BLG 317E
DATABASE SYSTEMS

WEEK 11

Relational Model

- relational model is not the best solution for all types of problems
  - storing user preferences
  - processing data from Wikipedia pages
  - building a social network

Example: User Preferences

- fields: user, preference type, selected option
- example task:
  retrieve notification setting of a given user
- no complex queries that would require SQL

Example: User Preferences

- fields: user, preference type, selected option
- example task:
  retrieve notification setting of a given user
- no complex queries that would require SQL

Example: Wikipedia Pages

Casino Royale (2006 film)

From Wikipedia, the free encyclopedia

This article is about the 2006 film. For the 1967 film, see Casino Royale (1967 film). For other uses, see Casino Royale (disambiguation).

Casino Royale (2006) is the twenty-first film in the Eon Productions *James Bond* film series and the first to star Daniel Craig as the fictional MI6 agent James Bond. Directed by Martin Campbell and written by Neal Purvis & Robert Wade and Paul Haggis, the film marks the third screen adaptation of Ian Fleming's 1953 novel of the same name. *Casino Royale* is set at the beginning of Bond's career as Agent 007, just as he is earning his licence to kill. After preventing a terrorist attack at Miami International Airport, Bond falls in love with Vesper Lynd, the treasury employee assigned to provide the money he needs to bankrupt a terrorist financier, Le Chiffre, by beating him in a high-stakes poker game. The story arc continues in the following *Bond* film, *Quantum of Solace* (2008), with explicit references to characters and events in *Spectre* (2015).

Casino Royale reboots the series, establishing a new timeline and narrative framework not meant to precede or succeed any previous *Bond* film,[3][4] which allows the film to show a less experienced and more vulnerable Bond.[5] Additionally, the character Miss Moneypenny is, for the first time in the series, completely absent.[6] Casting the film involved a widespread search for a new actor to portray James Bond, and significant controversy surrounded Craig when he was selected to succeed Pierce Brosnan in October 2005. Location filming took place in the Czech Republic, the Bahamas, Italy and the United Kingdom with interior sets built at Pinewood Studios. Although part of the storyline is set in

Casino Royale

British cinema poster for *Casino Royale*, designed by Empire Design

Directed by	Martin Campbell
Produced by	Michael G. Wilson Barbara Broccoli
Screenplay by	Neal Purvis Robert Wade Paul Haggis
Based on	*Casino Royale* by Ian Fleming
Starring	Daniel Craig Eva Green Mads Mikkelsen Giancarlo Giannini Jeffrey Wright Judi Dench

- combination of structured and unstructured data
- example task:
  retrieve first paragraph of all James Bond movies starring Daniel Craig
- difficult to represent as a relation

Example: Wikipedia Pages

Casino Royale (2006 film)

From Wikipedia, the free encyclopedia

This article is about the 2006 film. For the 1967 film, see Casino Royale (1967 film). For other uses, see Casino Royale (disambiguation).

Casino Royale (2006) is the twenty-first film in the Eon Productions *James Bond* film series and the first to star Daniel Craig as the fictional MI6 agent James Bond. Directed by Martin Campbell and written by Neal Purvis & Robert Wade and Paul Haggis, the film marks the third screen adaptation of Ian Fleming's 1953 novel of the same name. *Casino Royale* is set at the beginning of Bond's career as Agent 007, just as he is earning his licence to kill. After preventing a terrorist attack at Miami International Airport, Bond falls in love with Vesper Lynd, the treasury employee assigned to provide the money he needs to bankrupt a terrorist financier, Le Chiffre, by beating him in a high-stakes poker game. The story arc continues in the following *Bond* film, *Quantum of Solace* (2008), with explicit references to characters and events in *Spectre* (2015).

Casino Royale reboots the series, establishing a new timeline and narrative framework not meant to precede or succeed any previous *Bond* film,[3][4] which allows the film to show a less experienced and more vulnerable Bond.[5] Additionally, the character Miss Moneypenny is, for the first time in the series, completely absent.[6] Casting the film involved a widespread search for a new actor to portray James Bond, and significant controversy surrounded Craig when he was selected to succeed Pierce Brosnan in October 2005. Location filming took place in the Czech Republic, the Bahamas, Italy and the United Kingdom with interior sets built at Pinewood Studios. Although part of the storyline is set in

Casino Royale

British cinema poster for *Casino Royale*, designed by Empire Design

Directed by	Martin Campbell
Produced by	Michael G. Wilson Barbara Broccoli
Screenplay by	Neal Purvis Robert Wade Paul Haggis
Based on	*Casino Royale* by Ian Fleming
Starring	Daniel Craig Eva Green Mads Mikkelsen Giancarlo Giannini Jeffrey Wright Judi Dench

- combination of structured and unstructured data
- example task:
  retrieve first paragraph of all James Bond movies starring Daniel Craig
- difficult to represent as a relation

Example: Social Network

- users: userid, name, age, gender, . . .
- friends: userid1, userid2
- example tasks:
  find all friends of a given user
  find all friends of friends of a given user
  find all female friends of male friends of a given user
  find all friends of friends of ... friends of a given user
- too many complicated joins

Example: Social Network

- users: userid, name, age, gender, . . .
- friends: userid1, userid2
- example tasks:
  find all friends of a given user
  find all friends of friends of a given user
  find all female friends of male friends of a given user
  find all friends of friends of ... friends of a given user
- too many complicated joins

Problems: Representation

- difficult to handle unstructured and semistructured data
- difficult to represent hierarchy and neighborhood
- rigid schemas: all rows need to store all fields
- even if not applicable
- fixed in advance
- to make changes: shut down, alter table, restart

Problems: Representation

- difficult to handle unstructured and semistructured data
- difficult to represent hierarchy and neighborhood
- rigid schemas: all rows need to store all fields
  - even if not applicable
  - fixed in advance
  - to make changes: shut down, alter table, restart

Problems: Scaling

- when volume of data increases:
  - scalability?
  - faster processor?
    - works up to a point!
- scale out: more processors
- commodity hardware

NoSQL Databases

NoSQL Databases

- NoSQL ≠ "don't use SQL"
  - Not Only SQL
  - Use relational for some parts and non-relational for other parts
  - The term NoSQL is a general term used for all non-relational databases. (not the name of a specific DBMS).
- Can store structured, semi-structured, and unstructured data.
- Mainly developed for distributed databases and web applications.
- Used by many internet companies such as Google, Amazon, Facebook.

