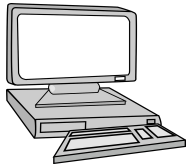
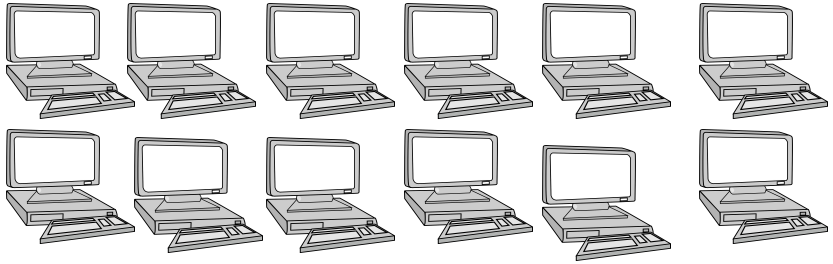
Databases and Big Data

MongoDB

Recap

NoSQL: Give up on correctness, but get scalability (not speed but your application can accommodate more and more users)





Relational Database		NoSQL Database
Schema: tables		No Schema: just blobs
SQL: join, select, ect.	Scale out	Simple API: put/get
Correctness		Not always correct
Speed*		Scalability

MongoDB

- Most vocal proponent of the NoSQL movement
  - Among the earliest (2009)
- But the database community weren't impressed
- Regardless, still the most popular NoSQL

Document Store

- What is a document?
  - Loose term
  - Anything that is *parse-able*
  - Examples: csv, text, etc.

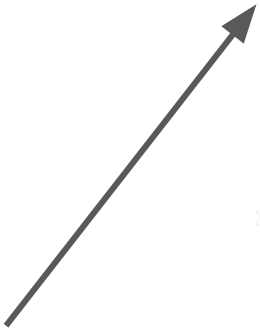
```
2,San Jose Diridon Caltrain Station,37.3297,-121.902,27,San Jose,2013-08-06,95113
3,San Jose Civic Center,37.3307,-121.889,15,San Jose,2013-08-05,95113
4,Santa Clara at Almaden,37.334,-121.895,11,San Jose,2013-08-06,95113
5,Adobe on Almaden,37.3314,-121.893,19,San Jose,2013-08-05,95113
6,San Pedro Square,37.3367,-121.894,15,San Jose,2013-08-07,95113
7,Paseo de San Antonio,37.3338,-121.887,15,San Jose,2013-08-07,95113
8,San Salvador at 1st,37.3302,-121.886,15,San Jose,2013-08-05,95113
9,Japantown,37.3487,-121.895,15,San Jose,2013-08-05,95113
10,San Jose City Hall,37.3374,-121.887,15,San Jose,2013-08-06,95113
```

```
Mr. Bingley was good-looking and gentlemanlike; he had a pleasant
countenance, and easy, unaffected manners. His sisters were fine women,
with an air of decided fashion. His brother-in-law, Mr. Hurst, merely
looked the gentleman; but his friend Mr. Darcy soon drew the attention
of the room by his fine, tall person, handsome features, noble mien, and
the report which was in general circulation within five minutes
after his entrance, of his having ten thousand a year. The gentlemen
pronounced him to be a fine figure of a man, the ladies declared he
was much handsomer than Mr. Bingley, and he was looked at with great
admiration for about half the evening, till his manners gave a disgust
which turned the tide of his popularity; for he was discovered to be
proud; to be above his company, and above being pleased; and not all
his large estate in Derbyshire could then save him from having a most
forbidding, disagreeable countenance, and being unworthy to be compared
with his friend.
```

Document Store

- We want semi-structure document:
  - Self-explaining documents
  - Use tags to capture semantics
  - Examples:
    - XML, JSON
    - ProtoBuffer

```
<?xml version="1.0" standalone="yes"?>
<BankAccount>
      <Number>1234</Number>
      <Type>Checking</Type>
      <OpenDate>11/04/1974</OpenDate>
      <Balance>25382.20</Balance>
      <AccountHolder>
            <LastName>Singh</LastName>
            <FirstName>Darshan</FirstName>
      </AccountHolder>
</BankAccount>
```



