Bases de dades i Gestió de flotes de dispositius

Màster Internet of Things

25 i **27de Maig** del 2020

Sessio 2
Bases de dades

Què és base de dades?

- **Base de dades:** Col·lecció de dades coherentment relacionades

- **Tipus de sistemes de bases de dades:**
  - Relacional
  - Grafs
  - Orientat a objectes

Relational Database Management Systems

- Robust software for managing data
- Data stored in tables that may be related
- Based on Structured Query Language (SQL)
  - Adopted by the American National Standards Institute (ANSI) and the International Standards Organization (ISO) as the standard data access language

File Versus Server-Based RDBMS

- File-based
  - Everything contained within a single file
  - Generally good for single user, desktop applications
  - Examples: SQLite, Microsoft Access
- Server-based
  - A database “server” manages databases and all transactions
  - Good for multiple, simultaneous connections and transactions
  - Examples: Oracle, Microsoft SQL Server, MySQL, PostgreSQL

Who are the main contenders?

- Commercial software
  - Sybase Adaptive Server Enterprise
  - IBM DB2
  - Oracle
  - Microsoft SQL Server
  - Teradata
- Free/GPL/Open Source:
  - MySQL
  - PostgreSQL

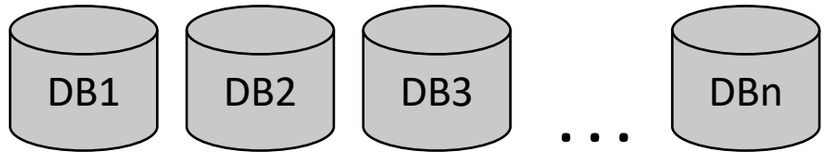
Relational Database Management Systems (RDBMS)

- File vs. server based
- Free vs. commercial
- Different data types
- Potentially different syntax for SQL queries
- Security models and concurrent users

Server-Based RDBMS – “Granules”

- Database Server

- Databases



- Tables

Site	Variable	Date	Value
1	Temperature	8/2/2007 14:00	12.4
1	Temperature	8/2/2007 14:30	12.7
1	Temperature	8/2/2007 15:00	13.1

- Records

1	Temperature	8/2/2007 14:00	12.4

Relational Database

- Review
- What is an Entity?

  An object of interest with properties or attributes

  Give some examples of entities?

University domain: student, lecturer, course, grade, room, facility, year-of-study, major, department, …

Entities Have Properties

- Properties have types
- Types: number, string, blob

In a **relational** DBMS, how do we represent entities?

Represent Entities as Tables

- Store data in rows and columns
- All tables have a name
  - Each column in the table has:
    - A name (property)
    - A type
  - Each row is a record
    - Records contain the data

Tables

Table Name: People

Columns

Rows

LastName	FirstName	Address
Hansen	Ola	Timoteivn 10
Svendson	Tove	Borgvn 23
Pettersen	Kari	Storgt 20

Tables can be related to one another

Relationships

- A **constraint** between two entities
  - Examples?
  - Each student has one grade per course
  - A student can take multiple courses
  - Types of relationships
  - one-to-one
  - many-to-one
  - many-to-many

Types of Relationships

- maximum number of relationship instances that an entity can participate in
- Ex: WORKS_FOR relation for DEPT:EMPL

1:1	Each DEPT can employ only 1 EMPL Each EMPL can work for only 1 DEPT
1:N	Each DEPT can employ any number of EMPL Each EMPL can work for only 1 DEPT
N:1	Each DEPT can employ only 1 EMPL Each EMPL can work for any number of DEPT
N:M	Each DEPT can employ any number of EMPL Each EMPL can work for any number of DEPT

Keys: More Constraints

- Primary Key
- Secondary Key(s)
- Candidate Key(s)
- Foreign Key(s)

Primary Key Constraint

- Uniquely identifies each record in a database table.

  - Primary keys **must** contain unique values.
  - A primary key column cannot contain NULL values.
  - Most tables should have a primary key, and each table can have only ONE primary key.

Secondary and Candidate Keys

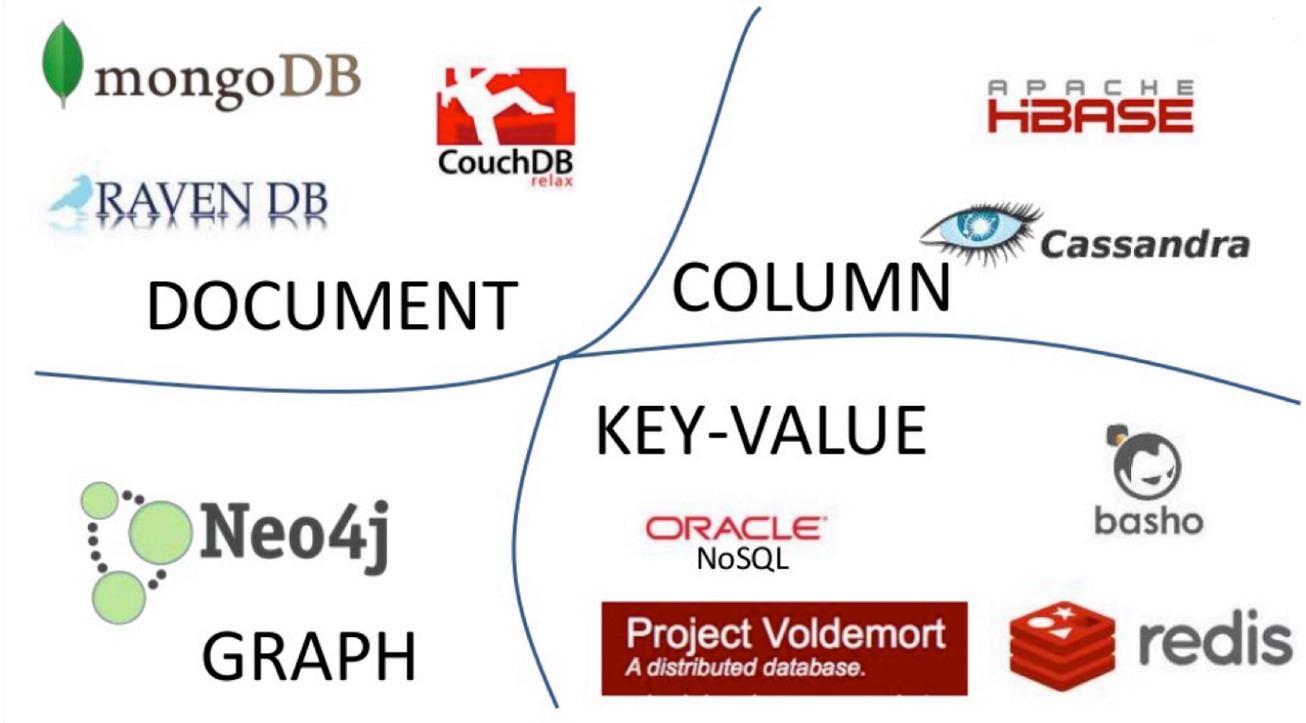
- Sometimes there are more than one key for a relation.
  - Database Administrator chooses the primary key.
  - The other keys are called **secondary** keys.
  - The **secondary** and **primary** keys are called **candidate** keys.

Foreign Keys

- Capture relationships across entities(tables).
- A **FOREIGN KEY** in one table points to a **PRIMARY KEY** in another table.

