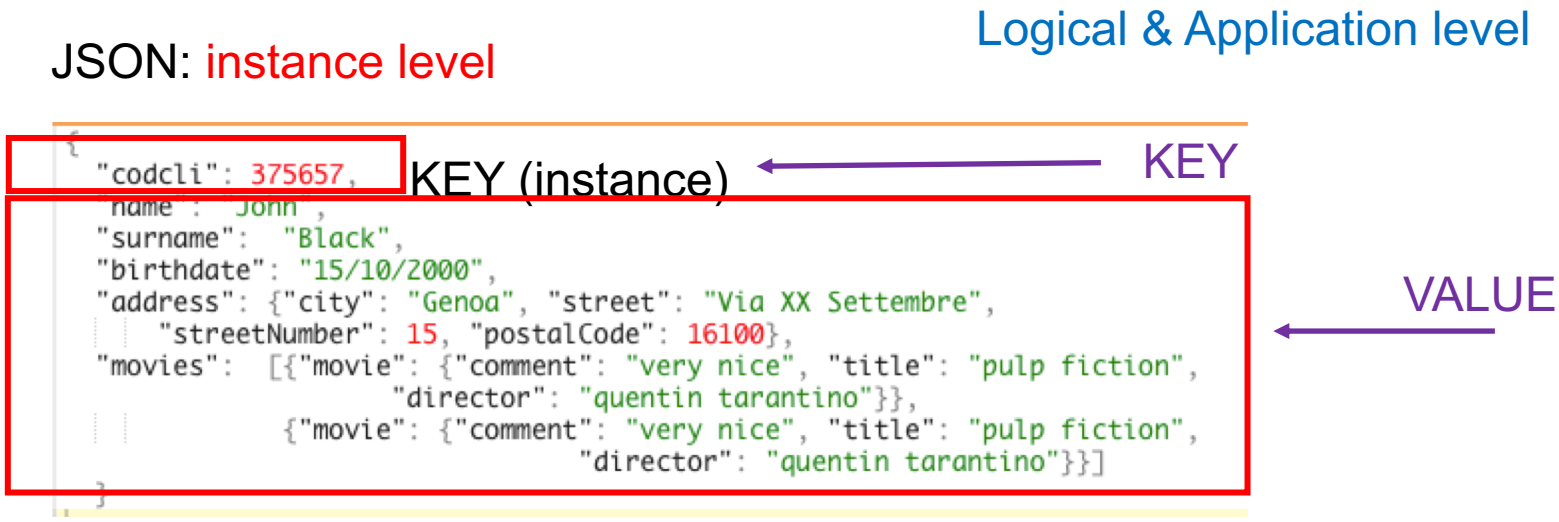
Document-oriented NoSQL data stores

Document-oriented data model at a glance

- Each data instance is represented in the form (key, value)
  - key is an identifier
  - value is the aggregate, corresponding to a document that, in turns, can be interpreted as a set of <name, nested-document> pairs
- Structure of the aggregate is visible at the logical and application level
  - All nested document components can be accessed during read and write operations
- Also in this case, data instances can be grouped into logical collections



Documents

- Can be represented using a semi-structured language like XML, JSON, BSON, and so on
- Are self-describing, hierarchical tree data structures
- Can consist of scalar values, collections, and maps
- Are structurally similar but not identical to each other
- Semi-structured data model

Example - document


```
{
  "codcli": 375657,
  "name": "John",
  "surname":  "Black",
  "birthdate": "15/10/2000",
  "address": {"city": "Genoa", "street": "Via XX Settembre",
     "streetNumber": 15, "postalCode": 16100},
  "movies":  [{"movie": {"comment": "very nice", "title": "pulp fiction",
                    "director": "quentin tarantino"}},
              {"movie": {"comment": "very nice", "title": "pulp fiction",
                                "director": "quentin tarantino"}}]
  }
```



```
▼ object {6}
    codcli : 375657
    name : John
    surname : ■ Black
    birthdate : 15/10/2001
  ▼ address {4}
      city : Genoa
      street : Via XX Settembre
      streetNumber : 15
      postalCode : 16100
  ▼ movies [2]
    ▼ 0  {1}
      ▼ movie {3}
          comment : very nice
          title : pulp fiction
          director : quentin tarantino
    ▼ 1  {1}
      ▼ movie {3}
          comment : very nice
          title : pulp fiction
          director : quentin tarantino
```

