Tecnologie NoSQL

ALBERTO BELUSSI

ANNO ACCADEMICO 2018/2019

Approcci NoSQL

- **Sistemi Key-Value**: tutti i dati sono rappresentati per mezzo di coppie (chiave, valore), dove la chiave identifica le istanze e valore può avere una qualsiasi struttura anche non omogenea nella collezione.
- **Sistemi Document-Store**: i dati sono rappresentati come collezioni di documenti (oggetti con struttura complessa). Ogni oggetto della collezione ha una struttura complessa (con dati incapsulati) senza schema fisso. Consentono la definizione di indici secondari su proprietà degli oggetti (attributi comuni). Esempi di questo approccio sono MongoDB e CouchDB.
- **Sistemi Extensible-Record-Store**: i dati sono rappresentati da collezioni di record, dette tabelle, che possono essere nidificate e a struttura variabile (BigTable di Google, HBase, HyperTable)

Sistemi document-store

Modelli dei dati dei sistemi Document-store

Le caratteristiche principali di tali modelli sono:

- I dati sono rappresentati da collezioni di oggetti (detti, documenti).
- Gli oggetti hanno struttura complessa (non sono tuple piatte ma contengono dati nidificati).
- La nidificazione (encapsulation) dei dati è una aspetto chiave di questi modelli.
- Gli oggetti di una collezione non devono avere necessariamente tutti la stessa struttura; tuttavia dovrebbero condividere un nucleo di proprietà comuni.

Sistemi document-store: modello dati

Progettazione dei dati come documenti

La decisione chiave nel processo di progettazione dei dati usando questo nuovo approccio riguarda: la struttura dei documenti e in particolare la rappresentazione delle relazioni (legami) tra i documenti.

Esistono in particolare due approcci per rappresentare le relazioni tra documenti:

- Attraverso riferimenti
- Attraverso documenti nidificati

Sistemi document-store: modello dati

L'uso di riferimenti per rappresentare i legami tra documenti porta a schemi normalizzati.

Ad esempio, in MongoDB:

Student document

```
{
_id: "ID-001"
name: "Mario"
surname: "Rossi"
}
```

Exam document

```
{
_id: "ID-991"
student_id: "ID-001"
course: "Basi di dati"
grade: 25
}
```

Contact document

```
{
_id: "ID-992"
student_id: "ID-001"
email: "mr@aaa.bb"
}
```

Sistemi document-store: modello dati

L'uso di documenti nidificati porta a schemi non normalizzati e parzialmente ridondanti, ma molto efficienti in lettura.

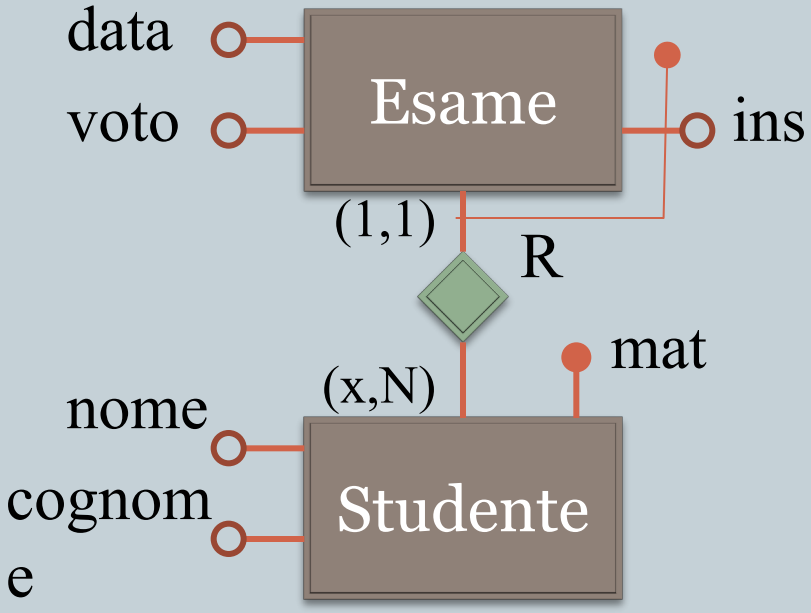
Ad esempio, in MongoDB: **Student document**

```
{
_id: "ID-001"
name: "Mario"
surname: "Rossi"
exams: {course: "Basi di dati"
        grade: 2 }
contacts: {email: "mr@aaa.bb"
           tel: 1234567 }
}
```

Sistemi document-store: modello dati

- L'approccio normalizzato consente di rappresentare il dato in modo simile a quanto avviene nel modello relazionale. Non c'è ridondanza e gli aggiornamenti sono più semplici (riguardano sempre una sola istanza).
- L'approccio embedded consente di rappresentare in un'unica istanza quanto viene rappresentato di solito nel modello ER con una entità insieme alle sue entità deboli. Vale a dire posso rappresentare nello stesso documento un oggetto con altre istanze di informazione in esso logicamente contenute. In questo caso il documento può essere aggiornato come oggetto complesso in modo atomico.

