Complex Data Types

Outline

- Semi-Structured Data
- Object Orientation
- Textual Data
- Spatial Data

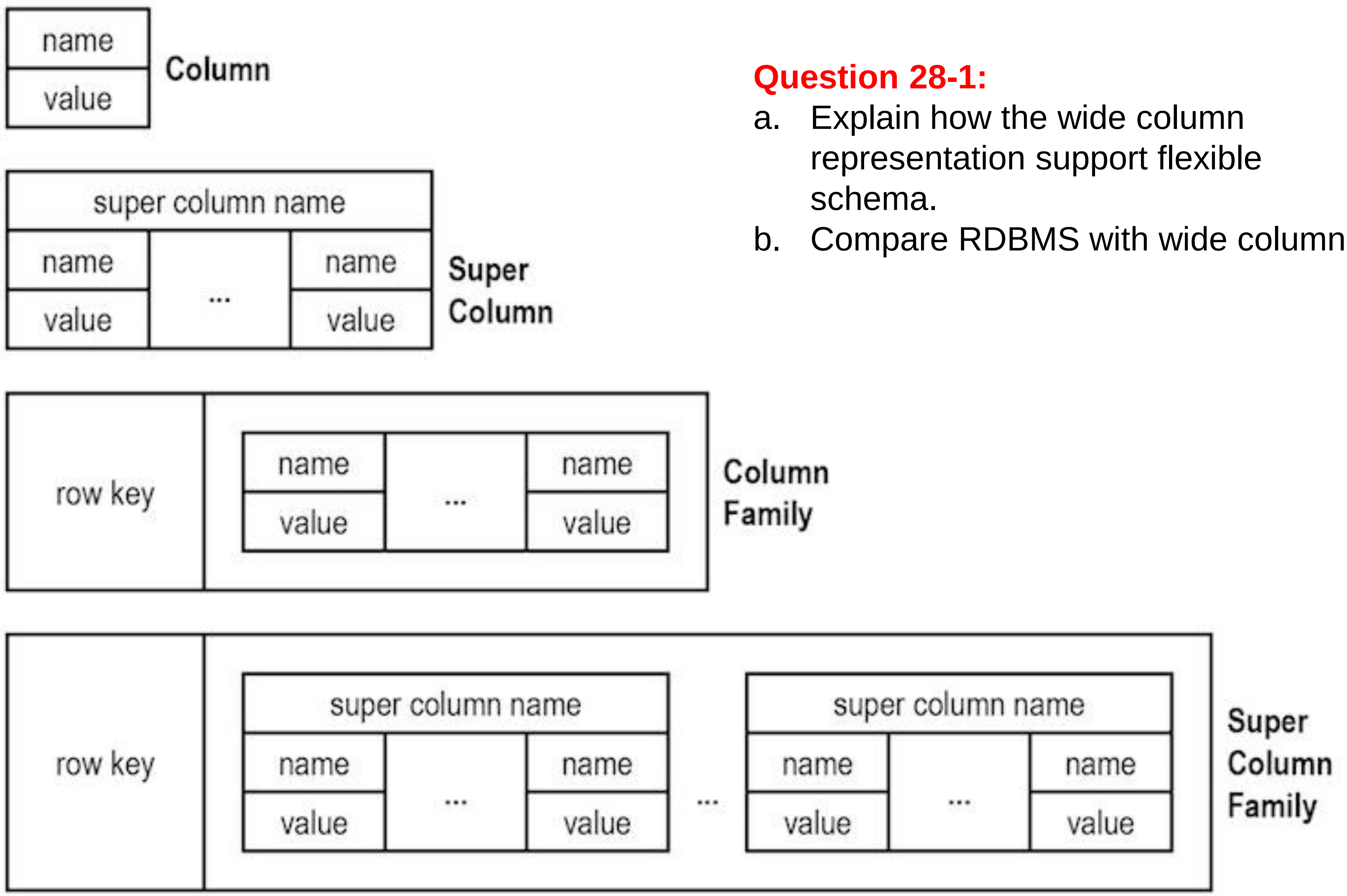
Semi-Structured Data

- Many applications require storage of complex data, whose **schema changes** often
- The relational model's requirement of atomic data types may be an overkill
  - E.g., storing set of interests as a set-valued attribute of a user profile may be simpler than normalizing it
- Data exchange can benefit greatly from semi-structured data
  - Exchange can be between applications, or between back-end and front-end of an application
  - Web-services are widely used today, with complex data fetched to the front-end and displayed using a mobile app or JavaScript
- JSON and XML are widely used semi-structured data models

