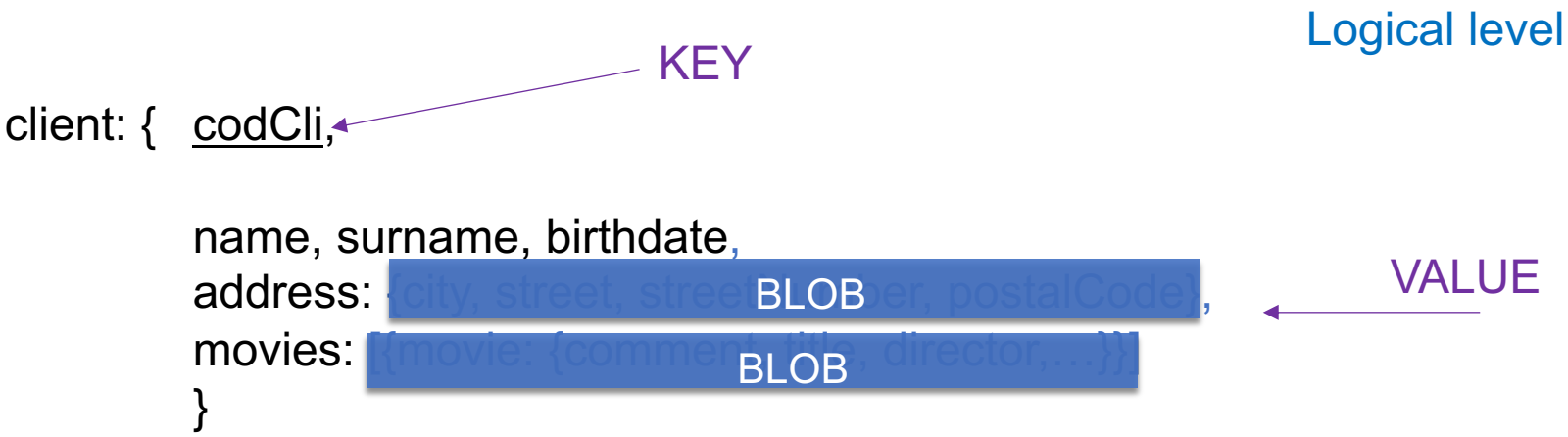
Column-family NoSQL data stores

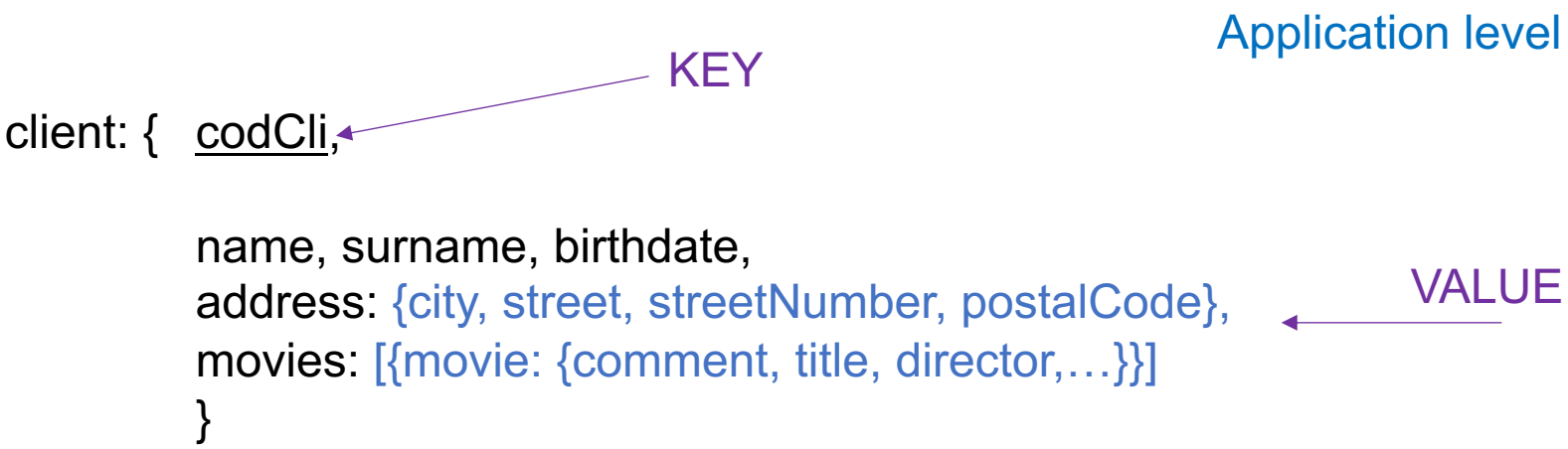
Column-family data model at a glance

- Each data instance is represented in the form (key, value)
  - key is an identifier
  - value is the aggregate, corresponding to a set of pairs (column-key, column-value)
- Colum-value can be either an atomic value or a complex one
- In this second case, the value structure is visible only at the application level, it is opaque (a blob) at the logical value
- Data instances can be grouped into logical collections



