9 ],
  memos: [ { memo: "on time", by: "payment" }, { memo: "delayed", by: "shipping" } ]
}
```

Q:Select all documents where the memos array contains in the 1st element a document written by 'shipping' department

Matching Arrays of Embedded Documents

```
{
  _id: 100,
  type: "food",
  item: "xyz",
  qty: 25,
  price: 2.5,
  ratings: [ 5, 8, 9 ],
  memos: [ { memo: "on time", by: "shipping" }, { memo: "approved", by: "billing" } ]
}

{
  _id: 101,
  type: "fruit",
  item: "jkl",
  qty: 10,
  price: 4.25,
  ratings: [ 5, 9 ],
  memos: [ { memo: "on time", by: "payment" }, { memo: "delayed", by: "shipping" } ]
}
```

db.inventory.find({ 'memos.**0**.by': 'shipping' }) //Returns 1st document

Means the 1st element in the array

Matching Arrays of Embedded Documents

```
{
  _id: 100,
  type: "food",
  item: "xyz",
  qty: 25,
  price: 2.5,
  ratings: [ 5, 8, 9 ],
  memos: [ { memo: "on time", by: "shipping" }, { memo: "approved", by: "billing" } ]
}

{
  _id: 101,
  type: "fruit",
  item: "jkl",
  qty: 10,
  price: 4.25,
  ratings: [ 5, 9 ],
  memos: [ { memo: "on time", by: "payment" }, { memo: "delayed", by: "shipping" } ]
}
```

Q: Select all documents where the memos array contains a document written “by” shipping department

Matching Arrays of Embedded Documents

```
{
  _id: 100,
  type: "food",
  item: "xyz",
  qty: 25,
  price: 2.5,
  ratings: [ 5, 8, 9 ],
  memos: [ { memo: "on time", by: "shipping" }, { memo: "approved", by: "billing" } ]
}

{
  _id: 101,
  type: "fruit",
  item: "jkl",
  qty: 10,
  price: 4.25,
  ratings: [ 5, 9 ],
  memos: [ { memo: "on time", by: "payment" }, { memo: "delayed", by: "shipping" } ]
}
```

db.inventory.find({ 'memos.by': 'shipping' }) // Returns both documents

Means any element in the array

Matching Arrays of Embedded Documents: Multiple Conditions

```
{
  _id: 100,
  type: "food",
  item: "xyz",
  qty: 25,
  price: 2.5,
  ratings: [ 5, 8, 9 ],
  memos: [ { memo: "on time", by: "shipping" }, { memo: "approved", by: "billing" } ]
}

{
  _id: 101,
  type: "fruit",
  item: "jkl",
  qty: 10,
  price: 4.25,
  ratings: [ 5, 9 ],
  memos: [ { memo: "on time", by: "payment" }, { memo: "delayed", by: "shipping" } ]
}
```

Q: Select all documents where memos array contains **a document** written by 'shipping' department and its content is "on time"

Matching Arrays of Embedded Documents: Multiple Conditions

```
db.inventory.find(
    {
        memos:
            {
                $elemMatch:
                    {
                        memo: 'on time',
                        by: 'shipping'
                    }
            }
    }
)
```

Matching Arrays of Embedded Documents: Multiple Conditions

```
{
  _id: 100,
  type: "food",
  item: "xyz",
  qty: 25,
  price: 2.5,
  ratings: [ 5, 8, 9 ],
  memos: [ { memo: "on time", by: "shipping" }, { memo: "approved", by: "billing" } ]
}

{
  _id: 101,
  type: "fruit",
  item: "jkl",
  qty: 10,
  price: 4.25,
  ratings: [ 5, 9 ],
  memos: [ { memo: "on time", by: "payment" }, { memo: "delayed", by: "shipping" } ]
}
```

```
db.inventory.find(
   {
     memos:
       {
          $elemMatch:
            {
               memo: 'on time',
               by: 'shipping'
            }
       }
    }
)
```

Summary: Query Operators

- http://docs.mongodb.org/manual/reference/operator/query/

- Comparison Operators
- Logical Operators
- Element Operators
- Evaluation Operators
- Array Operators

Query Operators: Comparison Op

Name	Description
$eq	Matches values that are equal to a specified value.
$gt	Matches values that are greater than a specified value.
$gte	Matches values that are greater than or equal to a specified value.
$lt	Matches values that are less than a specified value.
$lte	Matches values that are less than or equal to a specified value.
$ne	Matches all values that are not equal to a specified value.
$in	Matches any of the values specified in an array.
$nin	Matches none of the values specified in an array.