Features of Semi-Structured Data Models

- **Flexible schema**
  - **Wide column** representation: allow each tuple to have a different set of attributes, can add new attributes at any time
  - Some common **wide-column** store **database examples** include Apache Cassandra, Scylla, Apache HBase, Google BigTable, and Microsoft Azure Cosmos DB.
  - When it comes to a **wide-column database**, Cassandra is often mentioned first because of its pioneering work.

Wide column



Question 28-1:

a. Explain how the wide column representation support flexible schema.

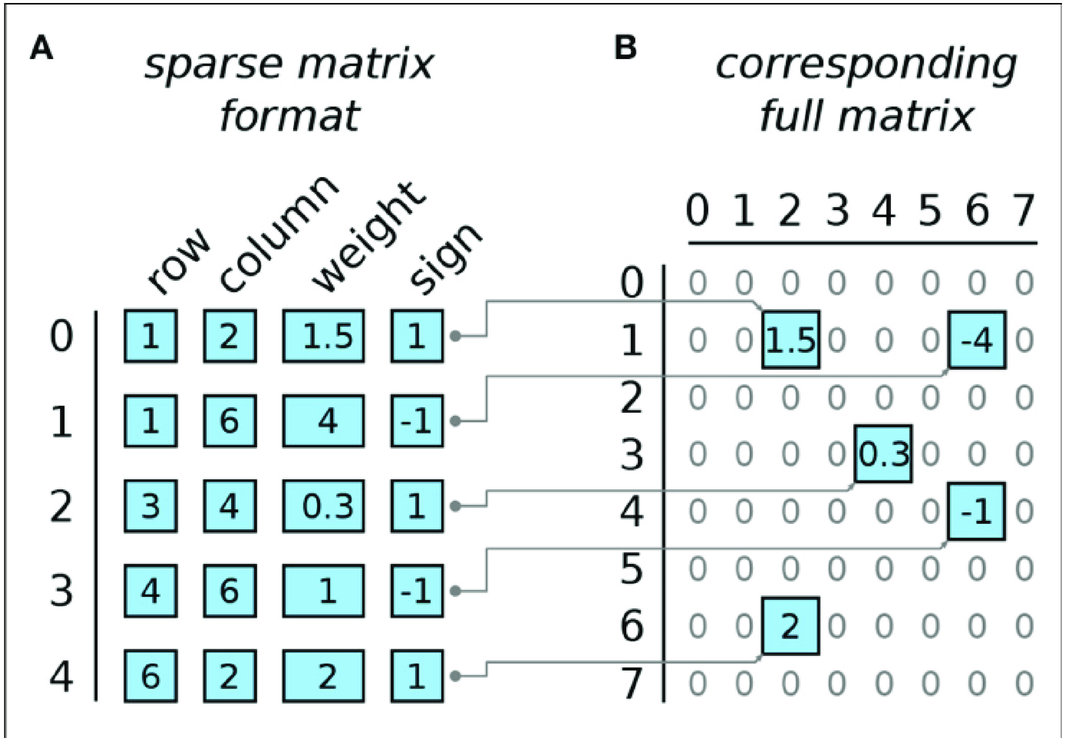
b. Compare RDBMS with wide column

Features of Semi-Structured Data Models

- **Flexible schema**
  - **Sparse column** representation: schema has a fixed but large set of attributes, by each tuple may store only a subset
  - Sparse columns are ordinary columns that have an optimized storage for null values.
  - Sparse columns reduce the space requirements for null values at the cost of more overhead to retrieve nonnull values.
  - Consider using sparse columns when the space saved is at least 20 percent to 40 percent.