Column-family 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 set of pairs (column-key, column-value)
- Colum-value can be either an atomic value or a complex one
- In this second case, the value structure is visible only at the application level, it is opaque (a blob) at the logical value
- Data instances can be grouped into logical collections



Column-family data model at a glance

- Columns can be organised into families: sets of related data (columns) that are often accessed together

Logical level

Client
Family: info: {codCli,

name, surname, birthdate,
address: {city, street, streetNumber, postalCode},

}

Column family associated with client personal information

BLOB

Client
Family:movies: {codCli,

movies: [{movie: {comment, title, director, ...}}]

}

Column family associated with client movie recommendations

BLOB

Column-family data model at a glance

- Columns can be organised into families: sets of related data (columns) that are often accessed together

Application level

Client
Family: info: {codCli,

Column family associated with client personal information

name, surname, birthdate,
address: {city, street, streetNumber, postalCode},

}

Client
Family:movies: {codCli,

Column family associated with client movie recommendations

movies: [{movie: {comment, title, director,…}}]
}

Instance structure

- Limited schema information: column-family names + column-names inside column-families
- Nested values are not visible at the NoSQL system level but only at the application level

6

Collections

- Pairs can be grouped into logical collections/namespaces: sets of aggregates, i.e., sets of (key, value) pairs sharing the same column-family
- Each collection becomes a sort of table

Client
Family: info: {codCli,

name, surname, birthdate,
address: {city, street, str BLOB ber, postalCode},

}

Client
Family: info

name	surname	birthdate	address

Collections and schema information

- Schema information can be provided either when inserting the data instances (DML)or before (DDL)
- Inside the same collection, the chunk of data corresponding to a column family might correspond (depending on the specific system) to different sets of column-names for each row
- Flexibility
- Columns must however belong to the same column family