```
db.inventory.find
( { qty: { $gte: 20 } } )
```

```
db.inventory.update(
      { "carrier.fee": { $gte: 2}},
      { $set: { price: 9.99 } }
)
```

Query Operators: Evaluation Op

Name	Description
`$mod`	Performs a modulo operation on the value of a field and selects documents with a specified result.
`$regex`	Selects documents where values match a specified regular expression.
`$text`	Performs text search.
`$where`	Matches documents that satisfy a JavaScript expression.

$Where Operator

- Passes a ***JavaScript expression or function*** to query system
- Very flexible in expressing complex conditions
- But relatively slow as it evaluates for each document (no index)
- Similar to using ***UDF*** in ***WHERE*** clause in relational database

```
db.myCollection.find( { $where: "this.credits == this.debits" } );
db.myCollection.find( { $where: "obj.credits == obj.debits" } );

db.myCollection.find( { $where: function() { return (this.credits == this.debits) } } );
db.myCollection.find( { $where: function() { return obj.credits == obj.debits; } } );
```

$Where Operator

- Can combine MongoDB operators with $Where

```
db.myCollection.find(
  { active: true, $where: "this.credits - this.debits < 0" } );


db.myCollection.find(
  { active: true,
    $where: function() { return obj.credits - obj.debits < 0; } }
);
```

Collection Modeling

Collection Modeling

- Modeling multiple collections that reference each other
- In Relational DBs ➔ FK-PK Relationships
- In MongoDB, two options:
  - Referencing
  - Embedding

FK-PK in Relational DBs

- Create “Students” relation

```
CREATE TABLE Students
        (sid CHAR(20),
         name CHAR(20),
         login CHAR(10),
         age INTEGER,
         gpa REAL);
```

- Create “Courses” relation

