# DS-2002: Data Systems

An Overview of NoSQL Databases

Prof. Jon Tupitza

# NoSQL Databases

Understanding the Principles Behind NoSQL Databases

UNIVERSITY of VIRGINIA

# An Explosion of Data… "BIG Data"

## A Rapid, Exponential Proliferation of New Devices: The Internet of Things (IoT)

**olume** (Size)

- Explosion in social media, mobile apps, digital sensors, RFID, GPS, and more have caused exponential data growth.

**elocity** (Speed)

- Sources like social networking and sensor signals create data at a tremendous rate; making it a challenge to capture, store, and analyze that data in a timely or economical manner.

**ariety** (Structure)

- Traditionally BI has sourced structured data, but now insight must be extracted from unstructured or poly-schematic data like large text blobs, digital media, sensor data, etc.

**eracity** (Quality)

- The anonymity of the WWW, incredible sources like social networking and duplicate systems bring into question the authenticity of the information being generated and collected.

# Architectural Models: How They're Used

**"Big Data" technologies aren't necessarily intended to replace existing database technologies; rather they play a critical role in extending the capabilities of a data management ecosystem.**

| | |
|---|---|
| **Standalone Data Analysis and Visualization** | Experiment with data sources to discover if they provide useful information. Handle data that can't be processed using existing systems. |
| **Data Transfer, Cleansing or ETL** | Extract and transform data before loading it into existing databases. Categorize, normalize, and extract summary results to remove duplication and redundancy. |
| **Data Warehouse or Data Storage** | Create robust data repositories that are reasonably inexpensive to maintain. Especially useful for storing and managing huge data volumes. |
| **Integrate with Existing EDW and BI Systems** | Integrate Big Data at different levels; EDW, OLAP, Excel PowerPivot. Also, APS enables querying Hadoop to integrate Big Data with existing dimension & fact data. |

# NoSQL Databases

- NoSQL databases are defined by a collection of characteristics that they share rather than having a formal definition:

  **Non-relational**

  **Scale-out**

  **Schema-less**

- They were all born out of this desire to address new needs of the internet world.

| | |
|---|---|
| Key–value databases | Column family databases |
| Document databases | Graph databases |

# NoSQL: Not Only SQL

| | |
|---|---|
| **Key-value stores** | • The simplest NoSQL database; based on “dictionaries” or “maps”.<br>• Items are stored in associative arrays; pairing a name (or "key"), with a value.<br>• **Riak, FoundationDB, and Redis** |
| **Column stores** | • Combines a key, value and timestamp for each item.<br>• Optimized for large datasets by storing columns of data together, rather than in rows.<br>• **Cassandra, BigTable and HBase** |
| **Document databases** | • Pairs keys with complex data structures (documents) using XML, JSON or BSON.<br>• Documents may contain key-value pairs, key-array pairs, and nested documents.<br>• **MongoDB, MarkLogic, and Apache CouchDB** |
| **Graph stores** | • Stores interrelated networks of data such as social connections, or network topologies.<br>• Optimized for interconnected data elements with an undetermined number of relations.<br>• **AllegroGraph, Neo4J and HyperGraphDB**. |