NoSQL Principles

- flexible schema
- focus on performance
- no join operations, nor transactions, and no normalizations
- massive scalability
- focus on availability
  - updates should always be allowed

NoSQL Principles

- flexible schema
- focus on performance
- no join operations, nor transactions, and no normalizations

- massive scalability
- focus on availability
  - updates should always be allowed

NoSQL Principles

- flexible schema
- focus on performance
- no join operations, nor transactions, and no normalizations
- massive scalability
- focus on availability
  - updates should always be allowed

NoSQL Principles

- flexible schema
- focus on performance
- no join operations, nor transactions, and no normalizations
- focus on availability
  - updates should always be allowed
- massive scalability

Scaling Methods of Database Servers

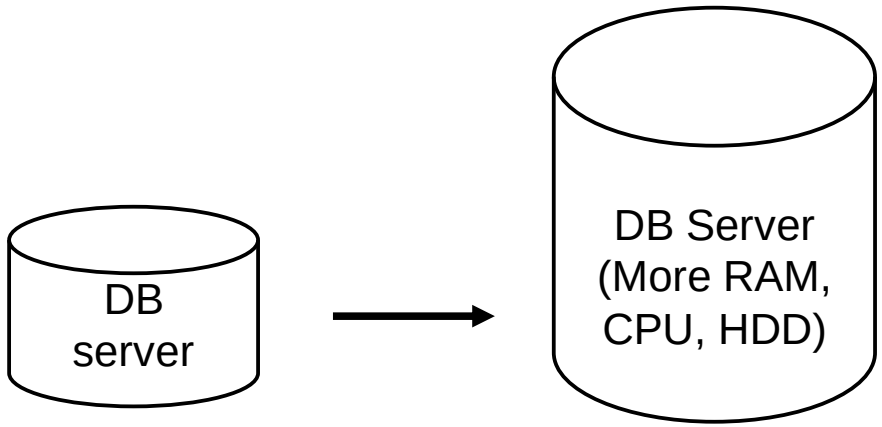
- **Scaling** means increasing/decreasing the hardware infrastructure.

- When the amount of data increases too much, a database server should scale to improve the query response time.

- There are two main scaling methods:
  - Scale-up (Vertical Scaling) method
  - Scale-out (Horizontal Scaling) method

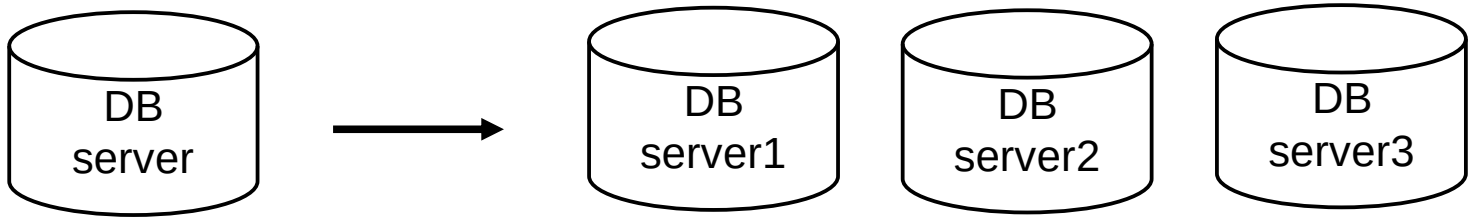
Scale-up (Vertical Scaling)

- An existing dedicated database server hardware is upgraded, by adding more RAM, more CPU, and more hard disks.
- This process is expensive, but used by most RDBMSs.



Scale-out (Horizontal Scaling)

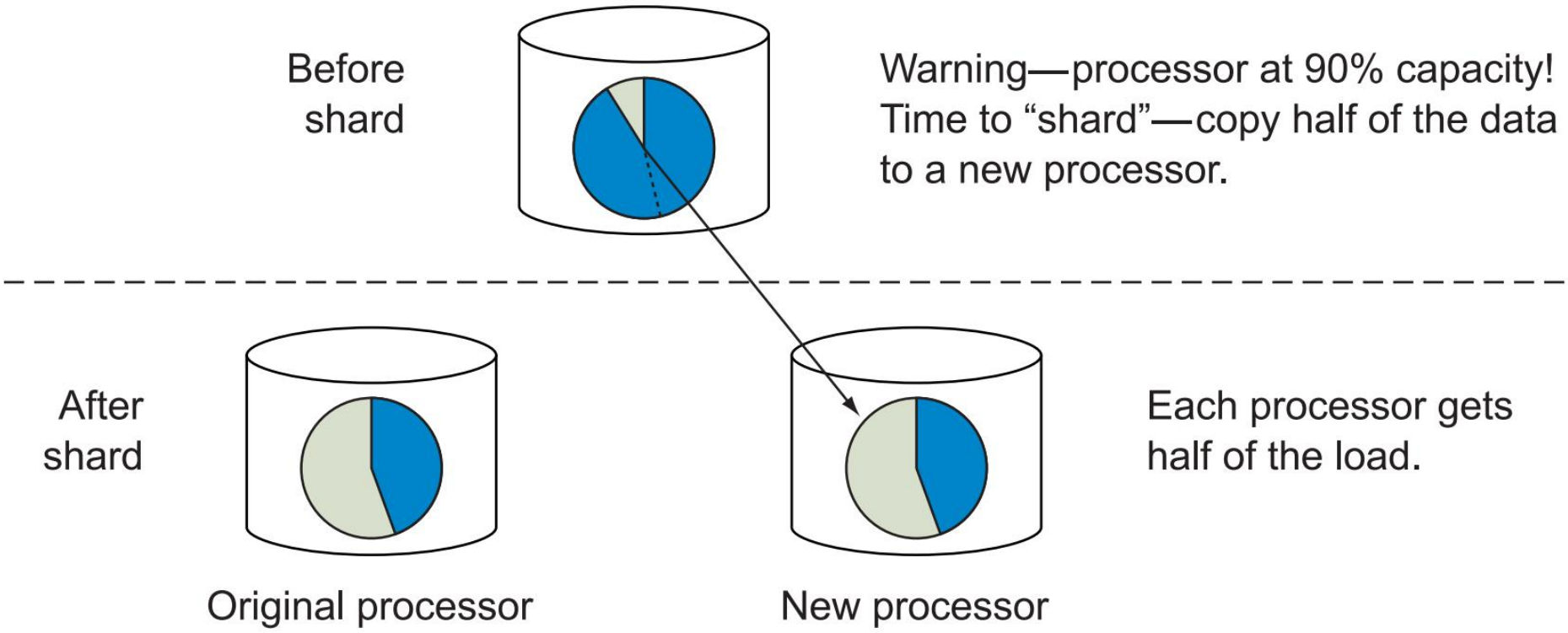
- Database load is distributed on multiple servers, whenever the load increases.
- Instead of an expensive dedicated database server, cheaper multiple servers are used in most NoSQL DBMSs.
- Commodity Hardware



