Database Systems
(CS 355 / CE 373)

Dr. Umer Tariq
Assistant Professor,
Dhanani School of Science & Engineering,
Habib University

Acknowledgements

- Many slides have been borrowed from the official lecture slides accompanying the textbook:

  Database System Concepts, (2019), Seventh Edition,
  Avi Silberschatz, Henry F. Korth, S. Sudarshan
  McGraw-Hill, ISBN 9780078022159

  The original lecture slides are available at:

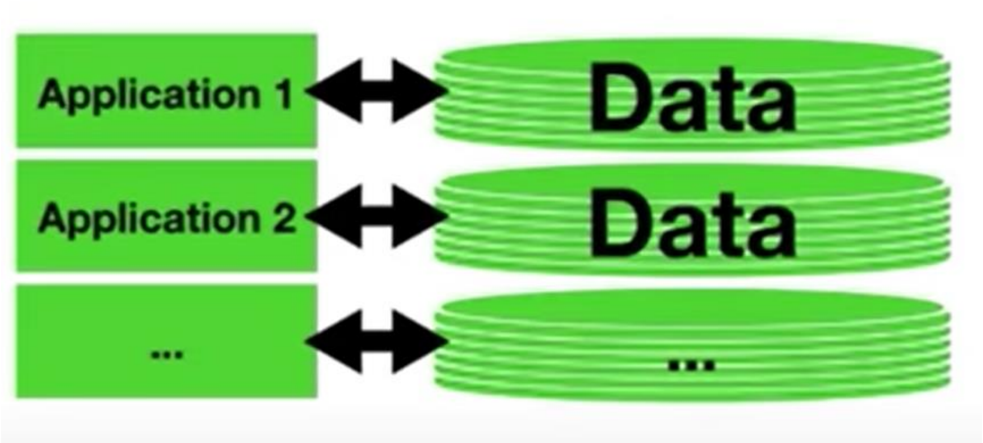
  https://www.db-book.com/

- Some of the slides have been borrowed from the lectures by Dr. Immanuel Trummer (Cornell University). Available at: (www.itrummer.org)

Outline: Week 13

- Introduction to “NoSQL” Databases
- Types of NoSQL Databases
- MongoDB
- MongoDB Query Language

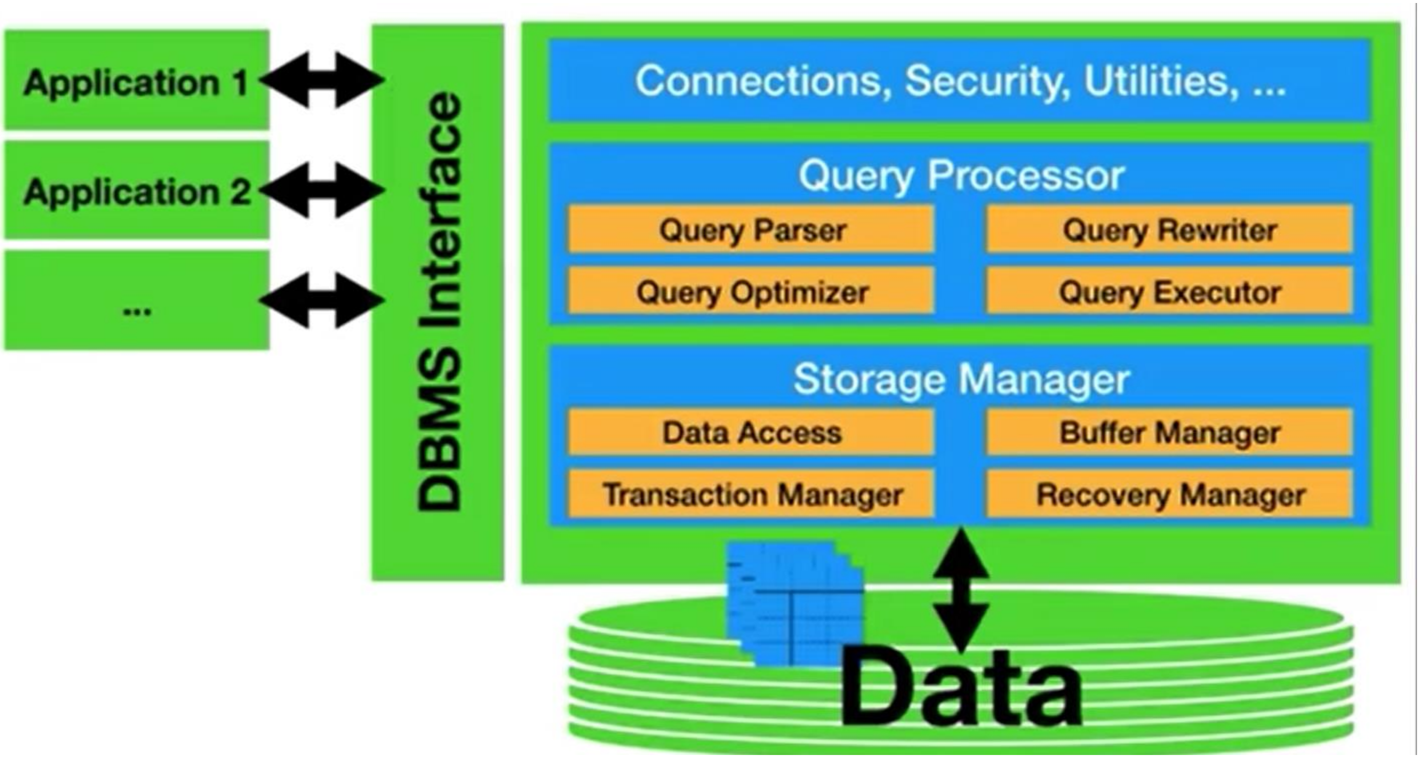
File-Based Approach



.csv

.csv

Database Management System (DBMS)



Application 1

Application 2

...

DBMS Interface

Connections, Security, Utilities, ...

Query Processor

Query Parser

Query Rewriter

Query Optimizer

Query Executor

Storage Manager

Data Access

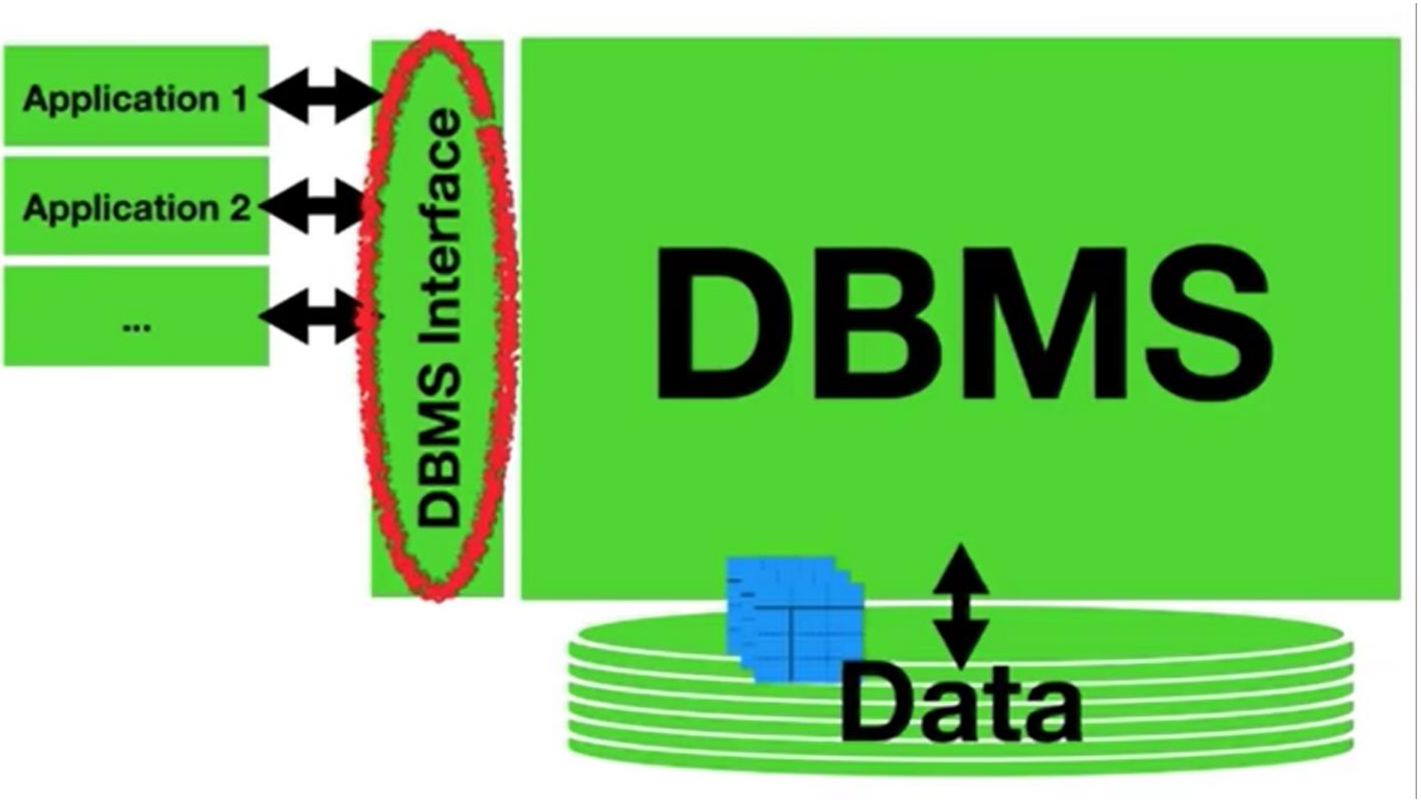
Buffer Manager

Transaction Manager

Recovery Manager

Data

Significance Of DBMS Interface

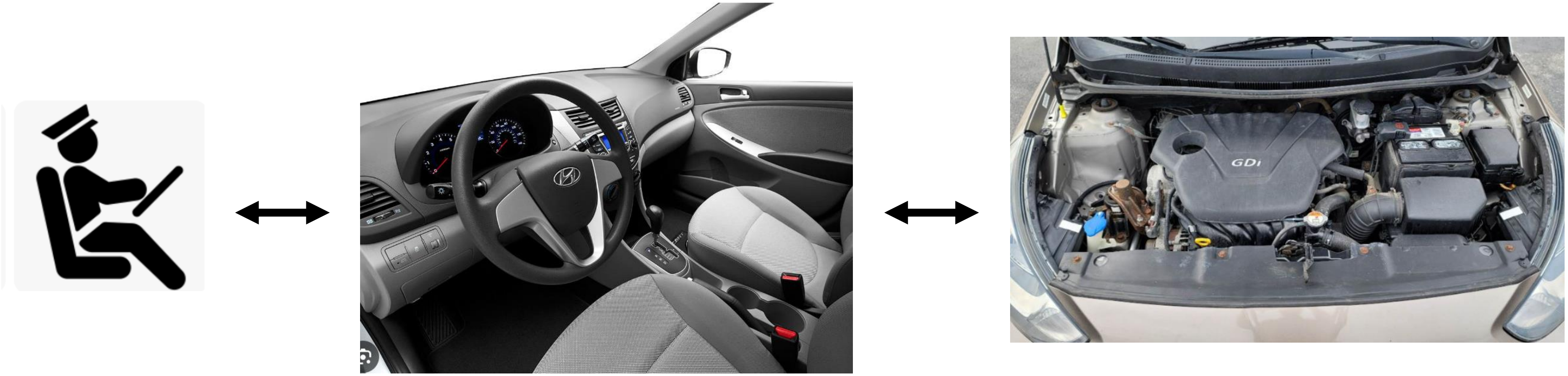


Why Focus on DBMS Interface?

- Managing complexity
  - Abstraction is your friend!