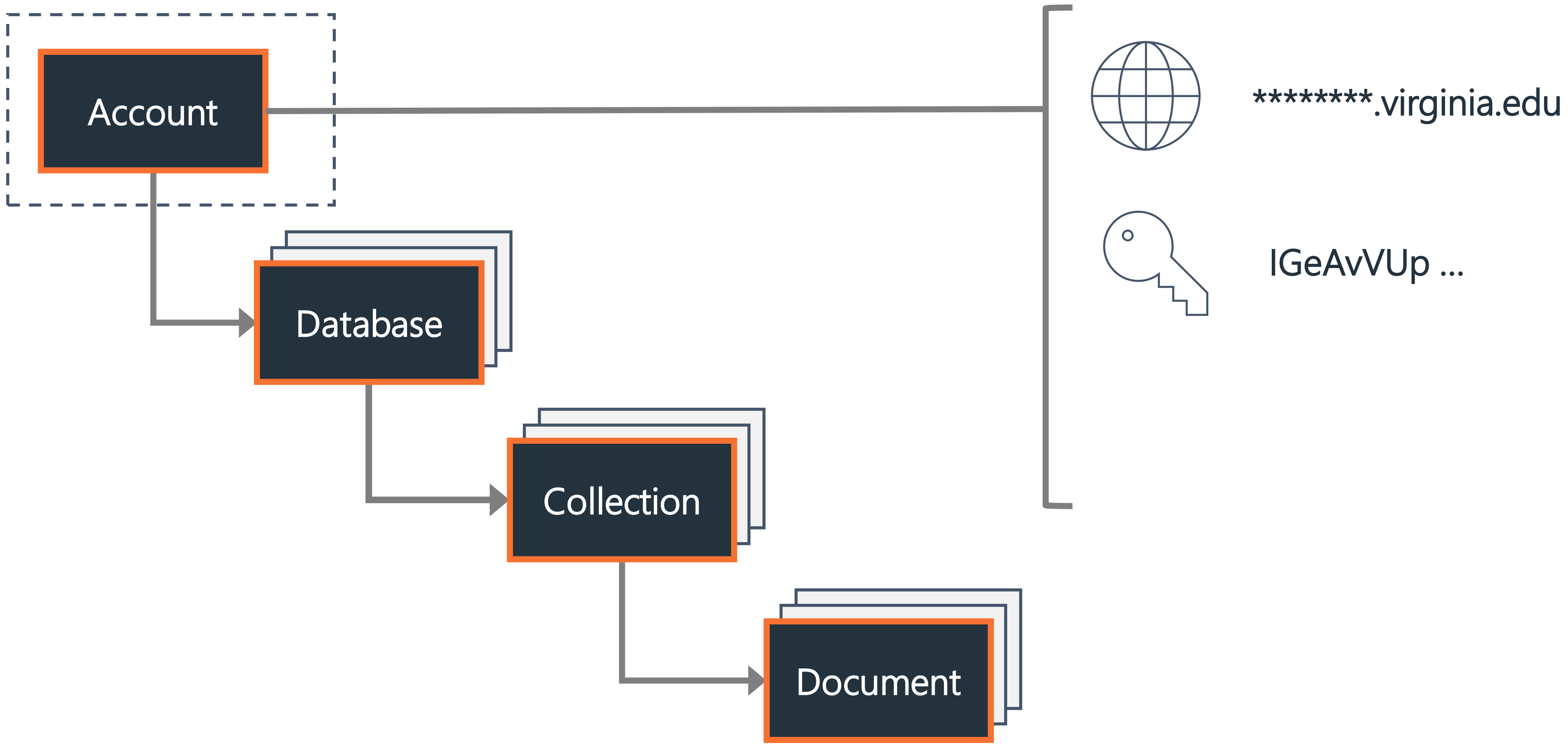
# Relational versus NoSQL Databases

| Relational Databases | NoSQL Databases |
|---|---|
| • Good for **structured data** and high-performance workloads | • Good for **non-relational** data |
| • Provides transactional consistency | • Provides eventual consistency |
| • **Predefined Schema Architecture** enforces integrity of data structure | • **Schema-less Architecture** allows for frequent structural changes and easy addition of varied data |
| • Products are Proven and Mature with a Variety of Available Tools | • **Poly-schematic:** Supports multiple document types |
| • **Limited Scalability** [Vertically]: Not well suited to distribution | • **Easily Scalable** [Horizontally]: Runs well on distributed systems |

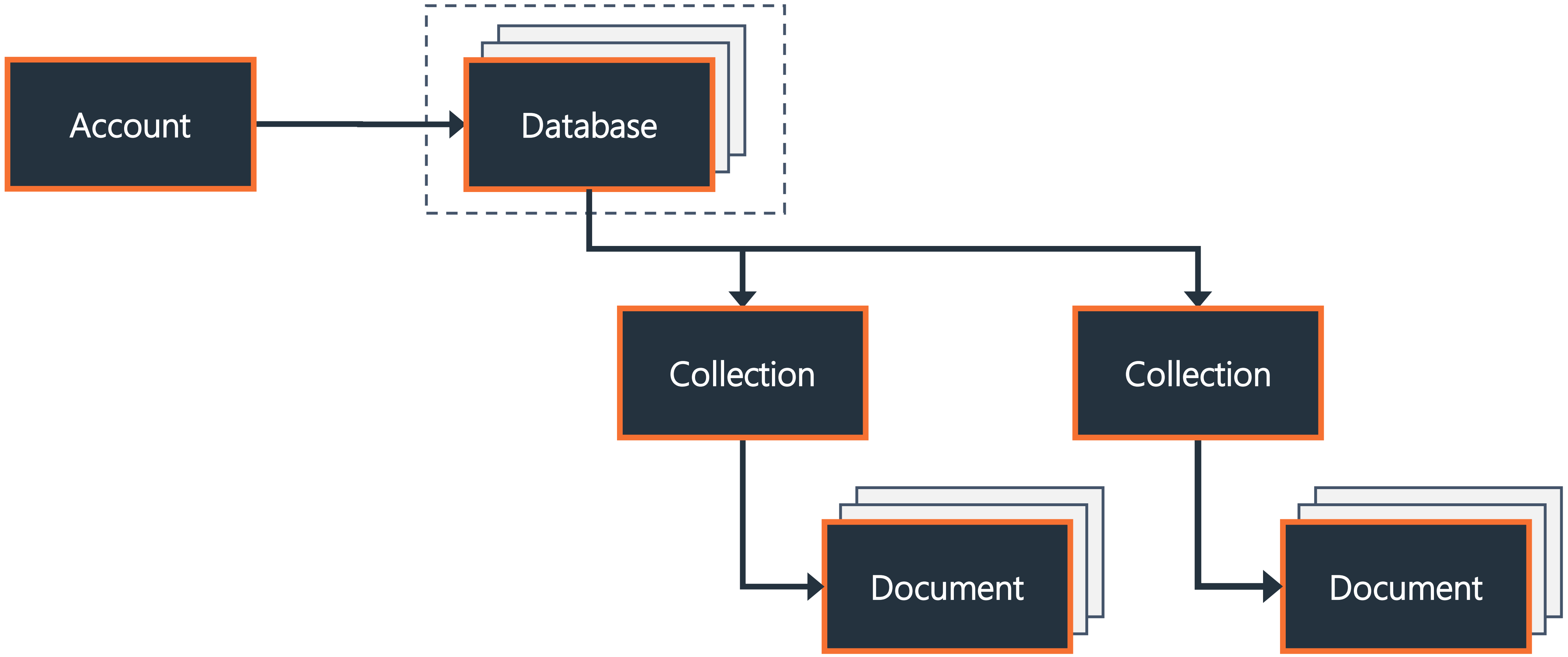
# SQL versus NoSQL Databases

| | SQL Databases | NoSQL Databases |
|---|---|---|
| **Types** | One: Relational Database | Many: Key-value, document, column, and graph databases |
| **Data Storage** | Records are stored as rows in tables that represent entities. Entity relationships are modeled by joining tables. | Records may be stored as "documents" using XML or JSON. Entity relationships are modeled by nesting them hierarchically. |
| **Schemas** | Static: Structure & data types are fixed at design time. Adding new elements requires schema design changes. | Dynamic: New elements can be added at runtime. Poly-schematic: Dissimilar data can be stored together as necessary. |
| **Scaling** | Vertically: A single server must be made increasingly powerful to cope with increasing demand. | Horizontally: New commodity servers and storage are added to an array. Data is automatically distributed across all servers. |
| **Transactions** | Full transactional consistency (ACID) | Single element (document) only. |
| **Manipulation** | Using platform-specific languages (SQL) | Using OSS API's, low-level lang., JavaScript |
| **Consistency** | Supports strong consistency | Per-product: Most are eventually consistent. |