Sharding

- when a server nears full capacity for data
- sharding: break data into chunks
- spread chunks across distributed servers
- increases efficiency
- more servers → more points of failure

Sharding


Before shard
Warning—processor at 90% capacity! Time to "shard"—copy half of the data to a new processor.
After shard
Each processor gets half of the load.
Original processor
New processor

Availability vs. Consistency

- focus on availability → relaxed consistency
- fewer transactional guarantees

- BASE instead of ACID:
  - Basic availability
  - Soft state
  - Eventual consistency

Replication

- replicate data between servers
- increases fault tolerance
- copies might diverge
  - eventual consistency: temporary inconsistency is allowed
  - when system stops, all copies will be the same

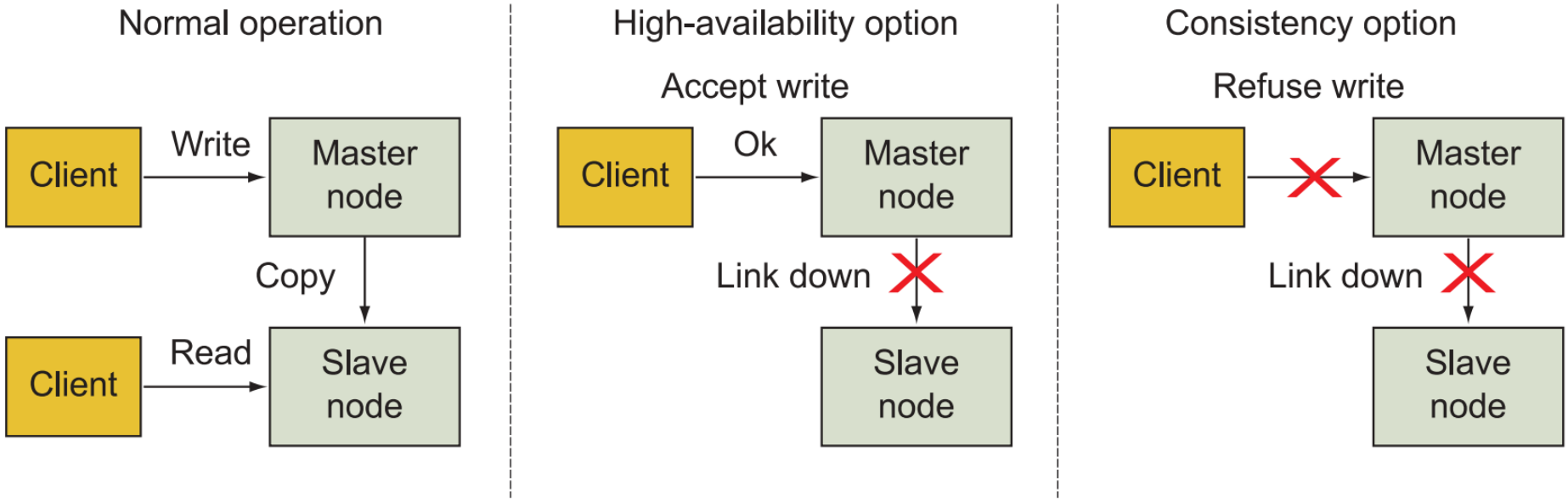
CAP Properties

- Consistency:
  all clients can read a single, up-to-date version of data
  from replicated partitions
- Availability:
  internal communication failures between replicated data
  don't prevent updates
- Partition tolerance:
  system keeps responding even if there is a communication failure
  (due to network partitioning) between nodes

CAP Theorem

- Any distributed database can provide at most two of the three CAP properties. (Eric Brewer - 2000)

CAP


Normal operation
Client
Write
Master node
Copy
Client
Read
Slave node
High-availability option
Accept write
Client
Ok
Master node
Link down
Slave node
Consistency option
Refuse write
Client
Master node
Link down
Slave node

Serialization

- in which format can an object be stored?
- simple solution: string
- serial format
- when writing: object → serial format (serialization)
- when reading: serial format → object (deserialization)

Serialization

- in which format can an object be stored?
- simple solution: string
- serial format
- when writing: object → serial format (serialization)
- when reading: serial format → object (deserialization)

Serialization Formats

- common formats: XML, JSON
- human-readable
- useful for data interchange
- useful for representing semistructured data

JSON Format (JavaScript Object Notation)

- Data is defined in Key-value pairs.
- A Key-value pair consists of a field name followed by a colon, then followed by a value.
  Example: **name: 'Will'**
- Objects: sets of key-value pairs.
- Data is separated by commas. Curly braces hold objects.
  Example: **{ name: 'Will' , city: 'NY' }**
- An array of documents is stored in brackets [ ].
  Example:

```
[
  {name: 'Will', city: 'NY', boss: 'Harold' },
  {name: 'Jeff', city: 'NJ', boss: 'Harold', hobbies: ['Swimming','Movies'] },
  {name: 'Harold', diploma: 'MBA', age: 70},
  {name: 'Matt', glasses: 'yes', office: '5th floor'}
]
```

JSON Example

- nested structure
- arrays of values

```
{
  "title": "The Usual Suspects",
  "year": 1995,
  "score": 8.7,
  "votes": 35027,
  "director": "Bryan Singer",
  "cast": [
    "Gabriel Byrne",
    "Benicio Del Toro"
  ]
}
```

JSON Example

- arrays of objects

```
[
 {
  "title": "The Usual Suspects",
  "year": 1995,
  ...
 },
 ...
 {
  "title": "Being John Malkovich",
  "year": 1999,
  ...
 }
]
```

Valid Documents

- JSON Schema
  - Describes your existing data format.
  - Clear, human- and machine-readable documentation.
  - Complete structural validation, useful for
    - automated testing.
    - validating client-submitted data.

Example JSON Schema

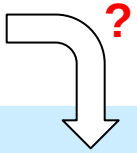
```
{
   "$schema": "http://json-schema.org/draft-04/schema#",
   "title": "Product",
   "description": "A product from Acme's catalog",
   "type": "object",

   "properties": {

      "id": {
         "description": "The unique identifier for a product",
         "type": "integer"
      },
      "name": {
         "description": "Name of the product",
         "type": "string"
      },

      "price": {
         "type": "number",
         "minimum": 0,
         "exclusiveMinimum": true
      }
   },

   "required": ["id", "name", "price"]
}
```




```
[
   {
      "id": 2,
      "name": "An ice sculpture",
      "price": 12.50,
   },

   {
      "id": 3,
      "name": "A blue mouse",
      "price": 25.50,
   }
]
```

Query Language

- no commonly used, declarative query language
- programmatic handling of data

Main Categories of NoSQL Databases

- The followings are main categories.
  - **Key-value Pair Based**
  - **Document-oriented**
  - **Column-oriented**
  - **Graph-based**

- Most of the NoSQL databases are multi-model DBMSs.
  - They support more than one DBMS type.
  - Example: CosmosDB can be used to store data in Key-value Based, Column-oriented, Document-oriented, or Graph-based formats.