NoSQL

mongoDB

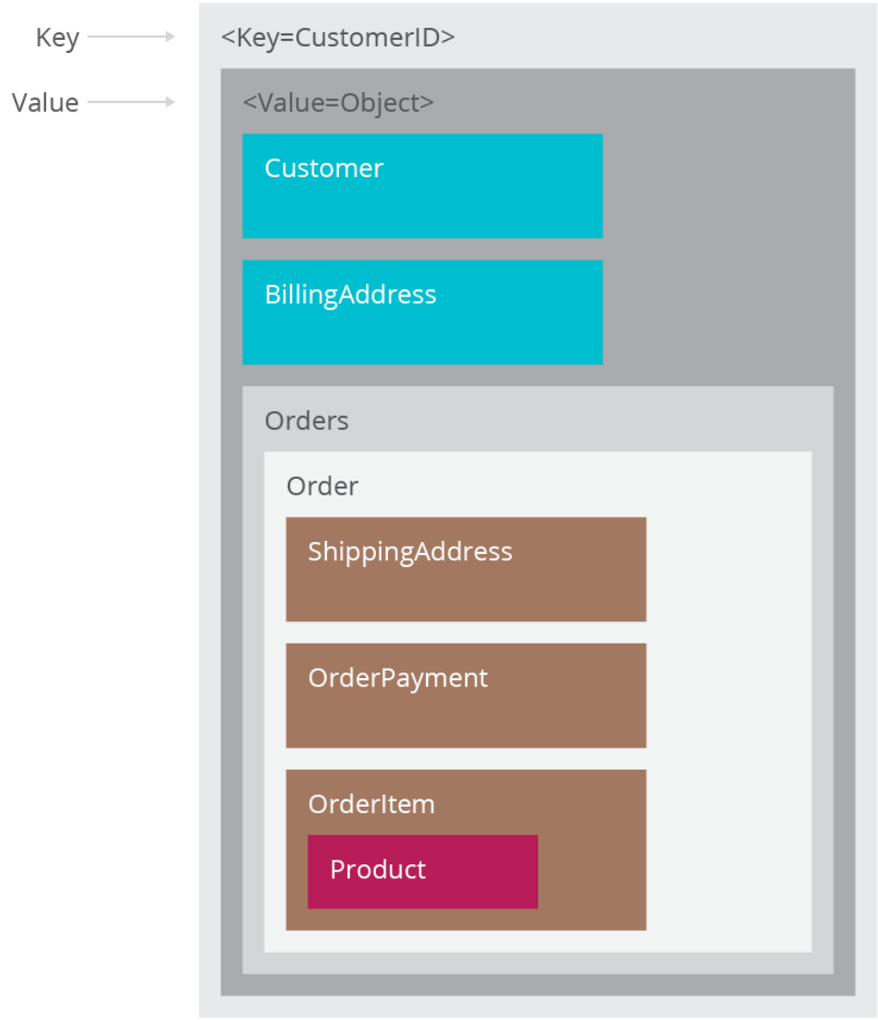


Types of NoSQL Databases: Key-Value Stores

- Key-value stores
  - The simplest NoSQL databases.
  - Every single item in the database is stored as an attribute name (or “key”), together with its value.
  - Some key-value stores allow each value to have a type, such as integer, which adds functionality.
  - Store session information, user profiles, shopping cart data, etc.
  - Examples: Riak, Berkeley DB and Redis.

Key-Value Stores, *cont'd*

Key-value store



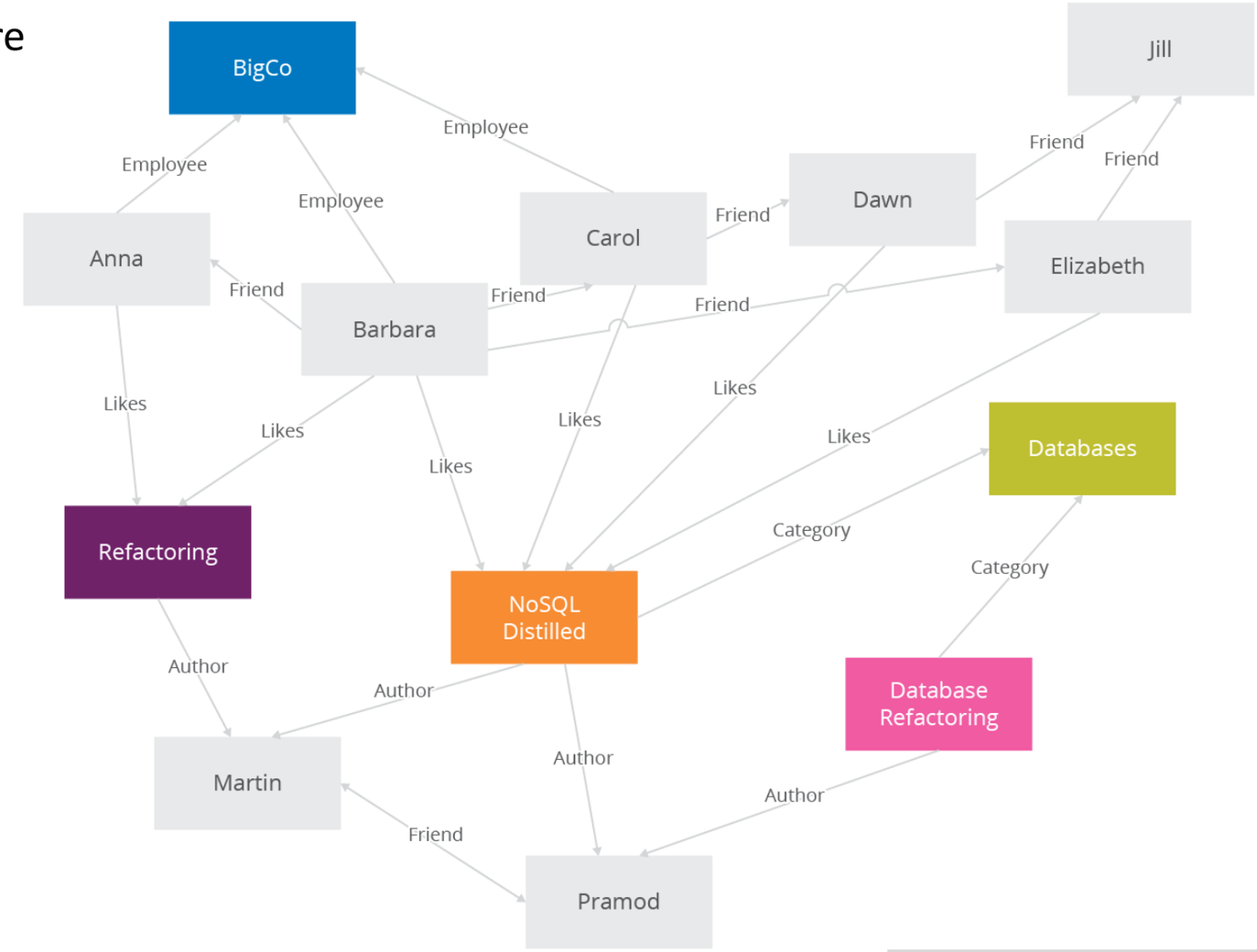
https://www.thoughtworks.com/insights/blog/nosql-databases-overview

Types of NoSQL Databases: Graph Stores

- Graph stores
  - Store information about connected data, such as social networks, spatial data, routing information for goods and money, recommendation engines, etc.
  - Examples: Neo4J and Giraph

Graph Stores, *cont'd*

Graph store



https://www.thoughtworks.com/insights/blog/nosql-databases-overview

Types of NoSQL Databases: Wide-Column Stores

- Wide-column stores
  - Optimized for queries over large datasets.
  - Store columns of data together, instead of rows.
  - The number of columns and the format of the data in a column can vary from row to row.
  - Useful for content management systems, blogging platforms, etc.
  - Examples: Apache Cassandra and Apache HBase

Wide-Column Stores, *cont'd*

mongoDB

Hbase Table

Table:Employees

	Column Family: Employee				Column Family: Address		
Row Key	Column: Employee First Name	Column: Employee Surname	Column: Employee Email	Column: Employee Date of Birth	Column: Address Line 1	Column: City	State
1	James	Dey	james_dey @hotmail.com			London	England
					Badger Farm	Winchester	England
2	Bill	Gates					
3	Larry	Ellison		17/08/1944	500 Oracle Parkway	Redwood Shores	

In comparison with the relational database tables, a column family can be considered to be similar to a relational table.