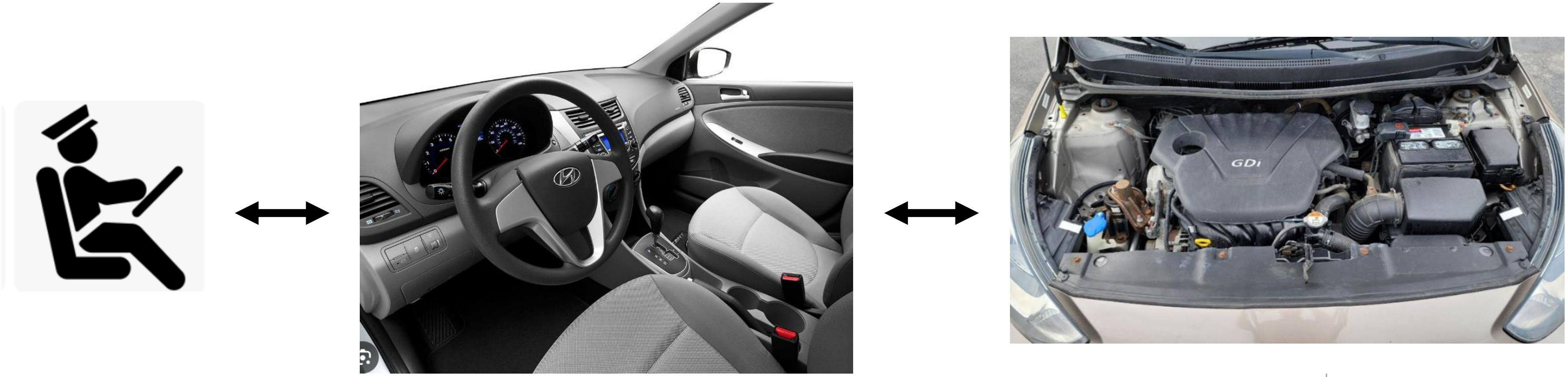
Why Focus on DBMS Interface?

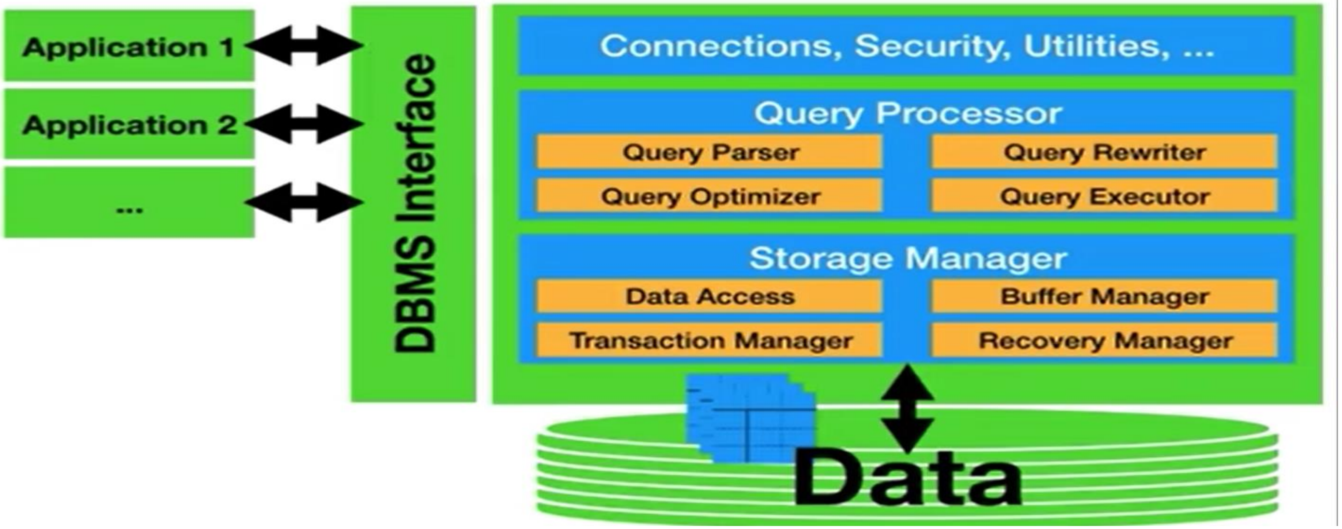
- Managing complexity
  - Abstraction is your friend!



Why Focus on DBMS Interface?

- Managing complexity
  - Abstraction is your friend!





Application 1

Application 2

...

DBMS Interface

Connections, Security, Utilities, ...

Query Processor

Query Parser

Query Rewriter

Query Optimizer

Query Executor

Storage Manager

Data Access

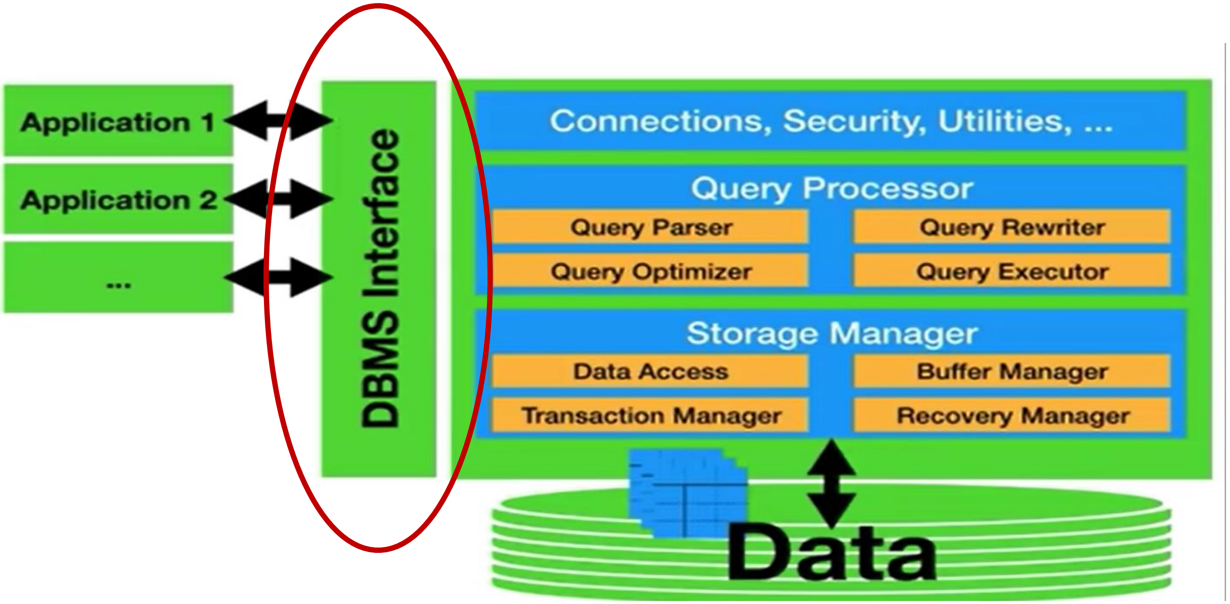
Buffer Manager

Transaction Manager

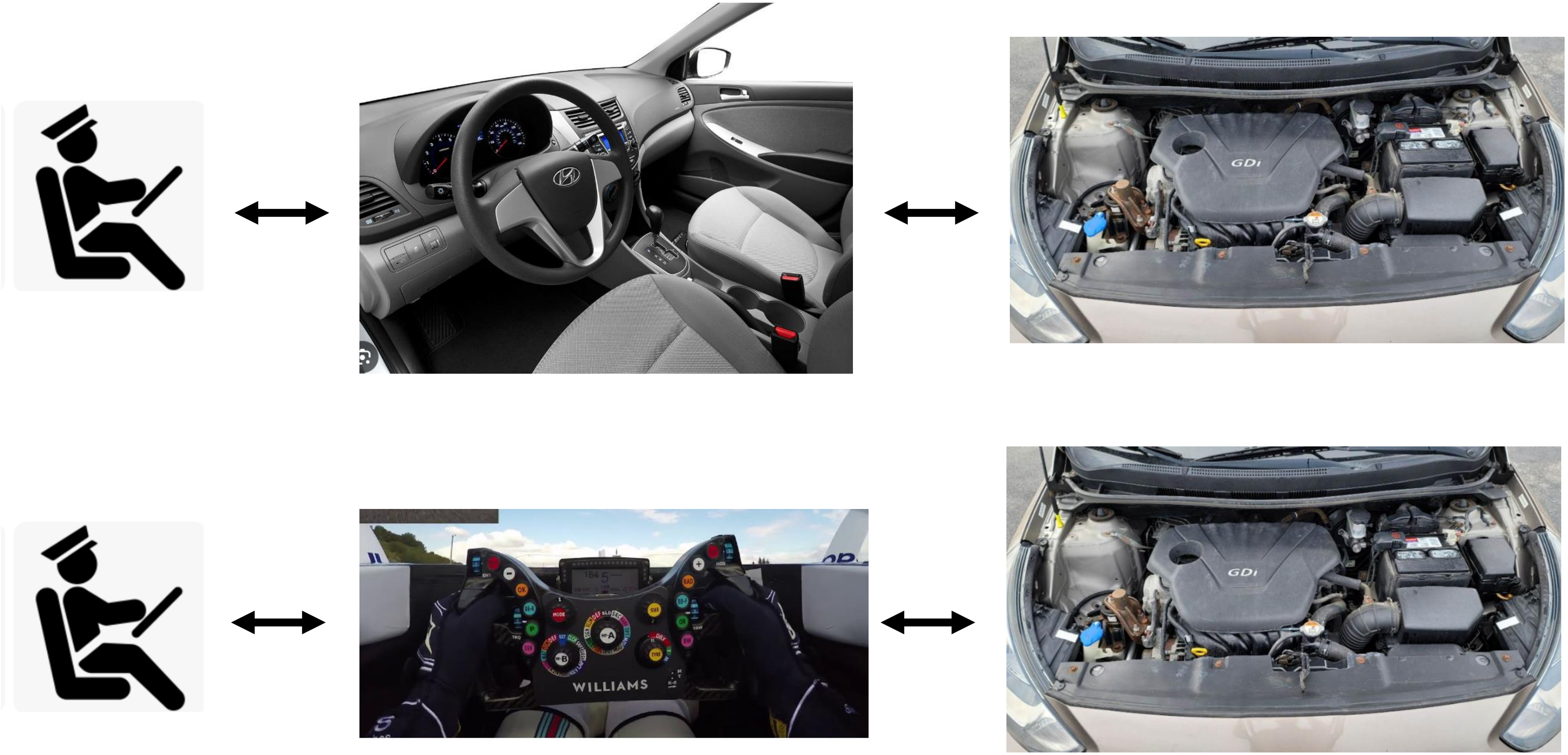
Recovery Manager

Data

What should be the DBMS Interface?

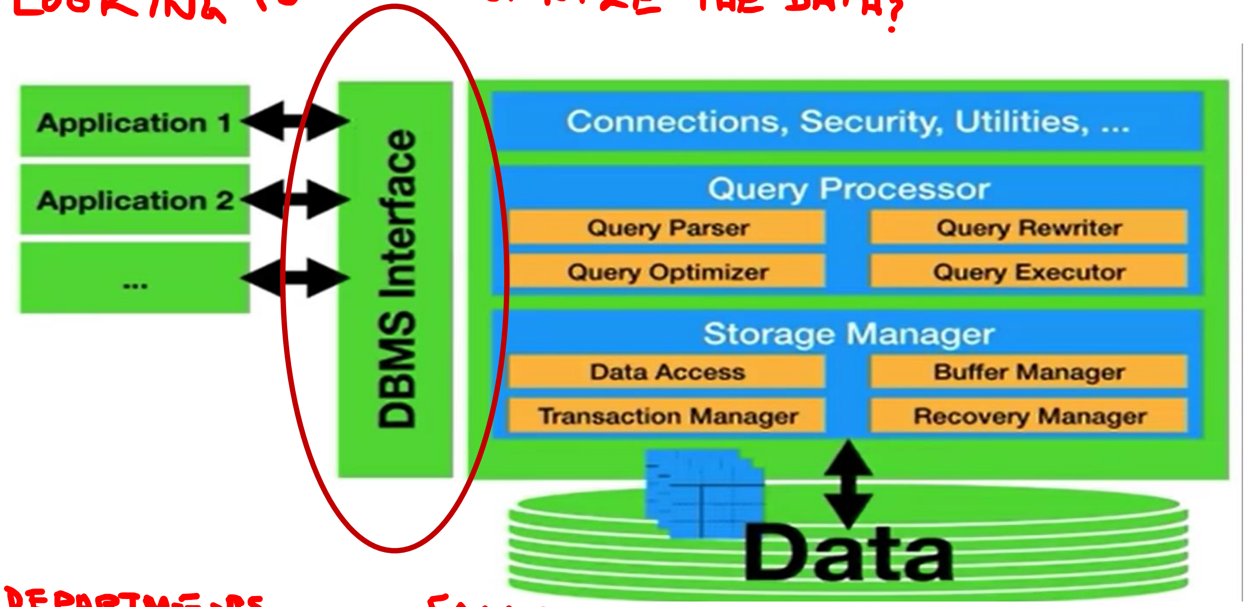


What Should be the DBMS Interface?



What should be the DBMS Interface?

"WE ARE LOOKING TO ORGANIZE THE DATA?