Key-Value Stores

Key-Value Stores

- Example Products: Redis, Riak, Amazon DynamoDB, ArangoDB, Oracle BerkeleyDB, FoundationDB, MemcacheDB.
- Data is stored in key and value pairs.
- It is also called as a dictionary.
- Usually implemented as a hash table where value can be a JSON, BLOB (Binary Large Objects), string, etc.
- Example:

Key	Value
EmployeeName	Joe Walton
Phone	212-100-1234
Email	jwalton@com

Key-Value Stores

- indexed by keys
- keys are distinct
- very simple interface: put, get, delete
- no queries on values

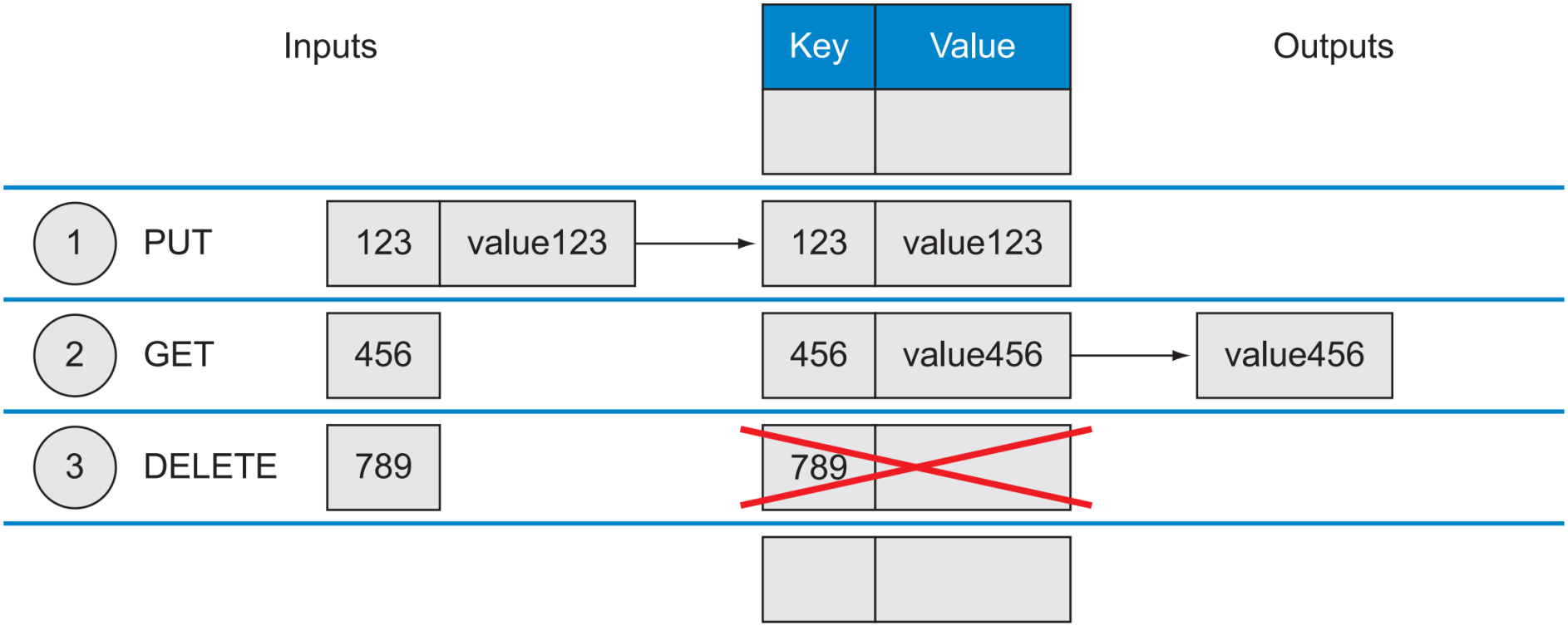
Key-Value Store Examples

- web page caching
- key: URL, value: web page
- image store
- key: path to image, value: image

Key-Value Store Examples

- web page caching
- key: URL, value: web page
- image store
- key: path to image, value: image

Key-Value Store



Key-Value Stores

- distribute records to computing nodes based on key
- advanced: data structures in value
- not just a blob of data

Document Stores

Document-oriented NoSQL

- Example Products: MongoDB, Apache CouchDB, Amazon DocumentDB, Azure DocumentDB, Lotus Notes, Microsoft Cosmos DB.
- It is similar to the Key-value pair DBMS.
- Stores and retrieves data as a Key-value pair, but the value part is stored as a document.
- The followings are data formats used in document-oriented DBMSs.
  - **JavaScript Object Notation (JSON)**
  - **Binary JSON (BSON)**
  - **Extensive Markup Language (XML)**
  - **Yet Another Markup Language (YAML)**
- Documents are easy to read, store, and parse using API libraries.
- Common use cases: content management systems (CMS) and catalogs. CMS lets users easily collect, store, and manage different types of content such as audio, image, etc.

MongoDB DBMS

- MongoDB is a Document-oriented NoSQL DBMS.

- The following editions are available:
  - MongoDB Community Server (free)
  - MongoDB Enterprise Server (commercial)
  - MongoDB Atlas (commercial cloud service)

- MongoDB has database drivers for many programming languages such as Python, Java, Javascript, C#, etc.

Features of MongoDB

- **High Performance** : Supports embedded data models to reduce I/O activity on database system. Indexes support fast queries.

- **High Availability** : Uses **Replication** (called replica sets) to provide automatic failover and data redundancy.
  A replica set is a group of MongoDB servers that maintain the same data set, providing redundancy and increasing data availability.
- MongoDB's storage engine is called as **WiredTiger**.
- A storage engine is a part of a database that is responsible for managing how data is stored and accessed, both in memory and on disk.
- WiredTiger features:
  - Document level locking.
  - Journal log (transaction log).
  - Compression for collections and indexes to minimize storage use.

