Relational VS NoSQL

Recap

- Relational DBMS software has worked very well for many decades
- Companies have invested lots of money in software built upon relational DBMS infrastructure
- Companies have invested in staff/talent skilled in RDBMS technologies

BUT

- RDBMS systems struggle to scale to support Big Data’s volume, variety, and velocity demands
- Big Data is exploding faster than RDBMS technologies can handle

Relational VS NoSQL

NoSQL (“Not Only SQL”)

- Uses Clusters:
    - Distribute the Data via Replication & Sharding
    - Distribute the Processing Across Multiple Nodes in a Cluster
- Uses Replication to provide
    - Redundancy
    - High Availability
    - Parallel Processing
- VERY Horizontally scalable

Relational VS NoSQL

NoSQL ("Not Only SQL")

- Requires Less Structure
  - Does not store data in tables with rigid row/column structure
  - Uses an "Aggregate" model (very *de*-normalized)
- Restricts join capabilities
- Relaxes ACID transaction compliance
- May use a *non-SQL* query language
- Typically open source, very low-cost software acquisition

Relational VS NoSQL

Relational

- Schema defines rigid structure
  - Tables, Rows, Columns
- Foreign Key relationships enable joins
- Uses SQL language
- Maintains ACID transaction compliance
- Normalized: store a value only once
- Clustering available (but challenging)
- Leading DBMS solutions are quite expensive

NoSQL

- Stores related values in aggregates
- Flexible structure:
  - – Ranges from none to some
- No joins
- Uses alternate query language
- Relaxed ACID compliance
- Data is De-normalized
- Designed to support clustering, replication, sharding
- Almost all players are open source and low-cost