TABLES:-

DEPARTMENTS

ID	Name	Building
1	CS	HU
2	CE	C-100
3	EE	C-101

FACULTY

ID	Name	Address
107	Uma	KHI

STUDENT

ID	NAME	ADDRESS	Major
1	AMIR	KHI	CS

QUERY:.

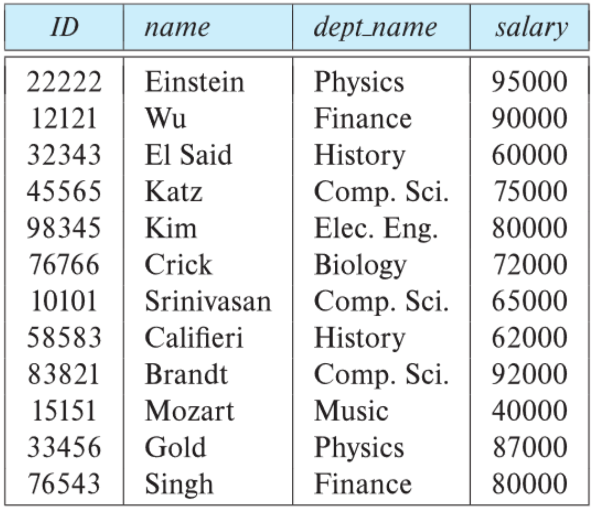
Give me a list of all CS students?

The "Relational" Model

(TABLE-based Data Model)

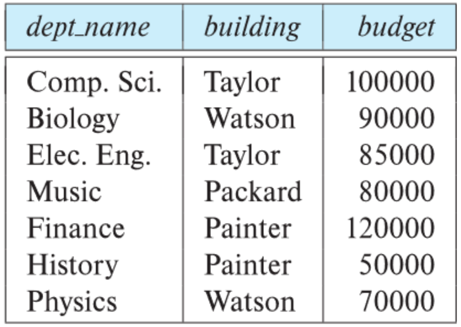
- The relational model uses a collection of tables to represent both data and the relationships among those data.

- Its conceptual simplicity has led to its widespread adoption; a vast majority of database products are based on the relational model.



ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

(a) The *instructor* table



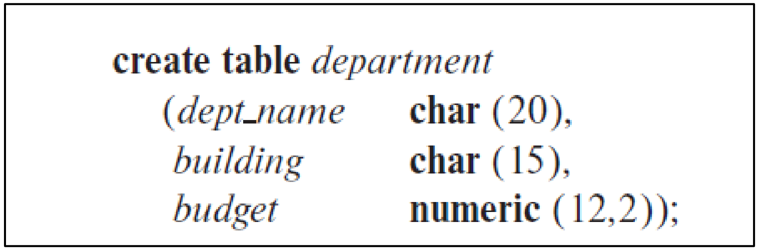
dept_name	*building*	*budget*
Comp. Sci.	Taylor	100000
Biology	Watson	90000
Elec. Eng.	Taylor	85000
Music	Packard	80000
Finance	Painter	120000
History	Painter	50000
Physics	Watson	70000

(b) The *department* table

Structured Query Language (SQL)

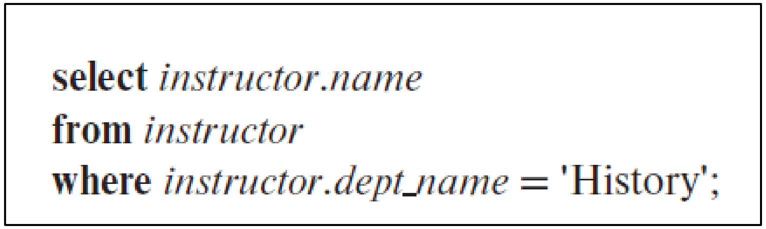
- The standard to access/retrieve/manipulate data in a **relational database**

- Examples of a Data Definition Language (DDL) Component

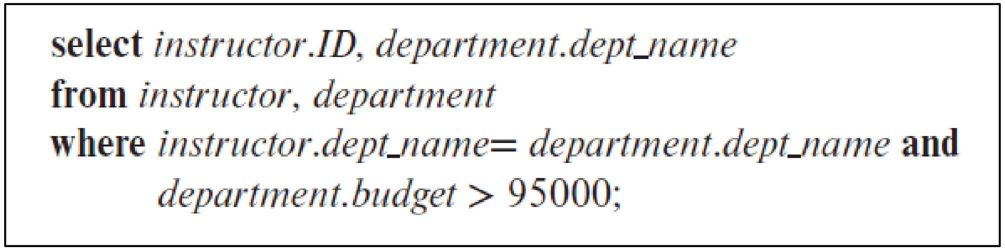


```
create table department
    (dept_name     char (20),
     building      char (15),
     budget        numeric (12,2));
```

- Examples of a Data Manipulation Language (DML) Component

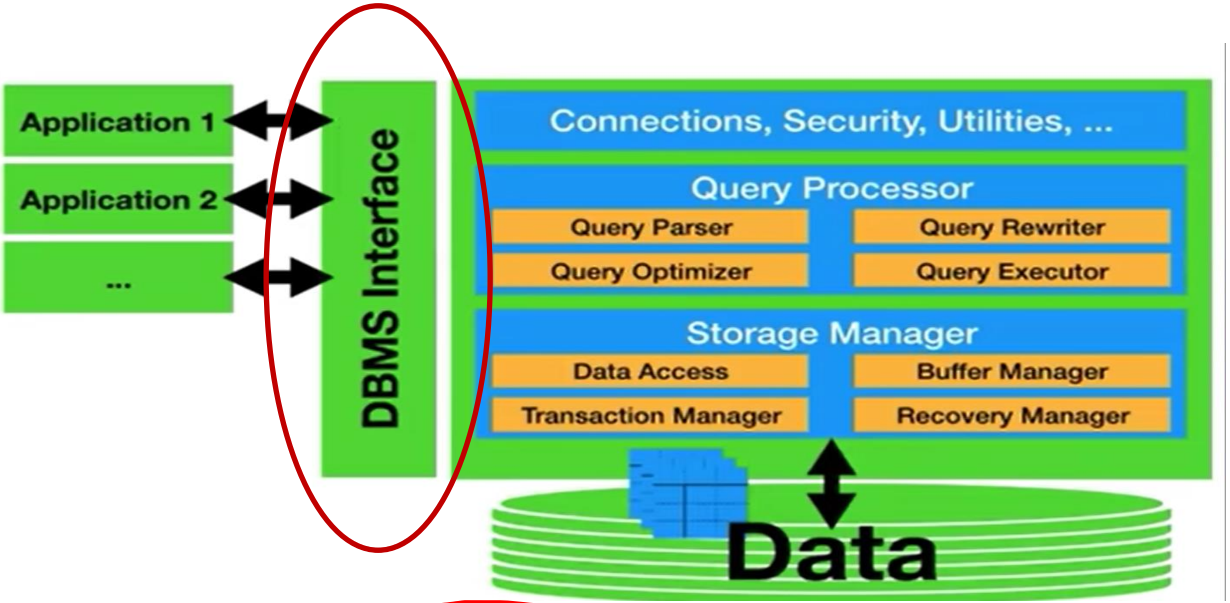


```
select instructor.name
from instructor
where instructor.dept_name = 'History';
```

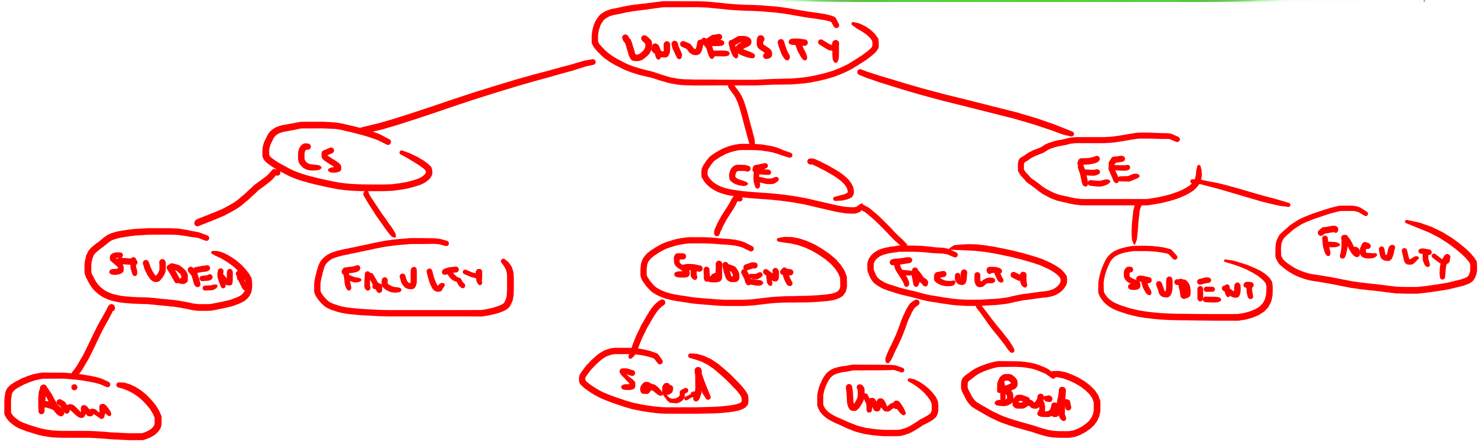


```
select instructor.ID, department.dept_name
from instructor, department
where instructor.dept_name= department.dept_name and
        department.budget > 95000;
```

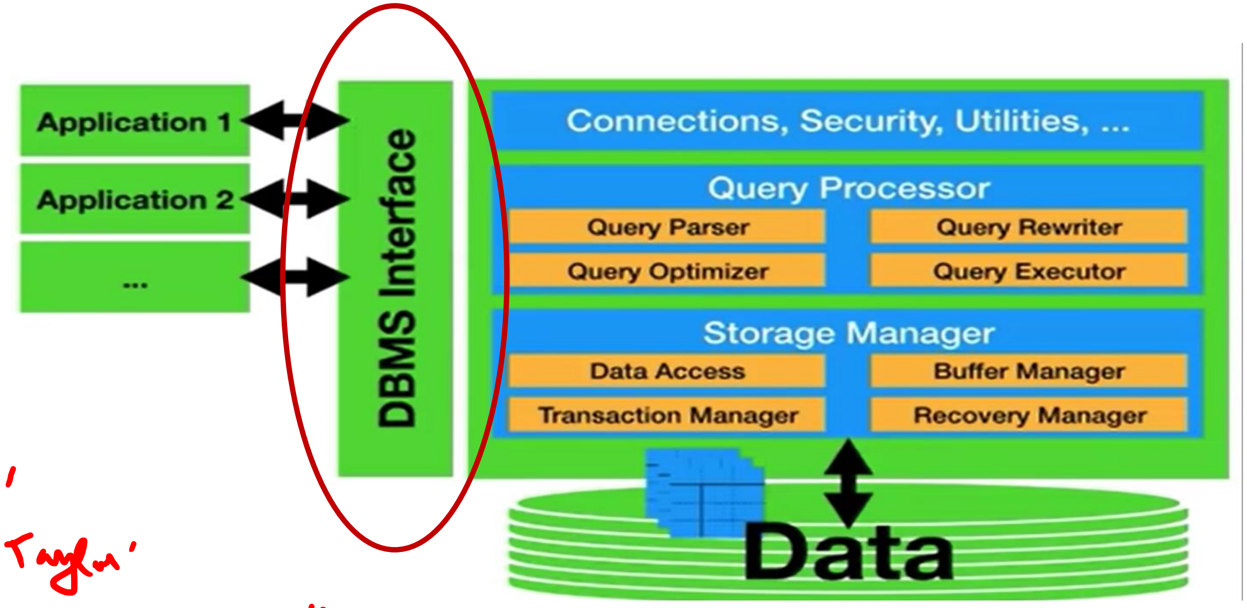
What should be the DBMS Interface?



GRAPHS:-



What should be the DBMS Interface?