Features of Semi-Structured Data Models

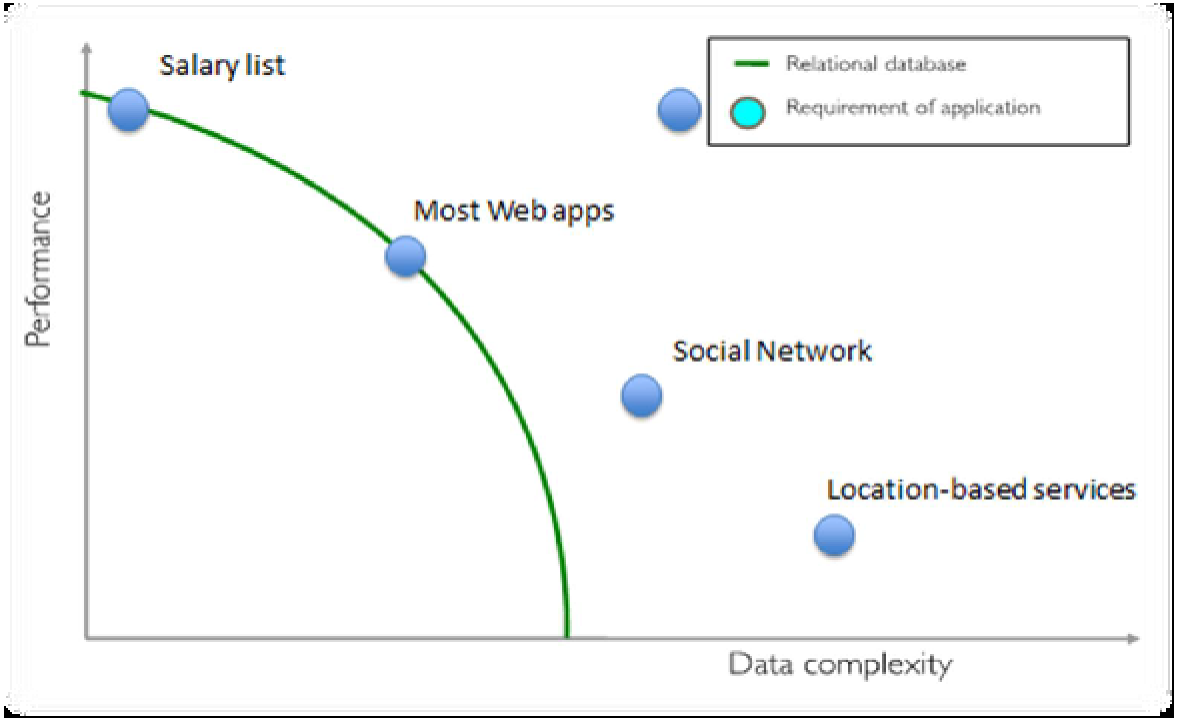
- **Flexible schema**
  - **Sparse column** representation: schema has a fixed but large set of attributes, by each tuple may store only a subset



	row	column	weight	sign
0	1	2	1.5	1
1	1	6	4	-1
2	3	4	0.3	1
3	4	6	1	-1
4	6	2	2	1

	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	1.5	0	0	0	-4	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0.3	0	0	0
4	0	0	0	0	0	0	-1	0
5	0	0	0	0	0	0	0	0
6	0	0	2	0	0	0	0	0
7	0	0	0	0	0	0	0	0

Relational DBMS VS Data Complexity



NoSQL why, what and when?