Tell ***us*** what the enclosed data means

Document Store

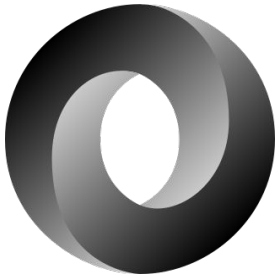
- Semi-structured documents


```
message Block {
    uint32 version = 1;
    google.protobuf.Timestamp timestamp = 2;
    repeated Transaction transactions = 3;
    bytes stateHash = 4;
    bytes previousBlockHash = 5;
    bytes consensusMetadata = 6;
    NonHashData nonHashData = 7;
}

// Contains information about the blockchain le
// block hash, and previous block hash.
message BlockchainInfo {

    uint64 height = 1;
    bytes currentBlockHash = 2;
    bytes previousBlockHash = 3;

}
```

Document Store

- Semi-structured documents

JavaScript Object Notation (JSON)

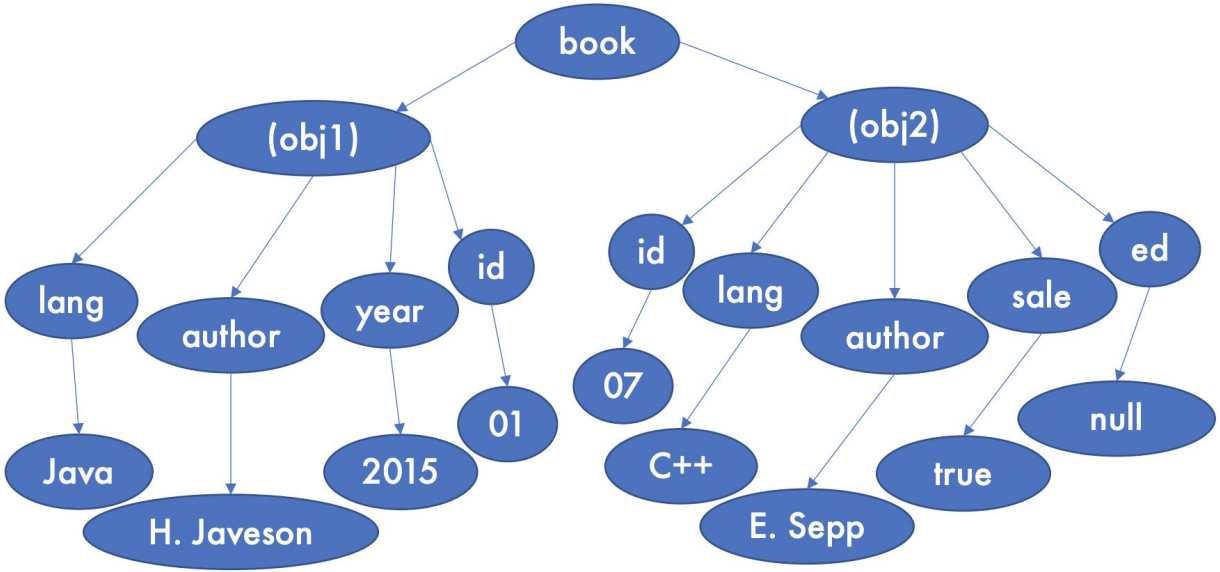


Many applications phasing out XML in favor of JSON


```
{
    "orders": [
        {
            "orderno": "748745375",
            "date": "June 30, 2088 1:54:23 AM",
            "trackingno": "TN0039291",
            "custid": "11045",
            "customer": [
                {
                    "custid": "11045",
                    "fname": "Sue",
                    "lname": "Hatfield",
                    "address": "1409 Silver Street",
                    "city": "Ashland",
                    "state": "NE",
                    "zip": "68003"
                }
            ]
        }
    ]
}
```