## Resource Model: Account URI and Credentials



## Resource Model: Database & Collection Resources



# Document Collections versus Tables

## JSON Documents
(Nested Hierarchies)

```
{
  “FirstName” : “Bob”,
  “LastName” : “Smith”,
  “BirthDate” : “03/11/1985”,
  “PhoneNumbers” :
   [ { “Home” : “571-555-1212” },
     { “Cell” : “703-525-1234” } ],
  “Interests: [ “golf”, “movies” ]
}
```

versus

## RDBMS Tables
(Inter-tabular Relational Integrity)

**Contacts**

| | |
|---|---|
| ContactID | 123 |
| FirstName | Bob |
| LastName | Smith |
| BirthDate | 03/11/1985 |

**Interests**

| | |
|---|---|
| InterestID | 3 |
| ContactID | 123 |
| Interest | Golf |

**PhoneNumbers**

| | |
|---|---|
| PhoneID | 1 |
| ContactID | 123 |
| Location | Home |
| Number | 571-555-1212 |

## Interacting with Data: Relational vs Document

### Relational Databases
**(SQL: Sequential Query Language)**

SELECT * FROM *[Table]* WHERE…

INSERT INTO *[Table]* (Col1, Col1)

UPDATE *[Table]  SET [Col1]* = *'value'*

DELETE FROM *[Table]  WHERE…*

**Filtering:** the WHERE clause
- WHERE LastName = 'Smith'
  AND InterestID  IN (1,2)

**Projection**: the SELECT clause
- SELECT FirstName, LastName… FROM

**versus**

### Document Collections
**(Documents: JSON Expressions)**

db.*collection*.find( { *criteria* }, { *projection* } )

db.*collection*.insert( { *field : 'value'* } )

db.*collection*.update( { *criteria* }, { *action* } )

db.*collection*.remove( { *criteria* } )

**Filtering:** More JSON Documents
- { "LastName" : "Smith",
  "Interests" : {"Golf", "Movies"} }

**Projection:** More JSON Documents
- { "FirstName" : 1, "BirthDate" : 0,
  "PhoneNumbers.Home" : 1 }

# The JSON Language

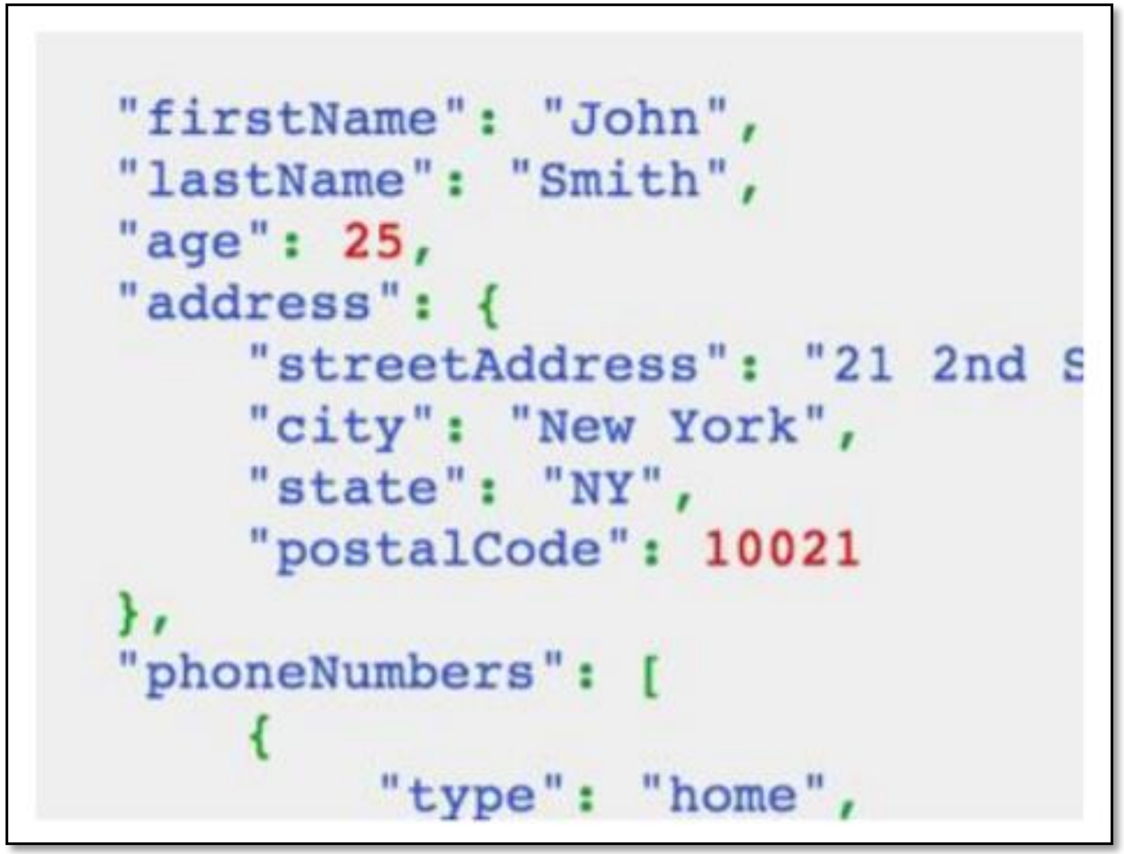
Understanding How JSON is Used to Store and Retrieve Data