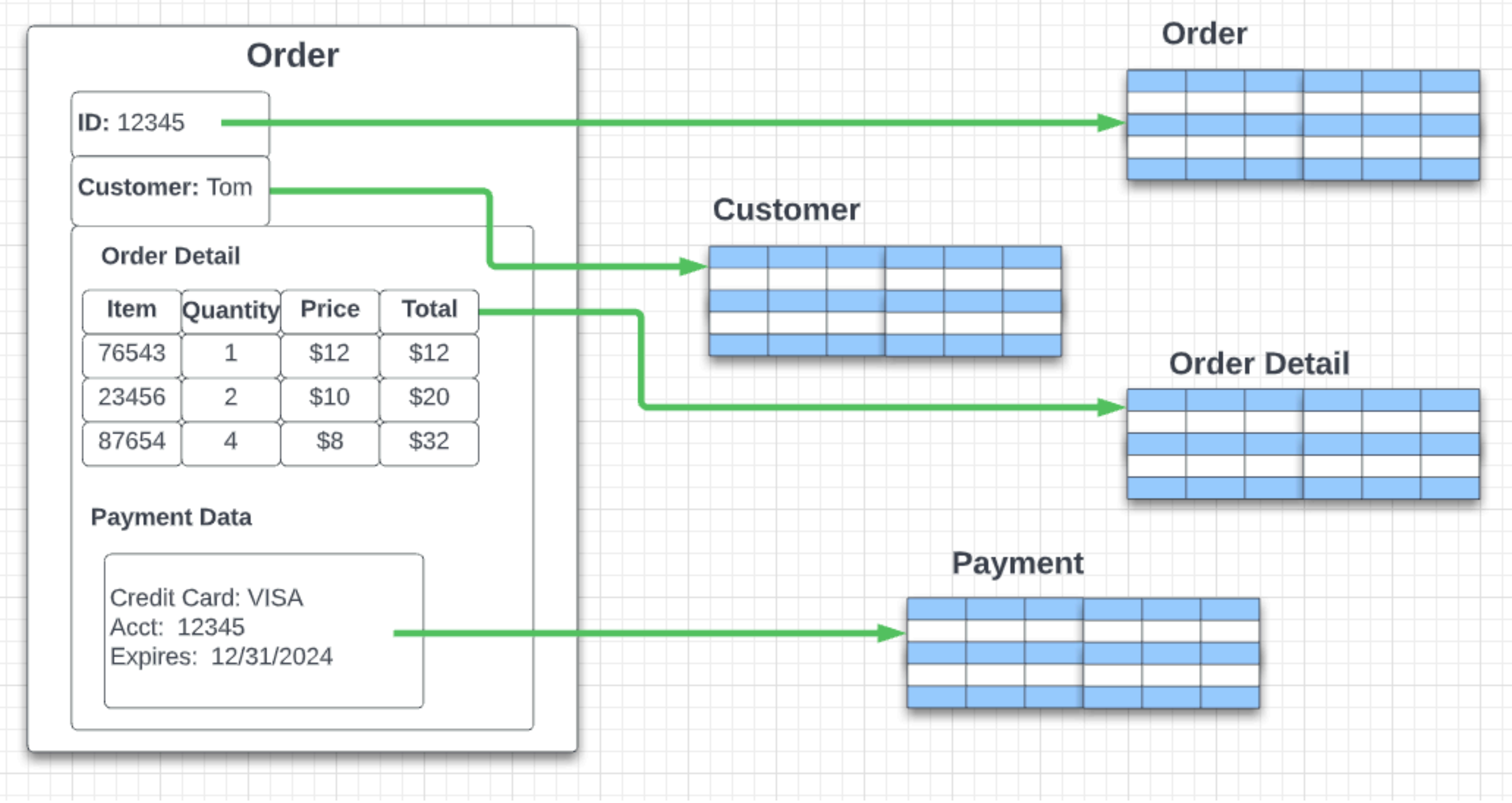
NOSQL Database Models

The “Aggregate” Data Model

- RDBMS requires Tables, Rows, Columns as data stores
  - Columns have “domains” of values
  - Third normal form – no multi-values, “store it once”

- NoSQL systems use AGGREGATES as data stores
  - Data values are grouped together as users need them
  - Think of an unnormalized document, or an “object”
  - Contains related values that are retrieved and manipulated together

NOSQL Database Models



Order

ID: 12345

Customer: Tom

Order Detail

Item	Quantity	Price	Total
76543	1	$12	$12
23456	2	$10	$20
87654	4	$8	$32

Payment Data

Credit Card: VISA
Acct: 12345
Expires: 12/31/2024

Aggregate

VS

3rd Normal Form

NOSQL Database Models

Aggregates are conceptually the opposite of 3rd Normal Form

Why aggregates?

- It is difficult to spread a relational model across nodes in a cluster
  - Replication and sharding introduce big challenges in consistency
- Each query should minimize the number of nodes being accessed across the cluster
- Data values that are accessed together should live on the same node

NOSQL Database Models

FOUR basic NOSQL Database Models

- **Document Store** (using XML or JSON format)
- **Graph** (using Node/Edge structures with Properties)
- **Key-Value** pairs
- **Wide Column Store** (rows with dynamic columns

holding key-value pairs)