Document structure

- No predefined schema, no DDL
- The schema of the data can differ across documents
- In documents, there are no empty attributes
  - if a given attribute is not found, it was not set or not relevant to the document
- New attributes can be created without the need to define them or to change the existing documents

Example – similar documents


```
{
  "codcli": 375657,
  "name": "John",
  "surname":  "Black",
  "birthdate": "15/10/2000",
  "address": {"city": "Genoa", "street": "Via XX Settembre",
     "streetNumber": 15, "postalCode": 16100},
  "movies":  [{"movie": {"comment": "very nice", "title": "pulp fiction",
                         "director": "quentin tarantino"}},
              {"movie": {"comment": "very nice", "title": "pulp fiction",
                                  "director": "quentin tarantino"}}]
  }
```



```
▼ object {6}
    codcli : 123456
    name : Jane
    surname : White
    phonenbr : 3406701353
    email : jane.white@gmail.com
  ▼ movies [1]
    ▼ 0  {1}
      ▼ movie {3}
            comment : best movie ever
            title : pulp fiction
            director : quentin tarantino
```



```
{
  "codcli": 123456,
  "name": "Jane",
  "surname": "White",
  "phonenbr": "3406701353",
  "email": "jane.white@gmai
  "movies": [
    {
      "movie": {
        "comment": "best movie ever",
        "title": "pulp fiction",
        "director": "quentin tarantino"
      }
    }
  ]
}
```



```
▼ object {6}
    codcli : 375657
    name : John
    surname : Black
    birthdate : 15/10/2001
  ▼ address {4}
      city : Genoa
      street : Via XX Settembre
      streetNumber : 15
      postalCode : 16100
  ▼ movies [2]
    ▼ 0  {1}
      ▼ movie {3}
            comment : very nice
            title : pulp fiction
            director : quentin tarantino
    ▼ 1  {1}
      ▼ movie {3}
            comment : very nice
            title : pulp fiction
            director : quentin tarantino
```

Collections

- Pairs can be grouped into logical collections: sets of aggregates, i.e., sets of (key, value) pairs
- A collection is a grouping of documents

Collections and schema information

- (nested) names in pairs representing the document
- collections do not enforce specific data structure
  - documents within a collection can have different fields (flexibility)
  - in practice, all documents in a collection are similar and have related purpose

Actually, in mongoDB we may require data validation against a JSON schema

Example - collection

```
{
  "codcli": 375657,
  "name": "John",
  "surname":  "Black",
  "birthdate": "15/10/2000",
  "address": {"city": "Genoa", "street": "Via XX Settembre",
      "streetNumber": 15, "postalCode": 16100},
  "movies":  [{"movie": {"comment": "very nice", "title": "pulp fiction",
                  "director": "quentin tarantino"}},
              {"movie": {"comment": "very nice", "title": "pulp fiction",
                              "director": "quentin tarantino"}}]
  }
{
  "codcli": 123456,
  "name": "Jane",
  "surname": "White",
  "phonenbr": "3406701353",
  "email": "jane.white@gmail.com",
  "movies": [
    {
      "movie": {
        "comment": "best movie ever",
        "title": "pulp fiction",
        "director": "quentin tarantino"
      }
    }
  ]
}
```