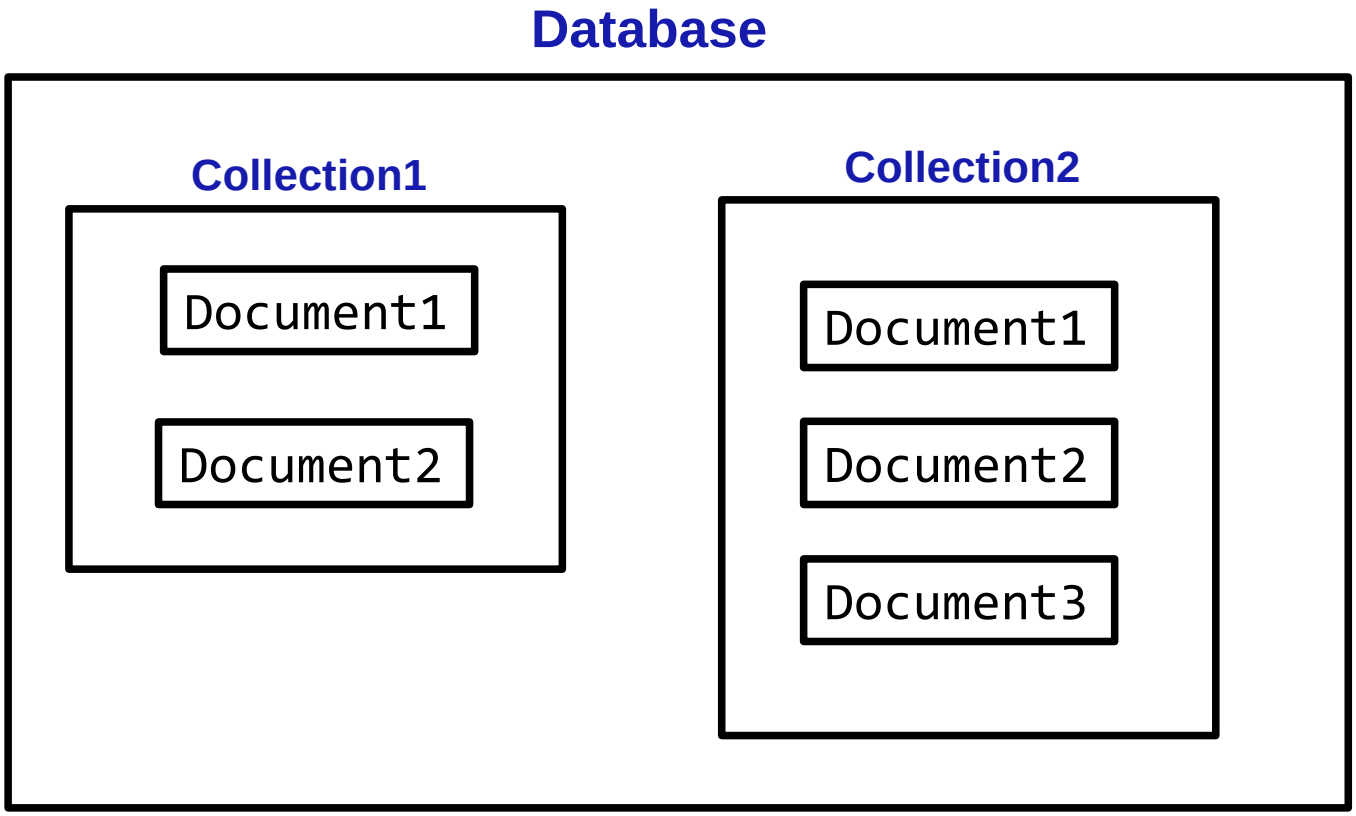
Features of MongoDB

- **Automatic Scaling** : Uses **Sharding** (partitioning) to provide horizontal scaling.
- Automatic sharding distributes data across a cluster of machines.
- Sharding is a database architecture that partitions data by key ranges and distributes the data among two or more database instances.

Comparison of SQL and MongoDB terms

SQL Terms	MongoDB Terms
Database	Database
Table	**Collection**
Row (Record, Tuple, Entity)	**Document** (Internal storage format is BSON) (Displayed format is JSON)
Column (Field, Attribute)	**Field (Key)**
Index	Index
Table Join operations	Embedded Documents
Primary Key	Primary Key (Default _id provided automatically as unique document identification)

MangoDB Database Organization



Collections in a Database

- The term **Collection** refers to a set of documents.
- A collection is equivalent to a **table** in RDBMS.
- Every document in a collection could have same or different data.
- MongoDB stores all documents in collections.
- A collection is a group of related documents that have a set of shared common indexes.

Documents in a Collection

- The term **Document** refers to a JSON object, not to a file in operating system.
- A document is equivalent to a **record** in RDBMS. (Also it is equivalent to a class object in object-oriented programming.)
- A document contains pairs of **Field-Value**.
- Fields are separated by commas, and enclosed by curly braces.

Document1

```
{ name: 'Will',
  city: 'NY',
  boss: 'Harold'
}
```

Pairs of field: value

Fields in a Document

- In the documents, the value of a field can be any of the JSON data types, including other documents, arrays, and arrays of documents.

- Field-Value pair is also called as Key-Value pair.

Document2

```
{ name: 'Jeff',
  city: 'NJ',
  boss: 'Harold',
  hobbies: ['Swimming','Movies']
}
```

The hobbies field is an array of strings. The brackets [ ] denote an array.

Example: EMPLOYEES Collection

EMPLOYEES Collection

Document1

```
{ name: 'Will',
  city: 'NY',
  boss: 'Harold'
}
```

Document2

```
{ name: 'Jeff',
  city: 'NJ',
  boss: 'Harold',
  hobbies: ['Swimming','Movies']
}
```