Semi-Structured Documents

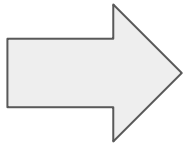
- It's a **tree**



Semi-Structured Documents

- From tables to trees
  - Not hard, because tables = flat trees

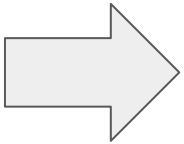
Name	Phone
Anh	12345
Dan	23093
Leo	09470



```
{
  "person":[
      {
        "name": "Anh",
        "phone": "12345"
      },
      {
        "name": "Dan",
        "phone": "23093"
      },
      {
        "name": "Leo",
        "phone": "09470"
      }
  ]
}
```

Semi-Structured Documents

Name	Phone
Anh	12345
Dan	23093
Leo	09470
Ben	NULL



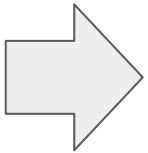
```
{
  "person":[
      {
        "name": "Anh",
        "phone": "12345"
      },
      {
        "name": "Dan",
        "phone": "23093"
      },
      {
        "name": "Leo",
        "phone": "09470"
      },
      {
        "name": "Ben"
      }
  ]
}
```

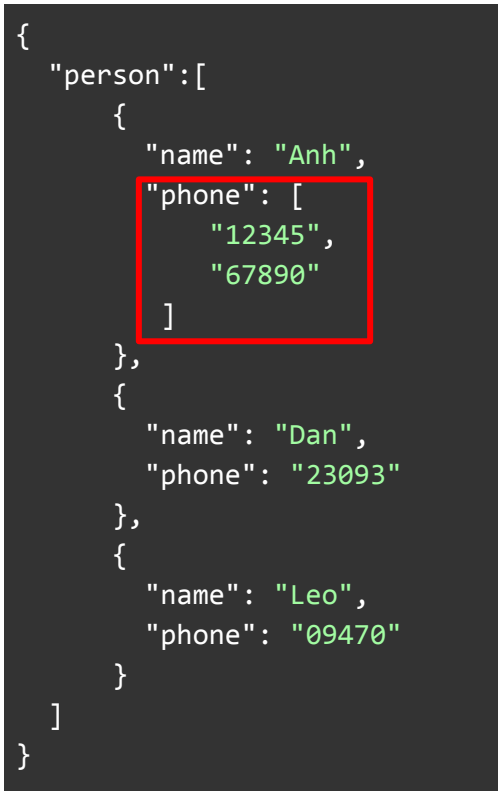
Field missing = NULL

Semi-Structured Documents

- But can we fit any tree into a table?
- Non-flat data:
  - **Array**
  - Multi-part

Name	Phone
Anh	???
Dan	23093
Leo	09470

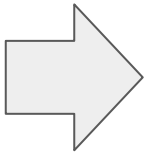




Semi-Structured Documents

- But can we fit any tree into a table?
- Non-flat data:
  - Array
  - **Multi-part**

Name	Phone
???	12345
Dan	23093
Leo	09470



```
{
  "person":[
      {
        "name": {
          "first": "Anh",
          "last": "Dinh"
        }
        "phone": "12345"
      },
      {
        "name": "Dan",
        "phone": "23093"
      },
      {
        "name": "Leo",
        "phone": "09470"
      }
  ]
}
```

Semi-Structured Documents

- Relational data model isn’t designed for nested data
  - Tables vs. trees
- Term: **impedance mismatch**

You have a model in mind but the database does not suit that.

Document Store

key value has no structure (more general)
but documents are stored in some structure.

- Handle trees
- Many implementations

MongoDB Language

- Remember for SQL, we have:
  - Data Definition Language (DDL): create, delete, etc.
  - Data Manipulation Language (DML): insert, update, etc.
  - Query Language: select … from … where
- MongoDB supports same categories
  - **Tables** in SQL → **Collections** in MongoDB