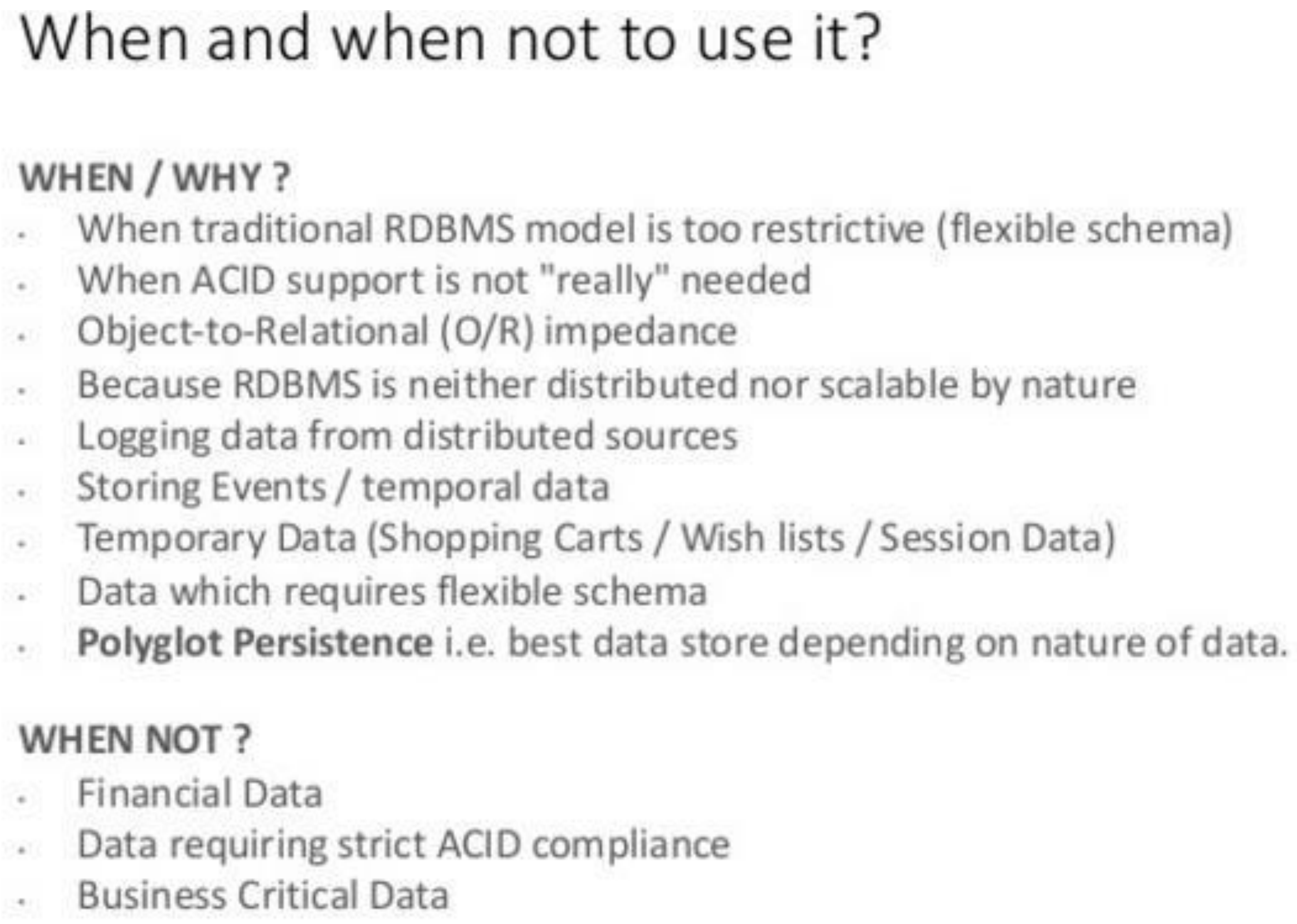
- □Relational databases were not built for **distributed applications.**

Because...

- □Joins are expensive
- □Hard to scale horizontally
- □Impedance mismatch occurs
- □Expensive (product cost, hardware, Maintenance)

- And It’s weak in:
- □Speed (performance)
- □High availability
- □Partition tolerance

NoSQL why, what and when?



When and when not to use it?

WHEN / WHY ?

- When traditional RDBMS model is too restrictive (flexible schema)
- When ACID support is not "really" needed
- Object-to-Relational (O/R) impedance
- Because RDBMS is neither distributed nor scalable by nature
- Logging data from distributed sources
- Storing Events / temporal data
- Temporary Data (Shopping Carts / Wish lists / Session Data)
- Data which requires flexible schema
- **Polyglot Persistence** i.e. best data store depending on nature of data.

WHEN NOT ?

- Financial Data
- Data requiring strict ACID compliance
- Business Critical Data

NoSQL why, what and when?

What is a schema-less data model?

In relational Databases:

□You can’t add a record which does not fit the schema

□You need to add NULLs to unused items in a row

□We should consider the datatypes. i.e : you can’t add a stirng to an interger field

□You can’t add multiple items in a field (You should create another table: primary-key, foreign key, joins, normalization, ... !!!)

NoSQL why, what and when?