Keys

- (key, value) pair

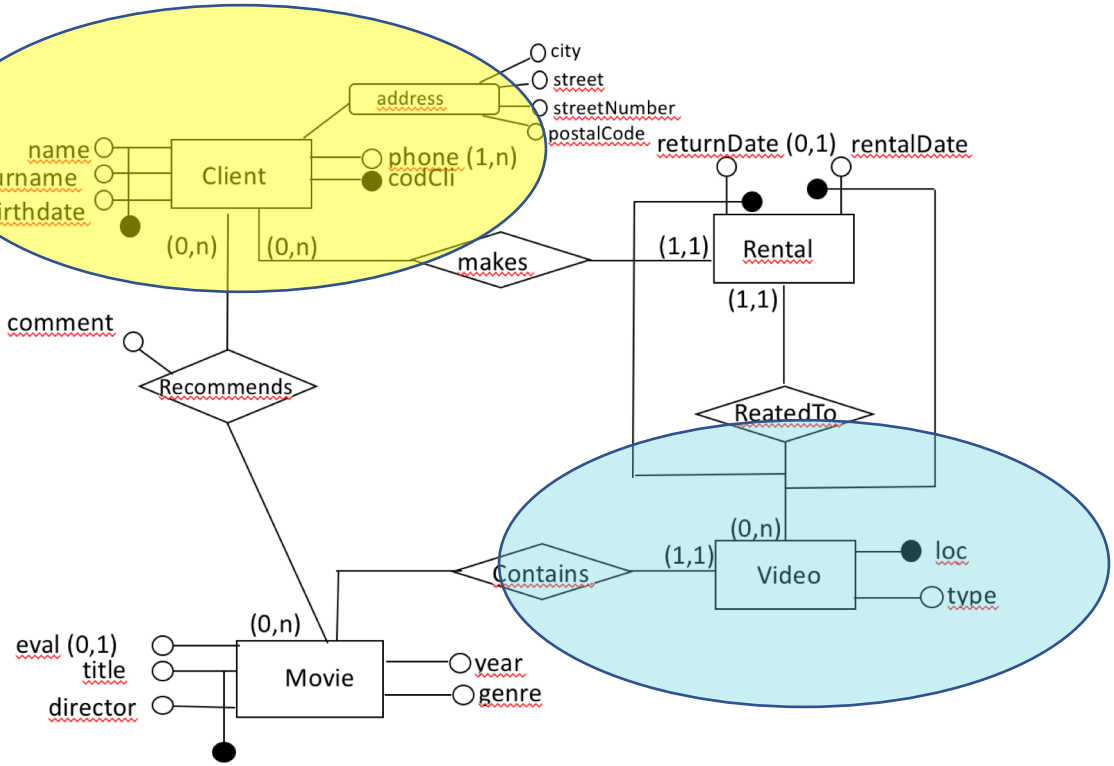
- At the logical level: identification + data retrieval
  - the key needs to assume unique values in the collection (i.e., it is an identifier)
  - mandatory
- At the physical level, the key could tell the system how to partition the data and where to store the data
  - partition key

- Partition key and identifier could to be different

- Aggregates can be directly retrieved by specifying values for attributes in the key or for nested attributes
- The structure of the aggregate is exposed
  - All nested sub-documents can be accessed

Keys

- Usually, one single attribute
- The type and the content identified starting from the designed aggregates