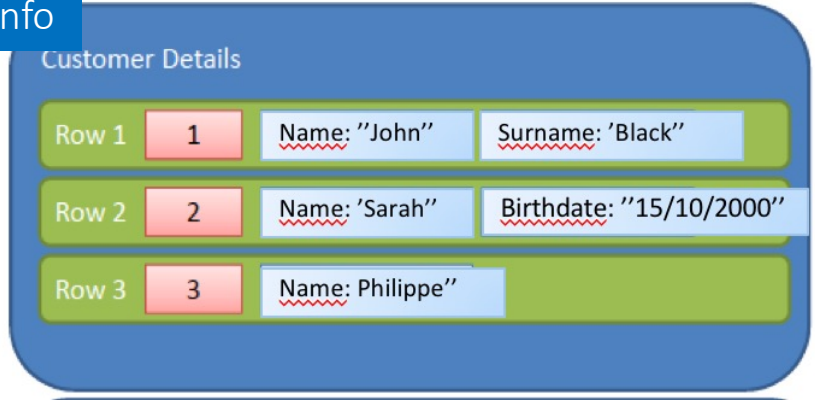
Client
Family: info: {codCli,

name, surname, birthdate,
address: {city, street, streetNumber, postalCode},

}

BLOB

client
Family: info



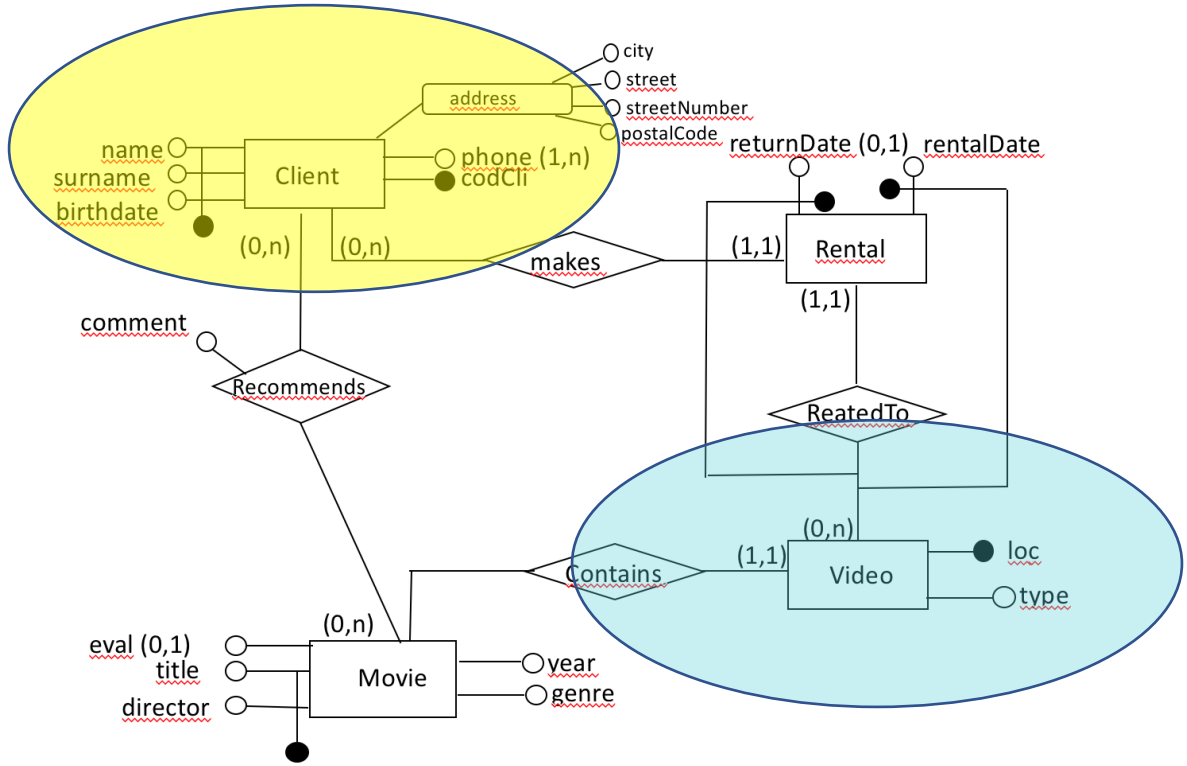
Keys

- (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)
- At the physical level, the key tells the system how to partition the data and where to store the data
  - partition key
- Usually, the partition key is a subset of or is equal to the primary key
- Aggregates can be directly retrieved only by specifying values for attributes in the partition key
  - primary key attributes can be used for retrieval together with the partition key, under some restrictions

Keys

- Usually, more than one attribute
- *Partition key subset of (or equal to) primary key (in bold in the examples)*
- The type and the content of the key is identified starting from the designed aggregates, taking into account the features of the system at hand