In NoSQL Databases:

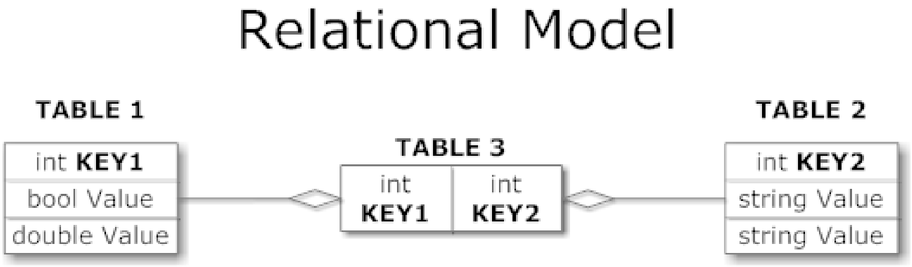
□There is no schema to consider

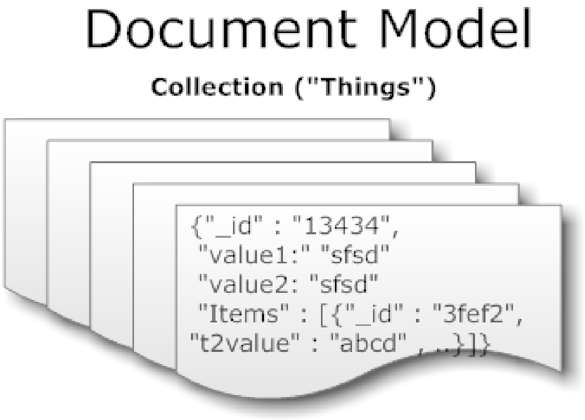
□There is no unused cell

□There is no datatype (implicit)

□Most of considerations are done in application layer

□We gather all items in an aggregate (document)





```
{"_id" : "13434",
 "value1:" "sfsd"
 "value2: "sfsd"
 "Items" : [{"_id" : "3fef2",
"t2value" : "abcd" , ...}]}
```

NoSQL why, what and when?

- A No SQL database provides a mechanism for storage and retrieval of data that employs less constrained consistency models than traditional relational database

- □No SQL systems are also referred to as "NotonlySQL“ to emphasize that they do in fact allow SQL-like query languages to be used.

Characteristics of NoSQLdatabases

- NoSQL avoids:
- □Overhead of ACID transactions
- □Complexity of SQL query
- □Burden of up-front schema design
- □DBA presence
- □Transactions (It should be handled at application layer)

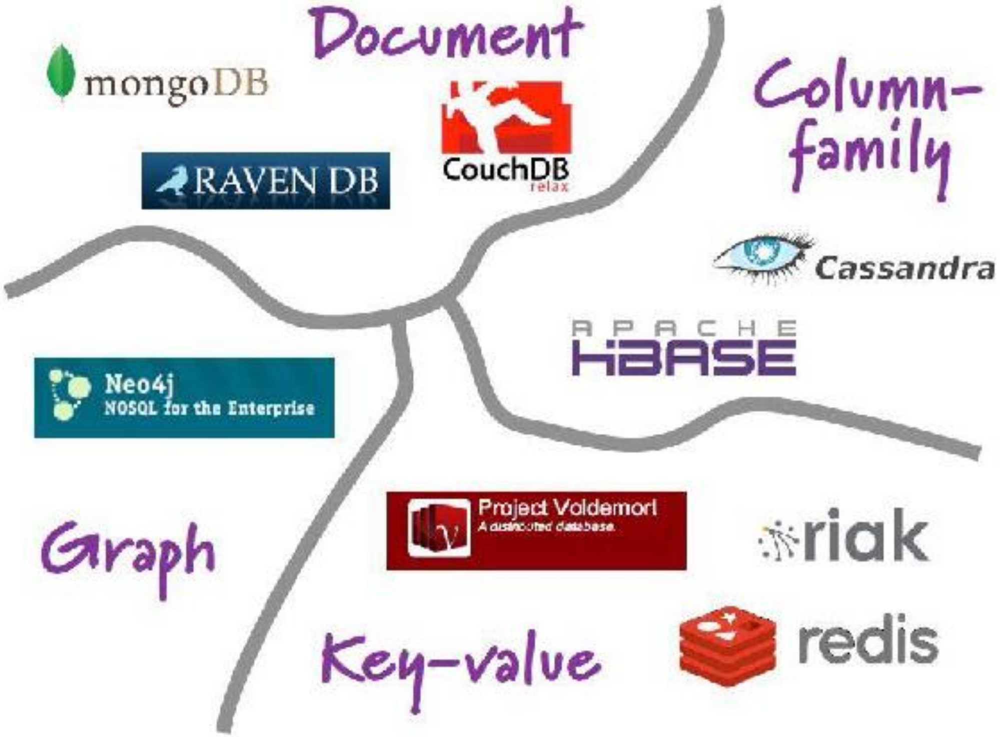
- Provides:
- □Easy and frequent changes to DB
- □Fast development
- □Large data volumes(eg.Google)
- □Schema less