There can be multiple column families per row

Like relational tables, column families have to be created beforehand, but like document stores, you don't have to include all of the columns in each row.

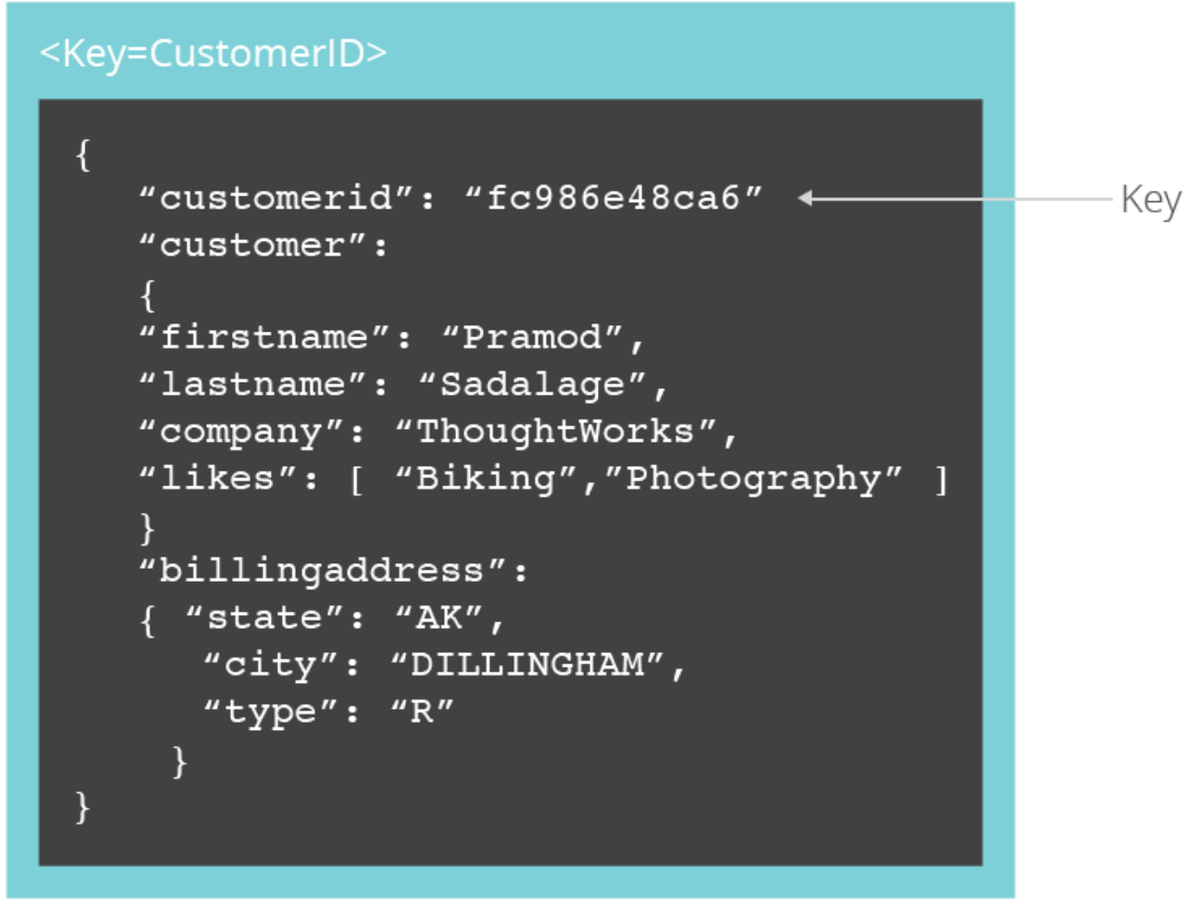
Wide-column store

https://bigdatalondon.files.wordpress.com/2015/06/hbase-table2.png

Types of NoSQL Databases: Document Databases

- Document databases
  - Document: Each key is paired with a complex data structure.
  - Documents can contain many different key-value pairs, or key-array pairs, or even nested documents.
  - Useful for content management systems, blogging platforms, web analytics, real-time analytics, ecommerce-applications, etc.
  - Example: MongoDB

Document Databases, *cont'd*



Document database

https://www.thoughtworks.com/insights/blog/nosql-databases-overview

Comparisons of NoSQL Database Types

Data Model	Performance	Scalability	Flexibility	Complexity	Functionality
Key–Value Store	high	high	high	none	variable (none)
Column-Oriented Store	high	high	moderate	low	minimal
Document-Oriented Store	high	variable (high)	high	low	variable (low)
Graph Database	variable	variable	high	high	graph theory
Relational Database	variable	variable	low	moderate	relational algebra