Sistemi document-store: esempio



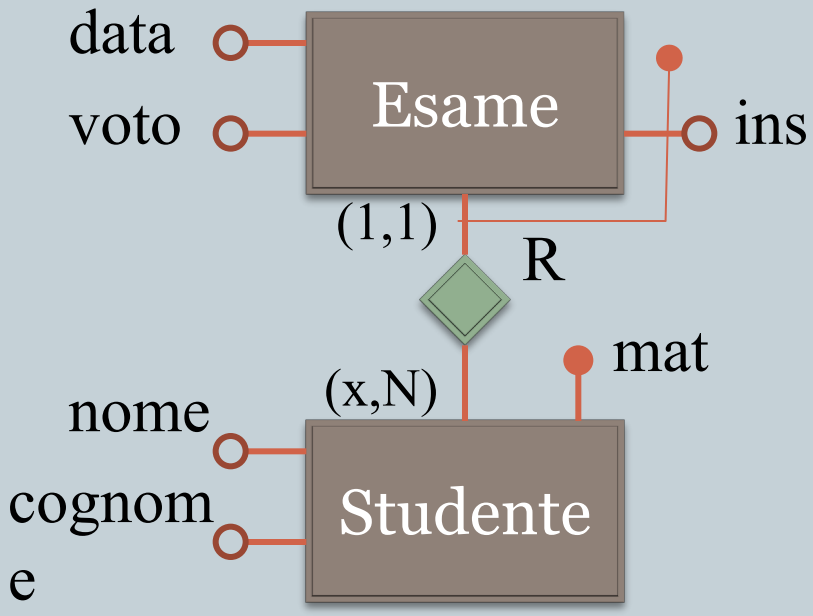
Modello relazionale

Esame(<u>ins, mat</u>, data, voto)

R

Studente(<u>mat</u>, nome, cognome)

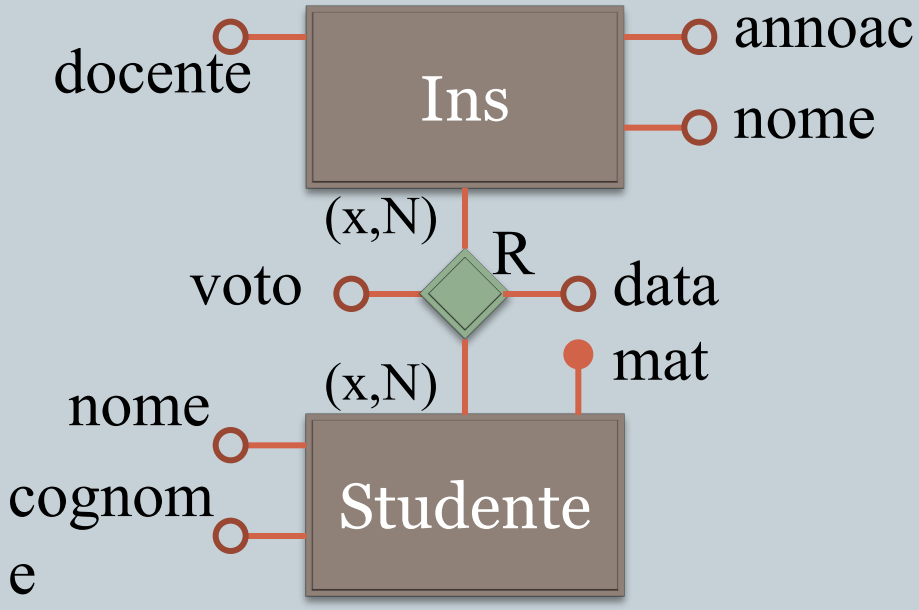
Sistemi document-store: esempio 1



MongoDB

```
{
 _id: "VR00010"
 nome: "Mario"
 cognome: "Rossi"
 esami: [
      {ins: "Basi di dati"
       voto: 22
       data: "1/7/2016" },
      {ins: "Algebra"
       voto: 26
       data: "5/7/2015" }
      ]
}
```