NoSQL Data Models

NoSQL databases are classified in four major datamodels:

- Key-value
- Document
- Column family
- Graph

Each DB has its own query language



Key-value data model

□Simplest NOSQL databases

□The main idea is the use of a hash table

□Access data (values) by strings called keys

□Data has no required format data may have any format

□Data model: (key, value) pairs

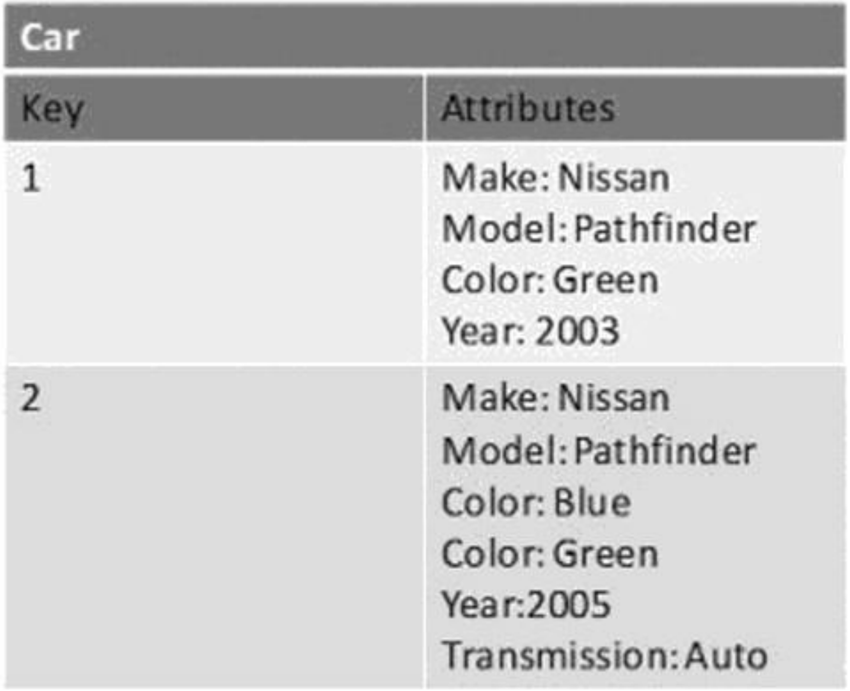
□Basic Operations:

Insert(key,value),

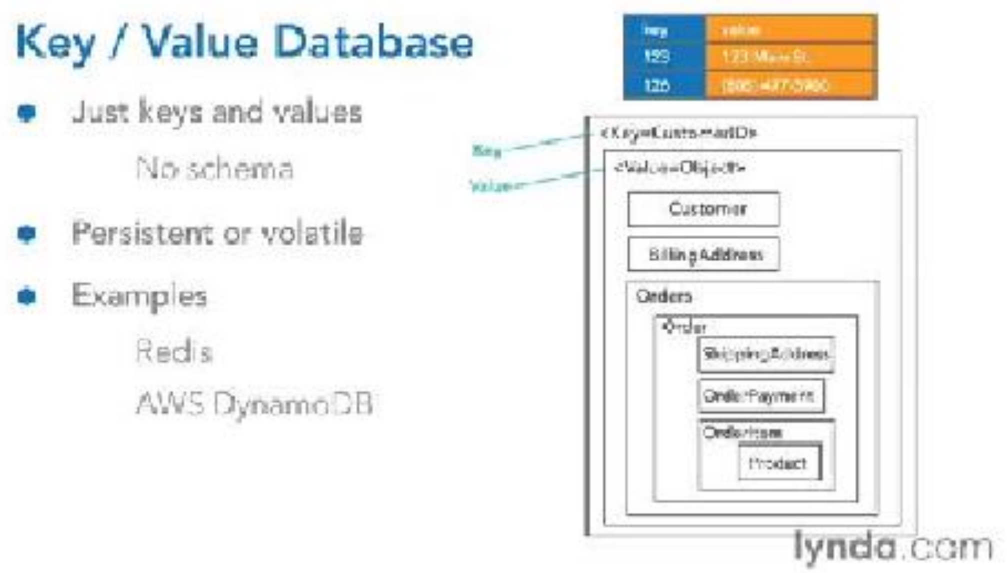
Fetch(key),

Update(key),

Delete(key)