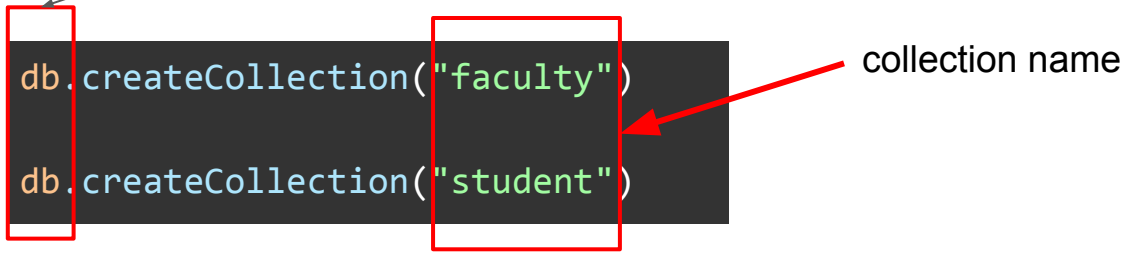
MySQL	**MongoDB**
Database	Database
Tables	Collections


{
"Name": "Anh"
"Phone": [
"12345",
"93932"
]
}
{
"Name": "Dan"
"Phone": "93752"
}
Collection
{
"Name": {
"First": "Albert",
"Last": "Einstein"
},
"Theory": "Particle Physics"
}
{
"Name": {
"First": "Kurt",
"Last": "Godel"
},
"Theory": "Incompleteness"
}
{
"Name": {
"First": "Sheldon",
"Last": "Copper"
},
}
Document
Database

MongoDB

- Create table

- Create collections



MongoDB

- Insert new document
  - Duplicates are allowed

```
db.faculty.insert({"Name": "Einstein",
"Theory": "Relativity"})
```

```
db.faculty.update({"Name": "Einstein"},
                  {"$set": {"Country": "Germany"}})
```

Guess what this does

```
{
  "Name": {
      "First": "Albert",
      "Last": "Einstein"
  },
  "Theory": "Particle Physics"
}

{
  "Name": "Einstein",
  "Theory": "Relativity"
}

...
...
```

MongoDB

- Read
  - Select all documents

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

```
db.faculty.find()
```

SELECT * from University;

(SQL CheatSheet)

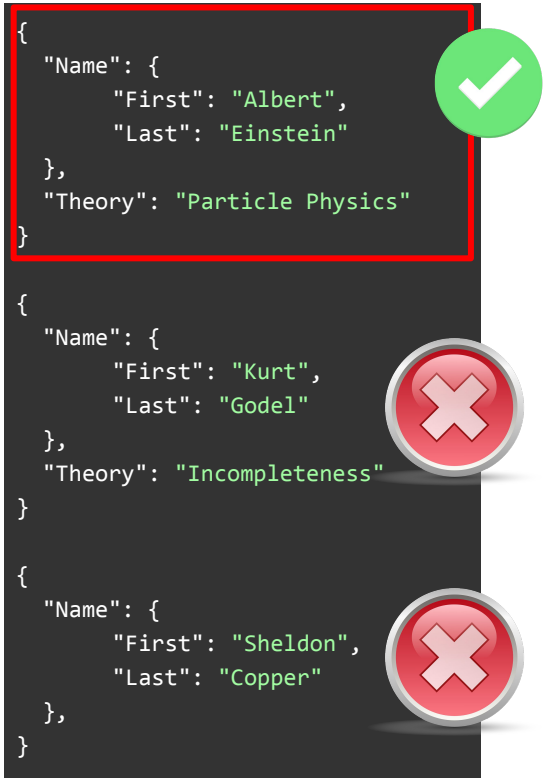
MongoDB

- Read condition

```
db.faculty.find(
      {"Theory": "Particle Physics"}
)
```

SELECT * from Faculty
WHERE Theory = “Particle Physics”

(SQL CheatSheet)



MongoDB

- Read projection

```
db.faculty.find(
      {"Theory": "Particle Physics"},
      {"Name": 1, "Last": 1}
)
```

SELECT Name, Last from Faculty
WHERE Theory = “Particle Physics”

(SQL CheatSheet)

