

CS544

LESSON 8

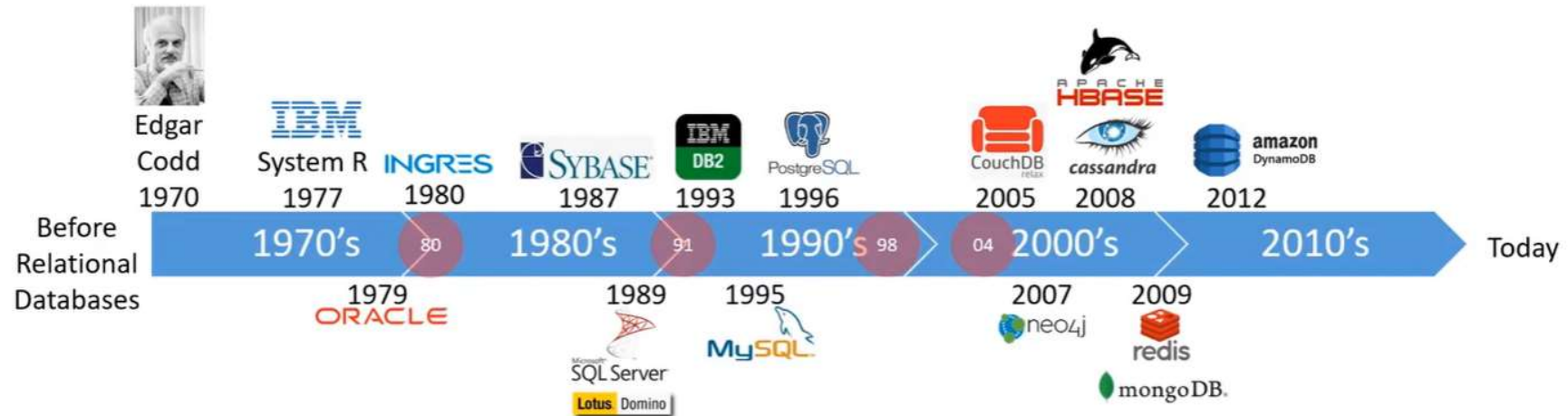
MONGODB

Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
March 28 Lesson 1 Enterprise Architecture introduction and Spring Boot	March 29 Lesson 2 Dependency injection AOP	March 30 Lesson 3 JDBC JPA	March 31 Lesson 4 JPA mapping 1	April 1 Lesson 5 JPA mapping 2	April 2 Lesson 6 JPA queries	April 3
April 4 Lesson 7 Transactions	April 5 Lesson 8 MongoDB	April 6 Midterm Review	April 7 Midterm exam	April 8 Lesson 9 REST webservices	April 9 Lesson 10 SOAP webservices	April 10
April 11 Lesson 11 Messaging	April 12 Lesson 12 Scheduling Events Configuration	April 13 Lesson 13 Monitoring	April 14 Lesson 14 Testing your application	April 15 Final review	April 16 Final exam	April 17
April 18 Project	April 19 Project	April 20 Project	April 21 Presentations			

SPRING MONGO

Today's requirements on databases

- Big data (large datasets)
- Agility
- Unstructured/ semi structured data



Problems with relational databases

- Scaling writes are is very difficult and limited
 - Vertical scaling is limited and is expensive
 - Horizontal scaling is limited and is complex
 - Queries work only within shards
 - Strict consistency and partition tolerance leads to availability problems

A relational database is hard to scale

Problems with relational databases

- The schema in a database is fixed
- Schema evolution
 - Adding attributes to an object => have to add columns to table
 - You need to do a migration project
 - Application downtime ...

A relational database is hard to change

Problems with relational databases

- Relational schema doesn't easily handle unstructured and semi-structured data
 - Emails
 - Tweets
 - Pictures
 - Audio
 - Movies
 - Text

Unstructured data

The university has 5600 students.
John's ID is number 1, he is 18 years old and already holds a B.Sc. degree.
David's ID is number 2, he is 31 years old and holds a Ph.D. degree. Robert's ID is number 3, he is 51 years old and also holds the same degree as David, a Ph.D. degree.