NOSQL Database Models

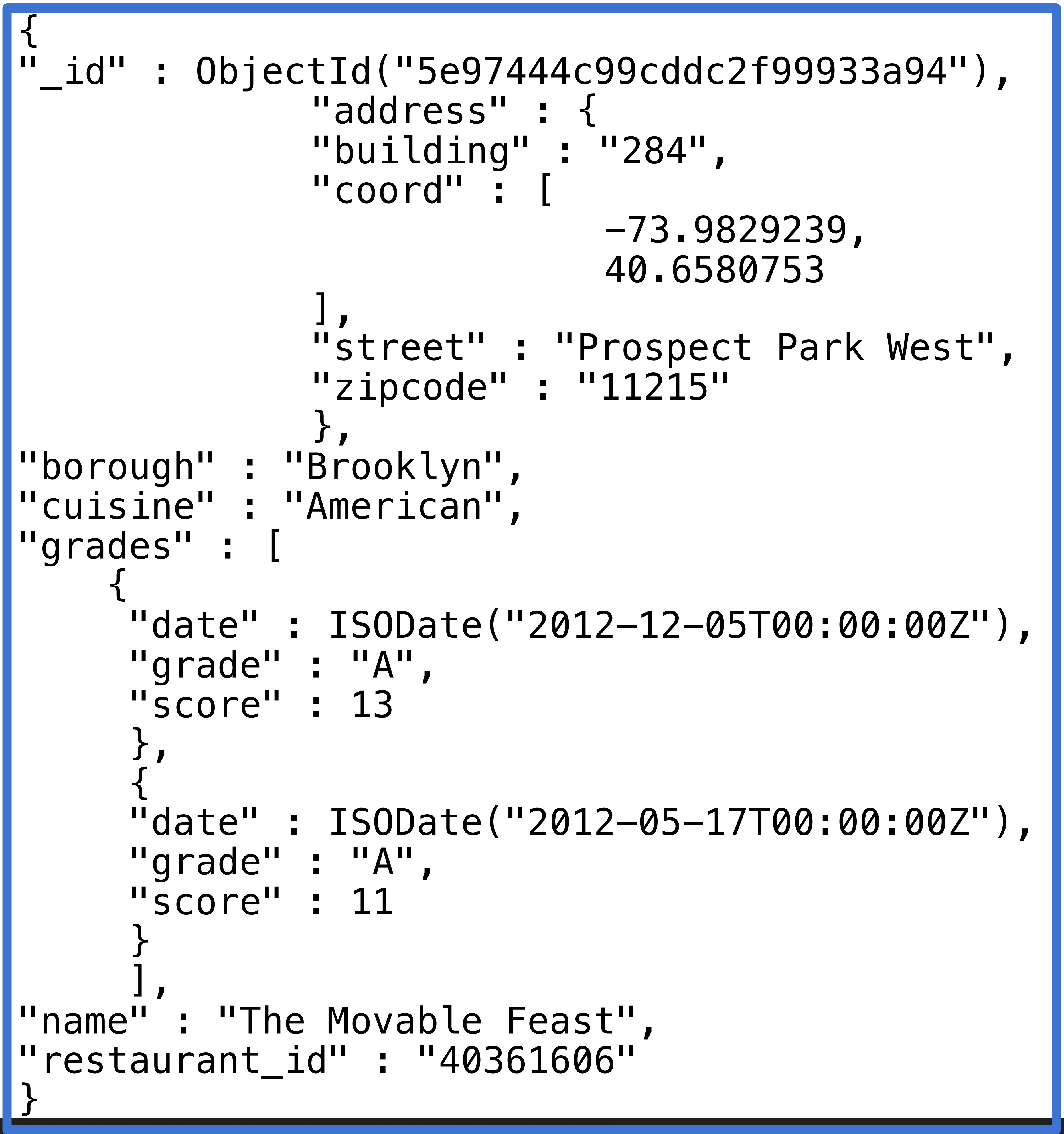
Document Database

- Organized around a “document” containing text
- Can handle very large data volumes
- Provides Speed and Scalability
- Document format is easily understood by humans
- No “schema”, but JSON/XML provides internal structure within a document
- Documents are indexed and stored within “collections”
- Supports full text search

Popular Implementations (open source)

- MongoDB
- CouchDB

NOSQL Database Models



```
{
"_id" : ObjectId("5e97444c99cddc2f99933a94"),
             "address" : {
             "building" : "284",
             "coord" : [
                          -73.9829239,
                          40.6580753
             ],
             "street" : "Prospect Park West",
             "zipcode" : "11215"
             },
"borough" : "Brooklyn",
"cuisine" : "American",
"grades" : [
    {
     "date" : ISODate("2012-12-05T00:00:00Z"),
     "grade" : "A",
     "score" : 13
     },
     {
     "date" : ISODate("2012-05-17T00:00:00Z"),
     "grade" : "A",
     "score" : 11
     }
     ],
"name" : "The Movable Feast",
"restaurant_id" : "40361606"
}
```

Sample Document

- JSON (Java Script Object Notation)
- Key:Value pairs provide some structure
- One primary key per document
- Not all documents must have the same key:value pairs