```
CREATE TABLE Courses
        (cid Varchar2(20),
         name varchar2(50),
         maxCredits integer,
         graduateFlag char(1));
```

Foreign key

Foreign key

- Create “Enrolled” relation

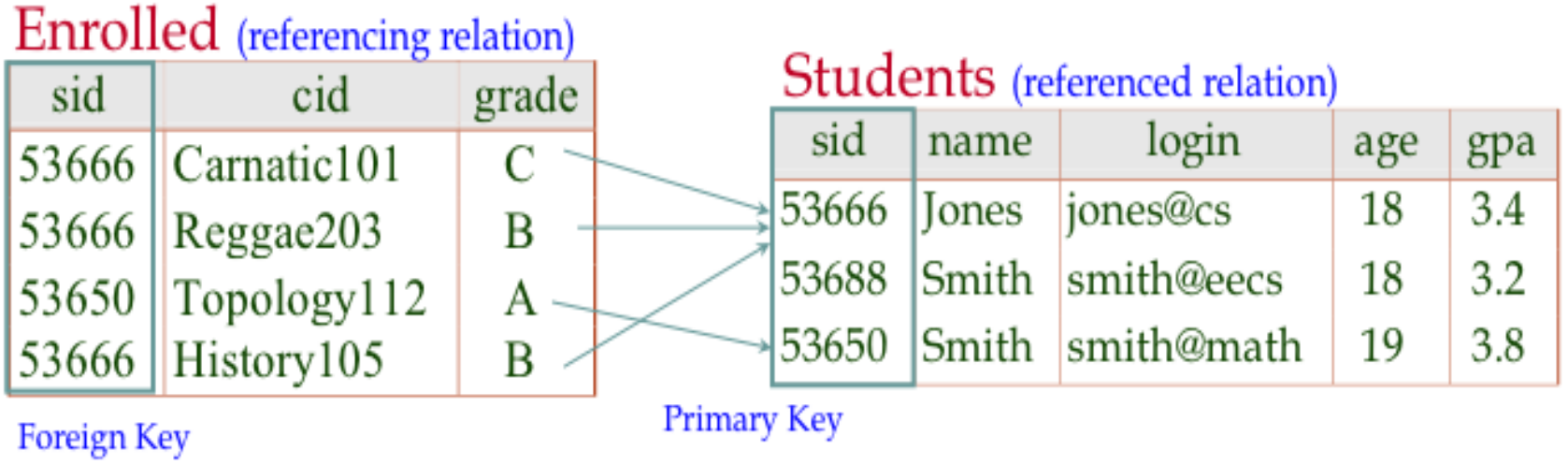
```
CREATE TABLE Enrolled
        (sid CHAR(20),
         cid Varchar2(20),
         enrollDate date,
         grade CHAR(2));
```

- **Each tuple in “Enrolled” reference a specific student and a specific course**

FK-PK in Relational DBs

It comes with an enforcement mechanism

- Cannot insert a FK for a non-existing PK
- You cannot delete a PK that has a FK

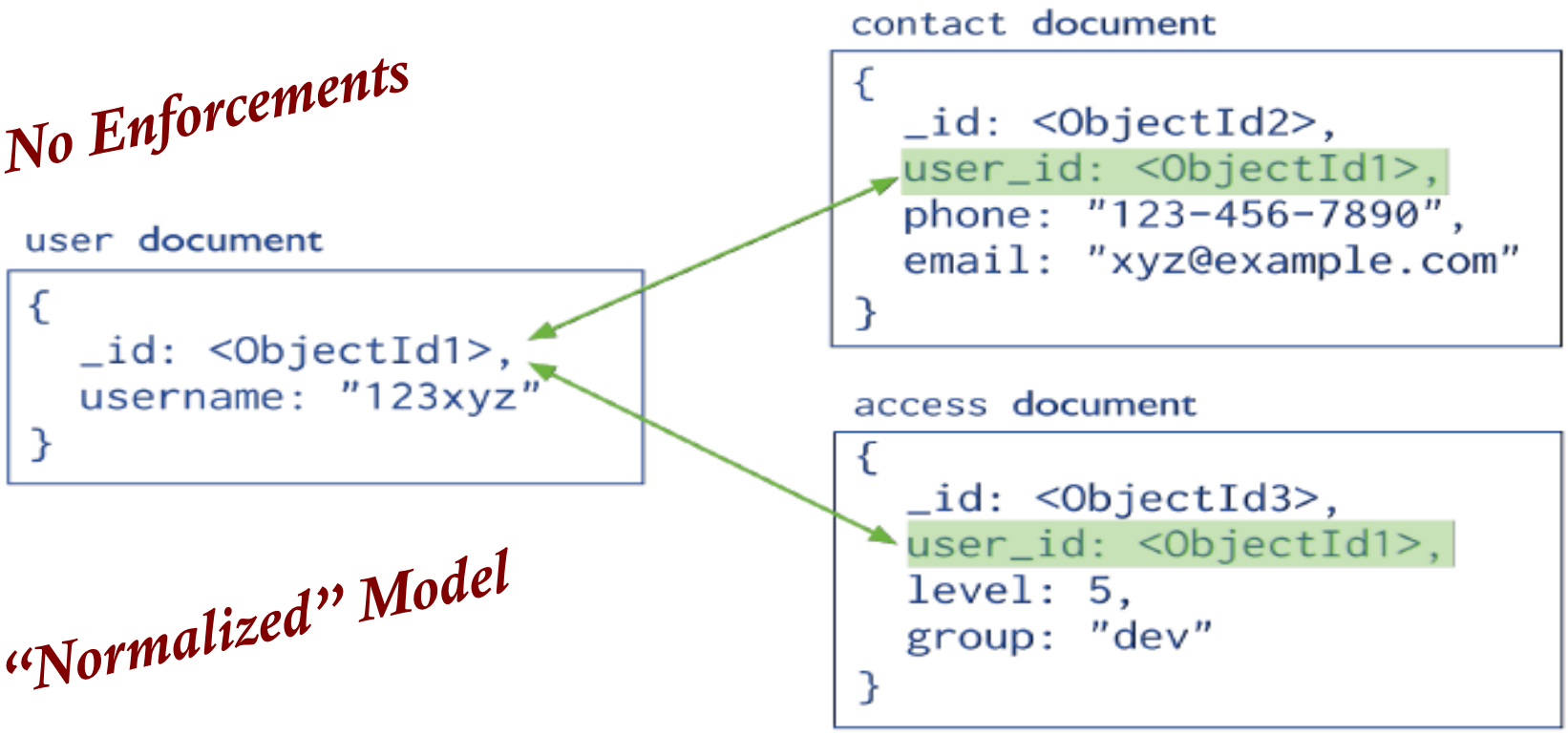


“Relationships” In MongoDB

- ***Referencing*** between two collections
  - Use Id of one and put in the other collection
  - Very similar to FK-PK in Relational Databases
  - **HOWEVER: Does not come with any enforcement !**

- ***Embedding*** between two collections
  - Put the document from one collection inside the other one

Modeling using Referencing



- Have three collections in the DB: "User", "Contact", "Access"
- Link them by _id (or any other field(s))

Embedding

De-Normalized Way

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

Embedded sub-document

Embedded sub-document

- Have one collection in DB as the "User" document
- The others are embedded inside a user's document model

Examples (1)

Referencing

- "Patron" & "Addresses"