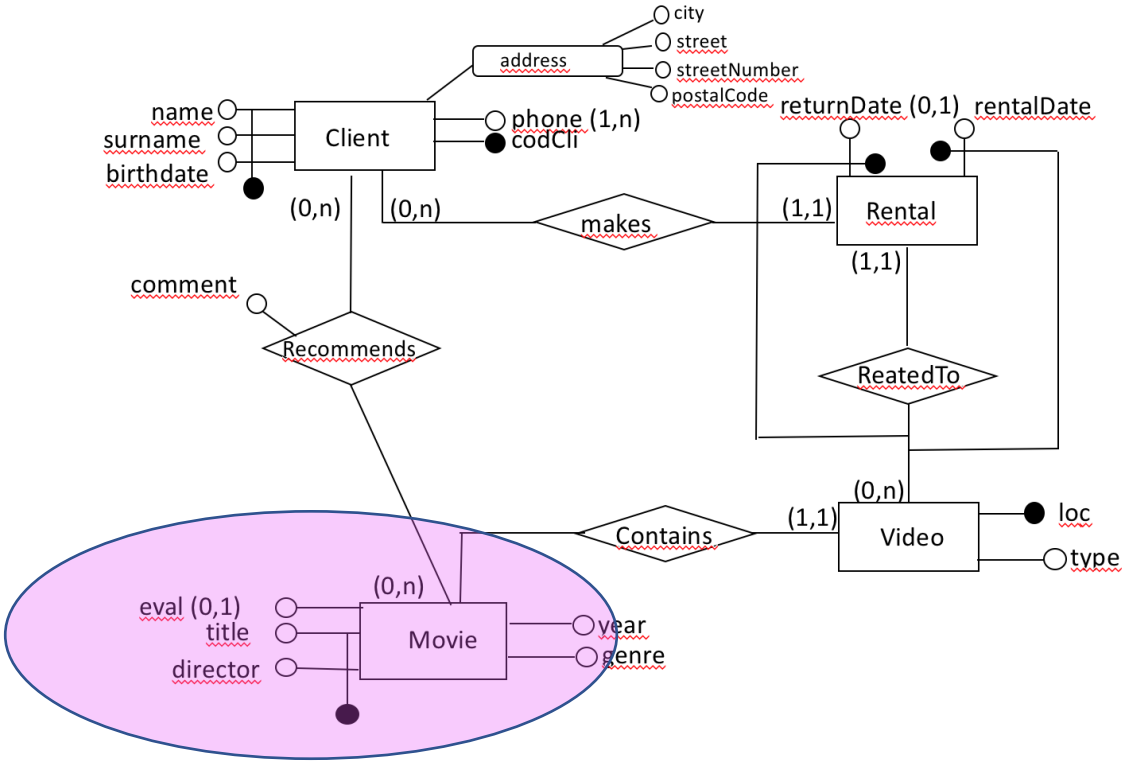
Client: {**codCli,** …}

primary key is codCli
(partition) key is codeCli

Video: {**loc,** …}

primary key is loc
(partition) key is loc

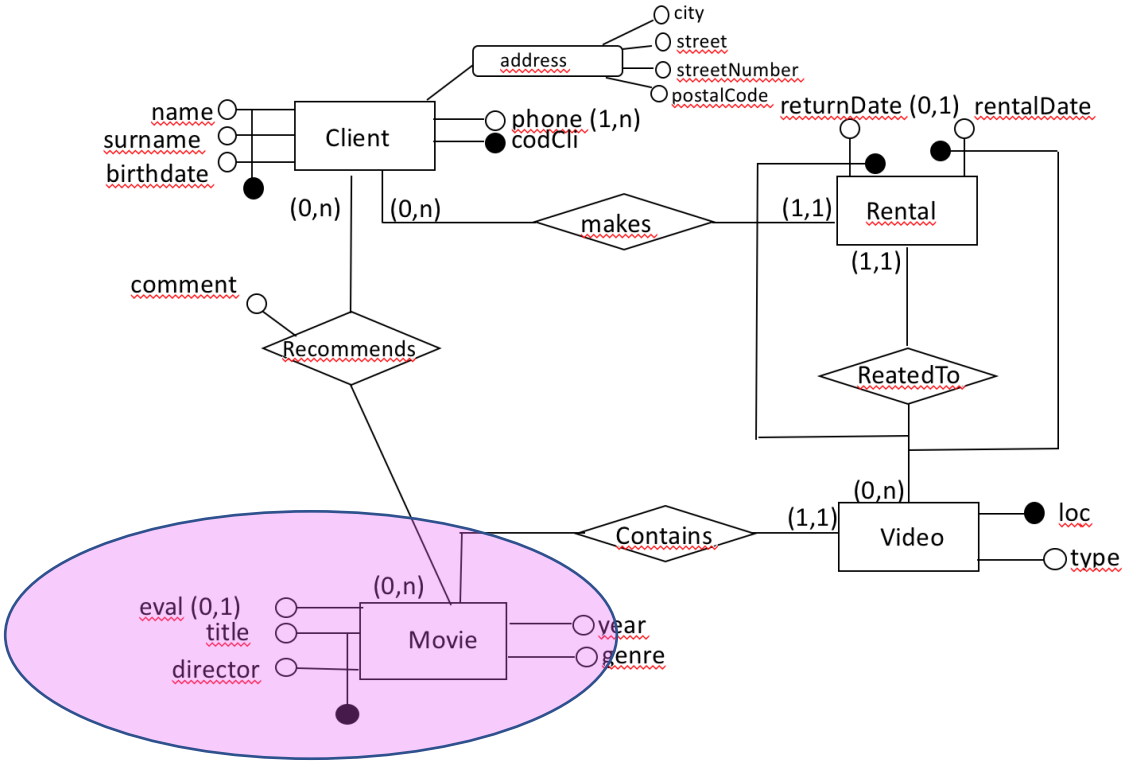
Keys



Movie : {**<u>title, director,</u>** …}

(partition) key could be either
(title, director)
title
director
→ we select (title, director) (in bold)