Semi-structured data

```
<University>
  <Student ID="1">
    <Name>John</Name>
    <Age>18</Age>
    <Degree>B.Sc.</Degree>
  </Student>
  <Student ID="2">
    <Name>David</Name>
    <Age>31</Age>
    <Degree>Ph.D. </Degree>
  </Student>
  ....
</University>
```

Structured data

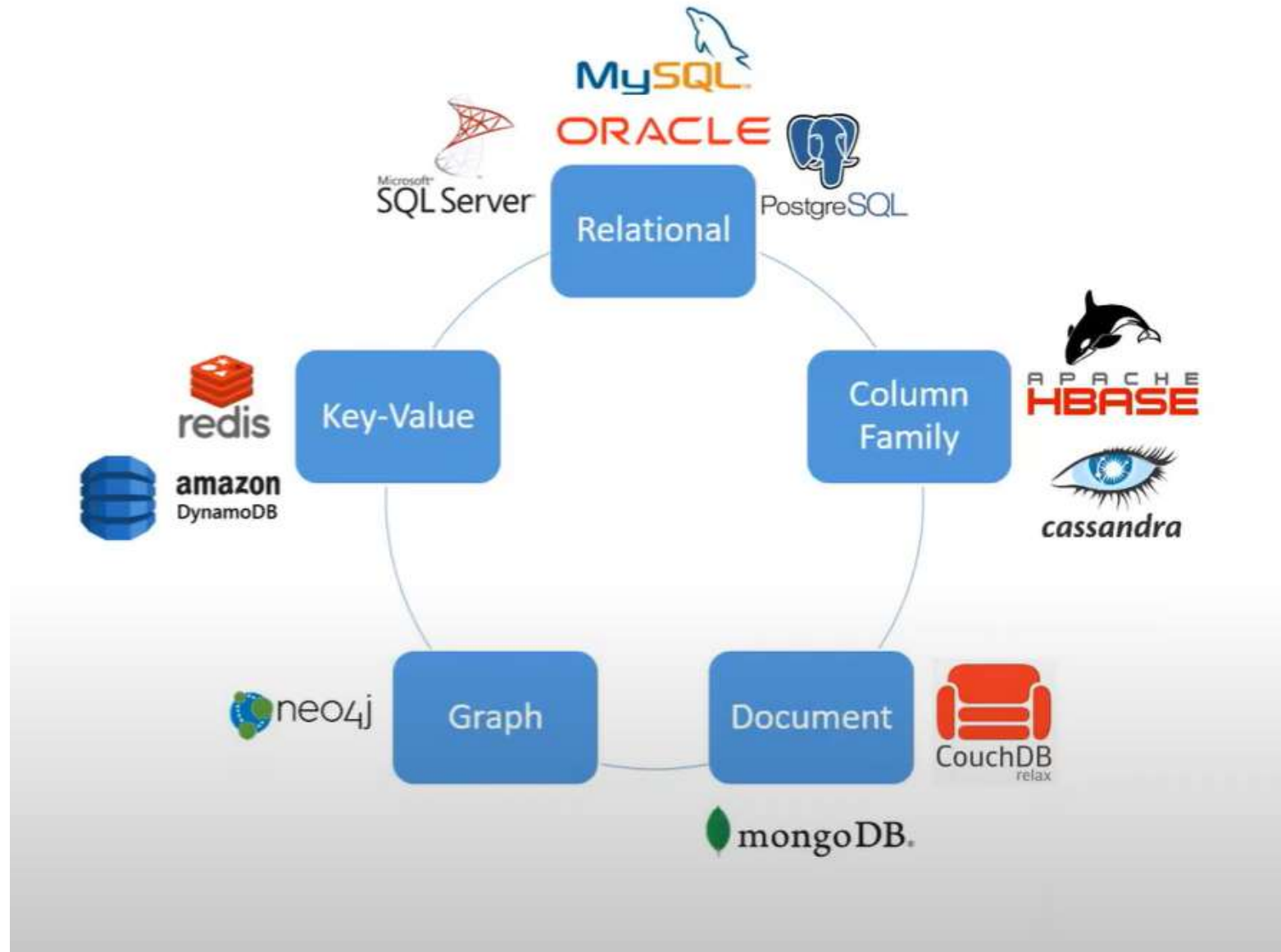
ID	Name	Age	Degree
1	John	18	B.Sc.
2	David	31	Ph.D.
3	Robert	51	Ph.D.
4	Rick	26	M.Sc.
5	Michael	19	B.Sc.

