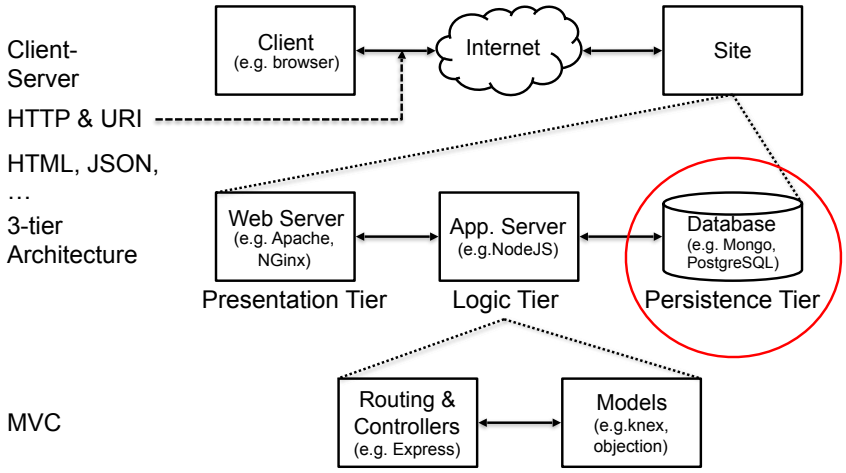


Why a database?


```
[{
    "adult": false,
    "backdrop_path": "/sEgULSEnywgdSesVHFHpPAbOijl.jpg",
    "genre_ids": [18, 12, 878],
    "id": 286217,
    "original_language": "en",
    "original_title": "The Martian",
    "overview": "During a manned mission to Mars, Astronaut Mark
 Watney is presumed dead after a fierce storm and left behind by
 his crew. But Watney has survived and finds himself stranded an
d alone on the hostile planet. With only meager supplies, he mus
t draw upon his ingenuity, wit and spirit to subsist and find a
way to signal to Earth that he is alive.",
    "release_date": "2015-10-02",
    "poster_path": "/AjbENYG3b8lhYSkdrWwlhVLRPKR.jpg",
    "popularity": 40.509541,
    "title": "The Martian",
    "video": false,
    "vote_average": 7.7,
    "vote_count": 447
  },
…
]
```

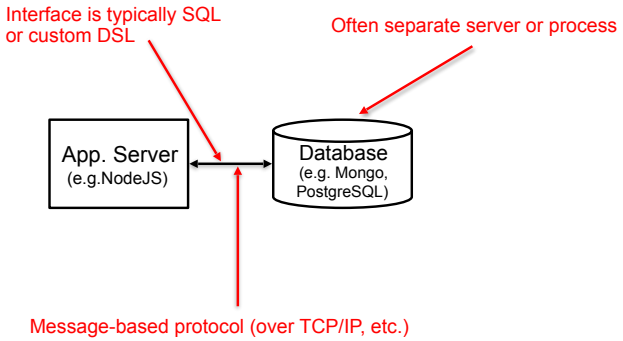

What's wrong with our approach of loading data from a JSON file and storing it in memory?

```
app.get('/api/films/:id', (request, response) => {
  response.send(films[request.params.id]);
});
```

Database Management Systems (DBMS)

- Efficient random access when total dataset is tool large to fit in memory
- Fast *and* complex queries (not fast *or* complex)
- Model relationships within the data
- Transactions and other forms of fault tolerance
- Security (and management tools)

Database client and server



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	CAP

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

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

RDBMS vocabulary

DB instance (e.g. PostreSQL)

Has 0+

Databases

Has 0+

Tables

Contains 0+

Rows

With 1+

Attributes/Columns

Each table has a schema with types, optional primary key, optional constraints

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

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

RDBMS mental model

Noun/Model, e.g. “Film” ⇔ Table
Model attributes, e.g., “title” ⇔ Columns

Schema (name and type)

Film table

id	title	overview	release_date	poster_path	vote_average	rating
int	string	text	string	string	float	int
1	Star…	Princes…	1977-05-25	/tvSLB…	7.7	3
2	2001: A…	Huma…	1968-04-05	/90T7…	7.5	4

Primary key: Unique identifier for record (can be 1+ columns)

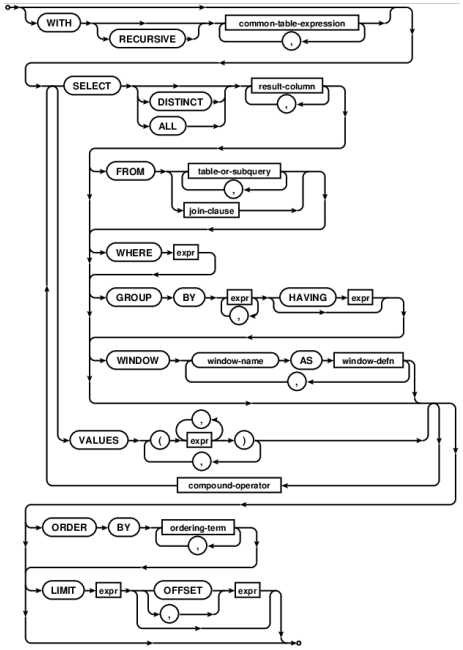
SQL statements

```
SELECT columns FROM table WHERE conditions;

INSERT INTO table(columns) VALUES (values);

UPDATE table SET column=value, … WHERE conditions;

DELETE FROM table WHERE conditions;


CREATE TABLE table (column Type, …);
DROP TABLE table;
```