UNIVERSITY of VIRGINIA

# JSON in NoSQL: How JSON Sparked NoSQL

Features of JSON

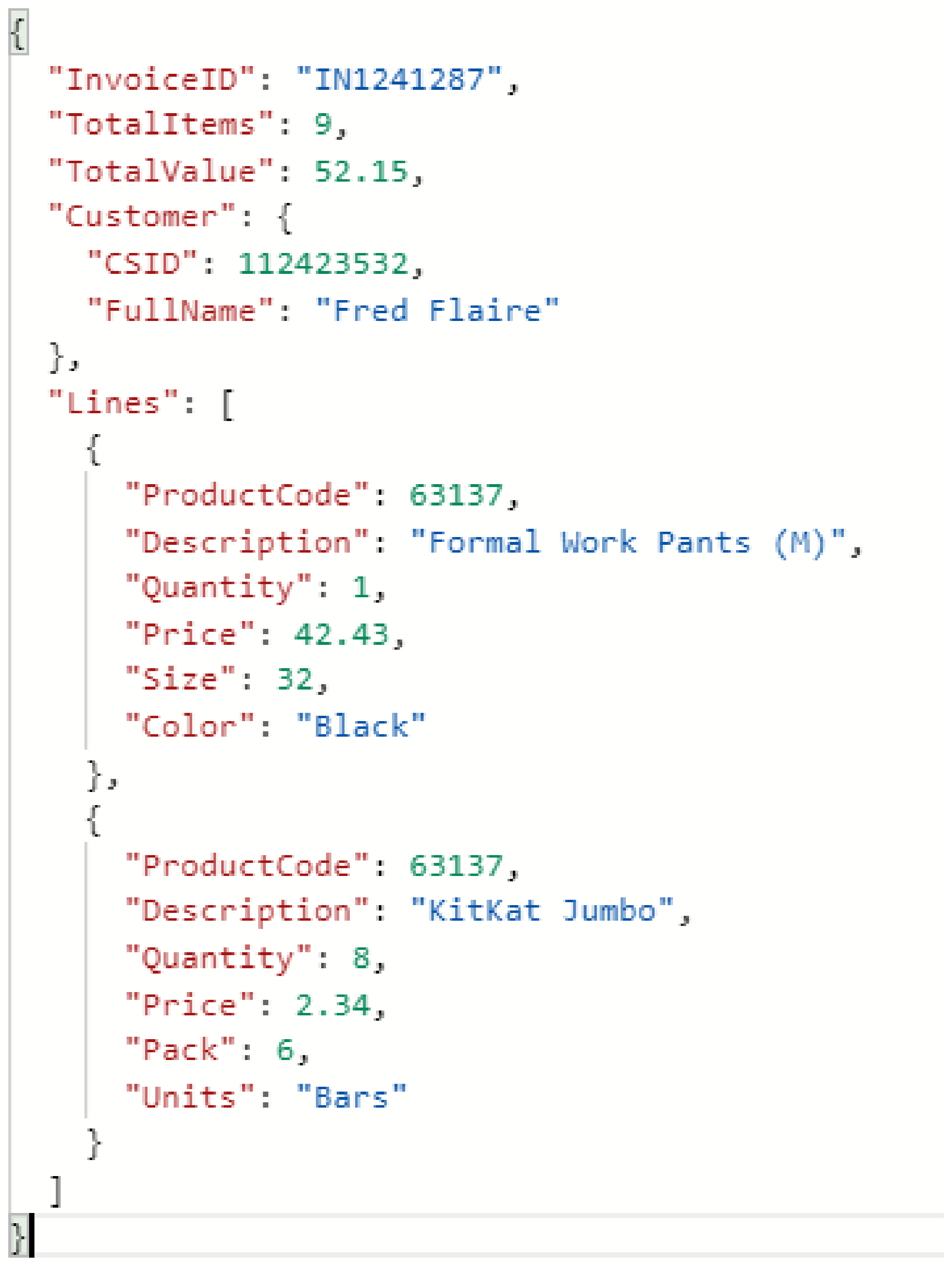
- Data Structure of the Web
- **Simple** Data Format
- Has Displaced More Complex Data Formats Such As XML and *ML
- **Developer Friendly:** Supported in Nearly Every Programming Language
- **Agility** of JSON Records
- **Extensible:** Lack of a Predefined Schema Makes Upgrades Easy



```
"firstName": "John",
"lastName": "Smith",
"age": 25,
"address": {
    "streetAddress": "21 2nd S
    "city": "New York",
    "state": "NY",
    "postalCode": 10021
},
"phoneNumbers": [
    {
        "type": "home",
```