Car	
Key	Attributes
1	Make: Nissan Model: Pathfinder Color: Green Year: 2003
2	Make: Nissan Model: Pathfinder Color: Blue Color: Green Year:2005 Transmission: Auto



Key / Value Database

- Just keys and values
  - No schema
- Persistent or volatile
- Examples
  - Redis
  - AWS DynamoDB

key	value
123	123 Main St.
126	[illegible]

Column family data model

Some statistics about Facebook Search (using **Cassandra**)

□MySQL > 50 GB Data

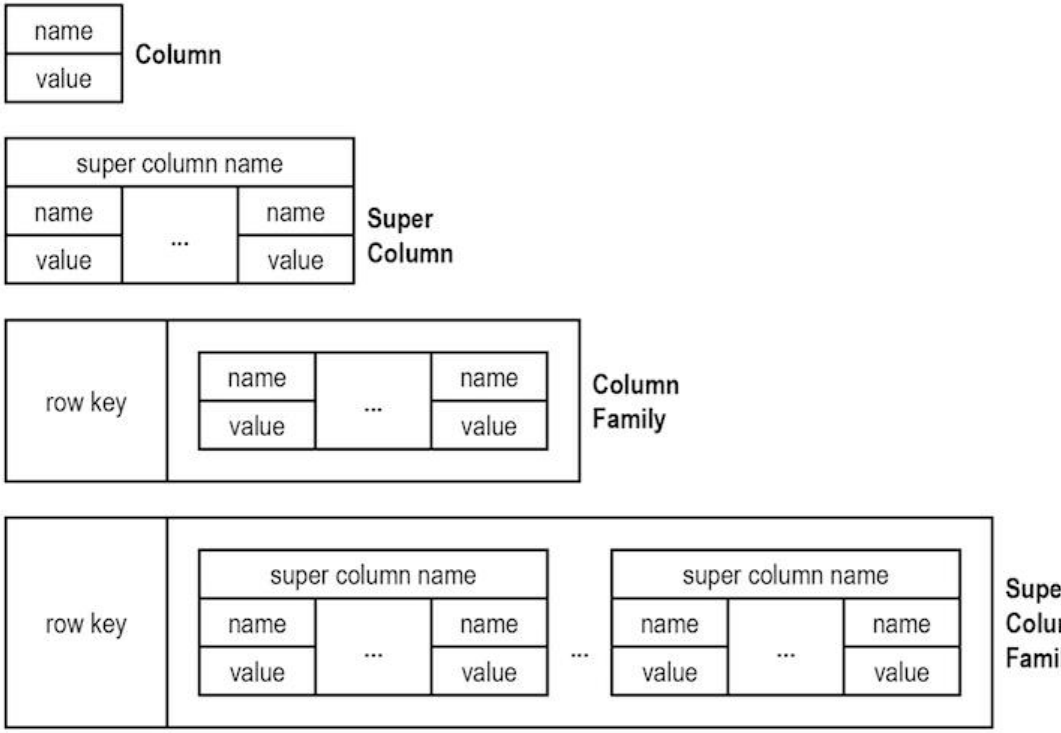
□Writes Average : ~300 ms

□Reads Average : ~350 ms

□Rewritten with Cassandra > 50 GB Data

□Writes Average : 0.12 ms

□Reads Average : 15 ms



Graph data model

□Based on Graph Theory.

□Scale vertically, no clustering.