"DOCUMENT-BASED"

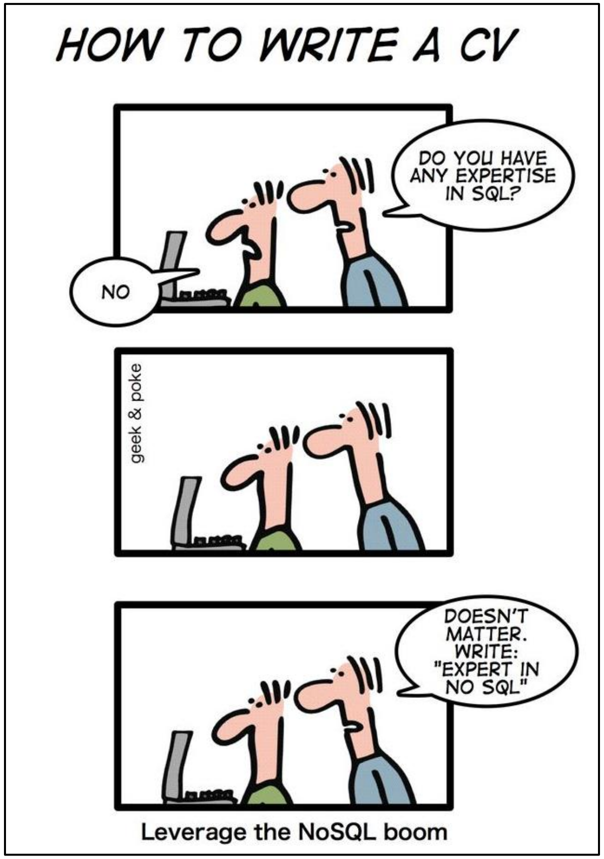
{ "ID" = 1
"Name" : 'CS'
"Building" = 'Taylor'
}

{ "ID = 2
" Name" = Physic
" Building" = "Watson"
}
: : :

{ ID = 1
'Name': Einstein'
"Dept.Name: Physics
}

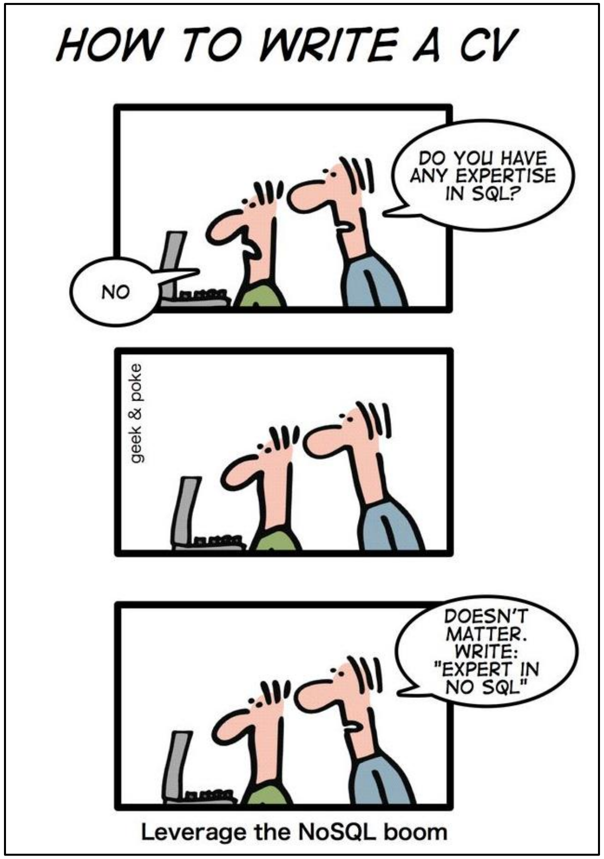
{ ID = 2
' Name: 'Wu'
Dept. Name: Finance
}

What are “NoSQL” Databases?


HOW TO WRITE A CV
DO YOU HAVE ANY EXPERTISE IN SQL?
NO
geek & poke
DOESN'T MATTER. WRITE: "EXPERT IN NO SQL"
Leverage the NoSQL boom

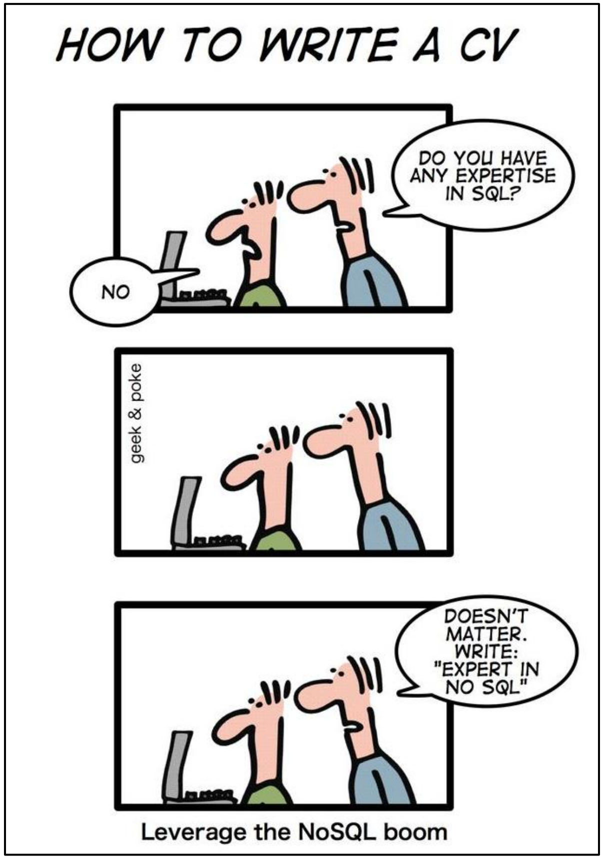
What are “NoSQL” Databases?

- An umbrella term for all databases that do not follow "Relational DBMS" principles.


HOW TO WRITE A CV
DO YOU HAVE ANY EXPERTISE IN SQL?
NO
geek & poke
DOESN'T MATTER. WRITE: "EXPERT IN NO SQL"
Leverage the NoSQL boom

What are "NoSQL" Databases?

- An umbrella term for all databases that do not follow Relational DBMS principles.
- Four major categories
  - Document based NoSQL Systems,
  - NoSQL Systems based on Key-Value Stores,
  - Column based or Wide Column NoSQL Systems,
  - Graph based NoSQL Systems.


HOW TO WRITE A CV
DO YOU HAVE ANY EXPERTISE IN SQL?
NO
geek & poke
DOESN'T MATTER. WRITE: "EXPERT IN NO SQL"
Leverage the NoSQL boom

NoSQL Databases: Document-Based

- These systems store data in the form of documents using well-known formats, such as JSON (JavaScript Object Notation).

- Examples
  - "MongoDB"
  - CouchDB

MongoDB

CouchDB
relax

NoSQL Databases: Key-Value Stores

- These systems have a simple data model based on fast access by the key to the value associated with the key.

- The value can be a record or an object or a document or even have a more complex data structure.

- Examples
  - Amazon Dynamo DB
  - Oracle NOSQL Database

amazon DynamoDB

ORACLE

NOSQL DATABASE

NoSQL Databases: Column-Based (or Wide Column)

- These systems partition a table by column into 'column families"
- Each column family is stored in its own files.
- Examples
  - Google Cloud Bigtable

NoSQL Databases: Graph-Based

- In these systems, data is represented as a graph, which is a collection of vertices (nodes) and edges.

- Both nodes and edges can be labeled to indicate the types of entities and relationships they represent,

- It is possible to store data associated with both individual nodes and individual edges.

- Examples
  - Neo4j
  - Graphbase.ai

graphbase.ai

The Graph DBMS
for Knowledge Graphs

NoSQL Databases: More than Just a Different DBMS Interface

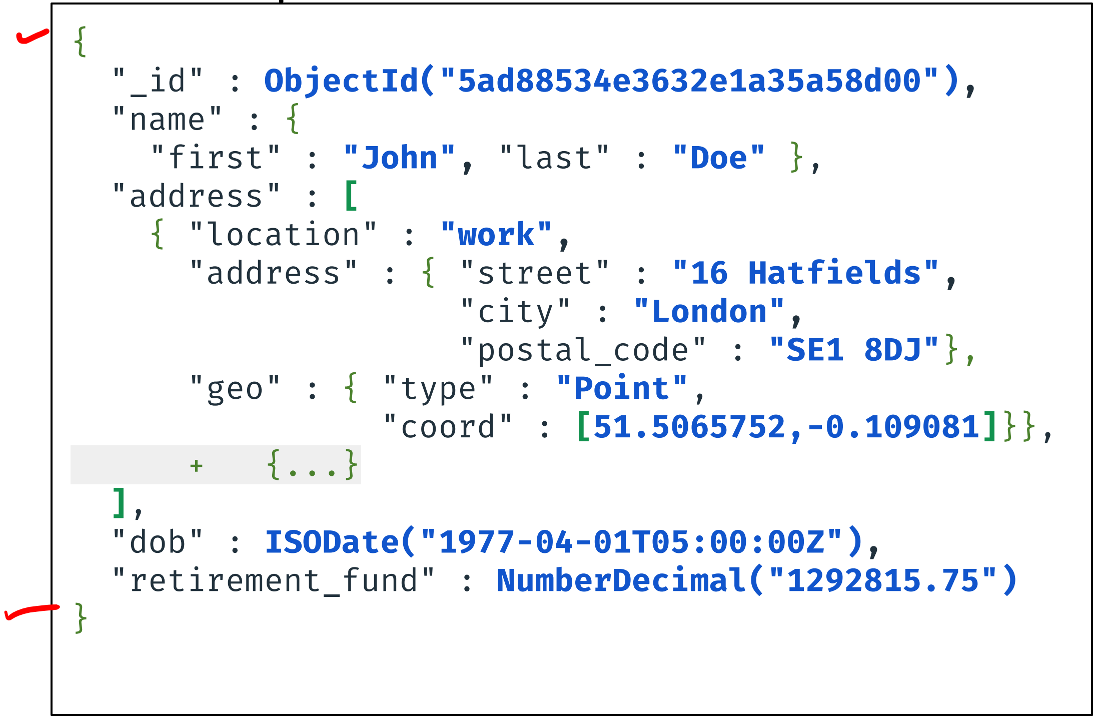
- In addition to a different DBMS interface, NoSQL databases have also tried to address some other shortcomings of the traditional relational DBMS:
  - 1) NoSQL databases aim to provide better support for horizontal scaling.
  - 2) With NoSQL databases, there is less of a mismatch between objects in a programming language and the constructs in a database (such as tables and documents).
  - 3) Many applications that use NoSQL systems require "continuous system availability". To accomplish this, data is replicated over nodes in a transparent manner, so that if one node fails, the data is still available on other nodes. Replication improves data availability and can also improve read performance. However, write performance becomes more cumbersome because an update must be applied to every copy of the replicated data items; this can slow down write performance if consistency is required.
  - 4) In many NoSQL applications, it is necessary to find individual records or objects (data items) from among the "millions of data records" or objects in a file. NoSQL databases aim to support this functionality

Document-based Data Model

- A way to organize and store data as a collection of documents where each document consists of a set of field-value pairs

- Similar to
  - Dictionaries in Python
  - Maps in Java
  - JSON Object in JavaScript