## Document = BSON Document

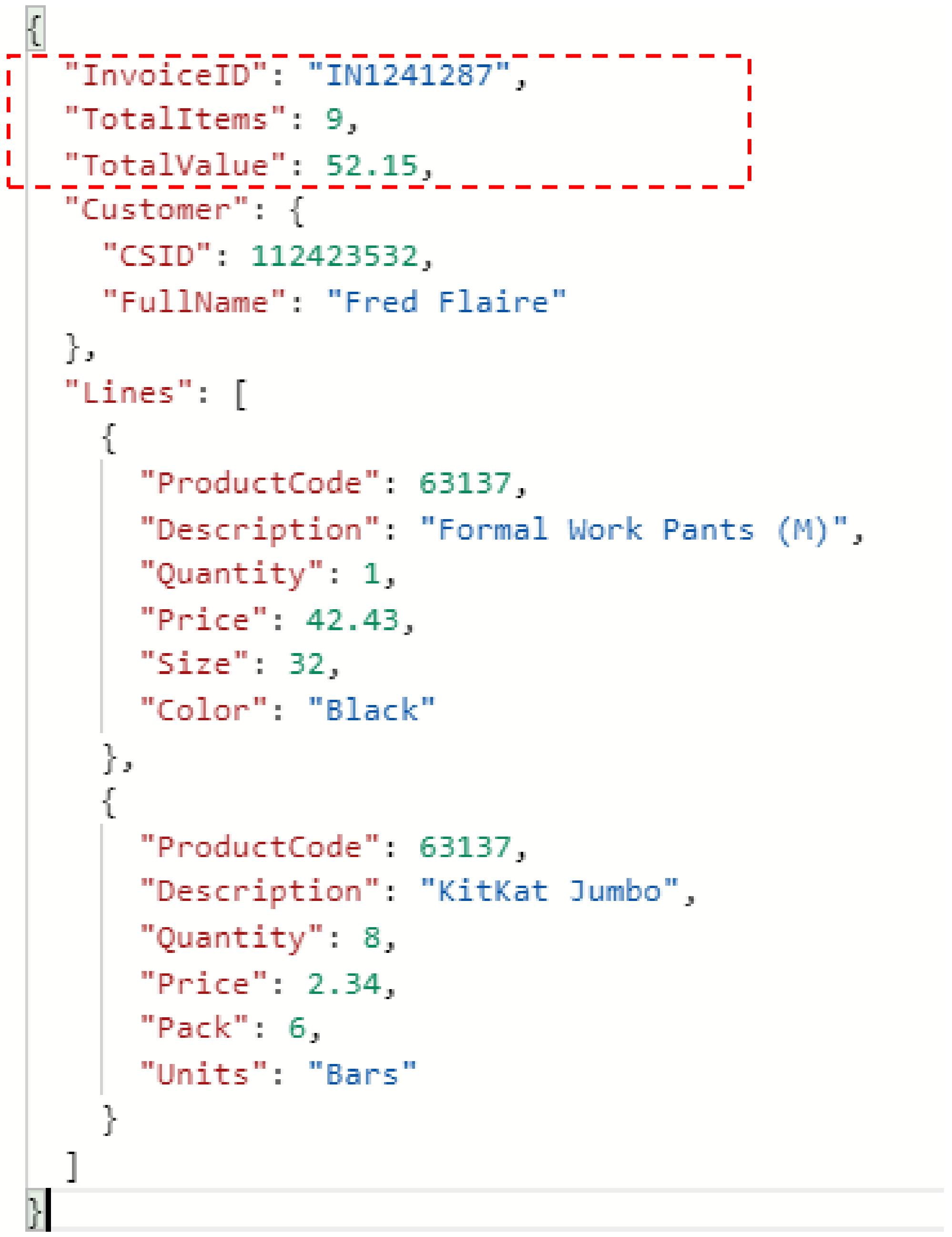
- BSON simply stands for “Binary JSON”
- Based in Open JSON document format
- Extended to add some optional non-JSON-native data types.
- MongoDB stores data in BSON format both internally, and over the network



```
{
  "InvoiceID": "IN1241287",
  "TotalItems": 9,
  "TotalValue": 52.15,
  "Customer": {
    "CSID": 112423532,
    "FullName": "Fred Flaire"
  },
  "Lines": [
    {
      "ProductCode": 63137,
      "Description": "Formal Work Pants (M)",
      "Quantity": 1,
      "Price": 42.43,
      "Size": 32,
      "Color": "Black"
    },
    {
      "ProductCode": 63137,
      "Description": "KitKat Jumbo",
      "Quantity": 8,
      "Price": 2.34,
      "Pack": 6,
      "Units": "Bars"
    }
  ]
}
```

## Document = BSON Document

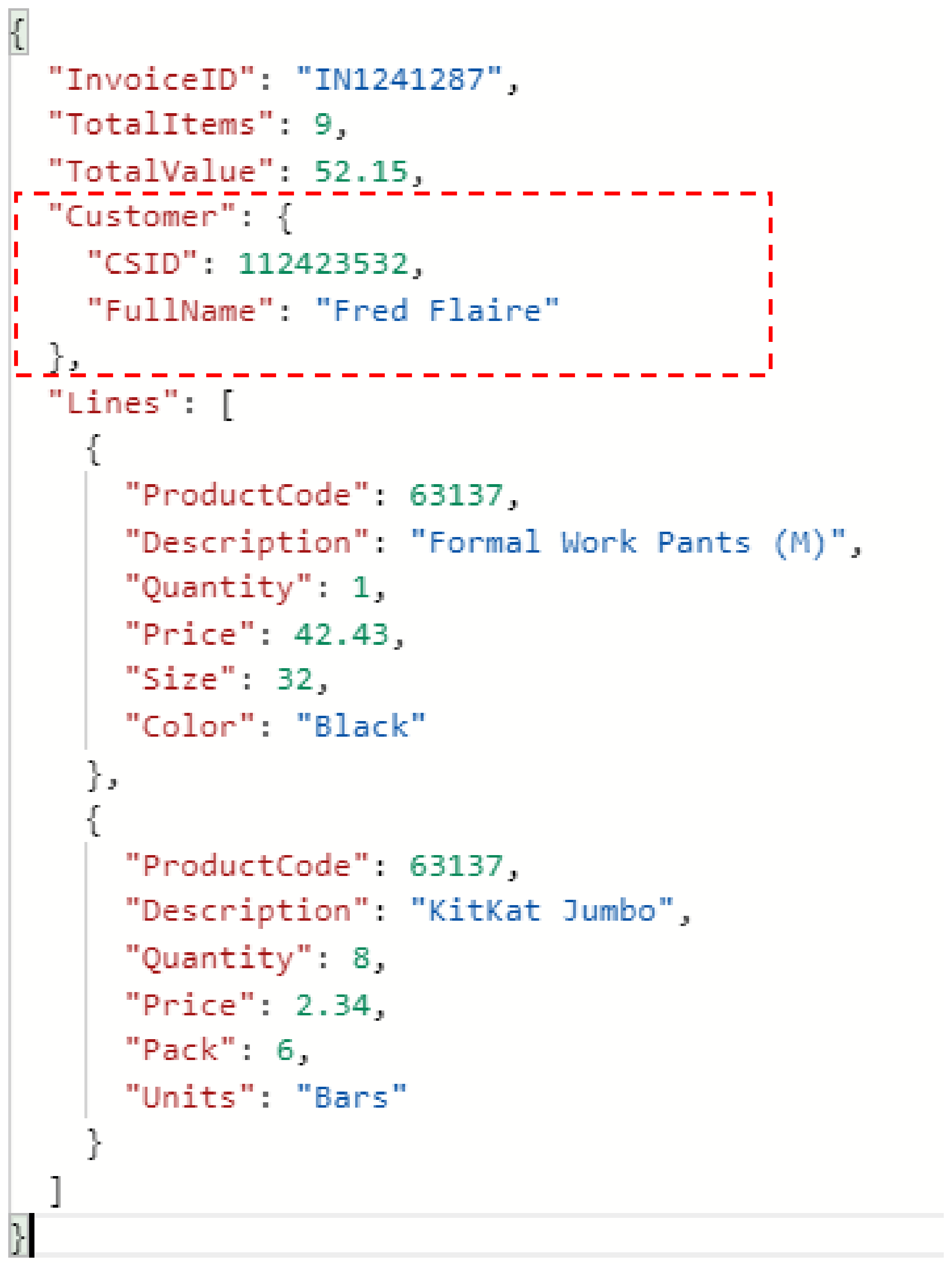
- Open JSON standard document format
- Language-independent data format
- Made up of attribute–value pairs
- Supports recursive embedding
- Supports embedded arrays
- Flexible schema



```
{
  "InvoiceID": "IN1241287",
  "TotalItems": 9,
  "TotalValue": 52.15,
  "Customer": {
    "CSID": 112423532,
    "FullName": "Fred Flaire"
  },
  "Lines": [
    {
      "ProductCode": 63137,
      "Description": "Formal Work Pants (M)",
      "Quantity": 1,
      "Price": 42.43,
      "Size": 32,
      "Color": "Black"
    },
    {
      "ProductCode": 63137,
      "Description": "KitKat Jumbo",
      "Quantity": 8,
      "Price": 2.34,
      "Pack": 6,
      "Units": "Bars"
    }
  ]
}
```