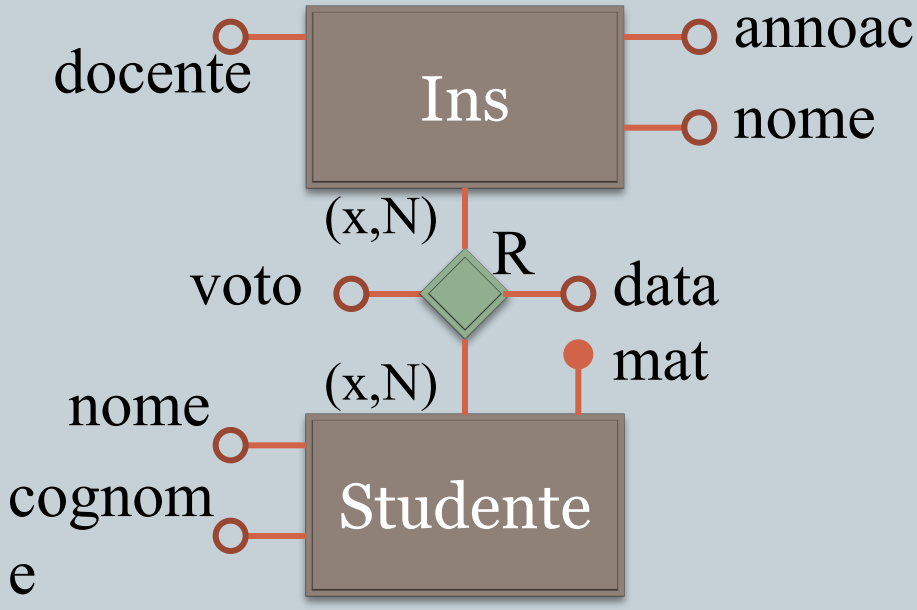
Sistemi document-store: esempio 2.1



MongoDB

```
{_id: "VR00010"
 nome: "Mario"
 cognome: "Rossi"
 esami: [{ins_id: "ins01"
          voto: 22
          data: "1/7/2016" },
         {ins_id: "ins02"
          voto: 26
          data: "5/7/2015" }]
}
{_id: "ins01"
 nome: "Basi di dati"
 annoac: "2016/2017"
 docente: "Belussi"}
{_id: "ins02"
 nome: "Algebra"
 annoac: "2015/2016"
 docente: "Gregorio"}
```