A relational database does not handle unstructured and semi structured data very well

NoSQL characteristics

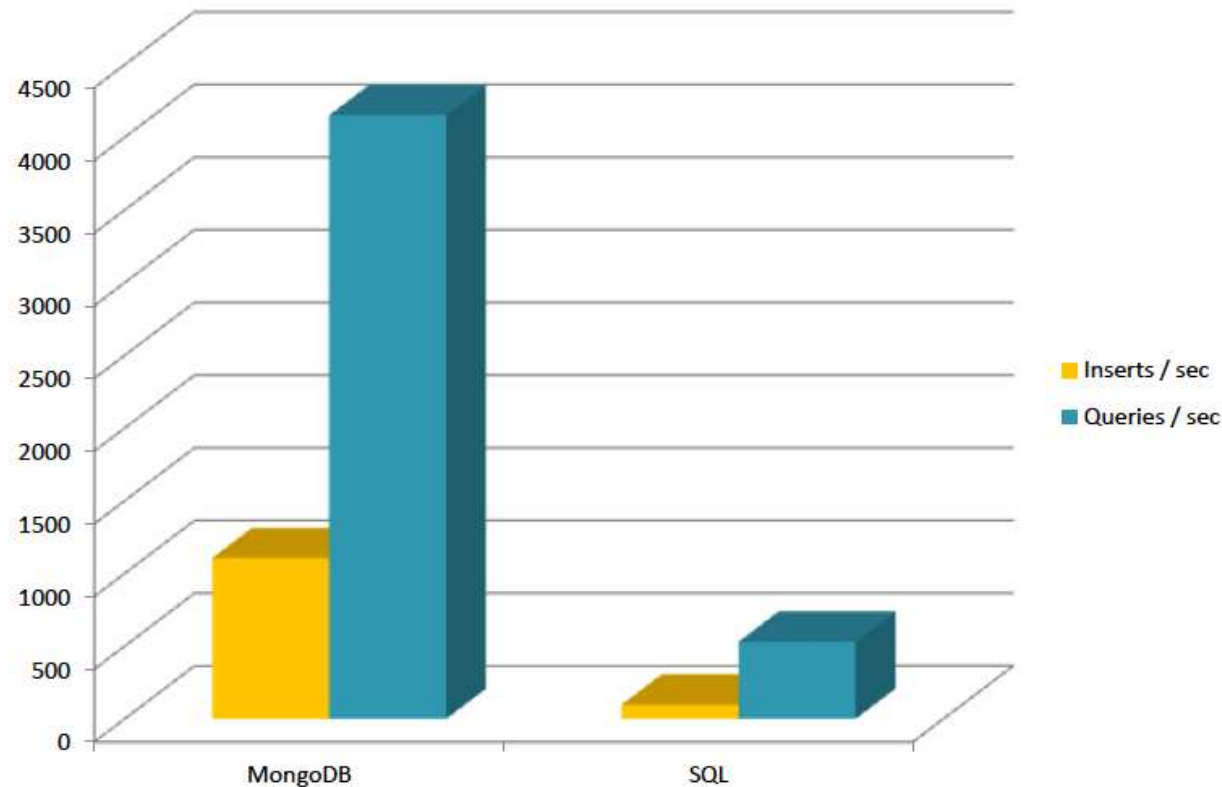
- Key-value store
- No fixed schema
- Can scale (almost) unlimited
 - Eventual consistency

Different types of databases



MongoDB

- Document database
- Fast
- Can handle large datasets



MongoDB

RDBMS		MongoDB
Database	→	Database
Table	→	Collection
Row	→	Document
Index	→	Index
Join	→	Embedded Document
Foreign Key	→	Reference