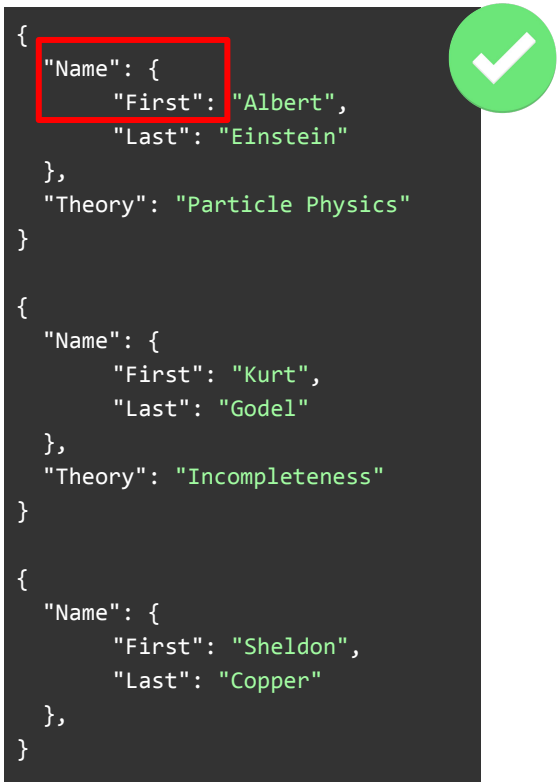
```
{
  "First": "Albert",
  "Last": "Einstein",
  "Theory": "Particle Physics"
}
{
  "Name": "Higgs",
  "Theory": "Particle Physics"
}
{
  "First": "Kurt",
  "Last": "Godel",
  "Theory": "Incompleteness"
}
```

{ “Last”: “Einstein”}
{ “Name”: “Higgs”}

MongoDB

- Read
  - Nested field

```
db.faculty.find({"Name.First": "Albert"})
```

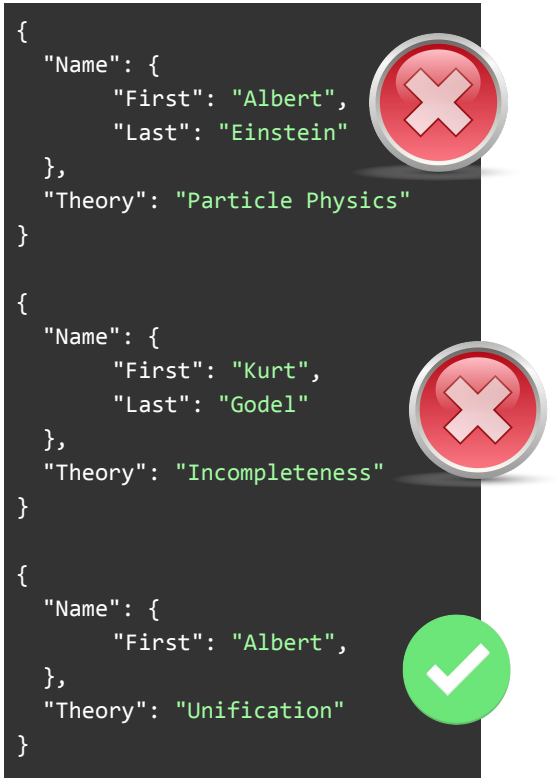


MongoDB

- Read
  - Nested field

```
db.faculty.find({"Name": {"First": "Albert"}})
```

Does a full match
(not a list!)



MongoDB

- Read
  - List matching

```
db.faculty.find({"Name": {"First": "Albert"}})
```

db.faculty.find({"Theory": "Special relativity"})

```
{
  "Name": {
      "First": "Albert",
      "Last": "Einstein"
  },
  "Theory": [
      "Special relativity",
      "General relativity"
  ]
}
{
  "Name": "Godel",
  "Theory": "Incompleteness"
}
```

MongoDB

- Read operators

Operator	Meaning
$gte, $eq, $ne, $gt, $lt, $lte	>=, =, !=, >, <, <=
$in, $nin	∈, ∉

```
db.faculty.find( {
      "NoPublications": {"$gte": 120}
})
```

```
{
  "Name": {
      "First": "Albert",
      "Last": "Einstein"
  },
  "NoPublications": 209
}
{
  "Name": "Godel",
  "NoPublications": 100
}
```