Document3

```
{ name: 'Harold',
  diploma: 'MBA',
  age: 70
}
```

Document4

```
{ name: 'Matt',
  glasses: 'yes'
  office: '5th floor'
}
```

Installing MongoDB

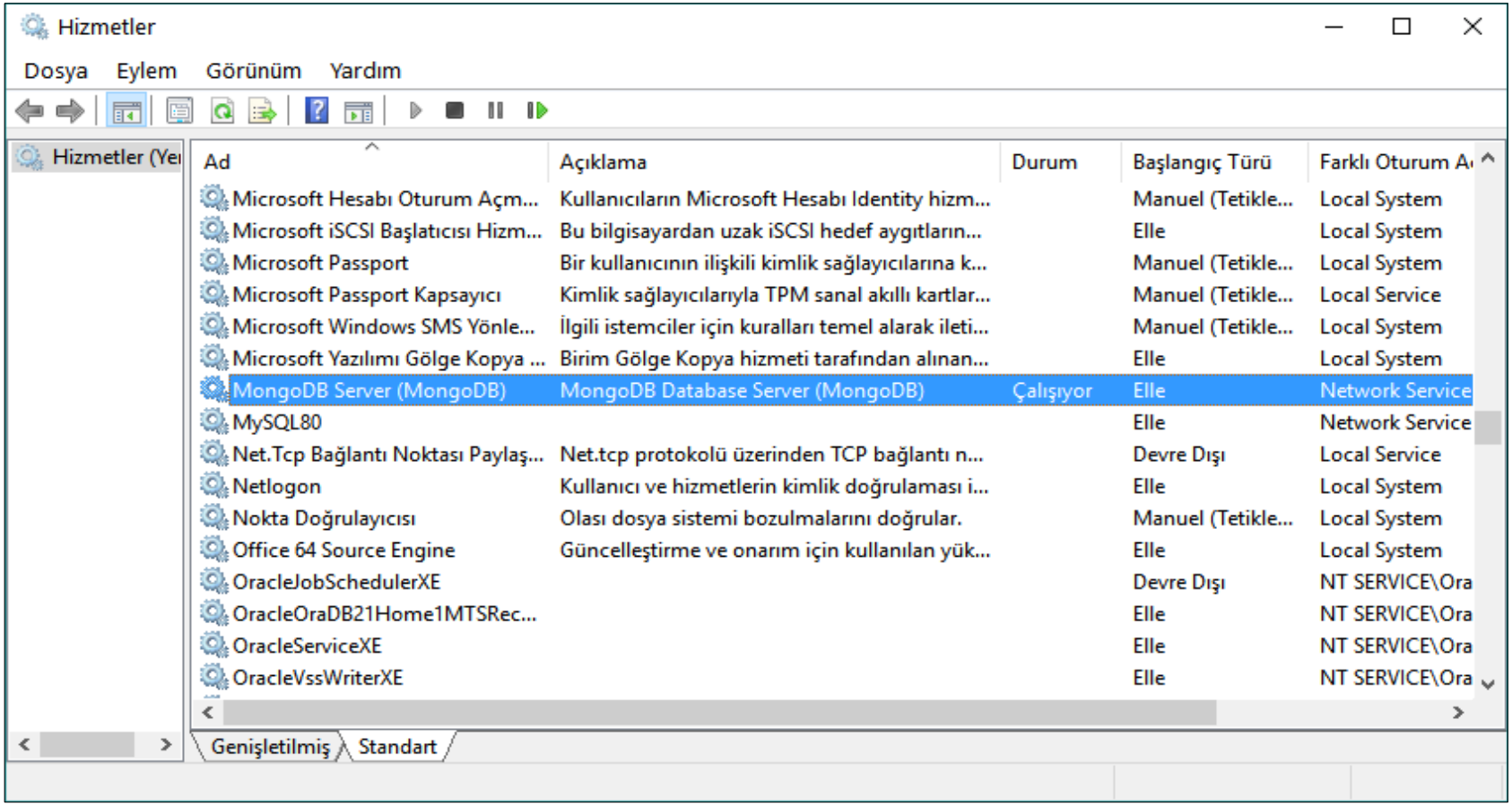
The followings can be downloaded from the MangoDB website.

Program	Description
MongoDB Community Server	Database server.
MongoDB Shell	• Command-line interface program. • Perform CRUD operations, query data, configure settings, and execute other actions. • Uses JavaScript as the query language.
MongoDB Compass	• GUI (Graphical User Interface) program. • Explore and manipulate (CRUD) the database. • Provides detailed schema visualizations, performance metrics, flexible querying.

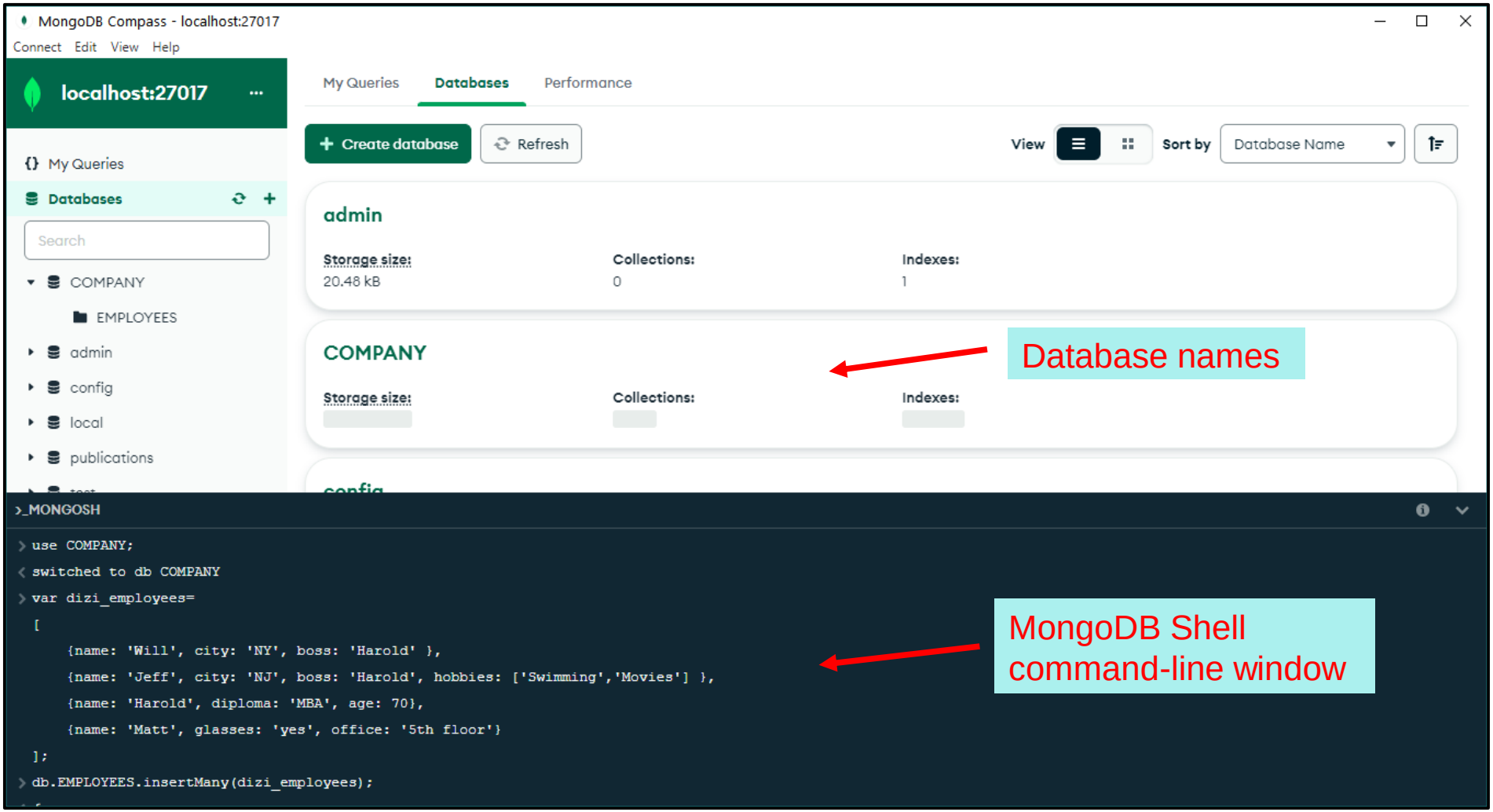
MongoDB Database Tools	• Tools are a collection of command-line utilities for working with a MongoDB deployment.
	  ➢ Data Import / Export (JSON, CSV formats).
	  ➢ Binary Import / Export (BSON format).
	  ➢ Diagnostic Tools.