## Document = BSON Document

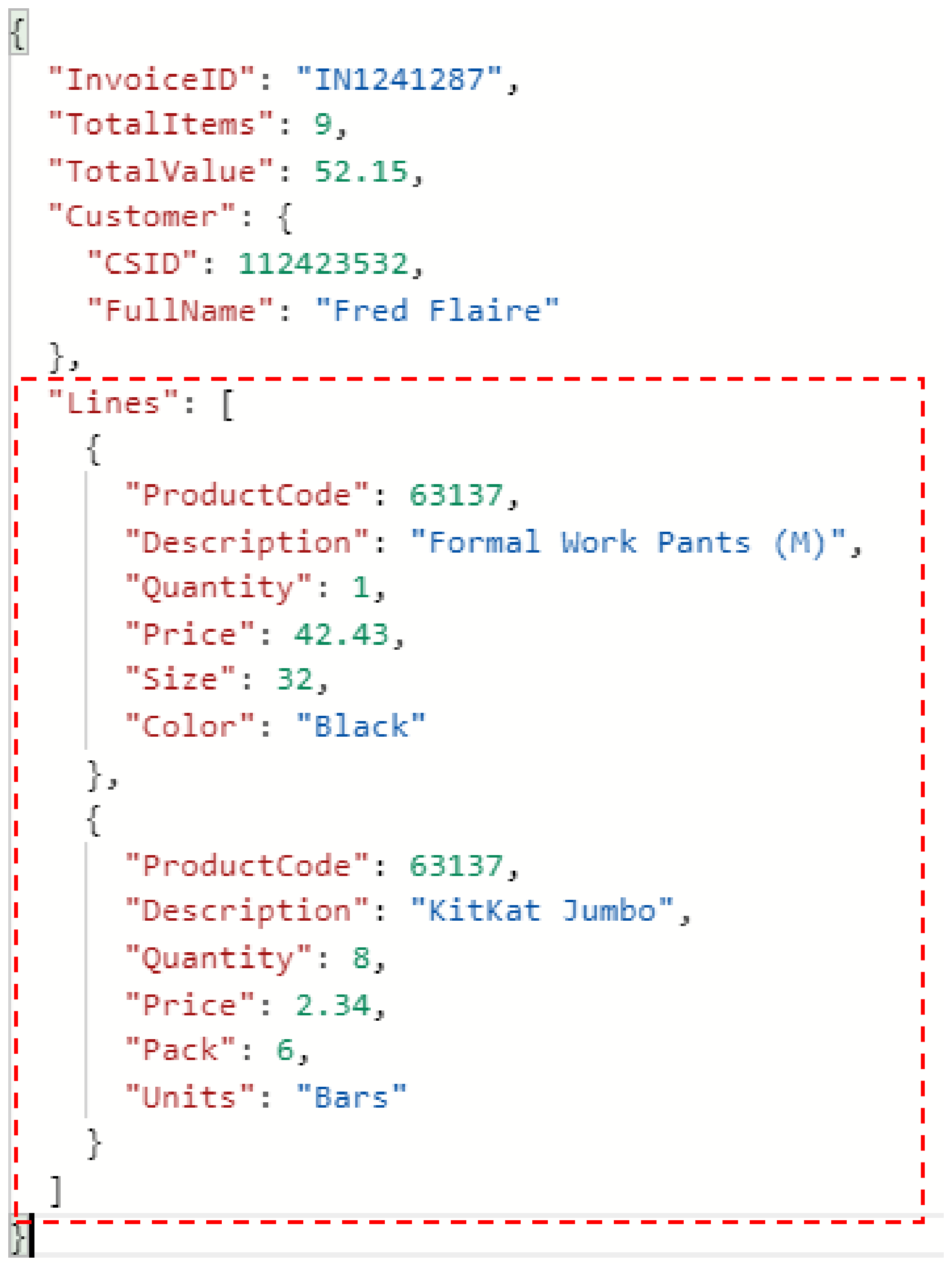
- Open JSON standard document format
- Language-independent data format
- Made up of attribute–value pairs
- Supports recursive embedding
- Supports embedded arrays
- Flexible schema



```
{
  "InvoiceID": "IN1241287",
  "TotalItems": 9,
  "TotalValue": 52.15,
  "Customer": {
    "CSID": 112423532,
    "FullName": "Fred Flaire"
  },
  "Lines": [
    {
      "ProductCode": 63137,
      "Description": "Formal Work Pants (M)",
      "Quantity": 1,
      "Price": 42.43,
      "Size": 32,
      "Color": "Black"
    },
    {
      "ProductCode": 63137,
      "Description": "KitKat Jumbo",
      "Quantity": 8,
      "Price": 2.34,
      "Pack": 6,
      "Units": "Bars"
    }
  ]
}
```