https://en.wikipedia.org/wiki/NoSQL

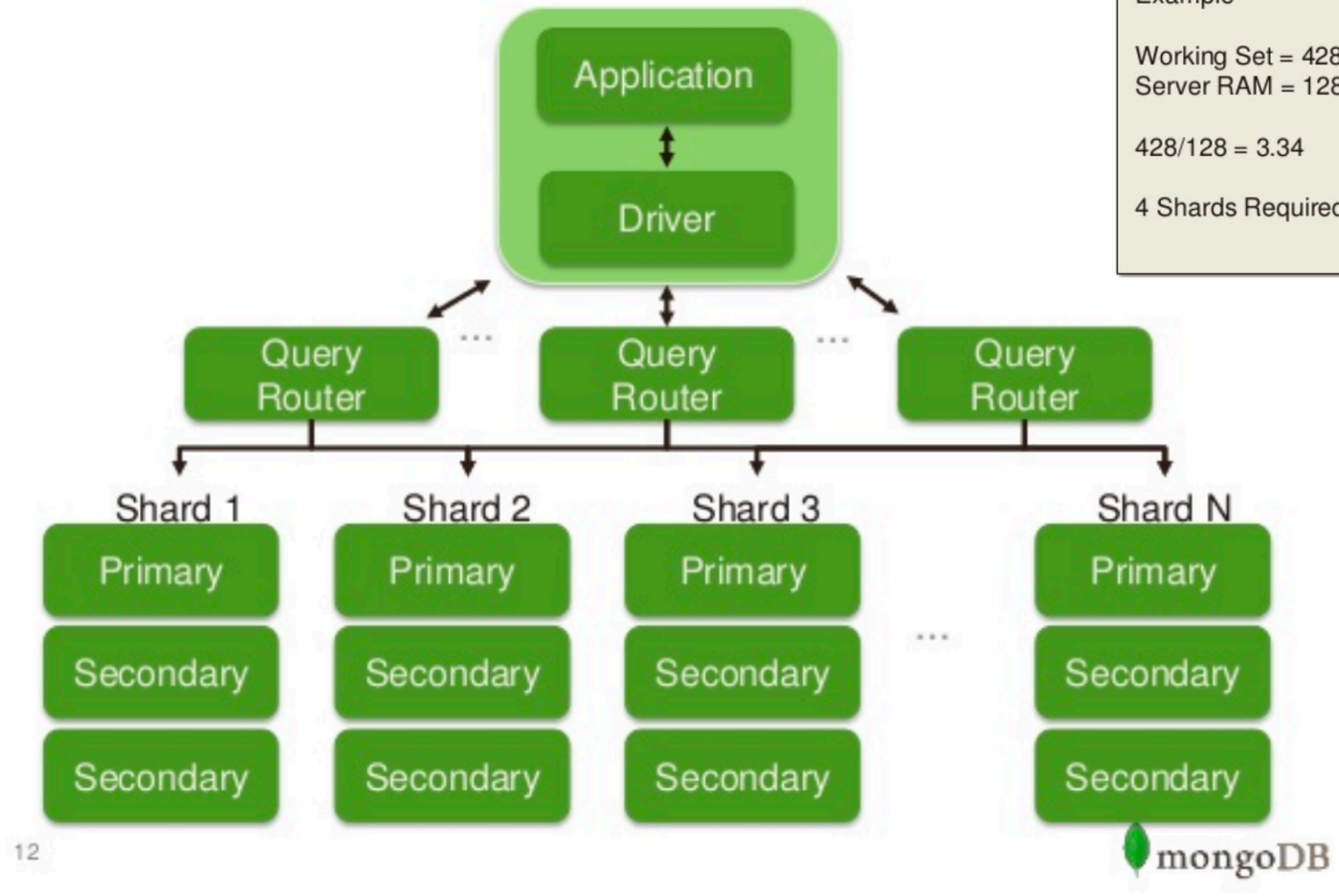
NoSQL Agile Development

- Unlike relational databases, NoSQL does not require schemas to be defined in advance.
  - **No database downtime to update a schema.**
- NoSQL can effectively address data that's completely unstructured or unknown in advance.
  - **Insert data without a predefined schema.**
- Apply validation rules within the database.
  - Enforce governance across data.
  - Maintain the agility benefits of a dynamic schema.

Scalability

- Scale-up
  - Relational databases scale vertically.
  - Increase the database server's capabilities in order to increase performance.
- Scale-out
  - Scale horizontally to support rapidly growing applications by adding more servers.
  - NoSQL databases support auto-sharding.

Sharding

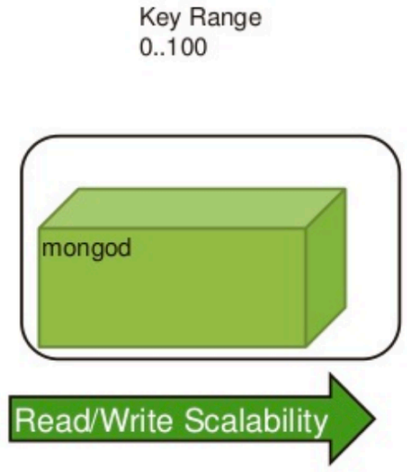


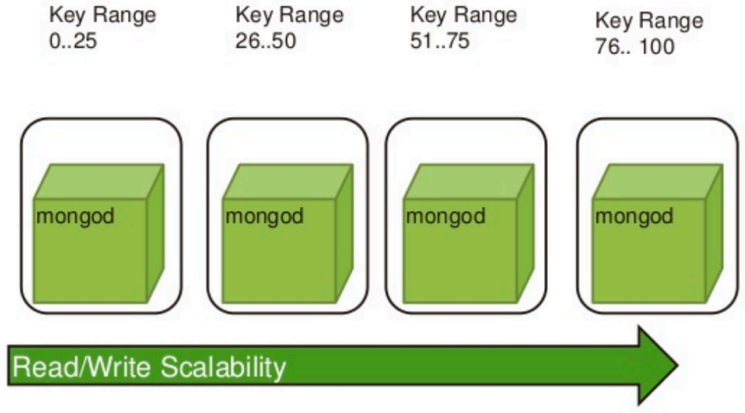
Example

Working Set = 428 GB
Server RAM = 128 GB

428/128 = 3.34

4 Shards Required





Auto-Sharding

- NoSQL databases natively and automatically spread data across an arbitrary number of servers.
- Applications do not need to be aware of the composition of the server pool.
- Data and query load are automatically balanced across servers.
- When a server goes down, it can be quickly and transparently replaced without application disruption.

Cloud Computing

- Auto-sharding works well with cloud computing.
- Example: Amazon Web Services
  - Provides virtually unlimited capacity on demand.
  - Takes care of all the necessary infrastructure administration tasks.
  - Developers no longer need to construct complex, expensive platforms to support their applications.

NoSQL Data Replication

- Automatic database replication maintains availability in the event of outages or planned maintenance events.
  - Automated failover and recovery.
- Distribute the database across multiple geographic regions to withstand regional failures and enable data localization.
  - No requirement for separate applications or expensive add-ons to implement replication.

NoSQL Data Caching

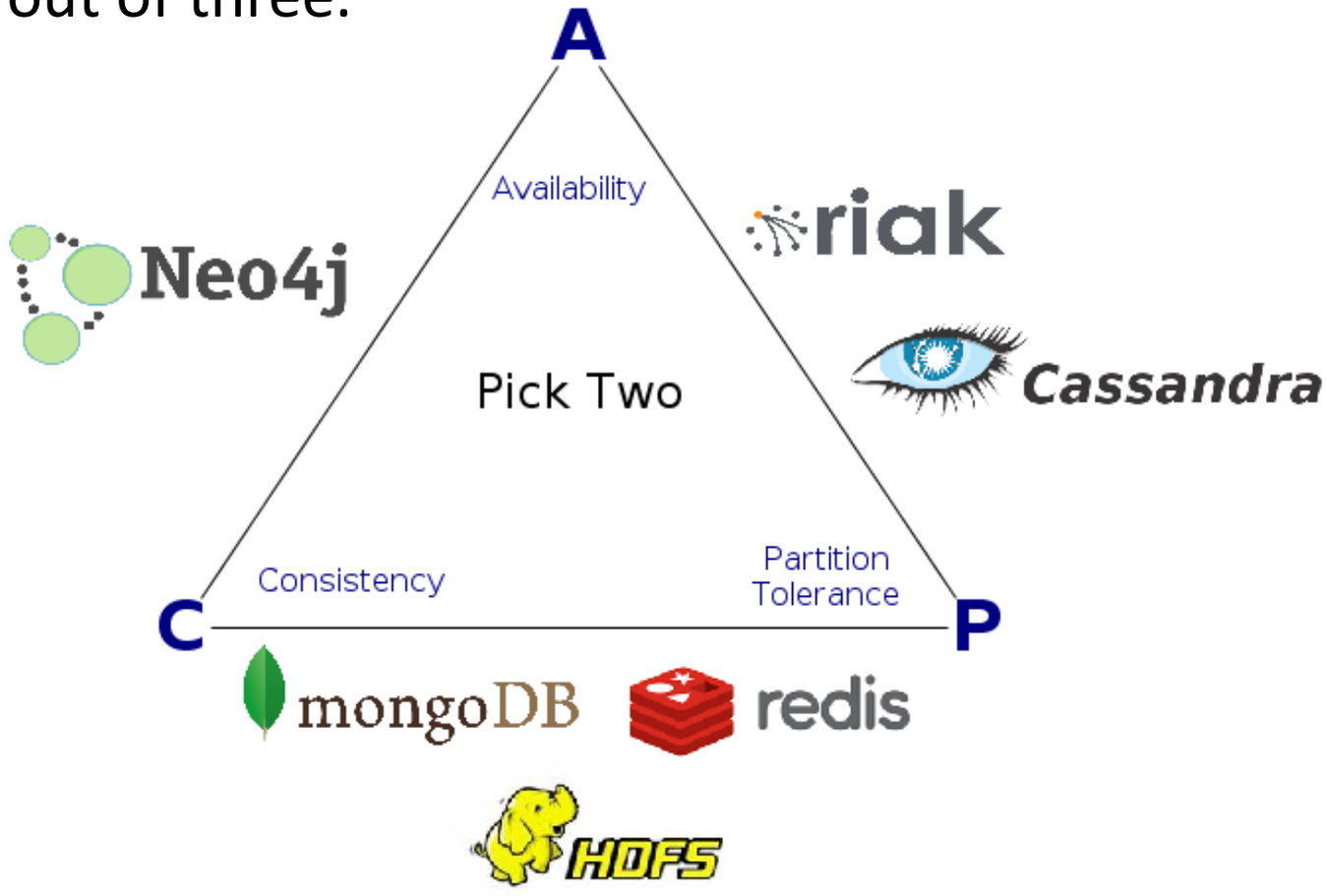
- Integrated caching capabilities.
  - Keep frequently-used data in system memory as much as possible.
  - No need for a separate caching layer.
- Available fully managed, integrated in-memory database management layer.
  - Support workloads that demand the highest throughput and lowest latency.