Starting MongoDB Server from Windows Services screen

- Open the Windows Services screen and find the MangoDB line.
- Click the Start button.

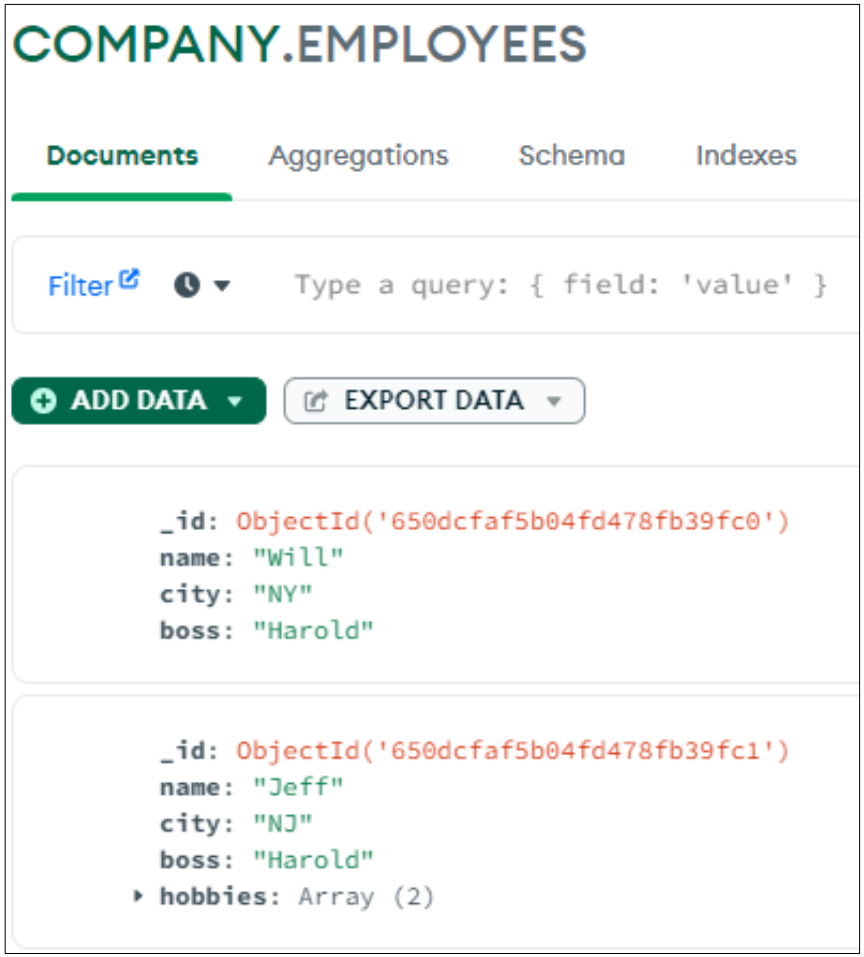


MongoDB Compass (GUI)



Example Data in Documents:
(COMPANY: Database name, EMPLOYEES: Collection name)

