From the aggregate-oriented logical schema to MongoDB logical/physical schema

Input

- The aggregate-oriented logical schema in meta-notation
- The annotated ER diagram
- One NoSQL system S (for today, MongoDB)

Output

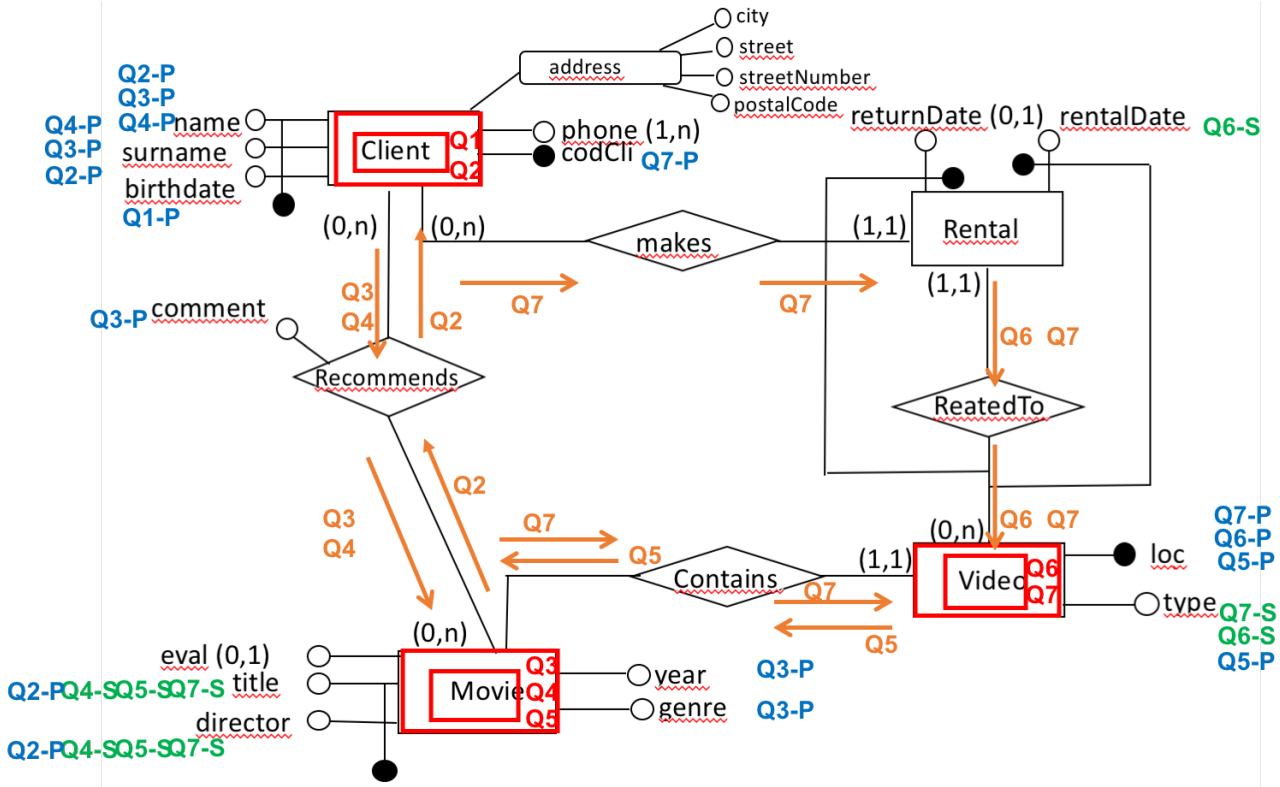
- A schema for S (in metanotation + + partition keys + indexes)

ASSUMPTION: all selection conditions are equality based

Issues

- From the aggregate-oriented logical schemas in meta-notation, the corresponding aggregation entity and the set of associated queries to a set of collections for S
- Each collection allows one subset of the queries to be executed
- From the set of selection attributes and the identifiers of the aggregation entity to the partition key and indexes

Input



- client: {<u>name, surname, birthdate</u>, recommends: [{title, director}]}
  - Q1, Q2
- movie: {<u>title, director</u>, year, genre, recommended_by: [{name, surname, comment}], contained_in: [{loc, type}]}
  - Q3, Q4, Q5
- video: {<u>loc</u>, type, rentals: [{rentalDate, codCli}], title, director}
  - Q6, Q7