MongoDB

- Read operators

Operator	Meaning
$gte, $eq, $ne, $gt, $lt, $lte	>=, =, !=, >, <, <=
$in, $nin	∈, ∉

```
db.faculty.find( {
      "University": {"$in": ["NUS", "SUTD"]}
})
```

```
{
  "Name": {
      "First": "Albert",
      "Last": "Einstein"
  },
  "NoPublications": 209
  "University": "SUTD"
}
{
  "Name": "Godel",
  "NoPublications": 100
  "University": "SMU"
}
```

-1 -> descending sort
1 -> ascending sort

MongoDB

- Read
  - Count
  - Sort
  - Limit
  - Duplicates
- Check them out yourself

```
db.faculty.find({"Theory": "Particle Physics"}).count()
```

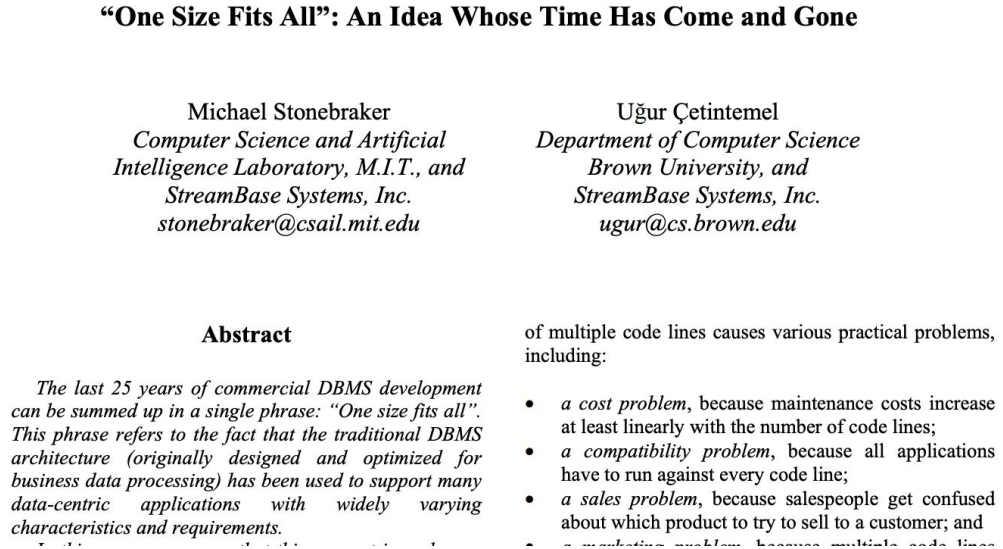
```
db.faculty.find({"Theory": "Particle Physics"}).sort({"Age": -1})
```

*db.faculty.find().limit(2) SELECT * FROM Faculty LIMIT 2;*

```
db.faculty.distinct("Name")
```

Summary

- MongoDB most popular for
  - Semi-structured data model
  - Or trees
- One size doesn’t fit all



“One Size Fits All”: An Idea Whose Time Has Come and Gone

Michael Stonebraker
Computer Science and Artificial Intelligence Laboratory, M.I.T., and StreamBase Systems, Inc.
stonebraker@csail.mit.edu

Uğur Çetintemel
Department of Computer Science Brown University, and StreamBase Systems, Inc.
ugur@cs.brown.edu

Abstract

The last 25 years of commercial DBMS development can be summed up in a single phrase: “One size fits all”. This phrase refers to the fact that the traditional DBMS architecture (originally designed and optimized for business data processing) has been used to support many data-centric applications with widely varying characteristics and requirements.

of multiple code lines causes various practical problems, including:

- *a cost problem*, because maintenance costs increase at least linearly with the number of code lines;
- *a compatibility problem*, because all applications have to run against every code line;
- *a sales problem*, because salespeople get confused about which product to try to sell to a customer; and