CAP Theorem

- Consistency
  - All nodes of a distributed system see the same data at the same time.
- Availability
  - A guarantee that every request gets a response whether the request succeeded or failed.
- Partition tolerance
  - The system continues to operate despite arbitrary network partitioning (splitting) due to device failures.

CAP Theorem, *cont'd*

- Consistency, availability, partition tolerance: You can only choose <u>two</u> out of three.
  - Different NoSQL databases emphasize different pairs of CAP.



SQL vs. NoSQL

	SQL	NoSQL
Types	SQL database	Key-value, graph, wide-column, document
Development history	Early 1970s	Late 2000s
Data model	Relational	Various
Schemas	Fixed	Dynamic
Scaling	Vertical	Horizontal
Development model	Mix of closed- and open-source	Open-source
Transaction support	Yes	At certain levels (e.g., document vs. database) depending on the application
Data manipulation	Via SQL language	Via object-oriented API
Consistency	Can be strong	Depends on the database product

MongoDB

- From “humongous” → Enorme, grandiós

- Scalable
- High performance
- Open-source
- Document-oriented
- Schema-free

- Scalable High-Performance Open-source, Document-orientated database.
- Built for Speed
- Rich Document based queries for Easy readability.
- Full Index Support for High Performance.
- Replication and Failover for High Availability.
- Auto Sharding for Easy Scalability.
- Map / Reduce for Aggregation.

MongoDB Features

- Horizontal scaling
- Indexing
- Replication/failover support

- Documents are stored in BSON (binary JSON)
  - Binary serialization of JSON-like objects.
  - Import or query any valid JSON object.

- Query syntax based on JavaScript.

MongoDB Documents and Collections

- Documents are analogous to records of a relational database.
  - JSON objects stored in a binary format.
- Collections are analogous to relational tables.
- A MongoDB query returns a cursor instead of records.

MongoDB Cursor

- A query returns a cursor.
  - Use the cursor to iterate through a result set.
- Better performance.
  - More efficient than loading all the objects into memory.

MongoDB Consistency

- The document model (and the graph model) can be consistent or eventually consistent.
  - Eventually consistent: Database changes are propagated to all nodes “eventually”.
    - Typically within milliseconds
  - Stale reads: Updated data not immediately returned.

- MongoDB consistency is tunable.
  - By default, all data is consistent.
  - All reads and writes access the primary data copy.
  - Read operations possible on the secondary copies.

Eventually Consistent Data

- Advantages
  - Fast data inserts.
- Disadvantages
  - Updates and deletes are more complex.
  - Periods of time during which not all copies of the data are synchronized.

MongoDB Interfaces

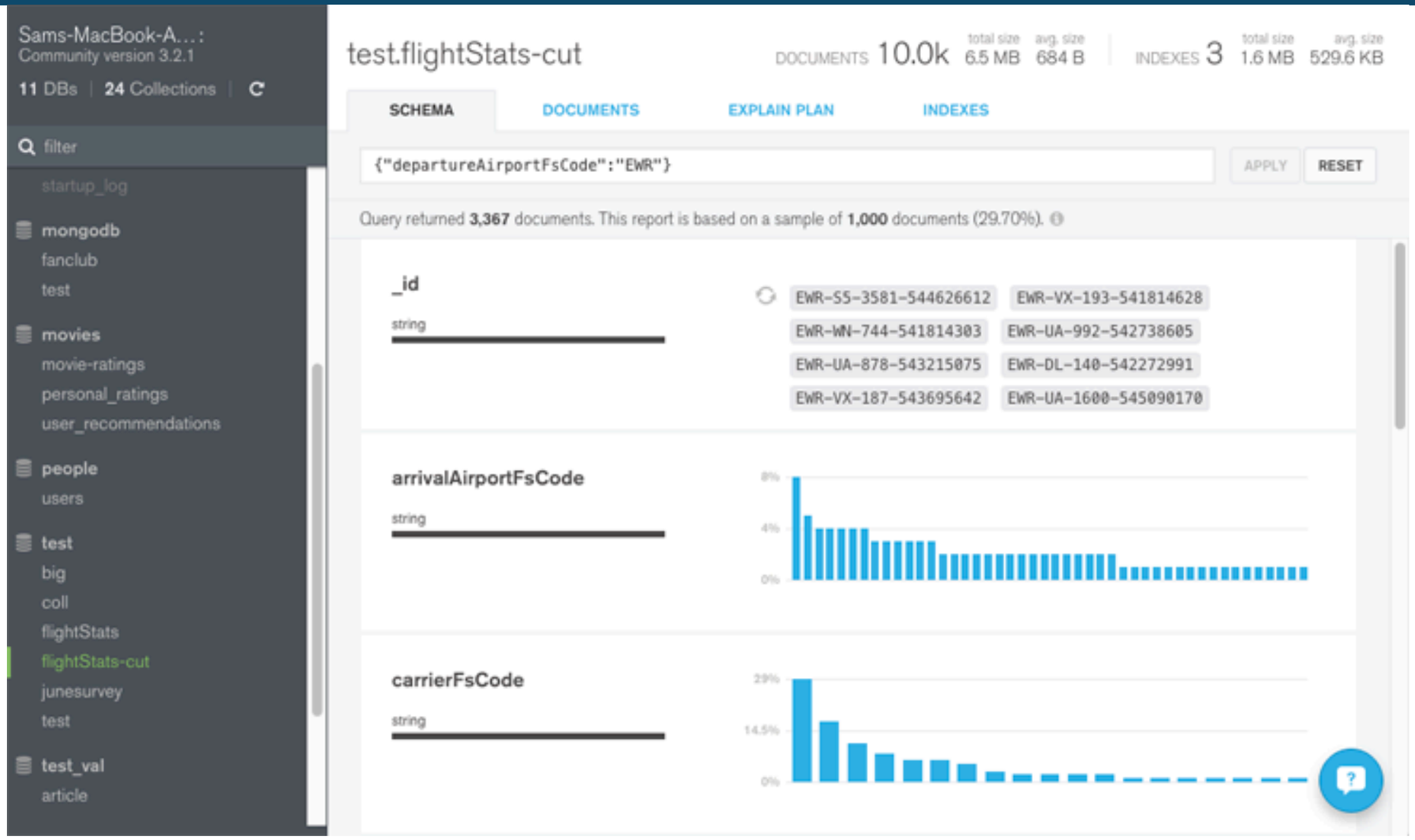
- Drivers
  - For many programming languages and development environments.
- GUI front-ends
  - Not included with the database.
  - Several third-party products for administration and data viewing.
  - Example: MongoDB Compass

MongoDB Compass

- Developed by the MongoDB engineers.