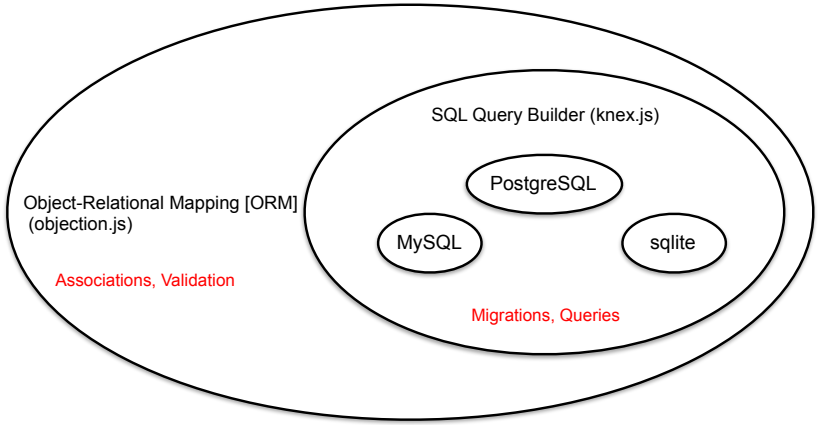
Example

```
SELECT title FROM Film WHERE rating >= 4;
```

SQLite SELECT Statement grammar



When in doubt, abstract!



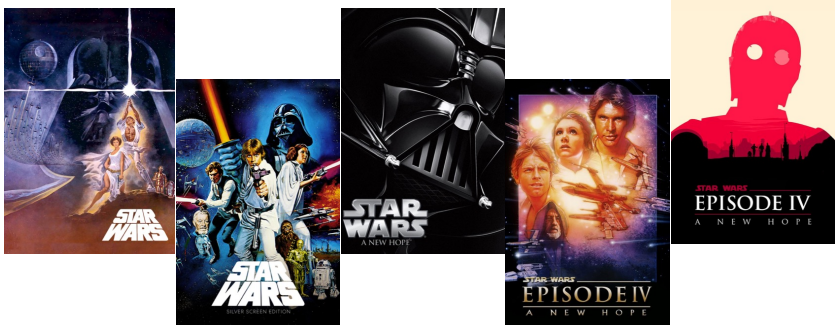
CRC cards to DB schema

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

Film				
id	**title**	**overview**	**poster**	**…**
11	Star Wars	"Princess Leia is captured…"	star-wars.jpg	
105	Back to the Future	"Eighties teenager Marty McFly…"	back.jpg	
…				

Example: Film posters

Film				
id	**title**	**overview**	**poster**	**…**
11	Star Wars	"Princess Leia is captured…"	star-wars.jpg	
105	Back to the Future	"Eighties teenager Marty McFly…"	back.jpg	
…				



Example: Film posters

Film				
id	**title**	**overview**	**poster**	**…**
11	Star Wars	"Princess Leia is captured…"	star-wars1.jpg	
11	Star Wars	"Princess Leia is captured…"	star-wars2.jpg	
11	Star Wars	"Princess Leia is captured…"	star-wars3.jpg	
11	Star Wars	"Princess Leia is captured…"	star-wars4.jpg	
11	Star Wars	"Princess Leia is captured…"	star-wars5.jpg	
105	Back to the Future	"Eighties teenager Marty McFly…"	back.jpg	
…				

Example: Film posters

Film				
id	**title**	**overview**	**poster**	**…**
11	Star Wars	"Princess Leia is captured…"	[star-wars1.jpg, star-wars2.jpg, star-wars3.jpg, star-wars4.jpg, star-wars5.jpg]	
105	Back to the Future	"Eighties teenager Marty McFly…"	back.jpg	
…				

Example: Film posters

Film				
id	**title**	**overview**	**poster**	**…**
11	Star Wars	"Princess Leia is captured…"	[{'path':'star-wars1.jpg, 'artist':'…', 'date':'…', …}, {'path':'star-wars2.jpg', …} …]	
105	Back to the Future	"Eighties teenager Marty McFly…"	back.jpg	
…				

Example: Film posters

Film			
id	**title**	**overview**	**…**
11	Star Wars	"Princess Leia is captured…"	
105	Back to the Future	"Eighties teenager Marty McFly…"	
…			

normalizing tables

primary key

Poster				
id	**filmId**	**path**	**year**	**…**
5	11	star-wars1.jpg	1977	
9	11	star-wars5.jpg	1992	
23	105	back.jpg	1985	
67	11	star-wars2.jpg	2002	
…				

foreign key

Example: Film posters

Database **Joins** are formed by **Cartesian Products**

```
SELECT * from Film, Poster WHERE Film.id  = Poster.filmId
```