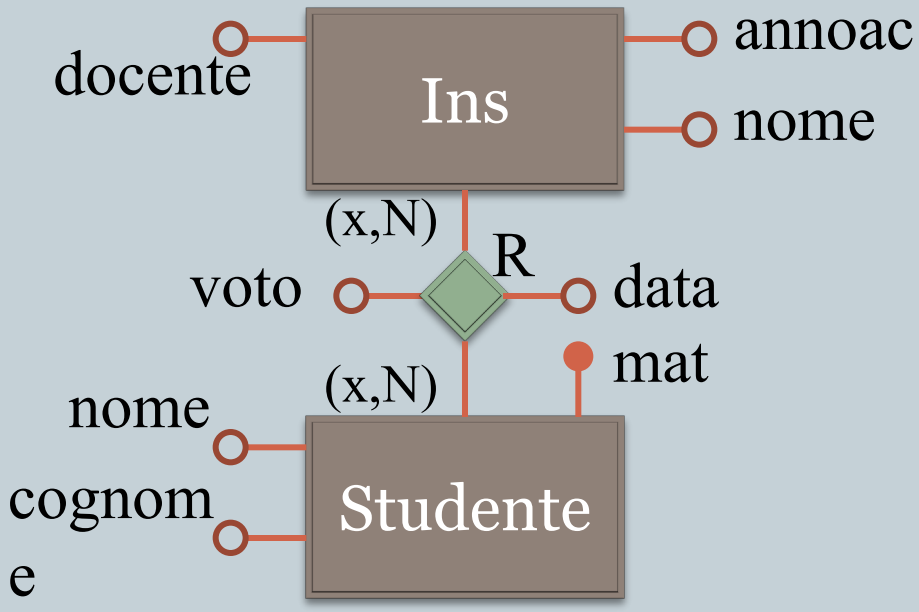
Sistemi document-store: esempio 2.2



MongoDB

```
{_id: "VR00010"
 nome: "Mario"
 cognome: "Rossi"
}
{_id: "ins01"
 nome: "Basi di dati"
 annoac: "2016/2017"
 docente: "Belussi"
 esami: [{stud_id: "VR00010"
          voto: 22
          data: "1/7/2016" }]}
{_id: "ins02"
 nome: "Algebra"
 annoac: "2015/2016"
 docente: "Gregorio"
 esami: [{stud_id: "VR00010"
          voto: 26
          data: "5/7/2015" }]}
```

Sistemi document-store: esempio 2.3



MongoDB

```
{_id: "VR00010"
 nome: "Mario"
 cognome: "Rossi" }
{_id: "ins01"
 nome: "Basi di dati"
 annoac: "2016/2017"
 docente: "Belussi" }
{_id: "ins02"
 nome: "Algebra"
 annoac: "2015/2016"
 docente: "Gregorio"}
{_id: "01" stud_id: "VR00010"
 ins_id: "ins01"
 voto: 22
 data: "1/7/2016" }
{_id: "02" stud_id: "VR00010"
 ins_id: "ins02"
 voto: 26
 data: "5/7/2015" }
```

Sistemi document-store: esempio

- Quale linguaggio di interrogazione è disponibile in questi sistemi?

Buona parte del linguaggio di interrogazione è realizzato attraverso il metodo `find`.

Vediamo alcune interrogazioni sull'esempio 1.

```
db.studenti.insertMany([
{_id: "VR00010", nome: "Mario", cognome: "Rossi"
 esami: [{ins: "Basi di dati", voto: 22, data: "1/7/2016" },
    {ins: "Algebra", voto: 26, data: "5/7/2015" }]}
{_id: "VR00011", nome: "Maria", cognome: "Bianchi"
 esami: [{ins: "Algebra", voto: 30, data: "1/7/2016" }]}
]);
```

Sistemi document-store: interrogazioni

Interrogazioni semplici:

- Trovare tutti gli studenti:

```
db.studenti.find( {} )
```

- Trovare tutti gli studenti di cognome “Rossi”:

```
db.studenti.find( {cognome: “Rossi”} )
```

- Trovare tutti gli studenti di cognome “Rossi” e nome “Mario”:

```
db.studenti.find( {cognome:“Rossi”, nome:“Mario”} )
```

Sistemi document-store: interrogazioni

Interrogazioni su dati incapsulati

- Trovare tutti gli studenti che hanno registrato almeno un esame con voto 30:

```
db.studenti.find( {“esami.voto”: 30} )
```

- Trovare tutti gli studenti che hanno registrato almeno un esame con voto maggiore di 22:

```
db.studenti.find( {esami.voto: { $gt : 22 } } )
```

Sistemi document-store: interrogazioni

Interrogazioni con proiezione

- Trovare il cognome degli studenti di nome "Mario":

```
db.studenti.find( {nome: "Mario"}, {cognome: 1} )
```

- Trovare il cognome degli studenti di nome "Mario" escludendo il campo _id:

```
db.studenti.find( {nome: "Mario"},
                  {cognome: 1, _id: 0} )
```

- Trovare la matricola e i voti di tutti gli studenti:

```
db.studenti.find( { }, {_id: 1, "esami.voto": 1} )
```

Sistemi document-store: interrogazioni

Interrogazioni con join (non consigliate dal sistema)

Per eseguire un join tra due collezioni di documenti è necessario usare la funzione `$lookup`

Interrogazione sull'esempio 2.3

```
db.studenti.aggregate([
{ $lookup:
      { from: esami,
        localField: _id,
        foreignField: stud_id,
        as: "esami_fatti"
      }
}
])
```

Sistemi document-store: interrogazioni

Interrogazioni con join

Risultato dell'interrogazione precedente:

```
{_id: "VR00010",
 nome: "Mario",
 cognome: "Rossi",
 "esami_fatti": [
  {_id: "01",
   stud_id: "VR00010",
   ins_id: "ins01",
   voto: 22,
   data: "1/7/2016" },
  {_id: "02", stud_id: "VR00010",
   ins_id: "ins02",
   voto: 26,
   data: "5/7/2015" } ]
}
```