```
{
    _id: "joe",
    name: "Joe Bookreader"
}
```

```
{
    patron_id: "joe",
    street: "123 Fake Street",
    city: "Faketon",
    state: "MA",
    zip: "12345"
}
```

- If it is 1-1 relationship
- If usually read the address with the name
- If address document usually does not expand

If most of these hold

➔

better use Embedding

Examples (2)

Embedding

- "Patron" & "Addresses"

```
{
    _id: "joe",
    name: "Joe Bookreader",
    address: {
                street: "123 Fake Street",
                city: "Faketon",
                state: "MA",
                zip: "12345"
              }
}
```

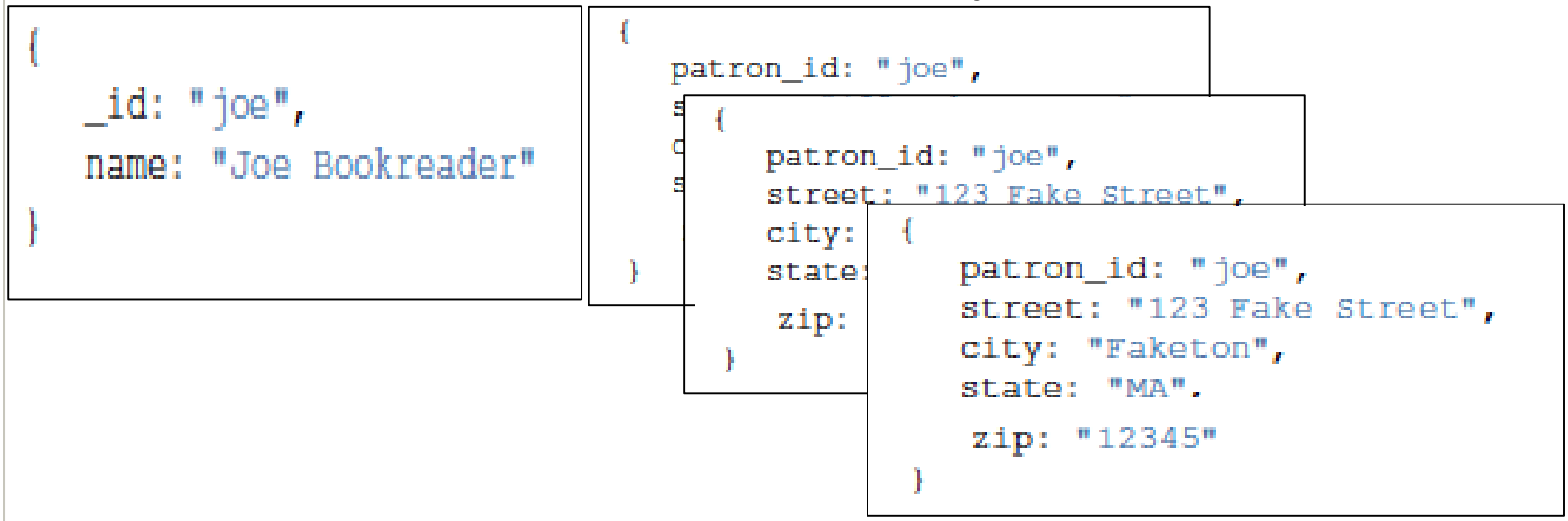
Advantages:

1. When you read, you get entire document at once
2. In Referencing ➔ Need to issue multiple queries

Examples (3)

Referencing

- What if a "Patron" can have many "Addresses" ?



- Do you read them together ➔ Go for Embedding
- Are addresses dynamic (e.g., add new ones frequently)

➔ Go for Referencing

Examples (4)

Embedding

If a "Patron" can have many "Addresses", how to embed?