# Items = JSON Document

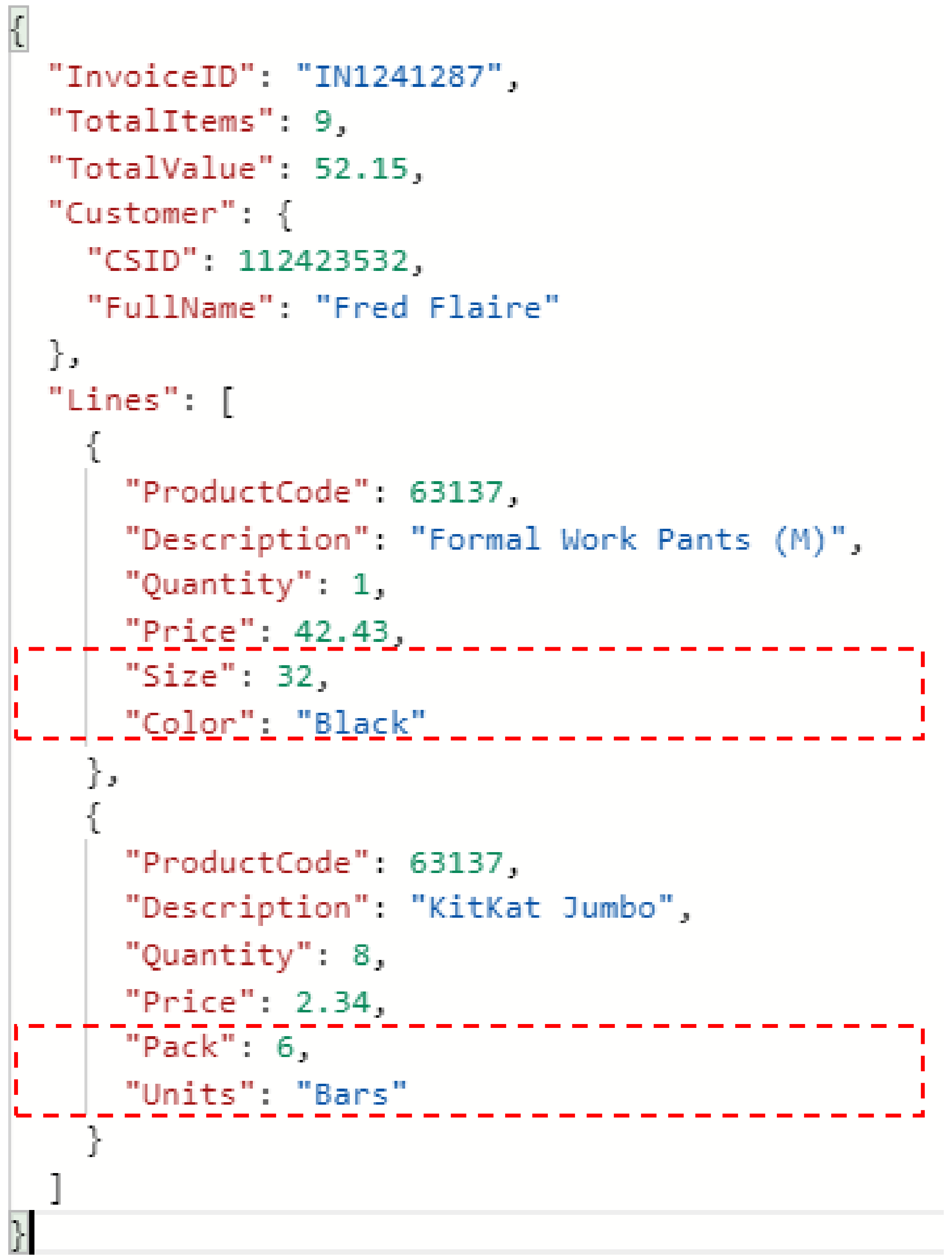
- Open standard document format
- Language-independent data format
- Made up of attribute–value pairs
- Supports recursive embedding
- Supports embedded arrays
- Flexible schema



```
{
  "InvoiceID": "IN1241287",
  "TotalItems": 9,
  "TotalValue": 52.15,
  "Customer": {
    "CSID": 112423532,
    "FullName": "Fred Flaire"
  },
  "Lines": [
    {
      "ProductCode": 63137,
      "Description": "Formal Work Pants (M)",
      "Quantity": 1,
      "Price": 42.43,
      "Size": 32,
      "Color": "Black"
    },
    {
      "ProductCode": 63137,
      "Description": "KitKat Jumbo",
      "Quantity": 8,
      "Price": 2.34,
      "Pack": 6,
      "Units": "Bars"
    }
  ]
}
```