NOSQL Database Models

```
<contact>
<firstname>Bob</firstname>
<lastname>Smith</lastname>
<phone type="Cell">(123)555-0178</phone>
<phone type="Work">(890)555-0133</phone>
<address>
          <type>Home</type>
          <street>123 Black St.</street>
          <city>Big Rock</city>
          <state>AR</state>
          <zip>23225</zip>
          <country>USA</country>
</address>
</contact>
```

Sample Document

- XML (Extended Markup Language)
- Tag:Value pairs provide some structure

NOSQL Database Models

Document Database

- Keeps related information together (not normalized into tables)
- Access to a document is fast (index/key/URL)
- ACID compliance is maintained only within a document
- Cannot “join” across documents
- Documents are kept in “collections"

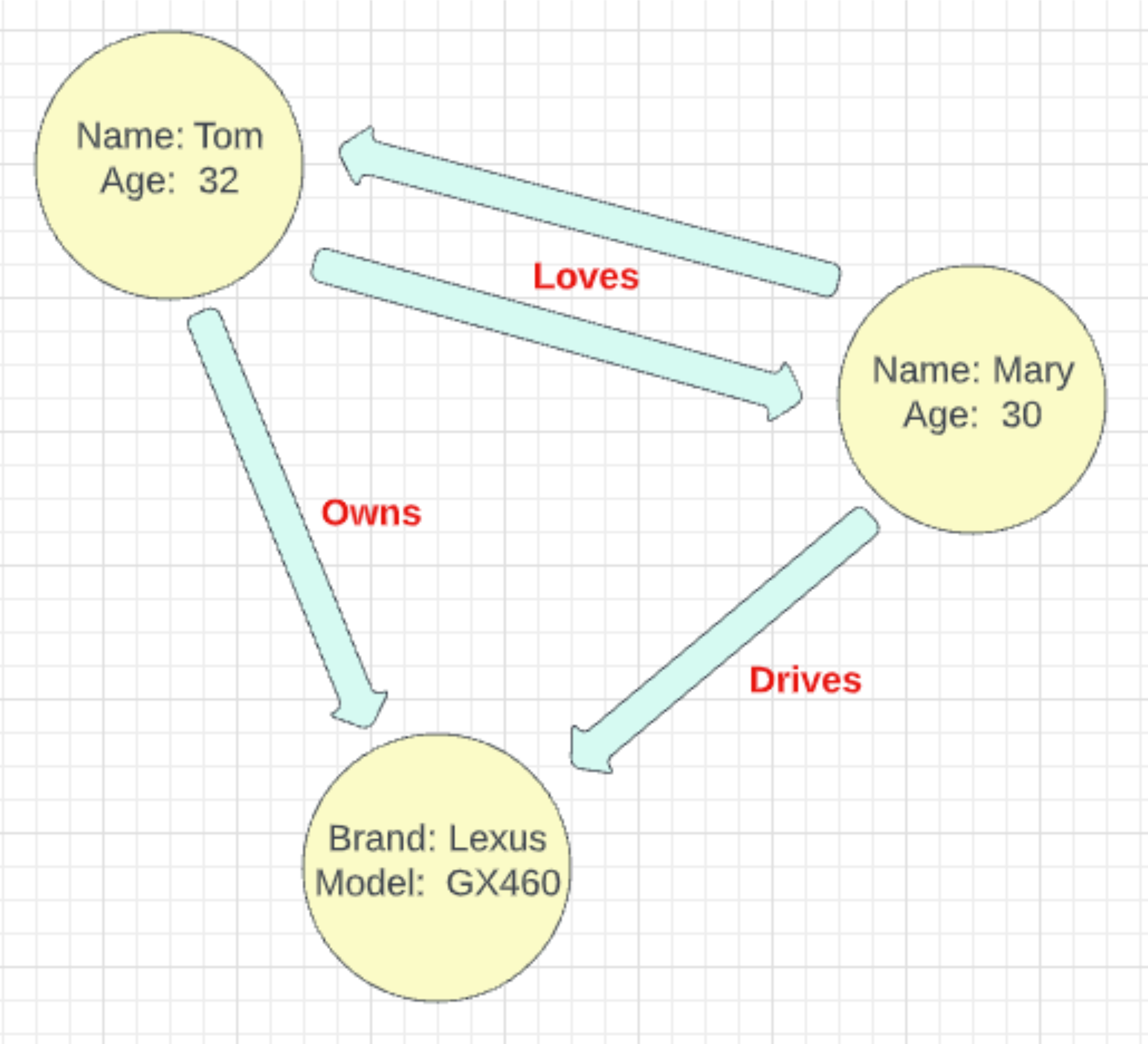
NOSQL Database Models

Graph Database

