SQL vs. NoSQL
Really: Relational vs. Non-Relational

	Relational (RDBMS)	Non-Relational
Data	Table-oriented	Document-oriented, key-value, graph-based, column-oriented, …
Schema	Fixed schema	Dynamic schema
Joins	Used extensively	Used infrequently
Interface	SQL	Custom query language
Transactions	ACID	BASE?

```
SELECT * FROM people
WHERE age > 25;
```

```
db.people.find(
  { age: { $gt: 25 } }
)
```

MongoDB vocabulary

MongoDB instance

Has 0+

Databases

Has 0+

Collections — Analogous to RDBMS tables

Contains 0+

Documents — Analogous to RDBMS rows

Are BSON objects with 0+

Fields — Analogous to RDBMS columns, Like JS properties

Index
Optimized lookup tables for specific fields (e.g. tree)

Cursor
Iterator into the result set that can obtain a few documents at a time

Flexible schema

Noun/Model, e.g. "Address" ⇔ Collection

Consider an "Address Book":

Student	cell, email, mailbox number, dorm room, …
Faculty	cell, email, office phone, office number, …
Office (e.g. Registrar)	email, office phone, office number, …

Some common fields, but many differences

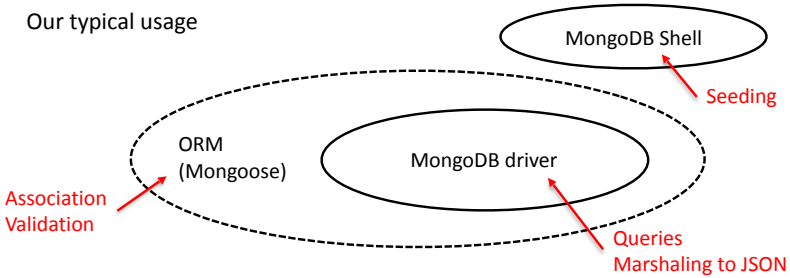
Document-oriented storage gives the flexibility to store just the exact information needed for each object

Example queries

"Raw" queries

```
db.collection.find({field: {  predicate }}, { fields });
db.collection.insertOne({ … });
db.collection.updateMany(
    { field: {  predicate } },
    { $set: { fields } }
);
```

Our typical usage



Recall: CRC cards model of Movies

User	
Responsibility	**Collaborator**
Knows user's name	
…	
Knows movies I rated	Rating

Movie	
Responsibility	**Collaborator**
Knows its title	
Knows its plot overview	
…	
Know which genres it is	Genre

Rating	
Responsibility	**Collaborator**
Knows rating	
Knows its owner	User
Knows its movie	Movie

Genre	
Responsibility	**Collaborator**
Knows its descriptor	

"many to many"

"one to many"

*Kent Beck & Ward Cunningham, OOPSLA 1989

Genres: One-to-few

Internal Mongo "id" analogous to primary key

```
> db.movies.findOne()
{
  "_id" : ObjectId("5a69eb43a1e7248f699794aa"),
  "id" : 135397,
  "title" : "Jurassic World",
  …
  "genre_ids" : [28, 12, 878, 53]
}
```

"Embedded" array

How to query?

```
// All movies with genre_id 28
db.movies.find({ genre_ids: 28 })

// All movies with genre_id 28 and 12
db.movies.find({ genre_ids: { $all: [28, 12] } })
```

What if genres are unbounded, i.e. "one-to-very many"?

Users ⇔ Ratings ⇔ Movies?

Should we embed movies in users (or vice versa)?

No. Models on both sides needs to stand-alone.

Do we need ratings from both users and movies?

Yes, e.g. show "my" ratings or movie's ratings.

```
{
  "_id" : ObjectId("5a69…"),
  "title": "Jurassic World",
  …
  "ratings": [{
    user: ObjectId("13a4…"),
    rating: 4
  }]
}
```

```
{
  "_id" : ObjectId("13a4…"),
  "name": "Alice Midd",
  …
  "ratings": [{
    movie: ObjectId("5a69…"),
    rating: 4
  }]
}
```

De-normalized (copied) data

\+ Fast/easy to access ratings
\- Two queries to add rating
\- Slow to update existing rating
\- Potential for inconsistent state

Should we use RDBMS or MongoDB?

No right or wrong answer, just tradeoffs

Is your data:

- Highly relational? *+RDBMS*
- Highly polymorphic? *+MongoDB*

Does your application have:

- Complex queries? *+RDBMS*
- Strong data integrity requirements? *+RDBMS*

Getting started cost:

- Uncertain initial schema *+MongoDB*