Design in MongoDB

Remarks

- The identifier in Mongodb always is the _id field (ObjectId type)
  - Automatically assigned
  - Monotonically increasing
  - Can be assigned at document insertion, but cannot be updated later
- The _id field is automatically indexed
- Partition/shard keys must be indexed

(in what follows, we use the term partition key for uniformity with aggregates, in Mongodb read shard key)

Aggregation entity Client

client: {name, surname, birthdate, recommends: [{title, director}]}

Q1. Average age of clients
Q2. Name and surname of clients and related recommended movies

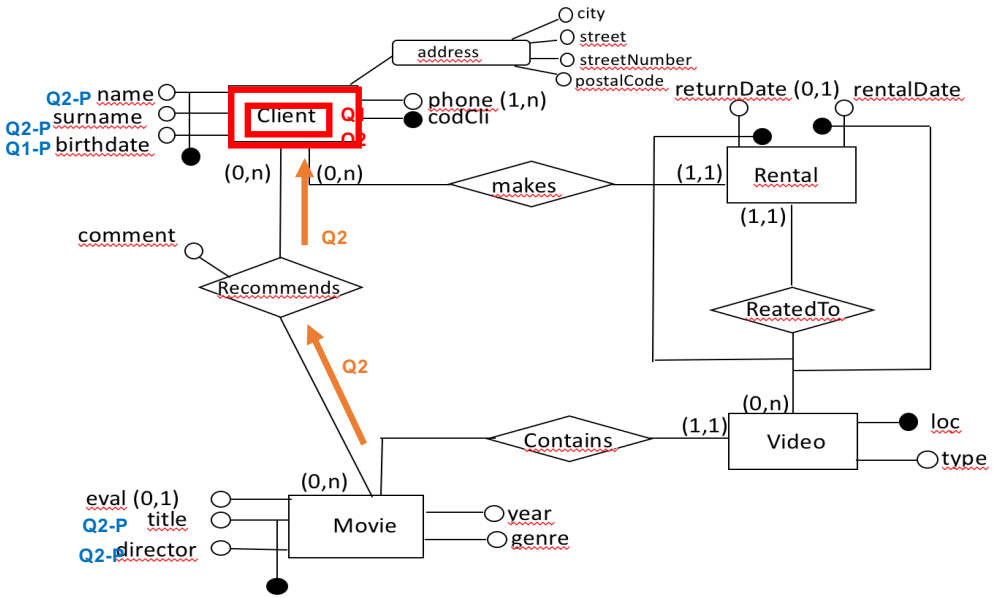
Queries associated with Client: Q1, Q2

Selection attributes for Q1: { }
Selection attributes for Q2: { }
Projection attributes for Q1: {birthdate}
Projection attributes for Q2: {name, surname, title, director}

No selection attribute → no need for a specific partition key

_id field automatically assigned could be partition key as well

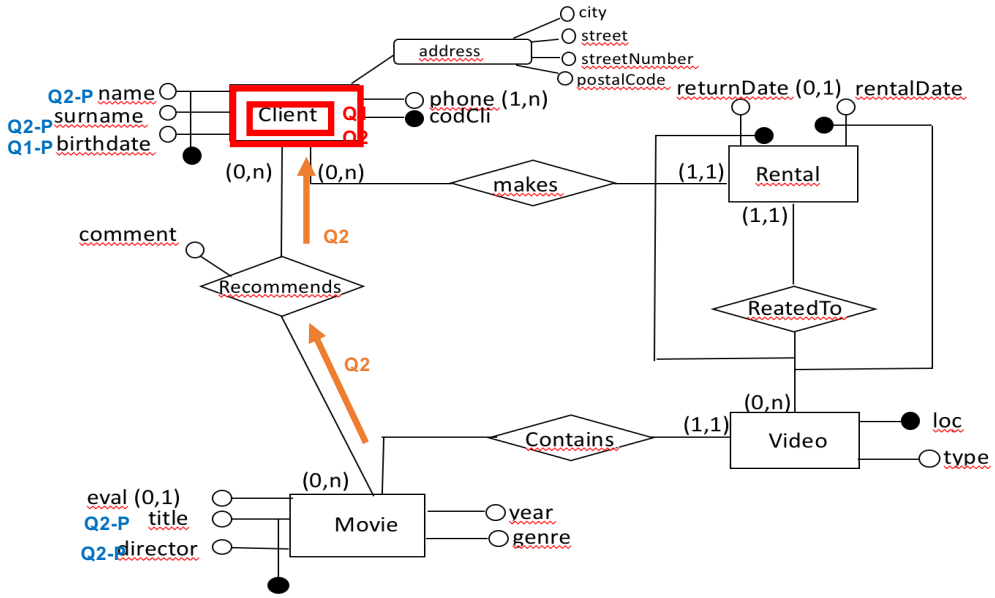
Other partition keys: whatever you want (name)