Keys



Movie : {**title, director**, …}

(partition) key could be either
(title, director)
title
director
→ we selected (title, director) (in bold)

Movie aggregate from our modeling example:

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

Identifier vs partition/sharding key

- In the Videos collection, videos are partitioned by loc
- In the Clients collection, clients are partitioned by codCli
- In the Movies collection, movies are partitioned by (title, director)

- This impacts the way data are stored
- This impacts the way data can be retrieved

Video Rental Example

Logical level (references exists but they are opaque in the system)

client: { **codCli**,

name, surname, birthdate,
address: BLOB,
movies: BLOB

]

}

REFERENCE

REFERENCE

movie: { **title, director**,

year, genre, recommended_by: BLOB
contained_in: BLOB

video: {**loc**,

type, rentals: BLOB title, director}}

Video Rental Example

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}}

Application level (references can be used at this level, by accessing the structure of opaque components)

Interaction

- Basic lookup based on the key and column name (of a given column family)
  - `void define(family)`
  - `void insert(key, family, columns)`
  - `columns get(key, family)`
  - `value get(key, family, column)`
  - `value get(key, family, column, value)`
  - `void put(key,family,column,value)`
- It is not possible to arbitrarily join data contained in different column-families (always possible at the application level)
- It is not possible to navigate the nested structure of the column-values
- In case collections are supported, collection name is also specified

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

Column family: All

video: {**loc**, type, rentals: [{rentalDate, codCli}], title, director}

```
{"loc": 1234,

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

```
value get(collection, key, family, column)
```

get(Videos, '1234', 'all', [title])

- loc is the partition key, we can retrieve data starting from it
- the title of the movie contained in the video can be directly retrieved from the data store

Example 2 – Find the videos containing a certain movie

Collection family: All

video: {**loc**, type, rentals: [{rentalDate, codCli}], title, director}

```
{"loc": 1234,
 "type":"dvd",
 "rentals": [{"rental": {"rentalDate":"15/10/2021",
                         "codCli": 375657}}],
 "title": "pulp fiction",
 "director": "quentin tarantino"}
```

```
value get(collection, key, column)
```

- Even if the title and director values are «visible» at the data store level, we have to retrieve data starting from the key
- We can only get all the videos, and
- At the application level, filter them by title and director

Example 3 – Find the videos rented by a certain client

Collection family: All

video: {**loc**, type, rentals: [{rentalDate, codCli}], title, director}

```
{"loc": 1234,
  "type":"dvd",
  "rentals": [{"rental": {"rentalDate":"15/10/2021",
                          "codCli": 375657}}],
  "title": "pulp fiction",
  "director": "quentin tarantino"}
```

```
value get(collection, key, family, column)
```

- Client data is «not visible» at the data store level
- At the data store level, we can only get all the videos, and
- At the application level, go inside video rentals and filter them by codCli

Example 2 – Find the videos containing a certain movie - with the movies collection

Collection family: All

movie: **{ title, director**,

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

```
value get(collection, key, family, column)
```

- get(Movies,'pulp fiction:quentin tarantino', 'all', contained_in)
- At the data store level, retrieve the information about the videos containing a certain movie (the whole blob)

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, 'all')

- At the data store level, find Video 1234 in collection Videos and retrieve all the information
- at the *application level*, we go inside the aggregate value and we discover that it it was rented by Client 375657
- at the *application level*, we can execute get(Clients, 375657, 'all', birthdate) to retrieve information about a/the customer birthdate that rented video 1234 (thus navigating the customer reference stored inside the video), and then the age is computed

Example 4 - Relationships