Video: {<u>loc</u>,…}

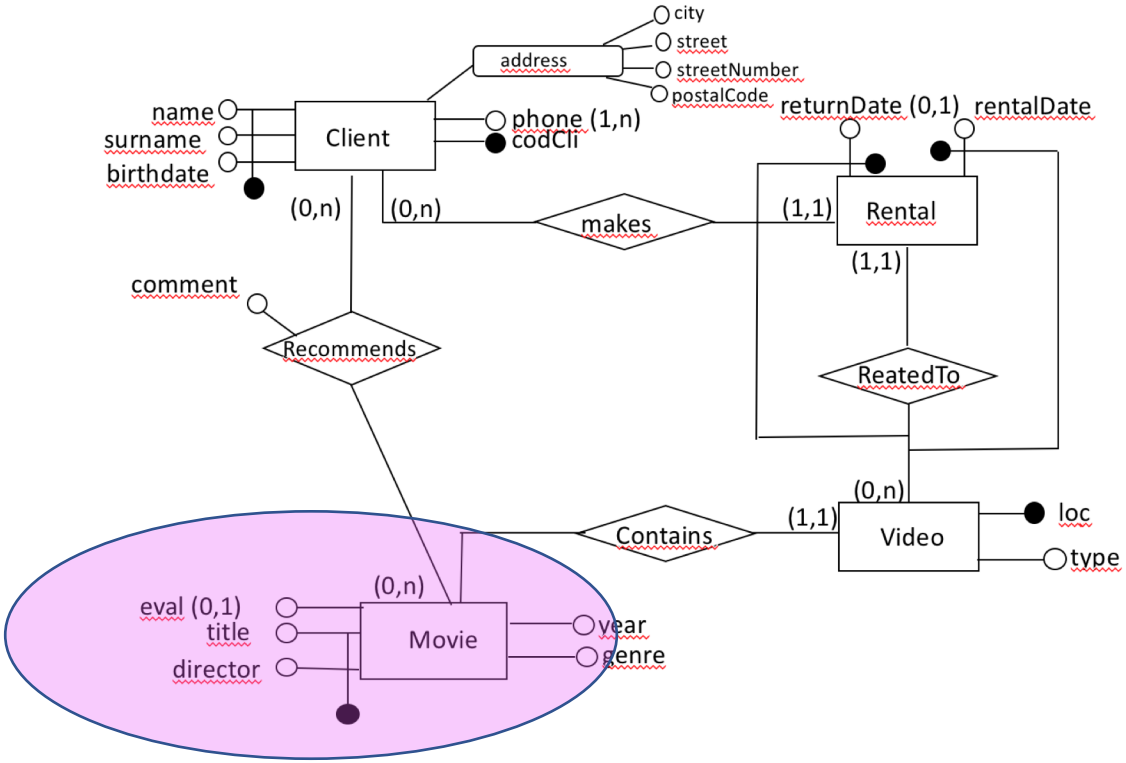
key is loc

Client: {<u>codCli</u>, …}

key is codCli

_id in MongoDB

Keys



Movie : {<u>title, director</u>,…}

key could be (title, director)
or we can use a new attribute **moveId** as the key (in this case values will be provided by the system)

Identifier vs partition/sharding key

- Ids mainly refer the way data can be referenced
  - In the Videos collection, videos are identified by loc
  - In the Clients collection, clients are indentified by codCli
  - In the Movies collection, clients are identified by title&director (or moviedId)
- In document-oriented data stores, the choice of an identifier does not impact how data can be retrieved
- Partition keys impact the way data are stored

Video Rental Example

Logical & application level

client: { codCli,

name, surname, birthdate,
address: {city, street, streetNumber, postalCode},
movies: [{movie: {comment, id:{title, director},…}}
]
}

REFERENCE

REFERENCE

movie: { id: {title,director},

year, genre, recommended_by: [{name, surname, comment}],
contained_in: [{video: {loc, type}}] }

video: {loc,

type, rentals: [{rental: {rentalDate, codCli}}], title, director}}

Query features

- Document data stores provide different query features
- In general:
  - you can query the data inside the document without having to retrieve the whole document by its key and then introspect the document

Interaction

- Basic lookup based on the key or field names in the aggregates
  - Document get(key)
  - Document get([name:value])
  - Void put(key, document)
  - Void set(key, [name:value])
  - Void remove(key)
- We can query inside (nested) documents, at any level
- Additional search constraints depending on the specific document representation (e.g, range constraints on values)
- We can retrieve part of the aggregate rather than the whole one (projection operations)
- In case collections are supported, the collection name has to be specified in each operation

Example 1 – Find the title of the movie in a certain video

```
{"loc": 1234,

  "type":"dvd",
  "rentals": [{"rental": {"rentalDate":"15/10/2021",
                          "codCli": 375657}}],
  "title": "pulp fiction",
  "director": "quentin tarantino"}
```

get(Video, 1234, [title])
Retrieve the video by the key (location number)
and project on the title

Example 2 – Find the videos containing a certain movie

```
{"loc": 1234,

  "type":"dvd",
  "rentals": [{"rental": {"rentalDate":"15/10/2021",
                          "codCli": 375657}}],
  "title": "pulp fiction",
  "director": "quentin tarantino"}
```

get(Video, [title:'pulp fiction'])
Retrieve the video by some filtering conditions on (nested) attributes and we return the whole videos