- Uses a graph structure consisting of
  - **Nodes** – Represents an entity (like a person)
  - **Edges** – Represents a relationship between entities
  - **Properties** – Attributes associated with Nodes and Edges
- Supports navigation along edges from a starting point node
- Designed for applications tracking the inter-connections among entities
  - Who is friends with whom? (In a social network application)
  - Who is following me, who am I following?
- Uses a pattern matching query language to navigate nodes & edges

- Popular Implementations (open source)
  - Neo4j

NOSQL Database Models



Sample Graph

- Nodes (with Properties)
- Edges (with Properties)
- Key:Value Pairs provice some structure

NOSQL Database Models

Key:Value Pairs Database

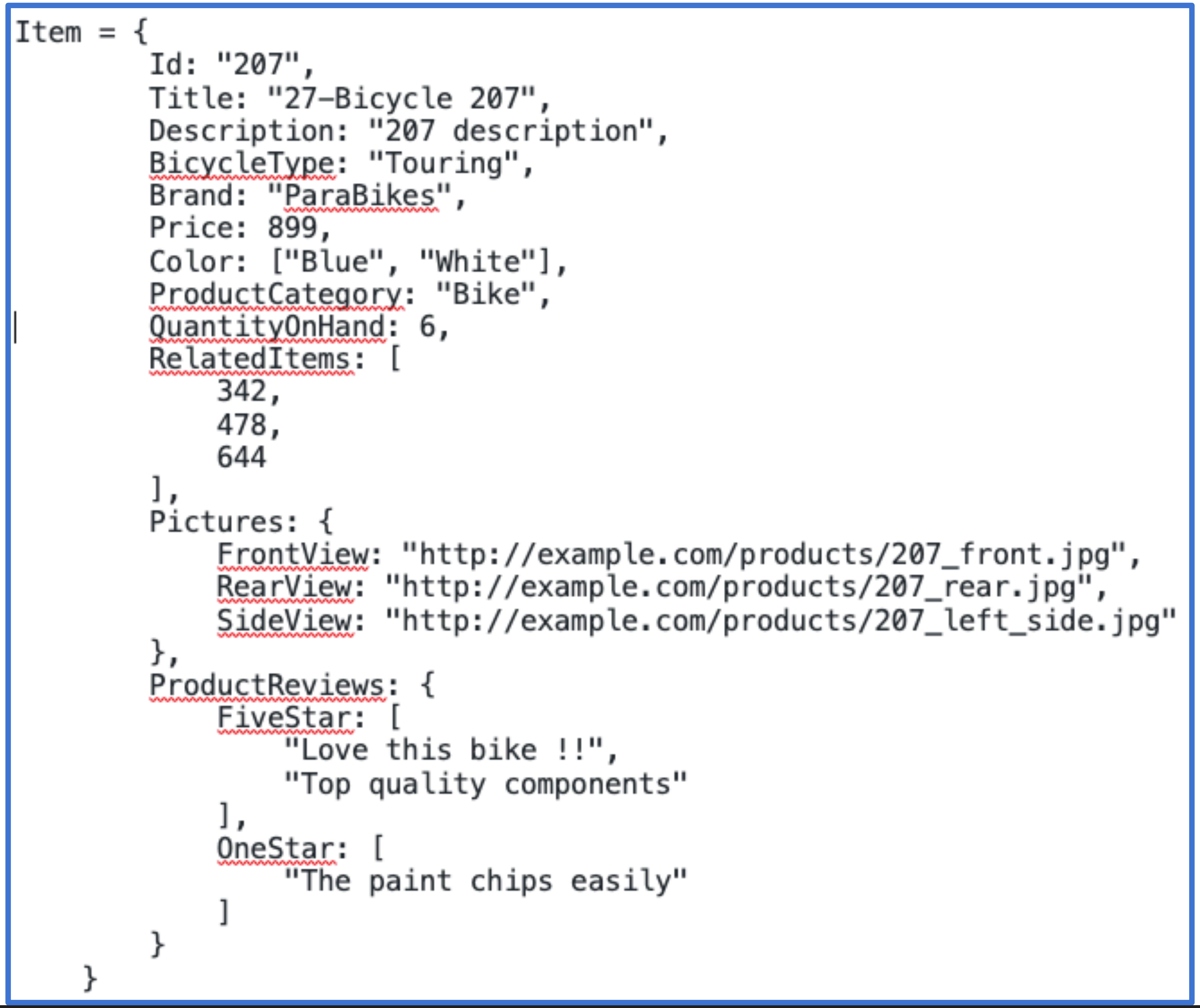
- “Schemaless” – no structure
- Maps a key to an opaque value