Aggregation entity Client

client: {_id, name, surname, birthdate, recommends: [{title, director}]}

Q1. Average age of clients
Q2. Name and surname of clients and related recommended movies



db.clients.createIndex({''name": 1, ''surname'':1, ''birthdate'':1}, {unique: true})

db.clients.createIndex({''name": 1}, {unique: false})
db.adminCommand({ shardCollection: "db.clients", key: { name: 1 },
field: "hashed" })

(non unique index on ''name'' automatically created if clients is empty)

Aggregation entity Movie

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

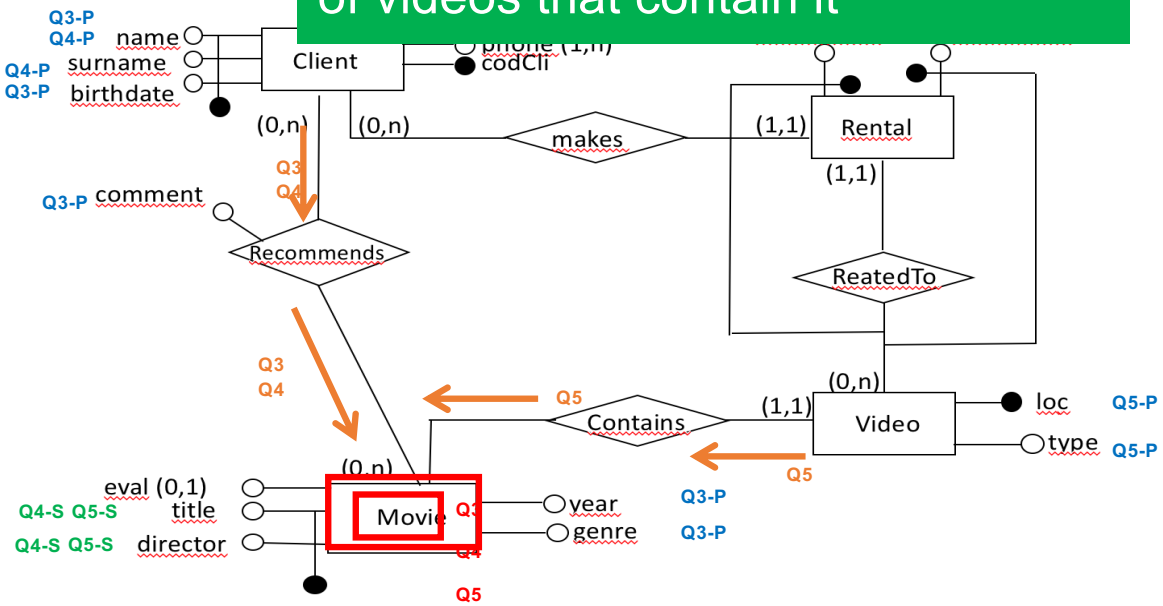
Queries associated with Movie: Q3, Q4, Q5

Selection attributes for Q3: { }
Selection attributes for Q4: { title, director }
Selection attributes for Q5: { title, director }

Q3. Genre and year of the movies and their related recommendations, together with the name and the surname of the client who made them
Q4. Name and surname of clients who recommended the movie 'pulp fiction' by 'quentin tarantino'
Q5. Given a movie, all information of videos that contain it