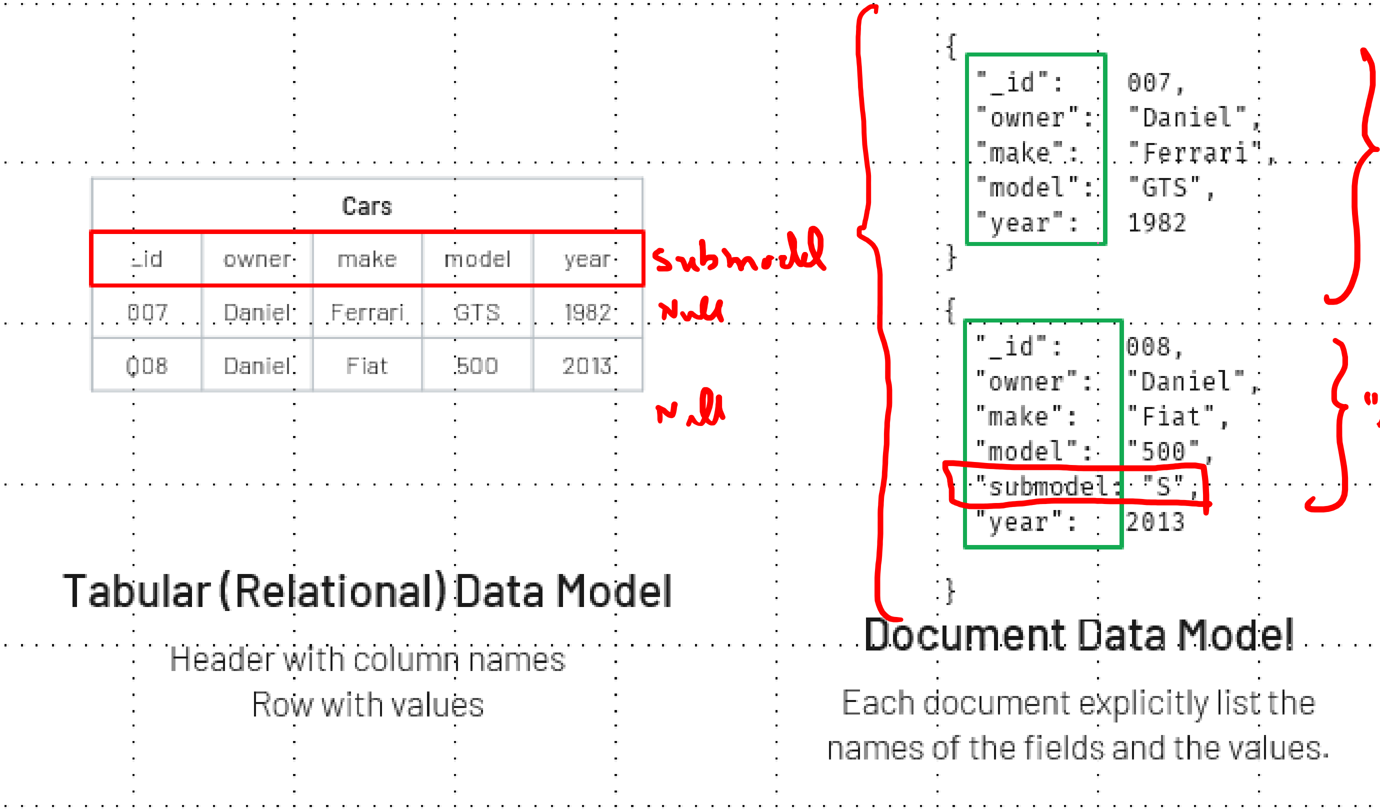
- Document constructs
  - Fields (Attributes)
  - Values
  - Sub-documents (Objects)
  - Arrays



```
{
  "_id" : ObjectId("5ad88534e3632e1a35a58d00"),
  "name" : {
    "first" : "John", "last" : "Doe" },
  "address" : [
    { "location" : "work",
      "address" : { "street" : "16 Hatfields",
                    "city" : "London",
                    "postal_code" : "SE1 8DJ"},
      "geo" : { "type" : "Point",
                "coord" : [51.5065752,-0.109081]}},
      +    {...}
  ],
  "dob" : ISODate("1977-04-01T05:00:00Z"),
  "retirement_fund" : NumberDecimal("1292815.75")
}
```

Document-based Data Model

"COLLECTION" named "Cars"



Cars				
_id	owner	make	model	year
007	Daniel	Ferrari	GTS	1982
008	Daniel	Fiat	500	2013

submodel

Null

Null

```
{
  "_id":      007,
  "owner":    "Daniel",
  "make":     "Ferrari",
  "model":    "GTS",
  "year":     1982
}

{
  "_id":      008,
  "owner":    "Daniel",
  "make":     "Fiat",
  "model":    "500",
  "submodel:  "S",
  "year":     2013

}
```

"Document"

"Document"

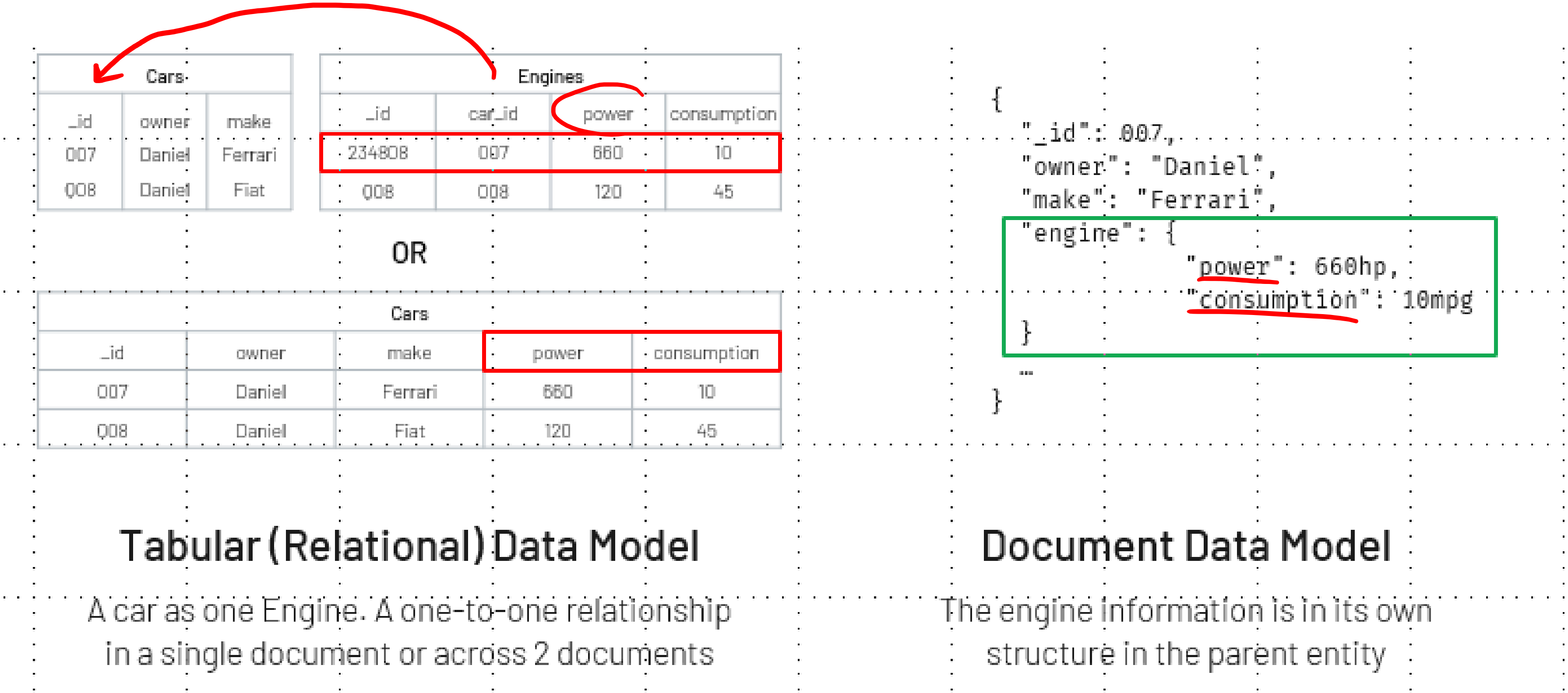
Tabular (Relational) Data Model

Header with column names
Row with values

Document Data Model

Each document explicitly list the names of the fields and the values.

Document-based Data Model



Cars		
_id	owner	make
007	Daniel	Ferrari
008	Daniel	Fiat

Engines			
_id	car_id	power	consumption
234808	007	660	10
008	008	120	45

OR

Cars				
_id	owner	make	power	consumption
007	Daniel	Ferrari	660	10
008	Daniel	Fiat	120	45

```
{
  "_id": 007,
  "owner": "Daniel",
  "make": "Ferrari",
  "engine": {
            "power": 660hp,
            "consumption": 10mpg
  }
  ...
}
```

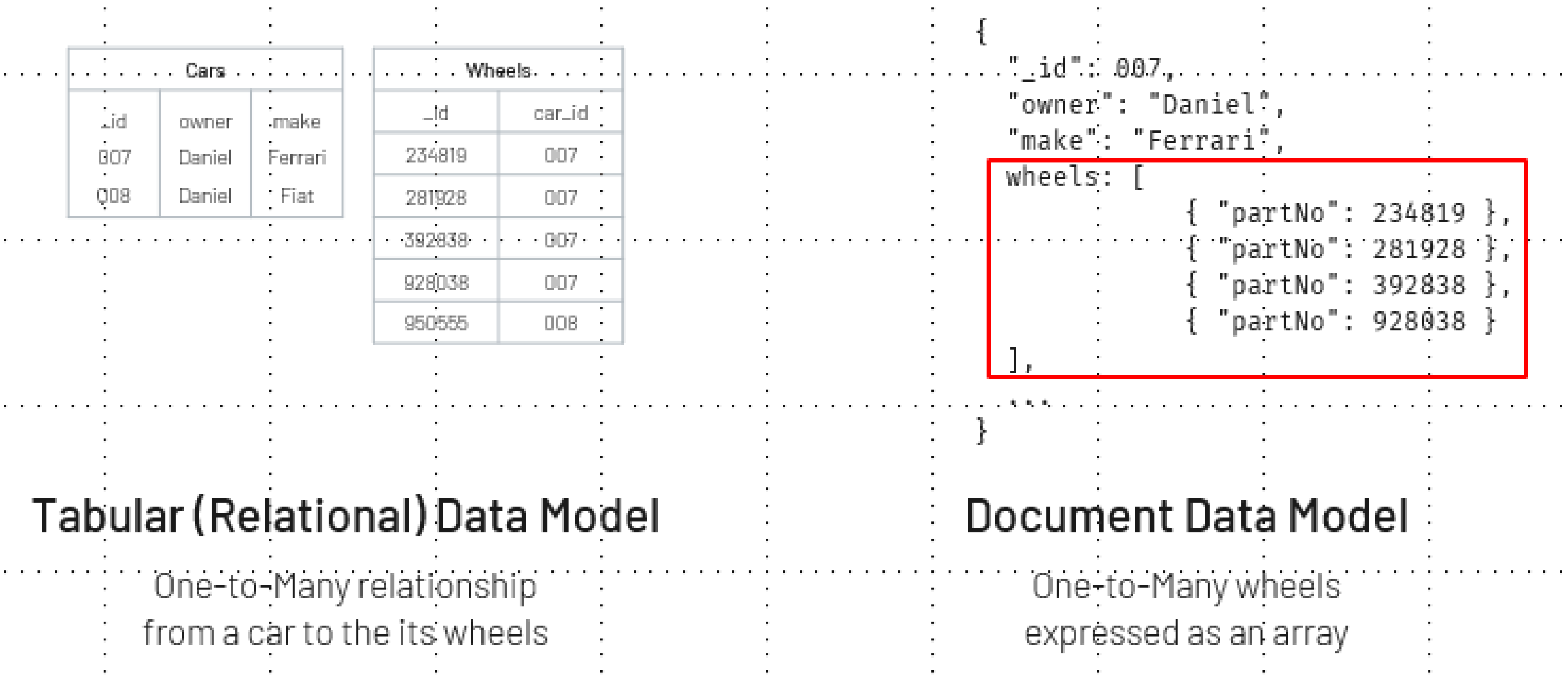
Tabular (Relational) Data Model

A car as one Engine. A one-to-one relationship in a single document or across 2 documents

Document Data Model

The engine information is in its own structure in the parent entity

Document-based Data Model



Cars

_id	owner	make
007	Daniel	Ferrari
008	Daniel	Fiat

Wheels

_id	car_id
234819	007
281928	007
392838	007
928038	007
950555	008

```
{
  "_id": 007,
  "owner": "Daniel",
  "make": "Ferrari",
  wheels: [
            { "partNo": 234819 },
            { "partNo": 281928 },
            { "partNo": 392838 },
            { "partNo": 928038 }
  ],
  ...
}
```

Tabular (Relational) Data Model

One-to-Many relationship
from a car to the its wheels

Document Data Model

One-to-Many wheels
expressed as an array

Document-based NoSQL Databases

- Document-based or document-oriented NOSQL systems typically store data as **collections** of similar **documents**.
  - These are also sometimes known as **document stores**.

- Individual **documents** are composed of ***<field>:<value>* pairs** with each document having a unique *"_id"* field.
  - Although the documents in a collection should be similar, they can have different data elements (attributes).
  - New documents can have new data elements that do not exist in any of the current documents in the collection.