- Visually explore the data.
- Interact with the data with full CRUD functionality.
- Run ad hoc queries.
- View detailed information about indexes.
- View execution plans in order to optimize query performance.

MongoDB Compass, *cont'd*



https://docs.mongodb.com/ecosystem/tools/administration-interfaces/

MongoDB Terminologies for RDBMS concepts

RDBMS		MongoDB
Database	⇒	Database
Table, View	⇒	Collection
Row	⇒	Document (JSON, BSON)
Column	⇒	Field
Index	⇒	Index
Join	⇒	Embedded Document
Foreign Key	⇒	Reference
Partition	⇒	Shard

BSON

mongoDB

"Binary JSON"

Binary-encoded serialization of JSON-like docs

Embedded structure reduces need for joins

Goals

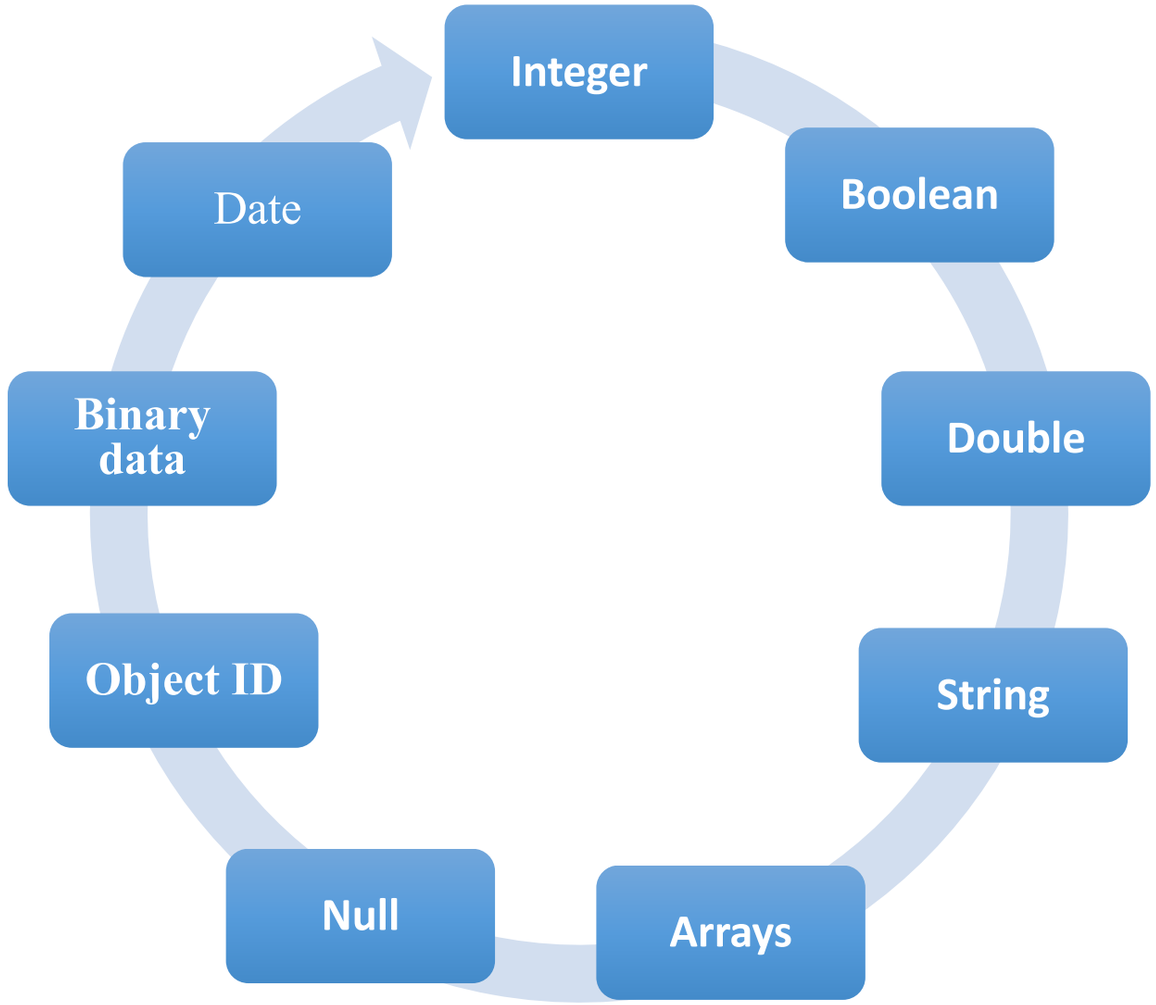
- Lightweight
- Traversable
- Efficient (decoding and encoding)

http://bsonspec.org/

BSON Example

```
{
"_id" :          "37010"
“City" :         “Nashik",
“Pin" :           423201,
"state" :        “MH",
“Postman” : {
                    name: “Ramesh Jadhav”
                    address: “Panchavati”
                  }
}
```

Data Types of MongoDB



Data Types

- **String** : This is most commonly used datatype to store the data. String in mongodb must be UTF-8 valid.
- **Integer** : This type is used to store a numerical value. Integer can be 32 bit or 64 bit depending upon your server.
- **Boolean** : This type is used to store a boolean (true/ false) value.
- **Double** : This type is used to store floating point values.
- **Min/ Max keys** : This type is used to compare a value against the lowest and highest BSON elements.
- **Arrays** : This type is used to store arrays or list or multiple values into one key.
- **Timestamp** : ctimestamp. This can be handy for recording when a document has been modified or added.
- **Object** : This datatype is used for embedded documents.