  (That is, the database doesn’t understand anything within the value)
- Simple query operations (put, get, remove, modify)
- Keys are unique in a collection
- May be a building block for other data models (such as key:value pairs within a document or a graph)

Popular Implementations

- Amazon Dynamo (available via AWS in the cloud)
- Redis (open source)

NOSQL Database Models



```
Item = {
        Id: "207",
        Title: "27-Bicycle 207",
        Description: "207 description",
        BicycleType: "Touring",
        Brand: "ParaBikes",
        Price: 899,
        Color: ["Blue", "White"],
        ProductCategory: "Bike",
        QuantityOnHand: 6,
        RelatedItems: [
            342,
            478,
            644
        ],
        Pictures: {
            FrontView: "http://example.com/products/207_front.jpg",
            RearView: "http://example.com/products/207_rear.jpg",
            SideView: "http://example.com/products/207_left_side.jpg"
        },
        ProductReviews: {
            FiveStar: [
                "Love this bike !!",
                "Top quality components"
            ],
            OneStar: [
                "The paint chips easily"
            ]
        }
    }
```

Sample Key:Value aggregate

- One primary key
- Each attribute has a key and a value
- May store multiple values in an array
- Key:Value Pairs provice some structure
- Not all documents will have the same key:value pairs

NOSQL Database Models

Key-Value Pairs Database Example (Amazon DynamoDB)

- The primary key value (Id) is 207.
- Most of the attributes have simple data types,

  such as String, Number, Boolean and Null.
- One attribute (Color) is a String Set in an array.
- The following attributes are document data types:
  - A List of Related Items. Each element is an Id for a related product.
  - A Map of Pictures. Each element is a short description of a picture, along with a URL for the corresponding image file.

NOSQL Database Models

Wide-Column (Column Family) Store Database

  - A TWO-LEVEL aggregate
  - Data is stored within “collections” of dynamic related columns
  - Similar to key:value with the pairs having columnar structure