List mode



JSON mode

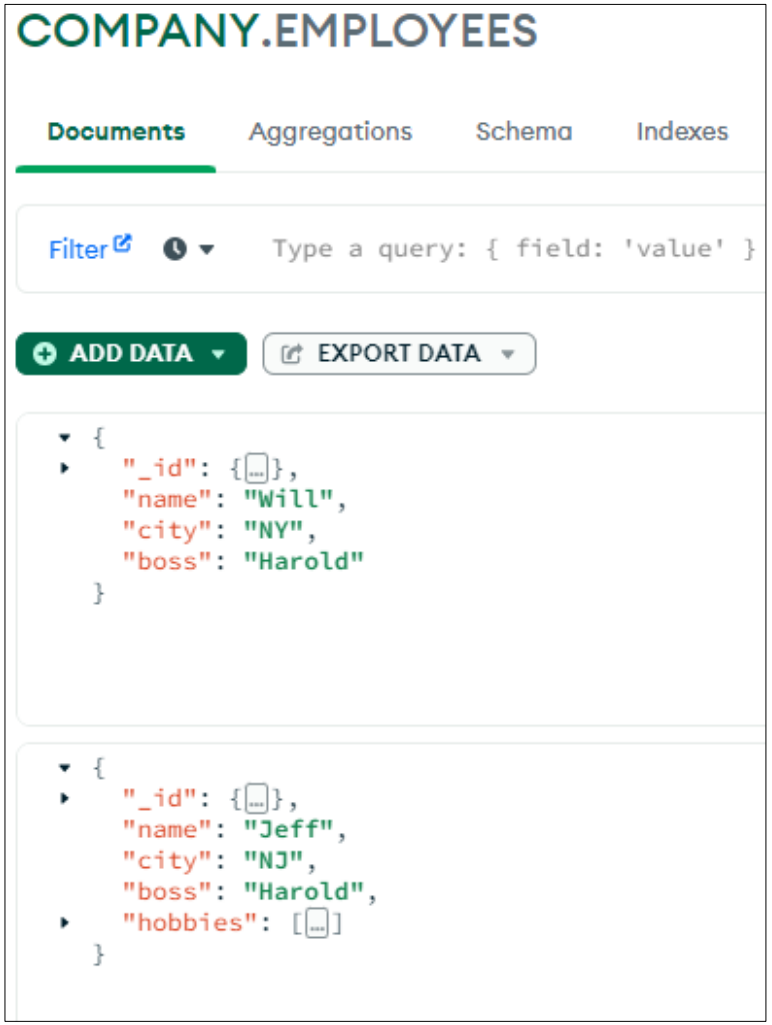
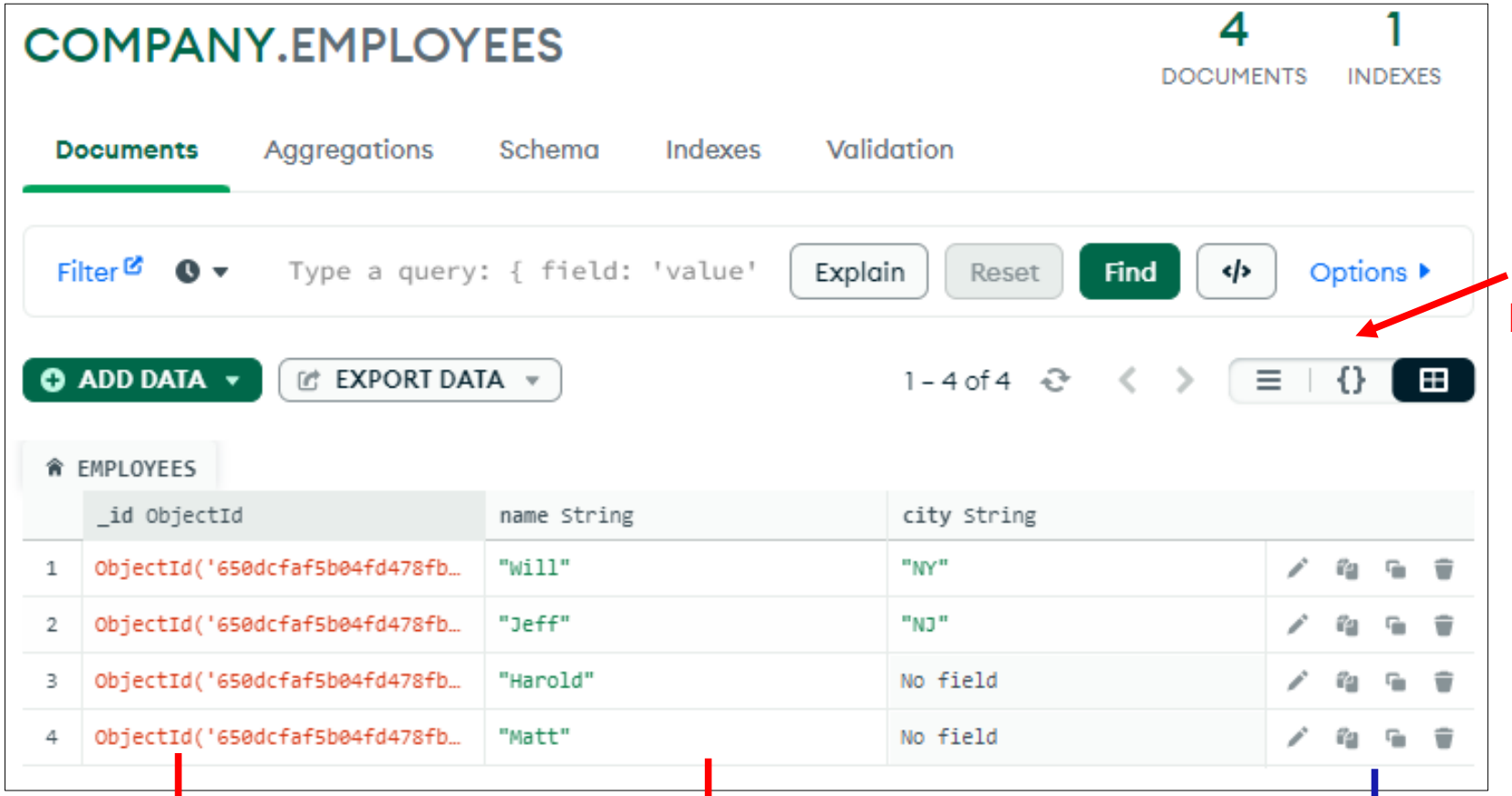


Table mode



Mode buttons

Unique Document Ids
(Assigned automatically)

Documents
(in each row)

Document buttons:
Edit, Copy, Clone, Delete

MongoDB Insert Example

```
itucsdb.movies.insert(
  {
    "title": "Ed Wood", "year": 1994,
    "score": 7.8,
    "votes": 6587, "director": "Tim Burton",
    "cast": [
      "Johnny Depp"
    ]
  }
)
```

MongoDB Insert Example

```
itucsdb.movies.insert(
  {
    "title": "Three Kings", "year": 1999,
    "score": 7.7,
    "votes": 10319, "cast": [
      "George Clooney", "Spike Jonze"
    ]
  }
)
```

MongoDB Find Example

```
itucsdb.movies.find()

itucsdb.movies.find(
  {"year": 1999}
)

itucsdb.movies.find(
  {"year": {$gt 1999}}
)
```

Example Python Application Program

- Install the MongoDB database driver: `pip install pymongo`

```
import pymongo

app = pymongo.MongoClient("mongodb://localhost:27017/")
db = app["test"]  #Database name
collection = db["PRODUCTS"]  #Collection name (cursor)

formatted_row = '{0:>4}  {1:>10}  {2:<20}  {3:>5}  {4:<15}'
    .format("ROW", "PRODUCT_ID", "NAME", "PRICE", "CATEGORY")
print(formatted_row)

counter = 1
for document in collection.find():
  #Get dictionary fields
  field1 = document["ProductID"]
  field2 = document["Name"]
  field3 = document["Price"]
  field4 = document["Category"]
  formatted_row = '{0:>4}  {1:>10}  {2:<20}  {3:>5}  {4:<15}'
    .format(counter, field1, field2, field3, field4)
  print(formatted_row)
  counter = counter + 1
```

Output Screen

ROW	PRODUCT_ID	NAME	PRICE	CATEGORY
1	5	Chang Syrup	19	Beverages
2	6	Chartreuse verte	18	Beverages
3	10	Cote de Blaye	264	Beverages
4	15	Guarana	5	Beverages
5	22	Ipoh Coffee	46	Beverages
6	4	Chais Coffee	18	Beverages
7	38	Sasquatch Ale	14	Beverages
8	45	Steeleye Stout	18	Beverages
9	1	Berry Spread	25	Condiments
10	3	Cajun Seasoning	22	Condiments
11	11	Cranberry Sauce	40	Condiments
12	14	Genen Shouyu	16	Condiments
13	16	Gula Coffee	19	Condiments
14	17	Gumbo Mix	21	Condiments
15	41	Seed Syrup	10	Condiments
16	7	Chocolade	13	Confections
17	8	Chocolate Biscuits	9	Confections
18	18	Gummibarchen	31	Confections
.....	.....	.....	.....	