```
{
   _id: "joe",
   name: "Joe Bookreader",
   addresses: [
                {
                  street: "123 Fake Street",
                  city: "Faketon",
                  state: "MA",
                  zip: "12345"
                },
                {
                  street: "1 Some Other Street",
                  city: "Boston",
                  state: "MA",
                  zip: "12345"
                }
              ]
 }
```

Use array of addresses

Examples (5)

- If addresses are added frequently … ?

```
{
    _id: "joe",
    name: "Joe Bookreader",
    addresses: [
                  {
                    street: "123 Fake Street",
                    city: "Faketon",
                    state: "MA",
                    zip: "12345"
                  },
                  {
                    street: "1 Some Other Street",
                    city: "Boston",
                    state: "MA",
                    zip: "12345"
                  }
                ]
}
```

This array will expand frequently

Size of "Patron" document increases frequently

May trigger re-locating the document each time

Bad !!!!!!

Document Size and Storage

- Each document contiguous on disk
- If doc size increases
  ➔ Document location must change
- If doc location changes
  ➔ Indexes must be updated
  ➔ leads to expensive updates

```
{
  _id: "joe",
  name: "Joe Bookreader",
  addresses: [
               {
                 street: "123 Fake Street",
                 city: "Faketon",
                 state: "MA",
                 zip: "12345"
               },
               {
                 street: "1 Some Other Street",
                 city: "Boston",
                 state: "MA",
                 zip: "12345"
               }
             ]
}
```

<u>In a newer version of MongoDB:</u>

Each document is allocated a ***power-of-2 bytes*** (smallest above its size)

System keeps some space empty for possible expansion

More Examples (6)

- **One-to-Many "Book", "Publisher"**
  - A book has one publisher
  - A publisher publishes many books

How Model ?
Pros and Cons ?

- **If embed "Publisher" inside "Book"**
  - Repeating publisher info inside each of its books
  - Very hard to update publisher's info (replicas, inconsistency!)

- **If embed "Book" inside "Publisher"**
  - Book becomes an array (many)
  - Frequent updates and increases in size (expensive)

More Examples (6)

- **One-to-Many "Book", "Publisher"**
  - A book has one publisher
  - A publisher publishes many books

Referencing is better in this case

- **If embed "Publisher" inside "Book"**
  - Repeating publisher info inside each of its books
  - Very hard to update publisher's info

- **If embed "Book" inside "Publisher"**
  - Book becomes an array (many)
  - Frequently update and increases in size

Summary:

Query Language in MongoDB:
Powerful document traversal and search operations