Example 3 – Find the videos rented by a certain client

```
{"loc": 1234,

  "type":"dvd",
  "rentals": [{"rental": {"rentalDate":"15/10/2021",
                          "codCli": 375657}}],
  "title": "pulp fiction",
  "director": "quentin tarantino"}
```

get(Video, '375657', [rentals.rental.codCli])
Retrieve the video by the key
and we return some nested attributes (dot notation)

Example 4 – Find the videos containing a certain movie - with the movies collection

movie: **{ id: {title, director},** year, genre,
recommended_by: [{name, surname, comment}],
contained_in: [{video: {loc, type}}] }

- get(Movie, {'pulp fiction','quentin tarantino'}, [contained_in.video])
- Retrieve all the information about the videos containing a certain movie

REMARK: the relationship is part of the aggregate, a single data access to retrieve all the relevant information (contained in a single data node)

Example 5 – Find the videos containing a certain movie - with the movies collection, Movieid

movie: **{ Movieid,**
title, director, year, genre,
recommended_by: [{name, surname, comment}],
contained_in: [{video: {loc, type}}] }

- get(Movies,[title:'pulp fiction', director: 'quentin tarantino'])
- At the data store level, retrieve the information about the videos containing a certain movie

REMARK: the relationship is part of the aggregate, a single data access to retrieve all the relevant information (contained in a single data node)

Example 4 - Relationships

Find the age(s) of customer(s) that rented a given video

```
{"loc": 1234,

  "type":"dvd",
  "rentals": [{"rental": {"rentalDate":"15/10/2021",
                          "codCli": 375657}}],
  "title": "pulp fiction",
  "director": "quentin tarantino"}
```

```
{
  "codcli": 375657,
  "name": "John",
  "surname":  "Black",
  "birthdate": "15/10/2000",
  "address": {"city": "Genoa", "street": "Via XX Settembre",
     "streetNumber": 15, "postalCode": 16100},
  "movies":  [{"movie": {"comment": "very nice", "title": "pulp fictio
                         "director": "quentin tarantino"}},
              {"movie": {"comment": "very nice", "title": "pulp fictio
                                    "director": "quentin tarantino"}}]
}
```

get(Videos,1234)

- At the data store level, find Video 1234 in collection Videos, no detailed information the clients that rented it
- at the *application level*, we go inside the aggregate value and we discover that it it was rented by Client 375657
- we can execute get(Clients, 375657) to retrieve information about a/the customer that rented video 1234 (thus navigating the customer reference stored inside the video)

Advanced interaction

- Some document oriented data stores support more expressive interaction
- Specifically, management of explicit references among documents (for relationships) and a sort of join operation traversing them
- MongoDB, e.g., supports the aggregation framework and the $lookup costruct that allows to navigate references (join)
- Embed vs reference
- More expressive query languages (e.g., group by)

Example 4 – Relationships – In MongoDB

Find the age(s) of customer(s) that rented a given video

```
{"loc": 1234,

  "type":"dvd",
  "rentals": [{"rental": {"rentalDate":"15/10/2021",
                          "codCli": 375657}}],
  "title": "pulp fiction",
  "director": "quentin tarantino"}
```

```
{
  "codcli":
  "na
```

```
: "pulp fictio
entin tarantino"}}]
```

L
via
s and we can join
ients that rented a video

REMARKS:

1. System support for join, but we need to explicitly tell the system what to look for and what to match
2. Since the relationship is not part of the aggregate, navigating it requires two distinct data accesses, at two possibly distinct data nodes (queries on embedded relationships are more efficient)

Popular document data stores

Each product has some features that may not be found in others

Popular document data stores

- MongoDB [http://www.mongodb.org/],
- CouchDB [http://couchdb.apache.org/],
- Terrastore [https://code.google.com/p/terrastore/],
- OrientDB [http://www.orientechnologies.com/],
- RavenDB [http://ravendb.net/],