MongoDB

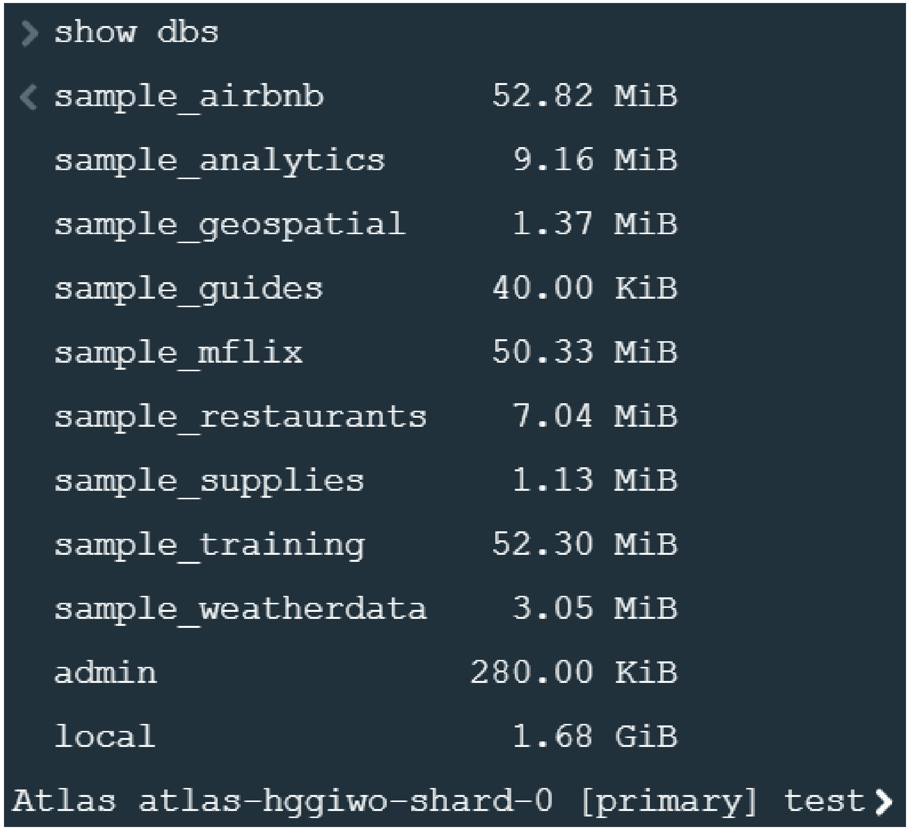
- MongoDB is a widely-used document-based NoSQL database system.
  - A database in MongoDB is composed of collections.
  - Each collection may consist of several documents.
  - A document in MongoDB is a data structure composed of field and value pairs, and has a unique “_id” field.
  - MongoDB documents are similar to JSON objects.
  - The values of fields may include other documents, arrays, and arrays of documents.

```
{
  "_id": 11,
  "user_id": “Saeed",
  "age": 29,
  "Status": "A"
}
```

```
{
  "_id": 12,
  "user_id": “Aamir",
  "age": 25,
  "Status": "A",
  "Country": “Pakistan"
}
```

MongoDB: Shell Commands

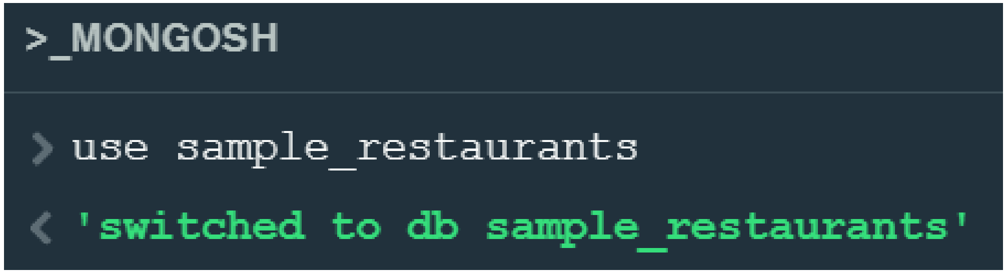
- The command `show dbs` is used to show all available databases.



```
> show dbs
< sample_airbnb        52.82 MiB
  sample_analytics      9.16 MiB
  sample_geospatial     1.37 MiB
  sample_guides        40.00 KiB
  sample_mflix         50.33 MiB
  sample_restaurants    7.04 MiB
  sample_supplies       1.13 MiB
  sample_training      52.30 MiB
  sample_weatherdata    3.05 MiB
  admin               280.00 KiB
  local                 1.68 GiB
Atlas atlas-hggiwo-shard-0 [primary] test>
```

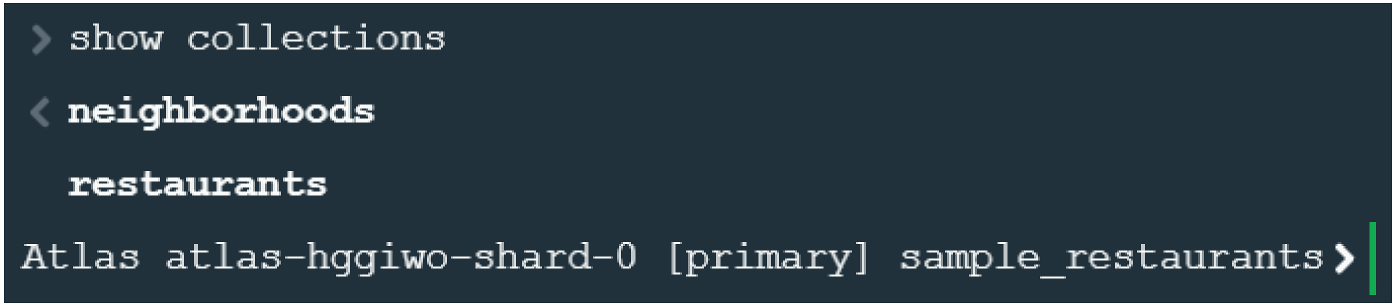
MongoDB: Shell Commands

- In order to use a database, use `use <database>` command.
  - If the database does not exist, it will be created.



```
>_MONGOSH

> use sample_restaurants
< 'switched to db sample_restaurants'
```

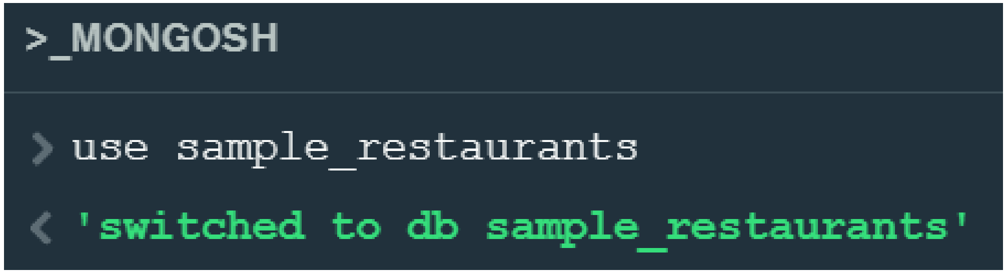
- To show collections in a database, use `show collections` command.



```
> show collections
< neighborhoods
  restaurants
Atlas atlas-hggiwo-shard-0 [primary] sample_restaurants>
```

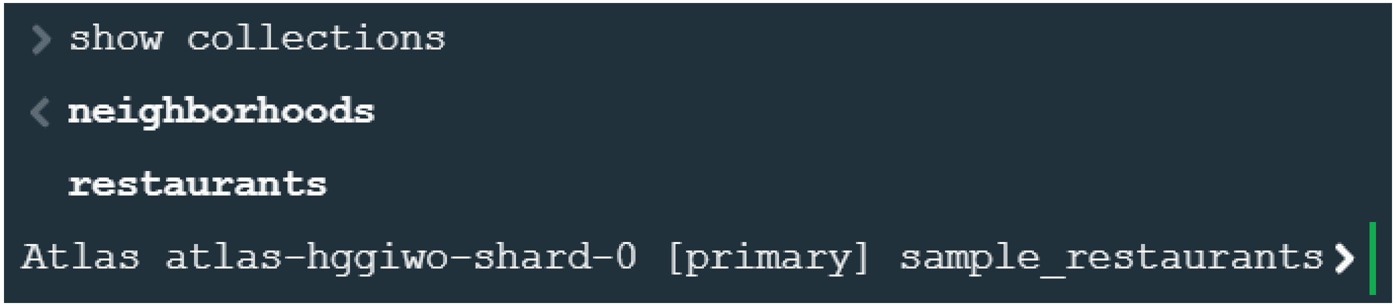
MongoDB: Shell Commands

- In order to use a database, use `use <database>` command.
  - If the database does not exist, it will be created.



```
>_MONGOSH

› use sample_restaurants
‹ 'switched to db sample_restaurants'
```

- To show collections in a database, use `show collections` command.



```
› show collections
‹ neighborhoods
  restaurants
Atlas atlas-hggiwo-shard-0 [primary] sample_restaurants›
```

MongoDB Query Language: Overview

- Designed for single-collection queries

- CRUD Operations
  - Create
  - Read
  - Update
  - Delete

MongoDB Query Language: Create

- Command to create a collection

```
db.createCollection(“cows”)
```

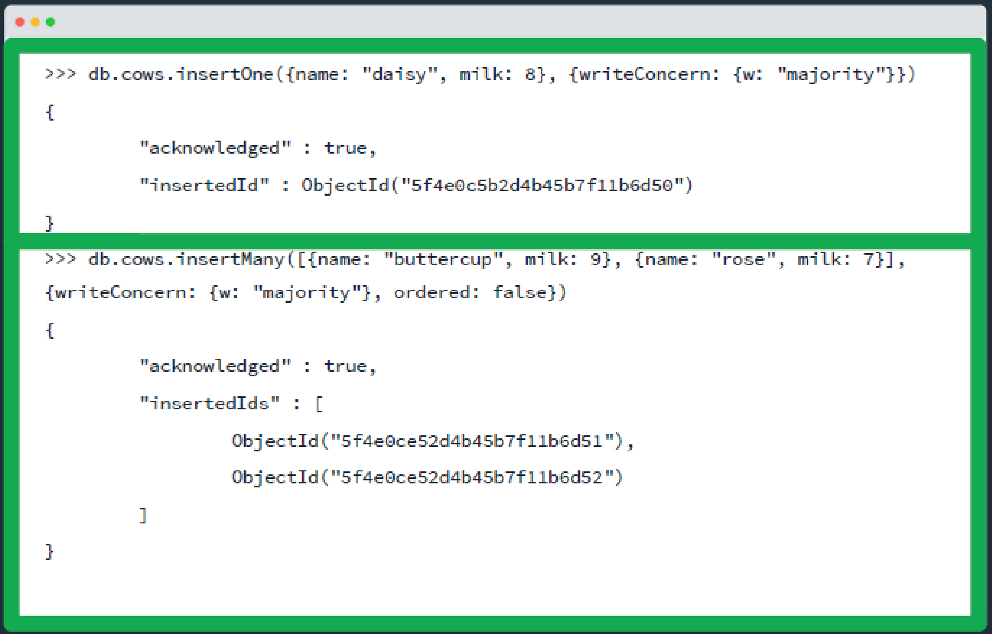
- Commands to insert documents into a collection

insertOne() Insert one document into a collection.

insertMany() Insert an array of documents into a collection.

writeConcern Sets the level of acknowledgment requested from MongoDB for write operations.

ordered For insertMany() there is an additional option for controlling whether the documents are inserted in ordered or unordered fashion.



```
>>> db.cows.insertOne({name: "daisy", milk: 8}, {writeConcern: {w: "majority"}})
{
        "acknowledged" : true,
        "insertedId" : ObjectId("5f4e0c5b2d4b45b7f11b6d50")
}
```

```
>>> db.cows.insertMany([{name: "buttercup", milk: 9}, {name: "rose", milk: 7}],
{writeConcern: {w: "majority"}, ordered: false})
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("5f4e0ce52d4b45b7f11b6d51"),
                ObjectId("5f4e0ce52d4b45b7f11b6d52")
        ]
}
```

SELECT
FROM
WHERE

MongoDB Query Language: Read

find() Selects documents and returns cursor.