Document data model (JSON)

Relational - Tables

Customer ID	First Name	Last Name	City
0	John	Doe	New York
1	Mark	Smith	San Francisco
2	Jay	Black	Newark
3	Meagan	White	London
4	Edward	Daniels	Boston

Account Number	Branch ID	Account Type	Customer ID
10	100	Checking	0
11	101	Savings	0
12	101	IRA	0
13	200	Checking	1
14	200	Savings	1
15	201	IRA	2



Document - Collections

```
{  customer_id : 1,
  first_name  : "Mark",
  last_name   : "Smith",
  city        : "San Francisco",
  accounts   : [ {
                    account_number : 13,
                    branch_ID      : 200,
                    account_type    : "Checking"
                  },
                  {
                    account_number : 14,
                    branch_ID      : 200,
                    account_type    : "IRA",
                    beneficiaries: [...]
                  }
                ]
}
```

Documents are rich structures



```
{  
  category: "glove",  
  model: "PRO112PT",  
  name: "Air Elite",  
  brand: "Rawlings",  
  price: 229.99,  
  available: Date("2013-03-31"),  
  position: ["infield", "outfield", "pitcher"]  
}
```



Fields can contain arrays

Documents are rich structures

```
{  
  category: "glove",  
  model: "PRO112PT",  
  name: "Air Elite",  
  brand: "Rawlings",  
  price: 229.99,  
  available: Date("2013-03-31"),  
  position: ["infield", "outfield", "pitcher"],  
  endorsed: {name: "Ryan Howard",  
             team: "Phillies",  
             position: "first base"},  
}
```

} Fields can contain sub-documents

Documents are rich structures

```
{
  category: "glove",
  model: "PRO112PT",
  name: "Air Elite",
  brand: "Rawlings",
  price: 229.99,
  available: Date("2013-03-31"),
  position: ["infield", "outfield", "pitcher"],
  endorsed: {name: "Ryan Howard",
              team: "Phillies",
              position: "first base"},
  history: [{date: Date("2013-03-31"), price: 279.99},
            {date: Date("2013-06-01"), price: 259.79},
            {date: Date("2013-08-15"), price: 229.99}]
}
```

Fields can contain
an array of sub-
documents

Documents are flexible



```
{  
  category: bat,  
  model: B1403E,  
  name: Air Elite,  
  brand: "Rip-IT",  
  price: 399.99  
  
  diameter: "2 5/8",  
  barrel: R2 Alloy,  
  handle: R2  
}
```

```
{  
  category: glove,  
  model: PRO112PT,  
  name: Air Elite,  
  brand: "Rawlings",  
  price: "229.99"  
  
  size: 11.25,  
  position: outfield,  
  pattern: "Pro taper",  
  material: leather,  
  color: black  
}
```


BSON

```
{ author: 'joe',  
  created : new Date('03/28/2009'),  
  title : 'Yet another blog post',  
  text : 'Here is the text...',  
  tags : [ 'example', 'joe' ],  
  comments : [  
    { author: 'jim',  
      comment: 'I disagree'  
    },  
    { author: 'nancy',  
      comment: 'Good post'  
    }  
  ]  
}
```



Remember it is stored in binary formats (BSON)

```
"\x16\x00\x00\x00\x02hello\x00  
\x06\x00\x00\x00world\x00\x00"  
  
"1\x00\x00\x00\x04BSON\x00&\x00  
\x00\x00\x020\x00\x08\x00\x00  
\x00awesome\x00\x011\x00333333  
\x14@\x102\x00\xc2\x07\x00\x00  
\x00\x00"
```

MongoDB model



document (e.g., one **tuple** in RDBMS)

```
{  
  name: "sue",  
  age: 26,  
  status: "A",  
  groups: [ "news", "sports" ]  
}
```

← field: value
← field: value
← field: value
← field: value

- The field names **cannot** start with the \$ character
- The field names **cannot** contain the . character
- Max size of single document 16MB

Find() method

SQL SELECT Statements

SELECT * **FROM** users

SELECT id, user_id, status **FROM** users

SELECT user_id, status **FROM** users

SELECT * **FROM** users **WHERE** status = "A"