```
{"loc": 1234,

  "type":"dvd"
  "rentals": [{"rental": {"rentalDate":"15/10
                          "codCli": 375
  "title": "pulp fiction",
  "director": "quentin tarant
```

Find the age(s) of c) that rented a give

REMARKS:

1. Who knows that inside **rentals** there is the code of the client that rented the video?
   And where to look for the client information and how to match?

The application, the data store is completely unaware of that!

2. Since the relationship is not part of the aggregate, navigating it requires a sort of join, no system support for it

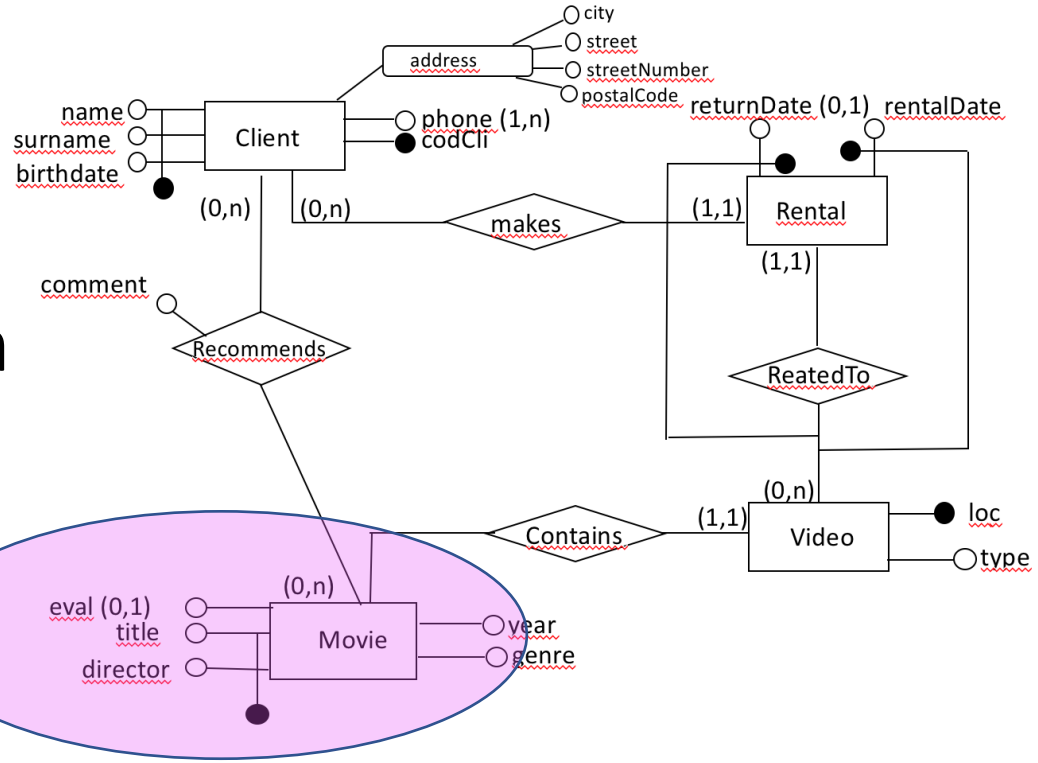
3. Since the relationship is not part of the aggregate, navigating it (at application level) requires two distinct data accesses, at two possibly distinct data nodes

- deos and retrieve
- ggregate value and we discover 657
- an execute get(Clients, 375657, 'all',
  bi e information about a/the customer birthdate that ren 1234 (thus navigating the customer reference stored inside the video), and then the age is computed

Advanced interaction

- Some column-family systems support SQL-like languages
- Joins are never supported
- *Only queries for which the set of nodes containing relevant data can be determined by the system through the partition key in advance are executed*
  - A value for the partition key has always to be specified
- The execution of other queries (but not joins) can however be forced or admitted through the creation of indexes

Keys
physical representation



Movie : {'**title,** director …}

primary key is (title, director)
(partition) key is title

movie: { **title**,

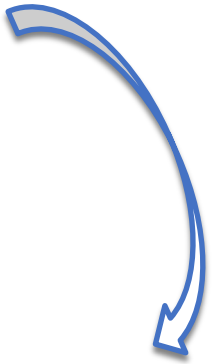
director, year, genre, recommended_by: [{name, surname, comment}],
contained_in: [{video: {loc, type}}] }

Keys physical representation

movie: { **title**,

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

title	director	year	genre	Recommended_by	Contained_in
A	B	1975	comedy	…	…
A	C	2010	comedy	…	…



title	**B:year**	**B:genre**	**…**	**C:year**	**C:genre**	**C:…**
A	1975	comedy		2010	comedy	

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

movie: { **title**,

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

(partition) key is title
primary key is (title, director)

SELECT contained_in
FROM Movies
WHERE title = 'pulp fiction' AND director = 'quentin tarantino'

- An equality condition with respect to the partition key has been specified, the query is admitted
- Single operation at the system level retrieves information about the videos containing a certain movie

Example – Find the videos containing a movie of a certain director - using the movies collection

movie: { **title**,

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

(partition) key is title
primary key is (title, director)

SELECT contained_in
FROM Movies
WHERE director = 'quentin tarantino'

- This query is not admitted because no partition key value (title) is specified

Popular column-family data stores

HYPERTABLE

Google Bigtable