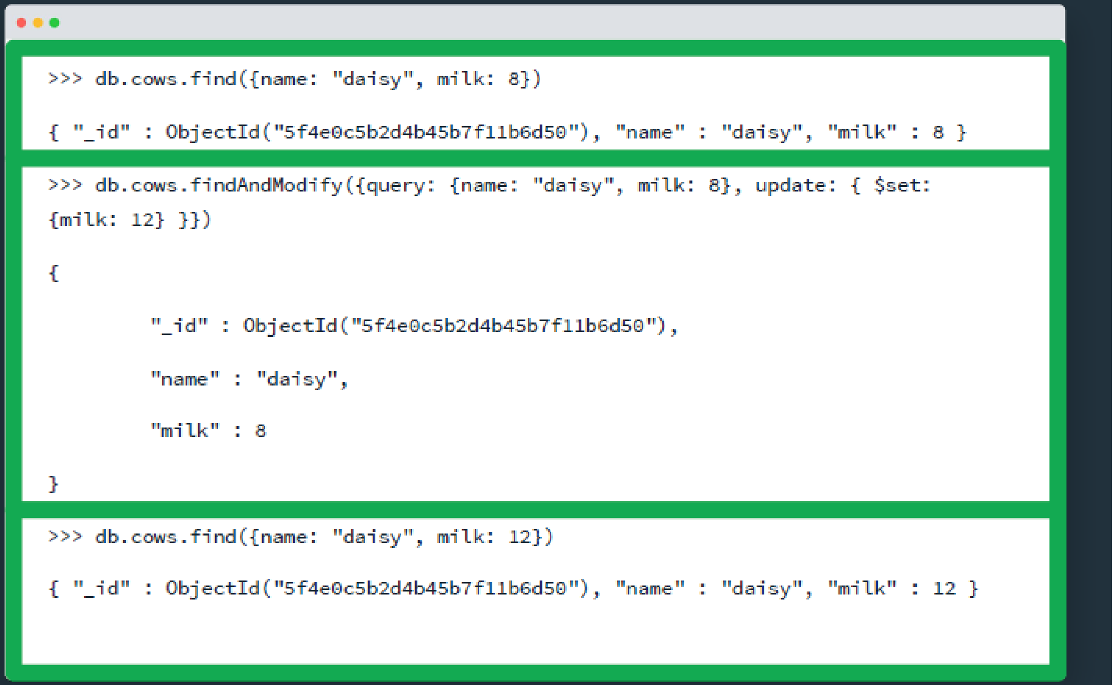
findOne() Returns first document that satisfies criteria.

findAndModify() Modifies and returns a single document.

findOneAndDelete() Deletes & returns the deleted document.

findOneAndUpdate() Updates a single document.

findOneAndReplace() Replaces a single document.



```
>>> db.cows.find({name: "daisy", milk: 8})

{ "_id" : ObjectId("5f4e0c5b2d4b45b7f11b6d50"), "name" : "daisy", "milk" : 8 }
```

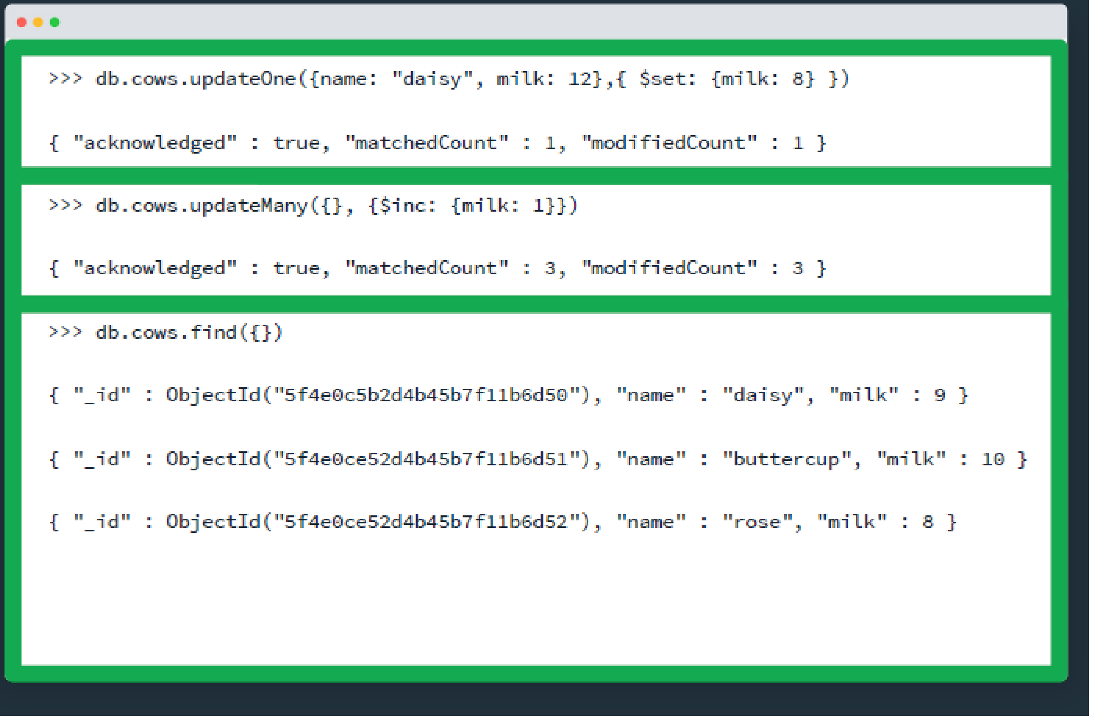
```
>>> db.cows.findAndModify({query: {name: "daisy", milk: 8}, update: { $set:
{milk: 12} }})

{

        "_id" : ObjectId("5f4e0c5b2d4b45b7f11b6d50"),

        "name" : "daisy",

        "milk" : 8

}
```

```
>>> db.cows.find({name: "daisy", milk: 12})

{ "_id" : ObjectId("5f4e0c5b2d4b45b7f11b6d50"), "name" : "daisy", "milk" : 12 }
```

MongoDB Query Language: Update

`updateOne()` Update one document into a collection.

`updateMany()` Update an array of documents into a collection.



```
>>> db.cows.updateOne({name: "daisy", milk: 12},{ $set: {milk: 8} })

{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
```

```
>>> db.cows.updateMany({}, {$inc: {milk: 1}})

{ "acknowledged" : true, "matchedCount" : 3, "modifiedCount" : 3 }
```

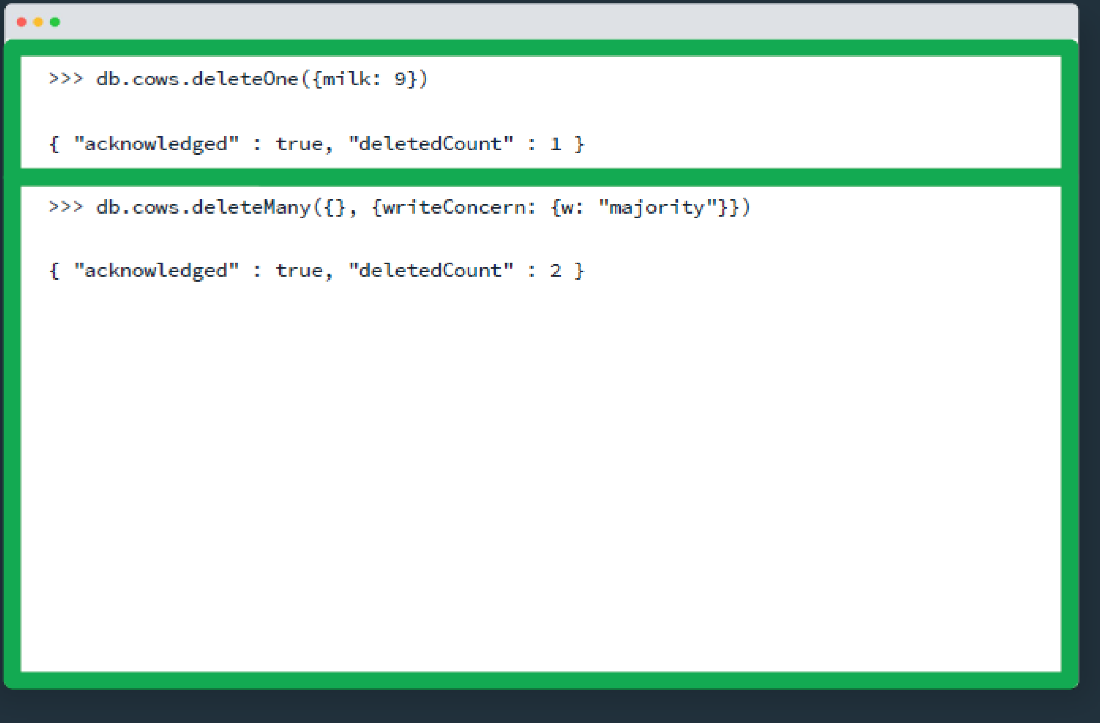
```
>>> db.cows.find({})

{ "_id" : ObjectId("5f4e0c5b2d4b45b7f11b6d50"), "name" : "daisy", "milk" : 9 }

{ "_id" : ObjectId("5f4e0ce52d4b45b7f11b6d51"), "name" : "buttercup", "milk" : 10 }

{ "_id" : ObjectId("5f4e0ce52d4b45b7f11b6d52"), "name" : "rose", "milk" : 8 }
```

MongoDB Query Language: Delete

`deleteOne()` Deletes one document from a collection.

`deleteMany()` Deletes many documents from a collection.

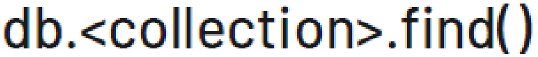
`writeConcern` Sets the level of acknowledgment requested from MongoDB for write operations.



```
>>> db.cows.deleteOne({milk: 9})

{ "acknowledged" : true, "deletedCount" : 1 }
```

```
>>> db.cows.deleteMany({}, {writeConcern: {w: "majority"}})

{ "acknowledged" : true, "deletedCount" : 2 }
```

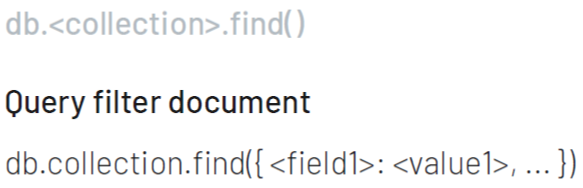
MongoDB Query Language: Focusing on Find()



```
db.<collection>.find()
```

- Find all documents in a collection

MongoDB Query Language: Focusing on Find()



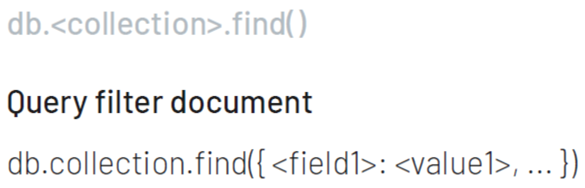
```
db.<collection>.find()
```

Query filter document

```
db.collection.find({ <field1>: <value1>, ... })
```

- Find documents in a collection that meet the search criteria.

MongoDB Query Language: Focusing on Find()



db.<collection>.find()

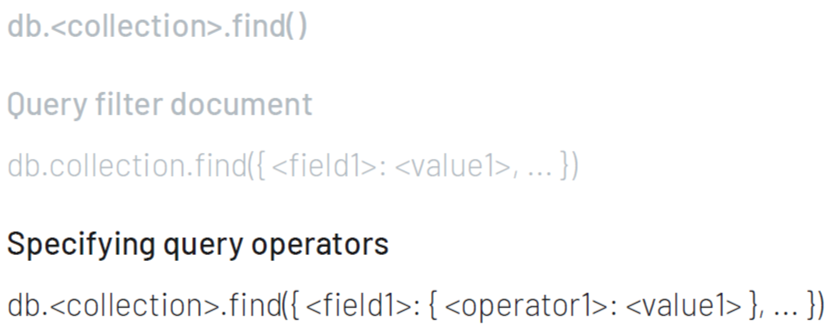
Query filter document

db.collection.find({ <field1>: <value1>, ... })

– Find documents in a collection that meet the search criteria.

```
    db.students.find({"LastName":"Baig"})

    db.students.find({"LastName":"Baig",
                      "Address.City":"Faisalabad"})
```

MongoDB Query Language: Focusing on Find()



db.<collection>.find( )

Query filter document

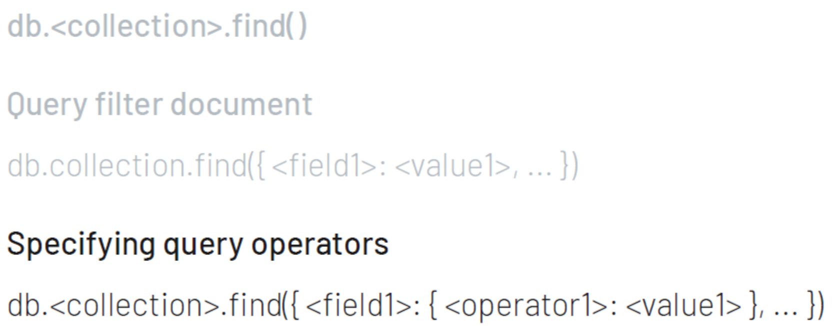
db.collection.find({ <field1>: <value1>, ... })

Specifying query operators

db.<collection>.find({ <field1>: { <operator1>: <value1> }, ... })

– Find documents in a collection that meet more complicated search criteria, specified through query operators

MongoDB Query Language: Focusing on Find()



db.<collection>.find()

Query filter document

db.collection.find({ <field1>: <value1>, ... })