## Document = BSON Document

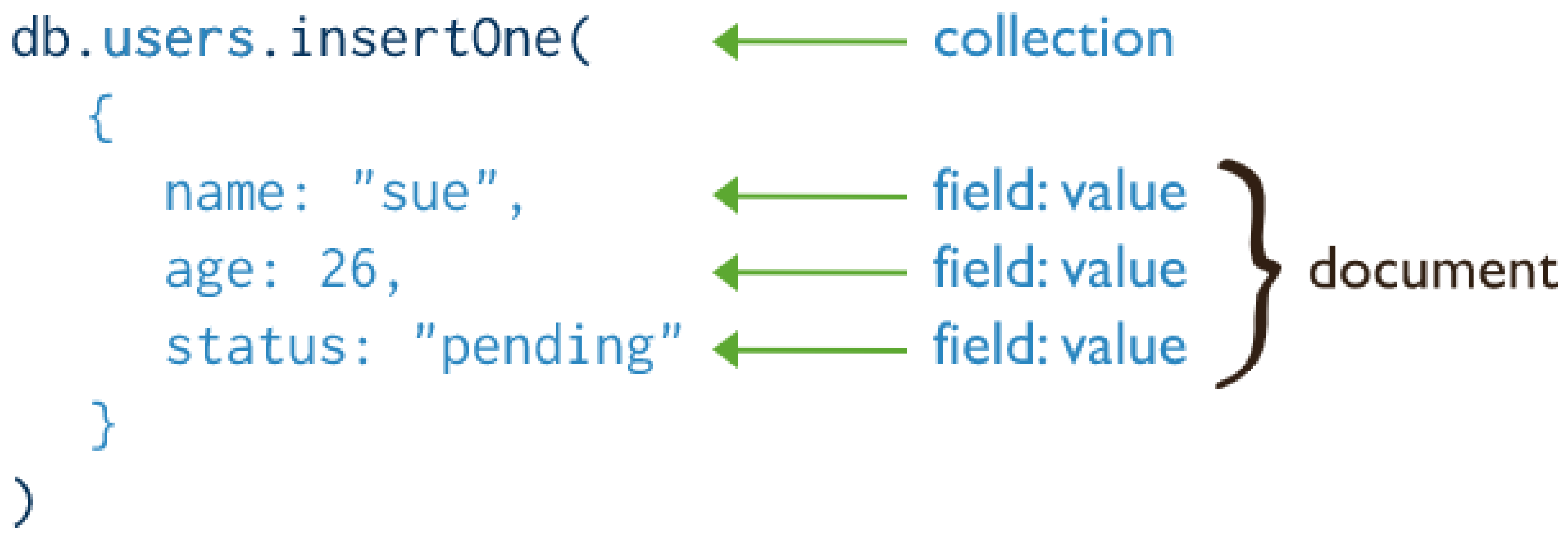
- Open JSON standard document format
- Language-independent data format
- Made up of attribute–value pairs
- Supports recursive embedding
- Supports embedded arrays
- Flexible schema



```
{
  "InvoiceID": "IN1241287",
  "TotalItems": 9,
  "TotalValue": 52.15,
  "Customer": {
    "CSID": 112423532,
    "FullName": "Fred Flaire"
  },
  "Lines": [
    {
      "ProductCode": 63137,
      "Description": "Formal Work Pants (M)",
      "Quantity": 1,
      "Price": 42.43,
      "Size": 32,
      "Color": "Black"
    },
    {
      "ProductCode": 63137,
      "Description": "KitKat Jumbo",
      "Quantity": 8,
      "Price": 2.34,
      "Pack": 6,
      "Units": "Bars"
    }
  ]
}
```

## MongoDB API CRUD Operations - Insert

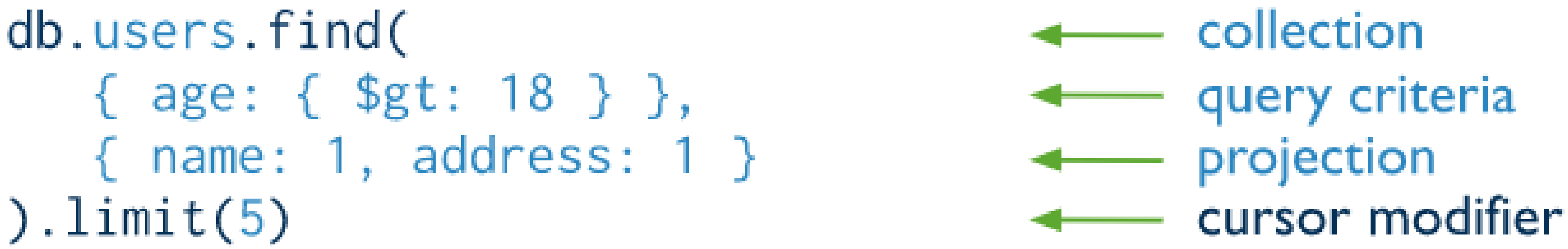
Inserting New Documents (Rows) Into a Collection (Table)



```
db.users.insertOne(
   {
      name: "sue",
      age: 26,
      status: "pending"
   }
)
```

# MongoDB API CRUD Operations - Read

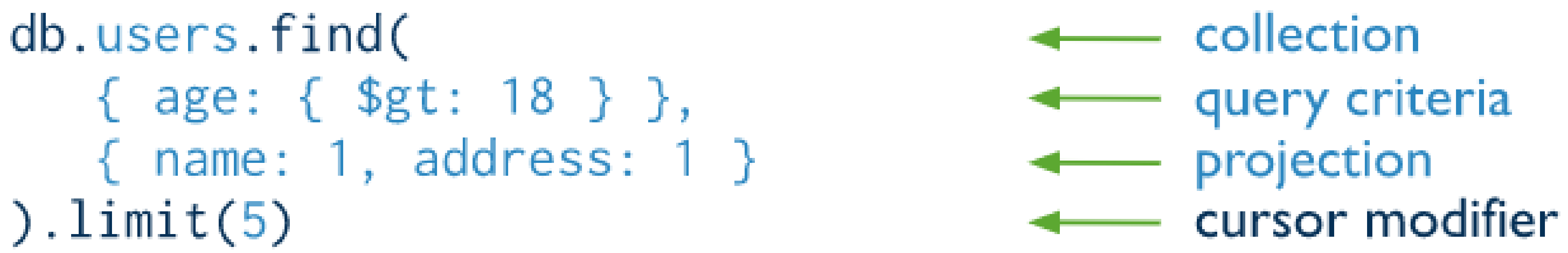
Retrieving Documents (Rows) From a Collection (Table)



```
db.users.find(
    { age: { $gt: 18 } },
    { name: 1, address: 1 }
).limit(5)
```

## MongoDB API Features – Creating Databases

Creating New Databases in MongoDB



```
db.users.find(
    { age: { $gt: 18 } },
    { name: 1, address: 1 }
).limit(5)
```

## MongoDB API Features – Creating Collections

Creating New Collections in a MongoDB Database

```
db.runCommand({
        customAction: "CreateCollection",
        collection: "testCollection",
        offerThroughput: 400
});

db.runCommand({
        customAction: "UpdateCollection",
        collection: "testCollection",
        offerThroughput: 1200 });
```

# MongoDB API Features – Indexing

| | |
|---|---|
| **Single Field Indexes:** | • db.coll.createIndex({name:1}) |
| **Compound indexes:** | • db.coll.createIndex({name:1,age:1}) |
| **Geospatial Indexes:** | • db.coll.createIndex({ location : "2dsphere" }) |
| **Wildcard Indexes:** | • db.coll.createIndex( { "$**" : 1 } ) |

# Q & A

A Survey of Data Management Systems