Partition key { title, director }
(with unique index)
hashed

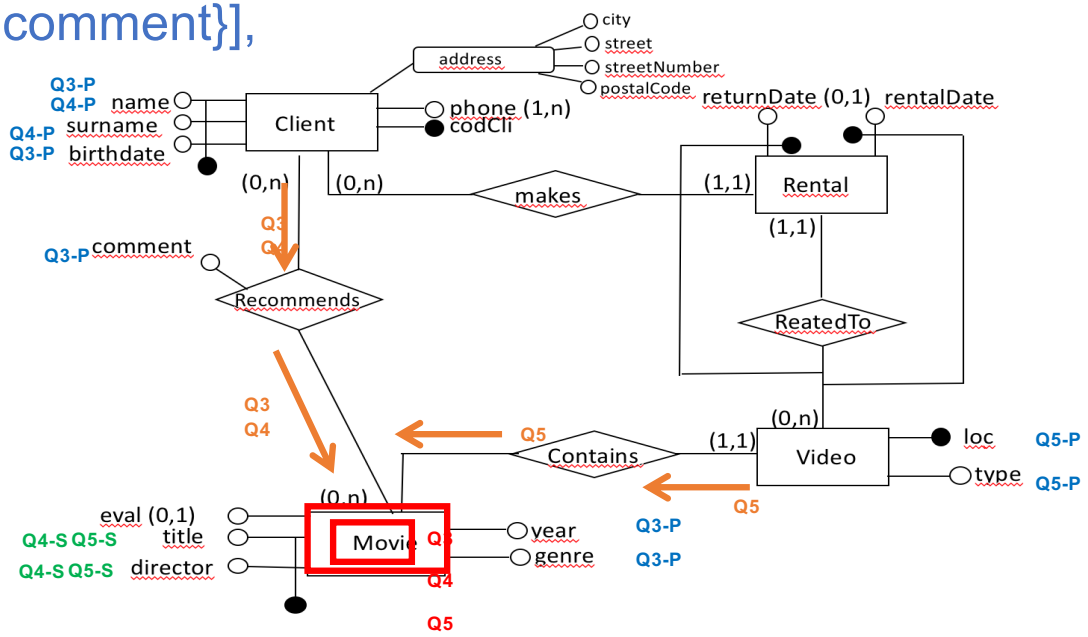
_id field automatically assigned
could be partition key as well



Aggregation entity Movie

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

Q3. Genre and year of the movies and their related recommendations, together with the name and the surname of the client who made them
Q4. Name and surname of clients who recommended the movie 'pulp fiction' by 'quentin tarantino'
Q5. Given a movie, all information of videos that contain it



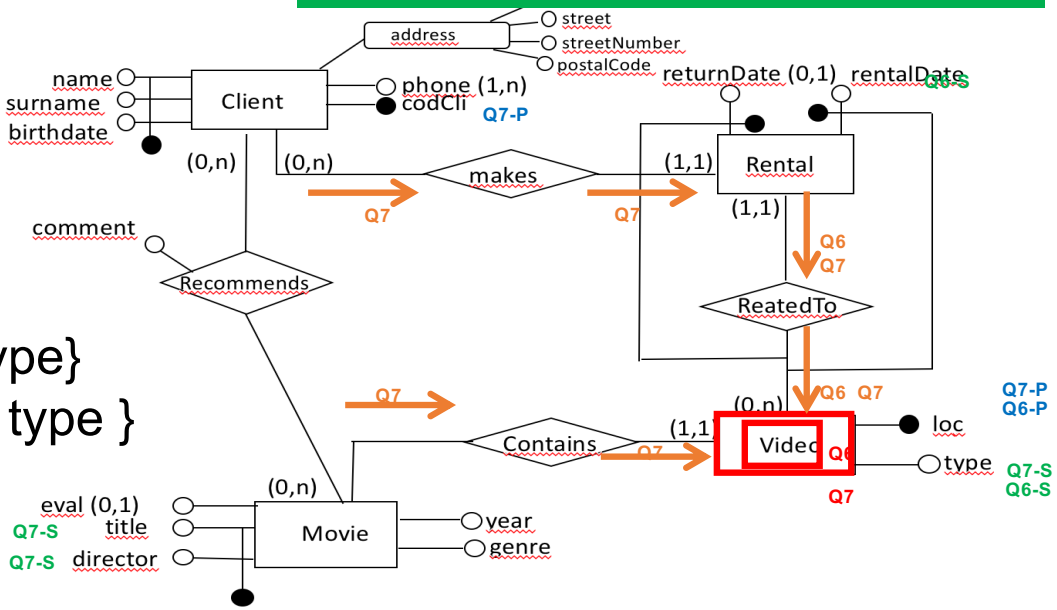

```
db.movies.createIndex( {"title": 1, "director":1}, {unique: true})
db.adminCommand( { shardCollection: "db.movies",
                   key: {"title": 1, "director":1},
                   field: "hashed" } )
```