SELECT user_id, status **FROM** users **WHERE** status = "A"

SELECT * **FROM** users **WHERE** status != "A"

SELECT * **FROM** users **WHERE** status = "A" **AND** age = 50

SELECT * **FROM** users **WHERE** status = "A" **OR** age = 50

SELECT * **FROM** users **WHERE** age > 25

MongoDB find() Statements

db.users.find()

db.users.find({}, { user_id: 1, status: 1 })

db.users.find({}, { user_id: 1, status: 1, _id: 0 })

db.users.find({ status: "A" })

db.users.find({ status: "A" }, { user_id: 1, status: 1, _id: 0 })

db.users.find({ status: { \$ne: "A" } })

db.users.find({ status: "A", age: 50 })

db.users.find({ \$or: [{ status: "A" }, { age: 50 }] })

db.users.find({ age: { \$gt: 25 } })

Find() method

```
SELECT * FROM users WHERE age < 25
```

```
db.users.find( { age: { $lt: 25 } } )
```

```
SELECT * FROM users WHERE age > 25 AND age <= 50
```

```
db.users.find( { age: { $gt: 25, $lte: 50 } } )
```

```
SELECT * FROM users WHERE user_id like "%bc%"
```

```
db.users.find( { user_id: /bc/ } )
```

```
SELECT * FROM users WHERE user_id like "bc%"
```

```
db.users.find( { user_id: /^bc/ } )
```

```
SELECT * FROM users WHERE status = "A" ORDER BY user_id ASC
```

```
db.users.find( { status: "A" } ).sort( { user_id: 1 } )
```

```
SELECT * FROM users WHERE status = "A" ORDER BY user_id DESC
```

```
db.users.find( { status: "A" } ).sort( { user_id: -1 } )
```

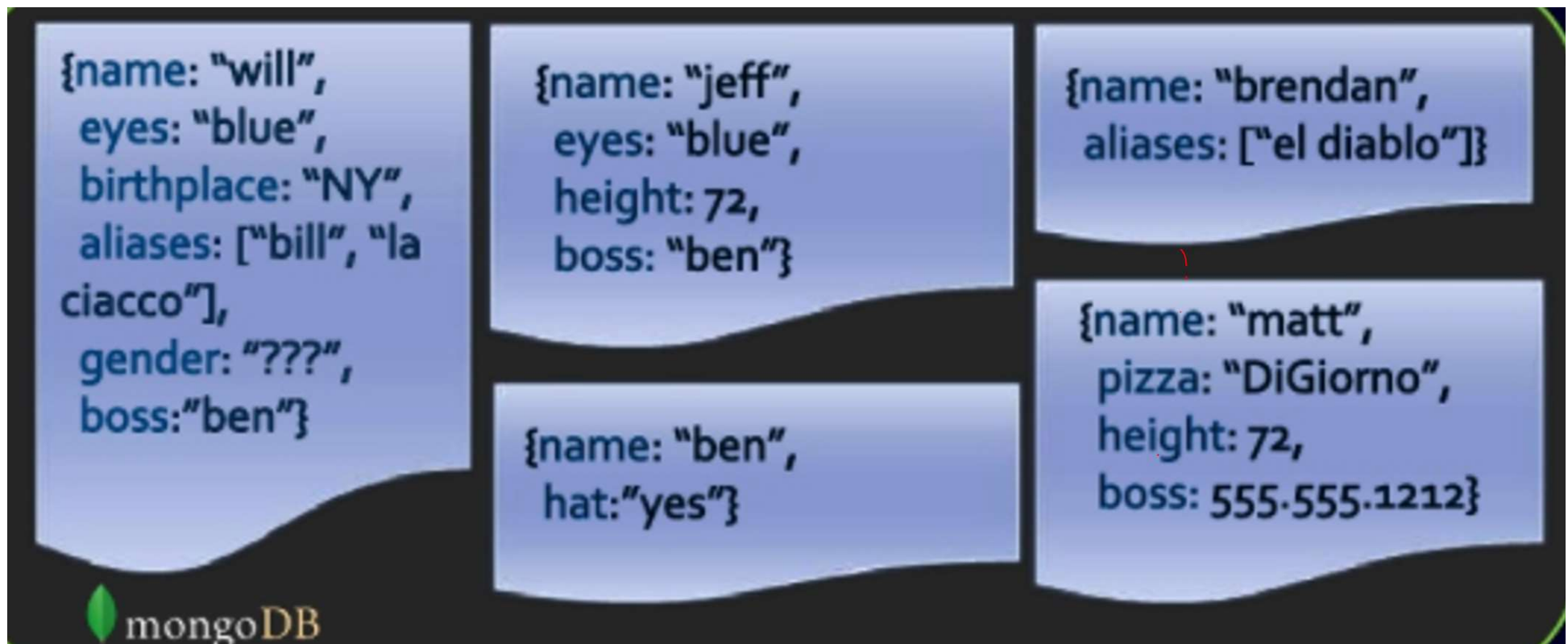
```
SELECT COUNT(*) FROM users
```

```
db.users.count()  
or  
db.users.find().count()
```

```
SELECT COUNT(user_id) FROM users
```

```
db.users.count( { user_id: { $exists: true } } )  
or  
db.users.find( { user_id: { $exists: true } } ).count()
```