Data Types

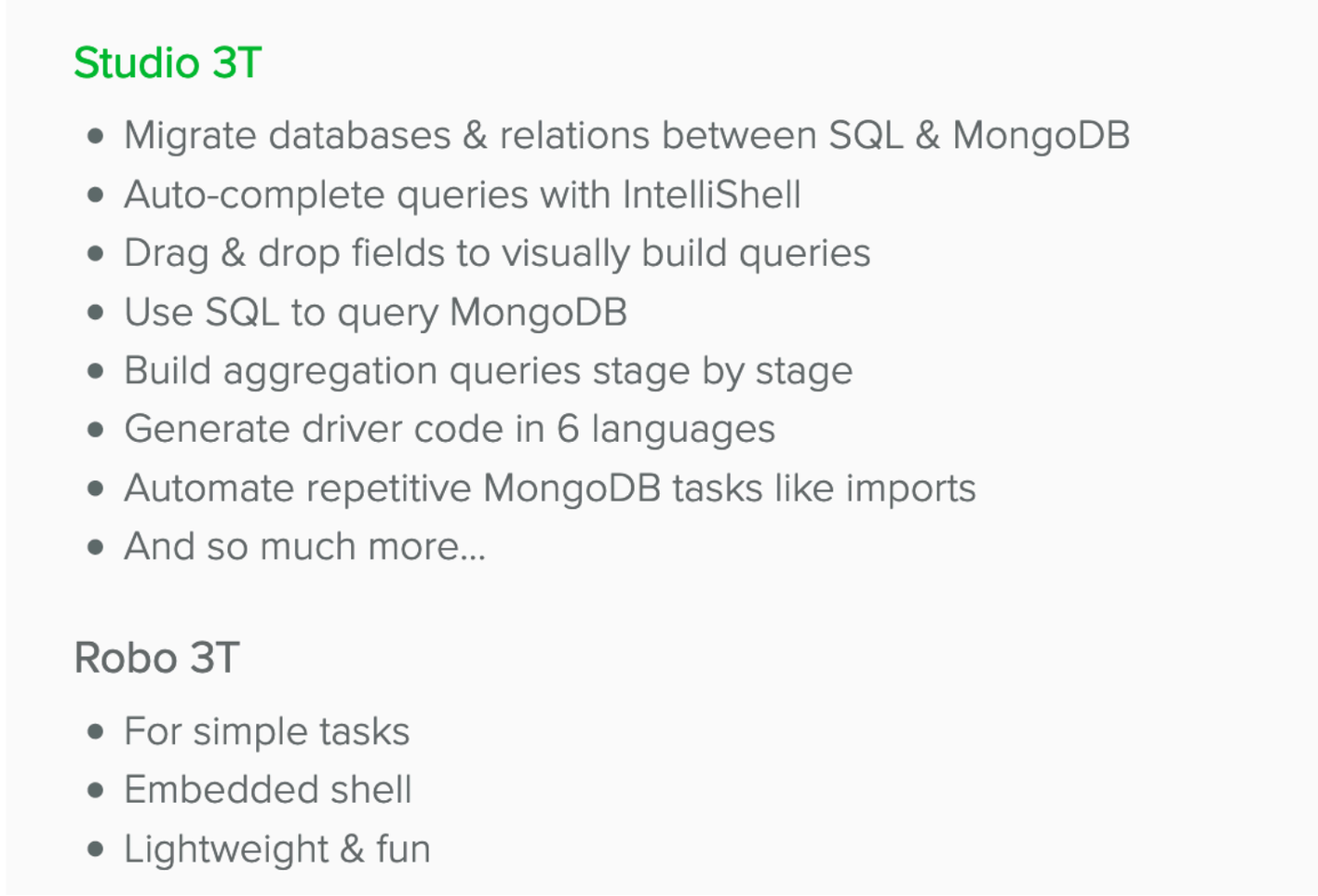
- **Null** : This type is used to store a Null value.
- **Symbol** : This datatype is used identically to a string however, it's generally reserved for languages that use a specific symbol type.
- **Date** : This datatype is used to store the current date or time in UNIX time format. You can specify your own date time by creating object of Date and passing day, month, year into it.
- **Object ID** : This datatype is used to store the document’s ID.
- **Binary data** : This datatype is used to store binay data.
- **Code** : This datatype is used to store javascript code into document.
- **Regular expression** : This datatype is used to store regular expression

MongoDB Demo

Install : Install Robo3T (database client)

And connect to the given URI, were we have

Setup a database specific for this session.



MongoDB Demo, *cont'd*

- Create the school database:

```
> use school
switched to db school
```

- Insert a document into the teacher collection:

```
> db.teacher.insert( {id: 7003, last: "Rogers", first: "Tom"} )
WriteResult({ "nInserted" : 1 })
```

- Who's in there now?

```
> db.teacher.find()
{ "_id" : ObjectId("582a7630f22e3c2f12d899ac"),
  "id" : 7003, "last" : "Rogers", "first" : "Tom" }
```

MongoDB Demo, *cont'd*

- Insert the rest of the teachers as an array of documents:

```
> db.teacher.insert([
... { id: 7008, last: "Thompson", first: "Art" },
... { id: 7012, last: "Lane", first: "John" },
... { id: 7051, last: "Flynn", first: "Mabel" }
... ])
```

MongoDB Demo, *cont'd*

- Find all of them and pretty print:

```
> db.teacher.find().pretty()
{
        "_id" : ObjectId("582a7630f22e3c2f12d899ac"),
        "id" : 7003,
        "last" : "Rogers",
        "first" : "Tom"
}
{
        "_id" : ObjectId("582a7808f22e3c2f12d899ad"),
        "id" : 7008,
        "last" : "Thompson",
        "first" : "Art"
}
{
        "_id" : ObjectId("582a7808f22e3c2f12d899ae"),
        "id" : 7012,
        "last" : "Lane",
        "first" : "John"
}
{
        "_id" : ObjectId("582a7808f22e3c2f12d899af"),
        "id" : 7051,
        "last" : "Flynn",
        "first" : "Mabel"
}
```

MongoDB Demo, *cont'd*

- A simple query with **`$or`**:

```
> db.teacher.find(
... { $or: [ {id: 7008}, {first: "John"} ] }
... ).pretty()
{
        "_id" : ObjectId("582a7808f22e3c2f12d899ad"),
        "id" : 7008,
        "last" : "Thompson",
        "first" : "Art"
}
{
        "_id" : ObjectId("582a7808f22e3c2f12d899ae"),
        "id" : 7012,
        "last" : "Lane",
        "first" : "John"
}
```

MongoDB Demo, *cont'd*

- Update a document:

```
> db.teacher.find()
{ "_id" : ObjectId("582a7630f22e3c2f12d899ac"), "id" : 7003, "last" : "Rogers", "first" : "Tom" }
{ "_id" : ObjectId("582a7808f22e3c2f12d899ad"), "id" : 7008, "last" : "Thompson", "first" : "Art" }
{ "_id" : ObjectId("582a7808f22e3c2f12d899ae"), "id" : 7012, "last" : "Lane", "first" : "John" }
{ "_id" : ObjectId("582a7808f22e3c2f12d899af"), "id" : 7051, "last" : "Flynn", "first" : "Mabel" }
```

```
> db.teacher.update(
... {id: 7051},
... {$set: {first: "Mabeline"}}
... )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

```
> db.teacher.find( {id: 7051} ).pretty()
{
        "_id" : ObjectId("582a7808f22e3c2f12d899af"),
        "id" : 7051,
        "last" : "Flynn",
        "first" : "Mabeline"
}
```

MongoDB Demo, *cont'd*

- Projection: Show only the ids and last names:

```
> db.teacher.find( {}, {_id: 0, id: 1, last: 1} )
{ "id" : 7003, "last" : "Rogers" }
{ "id" : 7008, "last" : "Thompson" }
{ "id" : 7012, "last" : "Lane" }
{ "id" : 7051, "last" : "Flynn" }
```

MongoDB Demo, *cont'd*

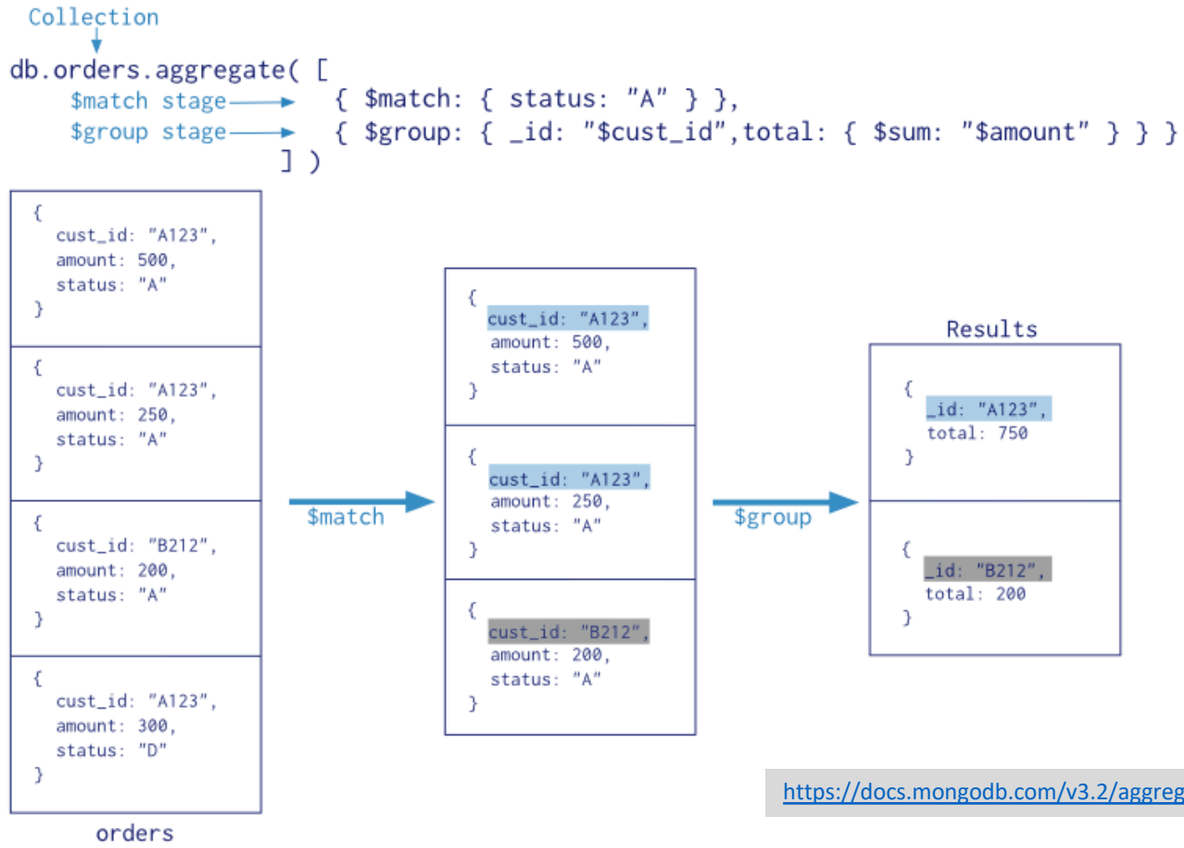
- Sort in ascending or descending order by last name:

```
> db.teacher.find( {}, {_id: 0, id: 1, last: 1} ).sort({last: 1})
{ "id" : 7051, "last" : "Flynn" }
{ "id" : 7012, "last" : "Lane" }
{ "id" : 7003, "last" : "Rogers" }
{ "id" : 7008, "last" : "Thompson" }
```

```
> db.teacher.find( {}, {_id: 0, id: 1, last: 1} ).sort({last: -1})
{ "id" : 7008, "last" : "Thompson" }
{ "id" : 7003, "last" : "Rogers" }
{ "id" : 7012, "last" : "Lane" }
{ "id" : 7051, "last" : "Flynn" }
```

Aggregation

- Documents enter a “pipeline” where the data is matched, grouped, and aggregated.

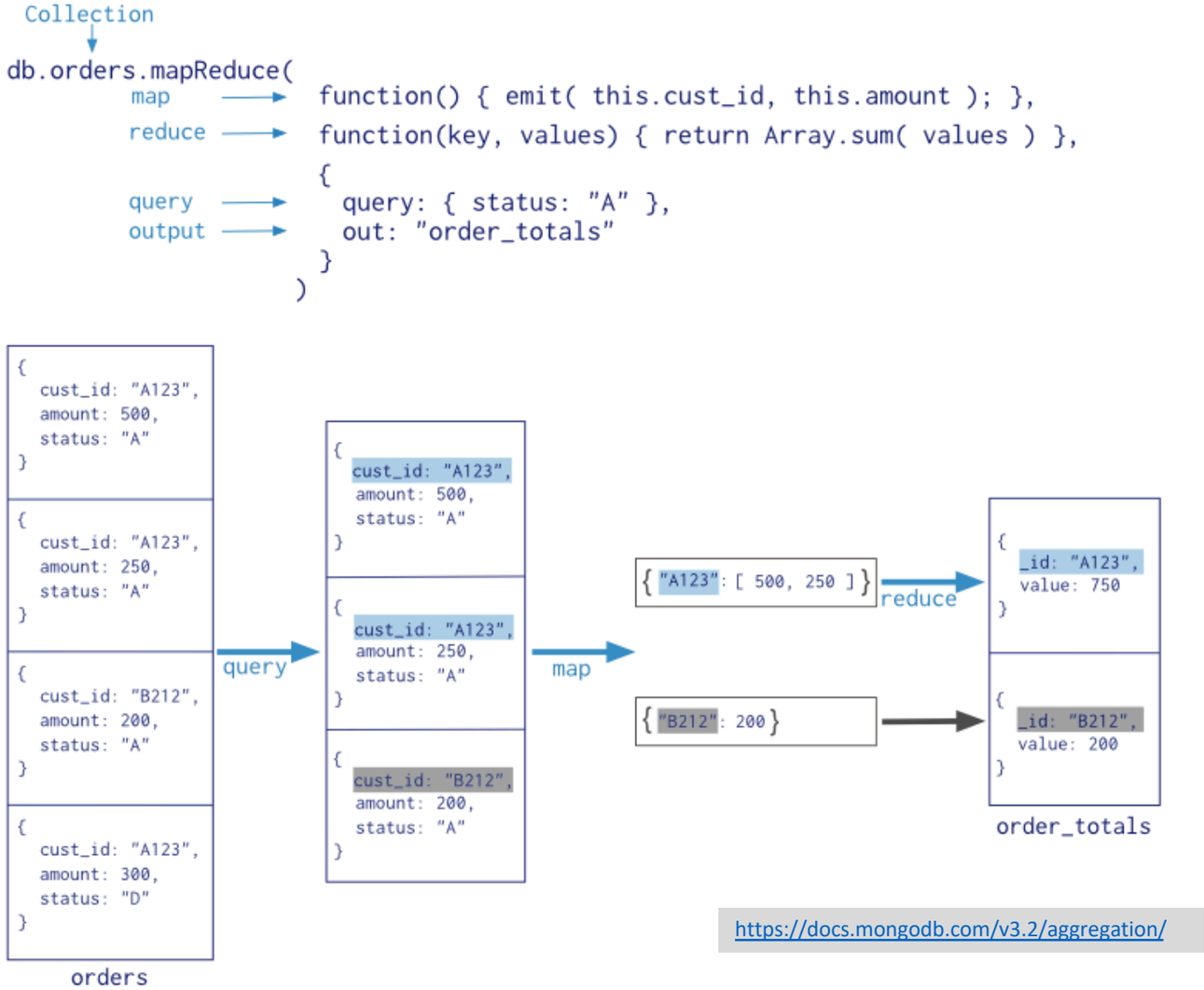


https://docs.mongodb.com/v3.2/aggregation/

Map-Reduce for Aggregation

- Map stage
  - Process each document.
  - Emit one or more objects per document.
- Reduce stage
  - Combine the output of the map operation.
- Finalize stage
  - Make final modifications to the result.

Map-Reduce for Aggregation, *cont'd*



https://docs.mongodb.com/v3.2/aggregation/

Task

- Crear una base de dades MongoDB
- Crear dues col·leccions a la BBDD i inserir documents d'exemple a cada col·lecció

- Per cada col·lecció:
  - Printar tots els documents
  - Crear tres consultes amb diferents paràmetres de cerca i ordenació
  - Fer una cerca amb el pipeline ***aggregation***.
  - Fer una consulta utilitzant l'esquema **map-reduce**

References

- https://docs.mongodb.com/manual/introduction/
- http://metadata-standards.org/Document-library/Documents-by-number/WG2-N1501-N1550/WG2_N1537_SQL_Standard_and_NoSQL_Databases%202011-05.ppt
- https://www.slideshare.net/raviteja2007/introduction-to-mongodb-12246792
- https://docs.mongodb.com/manual/core/databases-and-collections/
- https://docs.mongodb.com/manual/crud/