(non unique index on {title, director} automatically created if movies is empty)

Aggregation entity Video

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

Q6. Videos of type 'DVD', rented from a certain date
Q7. The videos of type 'VHS' containing the movie 'pulp fiction' by 'quentin tarantino' and the clients that rented them



Queries associated with Video: Q6, Q7

Selection attributes for Q6: { rentalDate, type}
Selection attributes for Q7: { title, director, type }

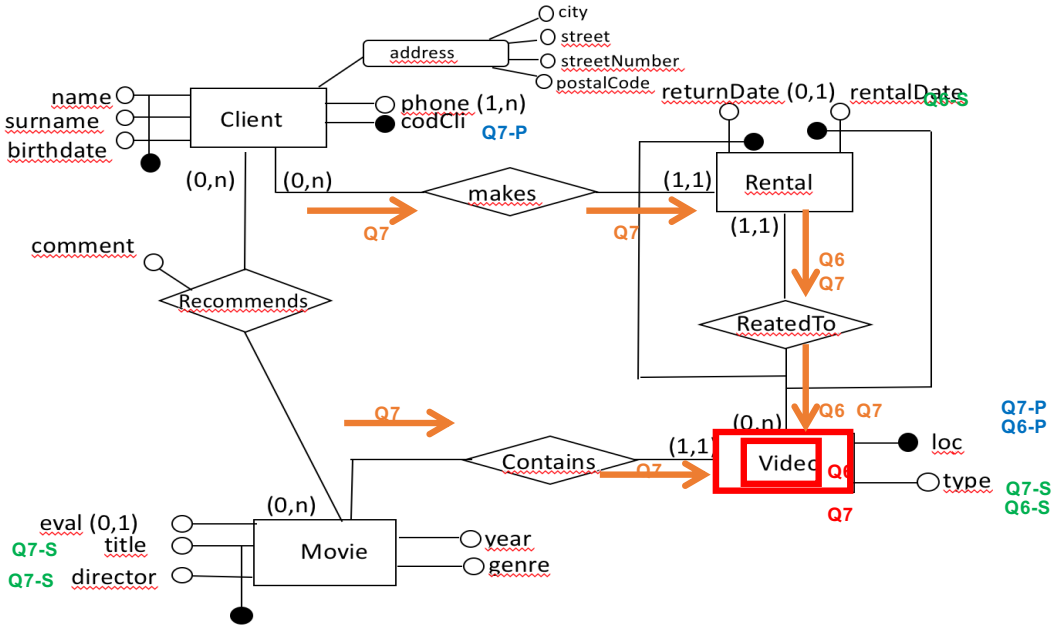
type appears in both → it becomes the partition key (an equality is always specified)

Partition key = { type }

A non-unique index has to be created on the partition key

Aggregation entity Video

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



What about the unique index on loc?

MongoDB does not support unique indexes across shards, except when the unique index contains the full shard key as a prefix of the index.

If you don't have an index on the shard key, you should at least have a compound index that starts with the shard key.

Unique index on type, loc

Index rules

- A unique index on _id is always created by the system
- All sharded collections must have an index that supports the shard key. The index can be:
  - an index on the shard key or
  - a compound index where the shard key is a prefix of the index
- If sharding is executed on an empty collection, a non-unique index is automatically created on the shard key
- For a ranged sharded collection, only the following indexes can be unique:
  - the index on the shard key
  - a compound index where the shard key is a prefix
  - the default _id index; however, the _id index only enforces the uniqueness constraint per shard if the _id field is not the shard key or the prefix of the shard key.

Index rules

- Although you can have a unique compound index where the shard key is a prefix, if using unique parameter, the collection must have a unique index that is on the shard key.
- You cannot specify a unique constraint on a hashed index
- The decision on whether an index must be unique or non-unique depends on the identifiers detected during the aggregate modeling