Schema free



Spring Mongo libraries



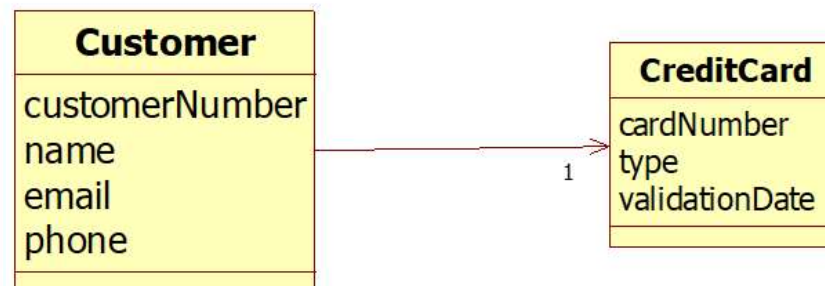
```
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-data-mongodb</artifactId>  
</dependency>
```

The Mongo Documents

@Document

```
public class Customer {  
    @Id  
    private int customerNumber;  
    private String name;  
    private String email;  
    private String phone;  
    private CreditCard creditCard;
```

```
public class CreditCard {  
    private String cardNumber;  
    private String type;  
    private String validationDate;
```



The repository

@Repository

```
public interface CustomerRepository extends MongoRepository<Customer, Integer> {  
    Customer findByPhone(String phone);  
    Customer findByEmail(String email);  
  
    @Query("{email : #{#email}}")  
    Customer findCustomerWithEmail(@Param("email") String email);  
  
    @Query("{ 'creditCard.type' : #{#cctype}}")  
    List<Customer> findCustomerWithCreditCardType(@Param("cctype") String cctype);  
}
```

Find in an Object

application.properties

```
spring.data.mongodb.host=localhost  
spring.data.mongodb.port=27017  
spring.data.mongodb.database=testdb
```


The application(1/2)

```
public class Application implements CommandLineRunner {
```

```
    @Autowired
```

```
    private CustomerRepository customerRepository;
```

```
    public static void main(String[] args) {  
        SpringApplication.run(Application.class, args);  
    }
```

```
    @Override
```

```
    public void run(String... args) throws Exception {
```

```
        // create customer
```

```
        Customer customer = new Customer(101, "John doe", "johnd@acme.com", "0622341678");
```

```
        CreditCard creditCard = new CreditCard("12324564321", "Visa", "11/23");
```

```
        customer.setCreditCard(creditCard);
```

```
        customerRepository.save(customer);
```

```
        customer = new Customer(109, "John Jones", "jones@acme.com", "0624321234");
```

```
        creditCard = new CreditCard("657483342", "Visa", "09/23");
```

```
        customer.setCreditCard(creditCard);
```

```
        customerRepository.save(customer);
```

```
        customer = new Customer(66, "James Johnson", "jj123@acme.com", "068633452");
```

```
        creditCard = new CreditCard("99876549876", "MasterCard", "01/24");
```

```
        customer.setCreditCard(creditCard);
```

```
        customerRepository.save(customer);
```