Film x Poster				
Film.id	**…**	**Poster.id**	**Poster.filmId**	**…**
11		5	11	
11		9	11	
11		23	105	
11		67	11	
105		5	11	
105		9	11	
105		23	105	
105		67	11	

returned rows

Example: Film posters

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

one-to-many

Poster	
Responsibility	**Collaborator**
Knows its path	
Knows its date	
Knows its artist	
Knows its film	Film
…	

Example: Film Ratings

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

many to many
(or "has many-through")

Rating	
Responsibility	**Collaborator**
Knows rating	
Knows its owner	User
Knows its film	Film

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

Thinking in relations/associations

- "HasOne" / "BelongsToOne"
  One-to-one relationship, e.g. Supplier and Account
- "HasMany" / "BelongsToOne"
  One-to-many relationship, e.g. Film and Poster
- "ManyToMany"
  Many-to-many relationship (often called "has many through"), e.g. User and Film through Rating

Where do the foreign keys go?

- "HasOne" / "BelongsToOne"
  Foreign key typically in the "BelongsToOne" side (although could be reversed)
- "HasMany" / "BelongsToOne"
  Foreign key in "BelongsToOne" side (the "many" model)
- "ManyToMany"
  Foreign keys in join model, e.g. Rating in "User and Film through Rating"

True or False? There can only be one relationship between two models.

You are developing an application for a veterinarian's office. How would you model the relation between Customer and Animal?

A. One-to-one, e.g. "HasOne"
B. One-to-many, e.g. "HasMany"
C. Many-to-many, e.g. "HasManyThrough"

Specifying schema: Migrations

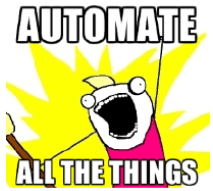
Customer data is critical! How do you evolve your application without destroying any data?

- Maintain multiple databases (e.g. test, development, production, …)
- Change schema/data with scripted ***migrations***

Migrations create/delete tables, add/remove/modify columns, modify data, etc.

Advantage of migrations:

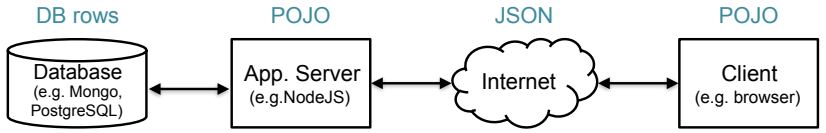
- Track all changes made to DB
- Manage with VCS
- Repeatable



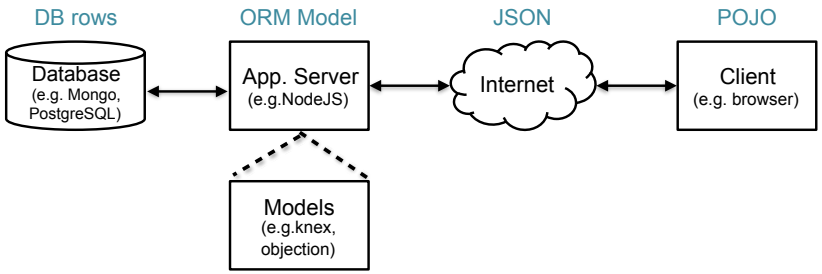
Example Migration

```
exports.up = function(knex, Promise) {
  return knex.schema
    .createTable('Film', table => {
      table
        .integer('id')
        .unsigned()
        .primary();
      table.text('overview');
      table.string('release_date');
      table.string('poster_path');
      table.string('title');
      table.float('vote_average');
      table.integer('rating');
    })
    .createTable('Genre', table => {
      table
        .integer('filmId')
        .unsigned()
        .references('id')
        .inTable('Film')
        .onDelete('CASCADE');
      table.integer('genreId');
      table.primary(['filmId', 'genreId']);
    });
};

exports.down = function(knex, Promise) {
  return knex.schema.dropTableIfExists('Genre').dropTableIfExists('Film');
};
```

Object Relational Mapping (ORM)



Object Relational Mapping (ORM)



Object Relational Mapping (ORM)

```
class Film extends Model {
  static get tableName() {
    return 'Film';
  }
static get jsonSchema() {
    return {
      type: 'object',
      required: [
        'id',
        'overview',
        'release_date',
        'poster_path',
        'title',
        'vote_average'
      ],

      properties: {
        id: { type: 'integer' },
        overview: { type: 'text' },
        release_date: { type: 'string' },
        poster_path: { type: 'string' },
        title: { type: 'string' },
        vote_average: { type: 'number' },
        rating: { type: ['integer', 'null']
, minimum: 0, maximum: 5 }
      }
    };
  }
```

```
class Genre extends Model {

  static get tableName() {
    return 'Genre';
  }


  static get idColumn() {
    return ['filmId', 'genreId'];
  }

    static get relationMappings() {
    return {
      film: {
        relation: Model.BelongsToOneRelation,
        modelClass: path.join(__dirname, 'Film
'),
        join: {
          from: 'Genre.filmId',
          to: 'Film.id'
        }
      }
    };
  }
}
```