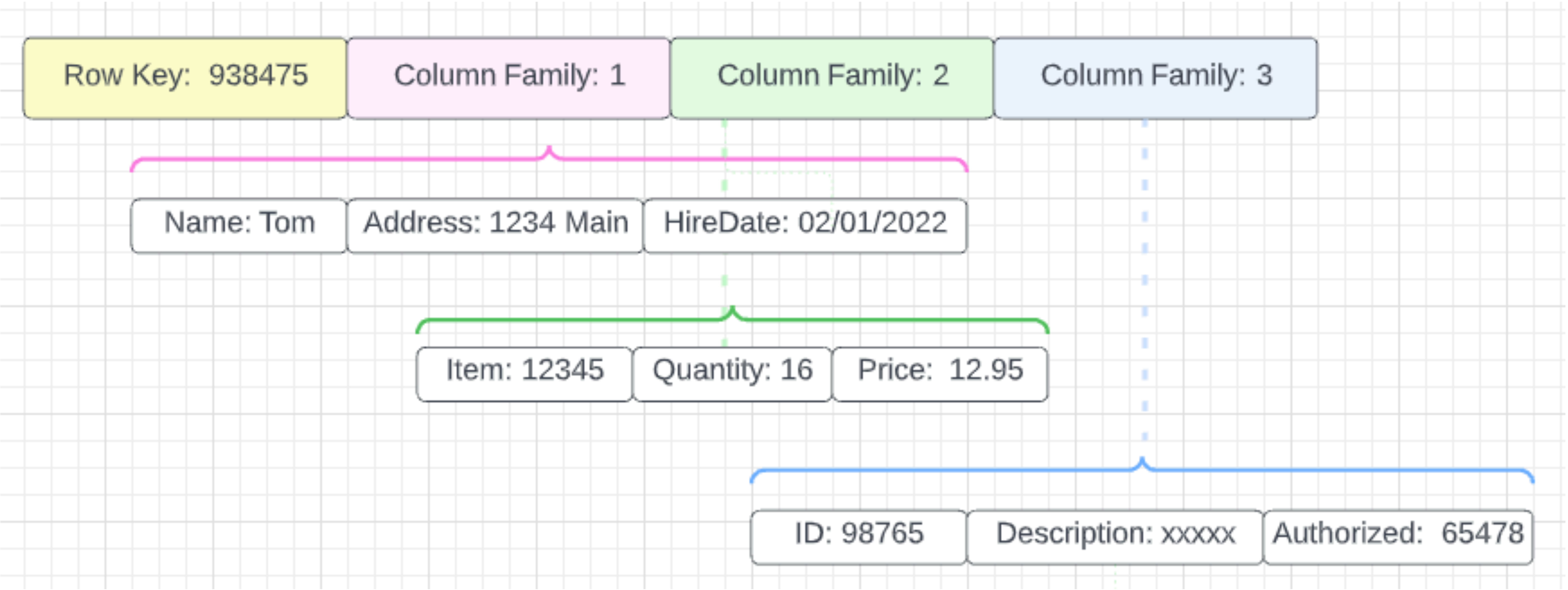
- Based on Google’s “Big Table”
- Popular Implementations (open source)
  - Cassandra
  - HBase

NOSQL Database Models

Wide-Column (Column Family) Store Database

- The first key is the row-key
  - The entire row is an aggregate of related data
  - The row consists of many key-value pairs (“columns”)
- The second key is the column family
  - Each column family consists of sparse key:value pairs
    - “Sparse” means the column value isn’t stored if it isn’t needed

NOSQL Database Models



Row Key: 938475

Column Family: 1

Column Family: 2

Column Family: 3

Name: Tom

Address: 1234 Main

HireDate: 02/01/2022

Item: 12345

Quantity: 16

Price: 12.95

ID: 98765

Description: xxxxx

Authorized: 65478

NOSQL Database Models

Benefits of a Wide Column Store

- Lookup of a row by the row key will be very fast
- In a distributed cluster system, the complete row is stored on a single node
- A row may be stored redundantly across the cluster
- A row can be retrieved with one access
- Massively scalable -- can scale to very large capacity with high availability
- The columns are "sparse"
  - That is, a column with no value is not stored

NOSQL Database Models

Issues with a Wide Column Store

- The data model is complex, and not very intuitive
- Generally, you create your data model based on your users' queries
  - The aggregation matches your users' query needs
  - Possibly taking into account how data is distributed across the cluster
  - A new query type might require a new data store

NOSQL Database Models

In Summary:

- **NoSQL Database Software can be adopted to allow organizations to better handle Big Data**
- **Four types of NoSQL databases:**
  - **Document**
  - **Graph**
  - **Key:Values Pairs**
  - **Wide Columnar**