The application(2/2)

```
//get customers
System.out.println(customerRepository.findById(66).get());
System.out.println(customerRepository.findById(101).get());
System.out.println("-----All customers -----");
System.out.println(customerRepository.findAll());
//update customer
customer = customerRepository.findById(101).get();
customer.setEmail("jd@gmail.com");
customerRepository.save(customer);
System.out.println("-----find by phone -----");
System.out.println(customerRepository.findByPhone("0622341678"));
System.out.println("-----find by email -----");
System.out.println(customerRepository.findCustomerWithEmail("jj123@acme.com"));
System.out.println("-----find customers with a certain type of creditcard -----");
List<Customer> customers = customerRepository.findCustomerWithCreditCardType("Visa");
for (Customer cust : customers){
    System.out.println(cust);
}
}
```

The database

testdb.customer

Documents

Aggregations

FILTER { field: 'value' }

ADD DATA



VIEW



{ }



```
_id: 101
name: "John doe"
email: "jd@gmail.com"
phone: "0622341678"
> creditCard: Object
_class: "customers.Customer"
```

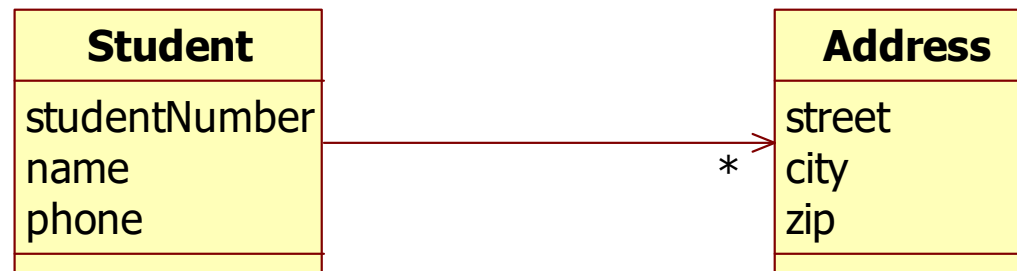
```
_id: 109
name: "John Jones"
email: "jones@acme.com"
phone: "0624321234"
> creditCard: Object
_class: "customers.Customer"
```

```
> _id: 66
name: "James Johnson"
email: "jj123@acme.com"
phone: "068633452"
> creditCard: Object
_class: "customers.Customer"
```

One to many relations

```
public class Customer {  
    @Id  
    private int customerNumber;  
    private String name;  
    private String email;  
    private String phone;  
    private List<CreditCard> creditCards = new ArrayList<CreditCard>();  
}
```

```
public class CreditCard {  
    private String cardNumber;  
    private String type;  
    private String validationDate;  
}
```



The repository

@Repository

```
public interface CustomerRepository extends MongoRepository<Customer, Integer> {  
    Customer findByPhone(String phone);  
    Customer findByEmail(String email);  
  
    @Query("{email : :#{#email}}")  
    Customer findCustomerWithEmail(@Param("email") String email);  
  
    @Query("{ 'creditCards': { $elemMatch: { 'type' : :#{#cctype} } }}")  
    List<Customer> findCustomerWithCreditCardType(@Param("cctype") String cctype);  
}
```

Find in an Array

Main point

- MongoDB is a document database that stores whole documents (including embedded data) in a collection. This gives data redundancy, but makes the data access very fast.

Science of Consciousness: The Unified Field is the source of all relative creation where there is no redundancy or loss of performance.

Connecting the parts of knowledge with the wholeness of knowledge

1. MongoDB is a document database where we store documents instead of relational data
2. Spring Boot Mongo makes it very easy to use the MongoDB in your application

-
3. **Transcendental consciousness** is the field where all intelligence resides.
 4. **Wholeness moving within itself:** In unity consciousness, one experiences that everything is an expression of one's own Self.