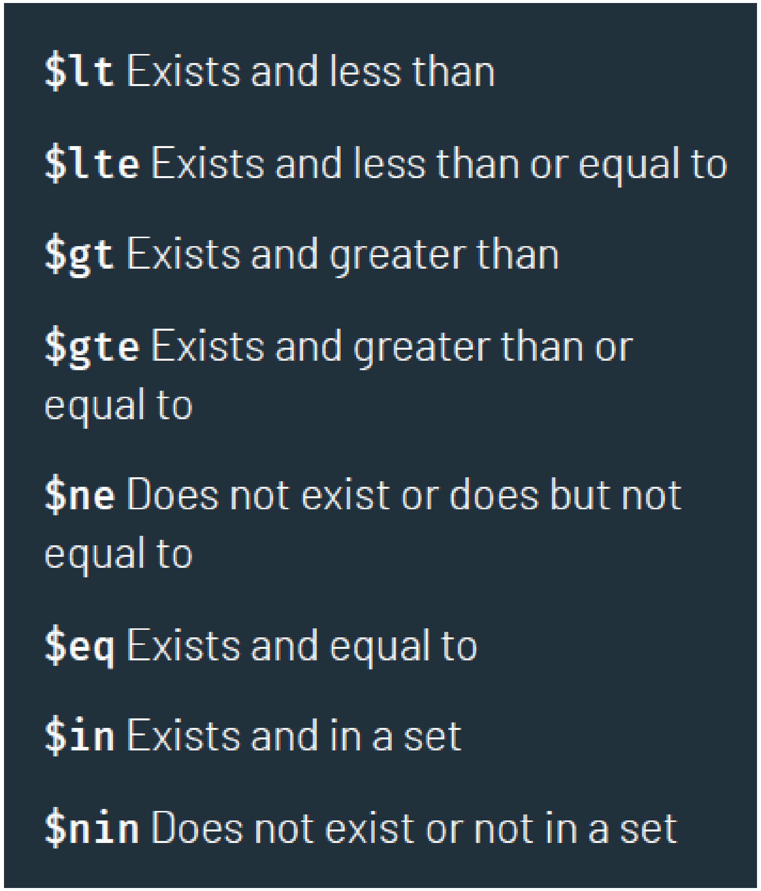
Specifying query operators

db.<collection>.find({ <field1>: { <operator1>: <value1> }, ... })

- Find documents in a collection that meet more complicated search criteria, specified through query operators

```
    db.students.find({"CGPA": {$lt:3.5}})

    db.students.find({"CGPA": {$gte:3.5}})
```

Query Operators in MongoDB Find(): Comparison



$lt Exists and less than

$lte Exists and less than or equal to

$gt Exists and greater than

$gte Exists and greater than or equal to

$ne Does not exist or does but not equal to

$eq Exists and equal to

$in Exists and in a set

$nin Does not exist or not in a set

Find all students with CGPA less than 3.5:

```
  db.students.find({"CGPA": {$lt:3.5}})
```

Find all students with CGPA of 3.5:

```
  db.students.find({"CGPA": {$eq:3.5}})
```

Find all students with CGPA other than 3.5:

```
  db.students.find({"CGPA": {$ne:3.5}})
```

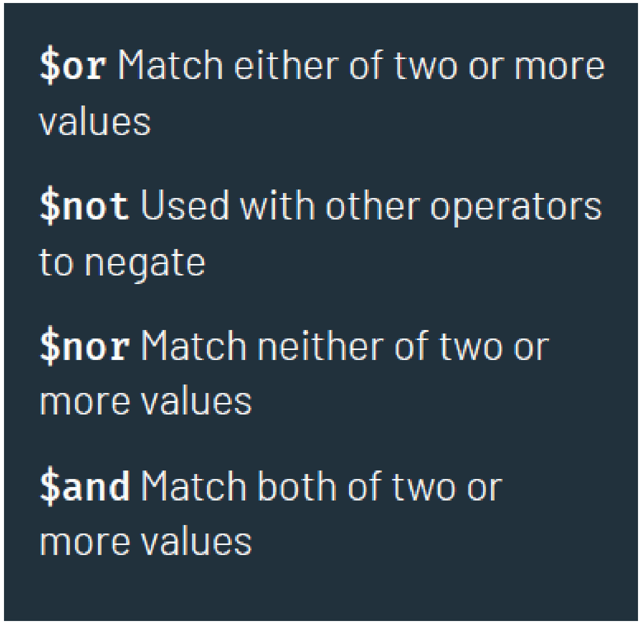
Find all students with CGPA of 2.9 or 3.2 or 3.5:

```
  db.students.find({"CGPA":{$in:[2.9,
     3.2,3.5]}})
```

Find all students with CGPA other than 2.9,3.2, or 3.5:

```
db.students.find({"CGPA":{$nin:[2.9, 3.2,
   3.5]}})
```

Query Operators in MongoDB Find(): Logical



$or Match either of two or more values

$not Used with other operators to negate

$nor Match neither of two or more values

$and Match both of two or more values

Find all students with CGPA between 3.0 and 3.5:

```
db.students.find({$and:[{"CGPA":
{$lt:3.5}}, {"CGPA": {$gt:3.0}}]})
```

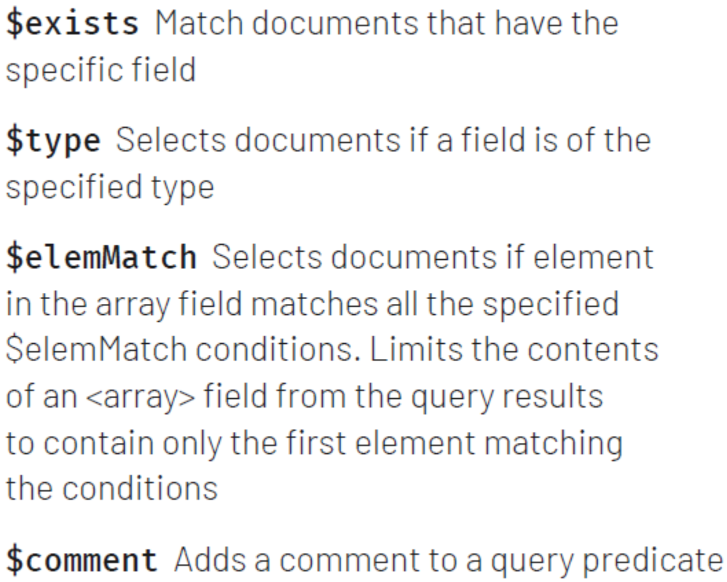
Find all students who are from Hunza or Faisalabad:

```
db.students.find({$or:[
  {"Address.City":{$eq:"Hunza"}},
  {"Address.City":{$eq:"Faisalabad"}}]})
```

Or

```
db.students.find({$or:[{"Address.City":
   "Hunza"},{"Address.City":"Faisalabad"}]})
```

Query Operators in MongoDB Find(): Various Categories



$exists Match documents that have the specific field

$type Selects documents if a field is of the specified type

$elemMatch Selects documents if element in the array field matches all the specified $elemMatch conditions. Limits the contents of an <array> field from the query results to contain only the first element matching the conditions

$comment Adds a comment to a query predicate

Find all students who do not have a Courses property/field:

```
   db.students.find({"Courses":{$exists:0 }})
```

Find all students who have taken the OOP course:

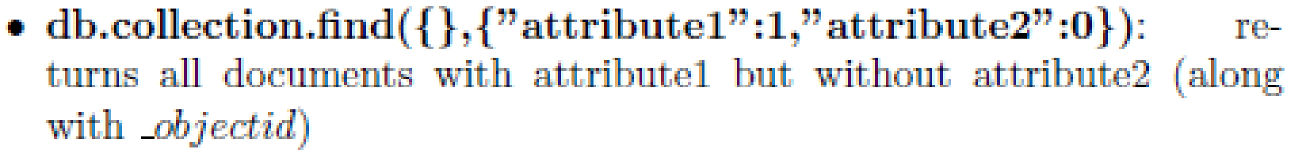
```
  db.students.find({"Courses":{$elemMatch:
     {"course_name":{$eq:"OOP"}}}})
```

Or

```
db.students.find({"Courses.course_name":"OOP"})
```

MongoDB Query Language: Focusing on Find()

- Projection



- **db.collection.find({},{”attribute1”:1,”attribute2”:0}):** returns all documents with attribute1 but without attribute2 (along with _*objectid*)

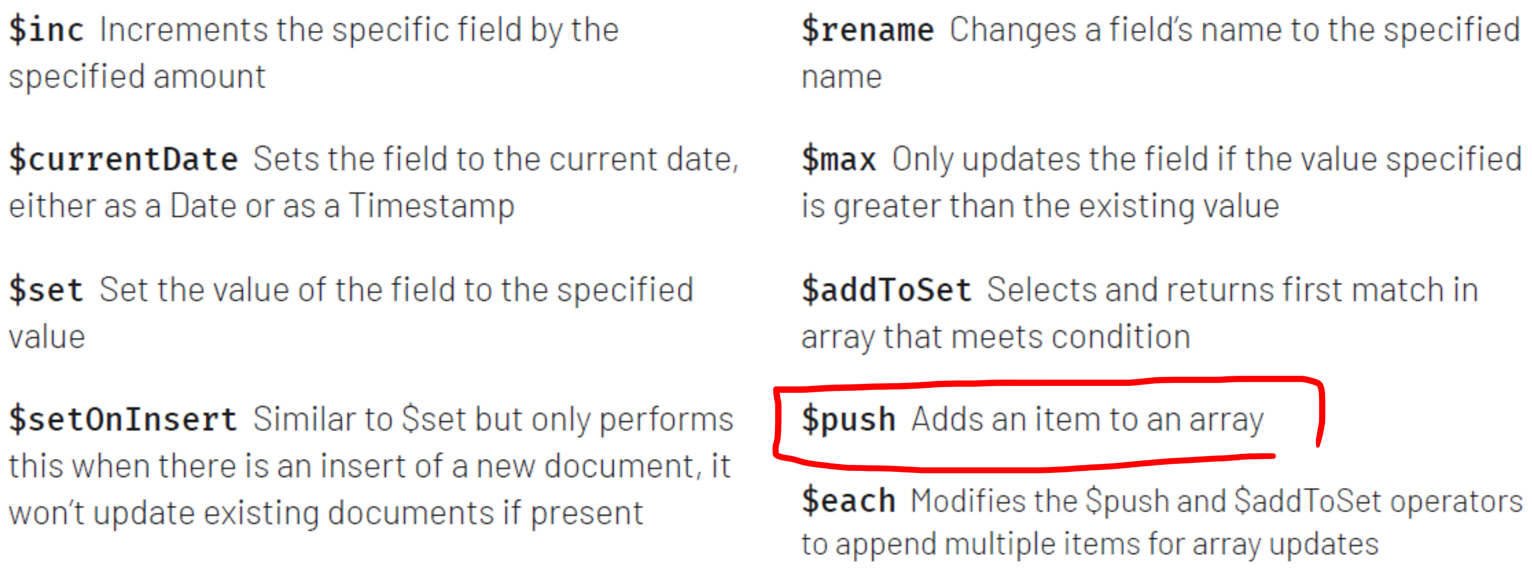
Display only the FirstName, LastName, and Major of all students:

```
   db.students.find({},{"FirstName":1,"LastName":1, "Major":1})
```

Display only the FirstName and LastName of students who are from Karachi:

```
  db.students.find({"Address.City":"Karachi"},{"FirstName":1,
                        "LastName":1})
```

Query Operators in MongoDB: Update Modifiers



$inc Increments the specific field by the specified amount

$currentDate Sets the field to the current date, either as a Date or as a Timestamp

$set Set the value of the field to the specified value

$setOnInsert Similar to $set but only performs this when there is an insert of a new document, it won't update existing documents if present

$rename Changes a field's name to the specified name

$max Only updates the field if the value specified is greater than the existing value

$addToSet Selects and returns first match in array that meets condition

$push Adds an item to an array

$each Modifies the $push and $addToSet operators to append multiple items for array updates

Set the CGPA of all CS students to 3.0:

```
    db.students.updateMany({"Major":"CS"},{$set: {"CGPA":3.0}})
```

Icrement the CGPA of all CS students by 0.1:

```
    db.students.updateMany({"Major":"CS"},{$inc: {"CGPA":0.1}})
```

MongoDB: Transactions

- Most MongoDB updates are atomic if they refer to a single document
- MongoDB also provides a pattern for specifying transactions on multiple documents.