□You can use graph algorithms easily

□Transactions

□ACID

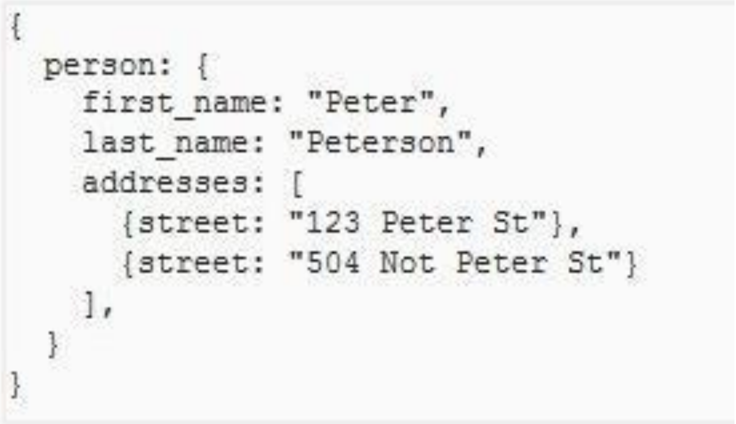
neo4j

Document based data model (MongoDB)

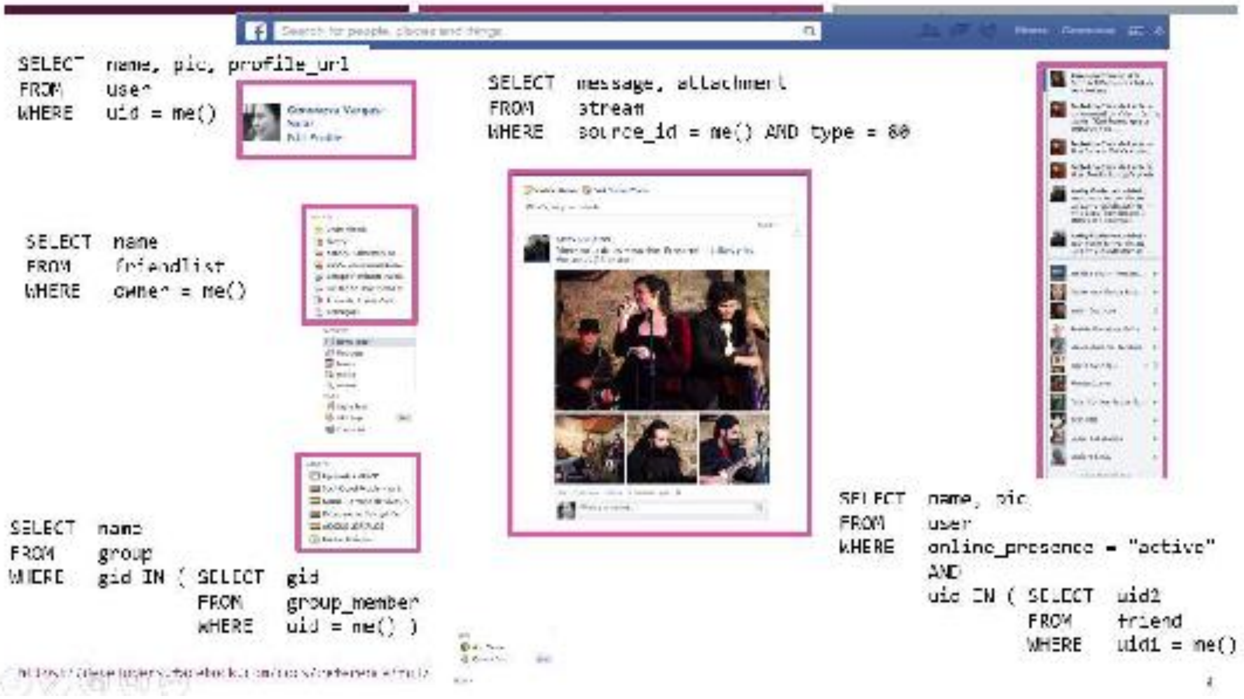
- Pair each key with complex data structure known as data structure.

- Indexes are done via B-Trees.

- Documents can contain many different key-value pairs, or key-array pairs, or even nested documents.



```
{
  person: {
    first_name: "Peter",
    last_name: "Peterson",
    addresses: [
      {street: "123 Peter St"},
      {street: "504 Not Peter St"}
    ],
  }
}
```



```
SELECT  name, pic, profile_url
FROM    user
WHERE   uid = me()
```

```
SELECT  message, attachment
FROM    stream
WHERE   source_id = me() AND type = 80
```

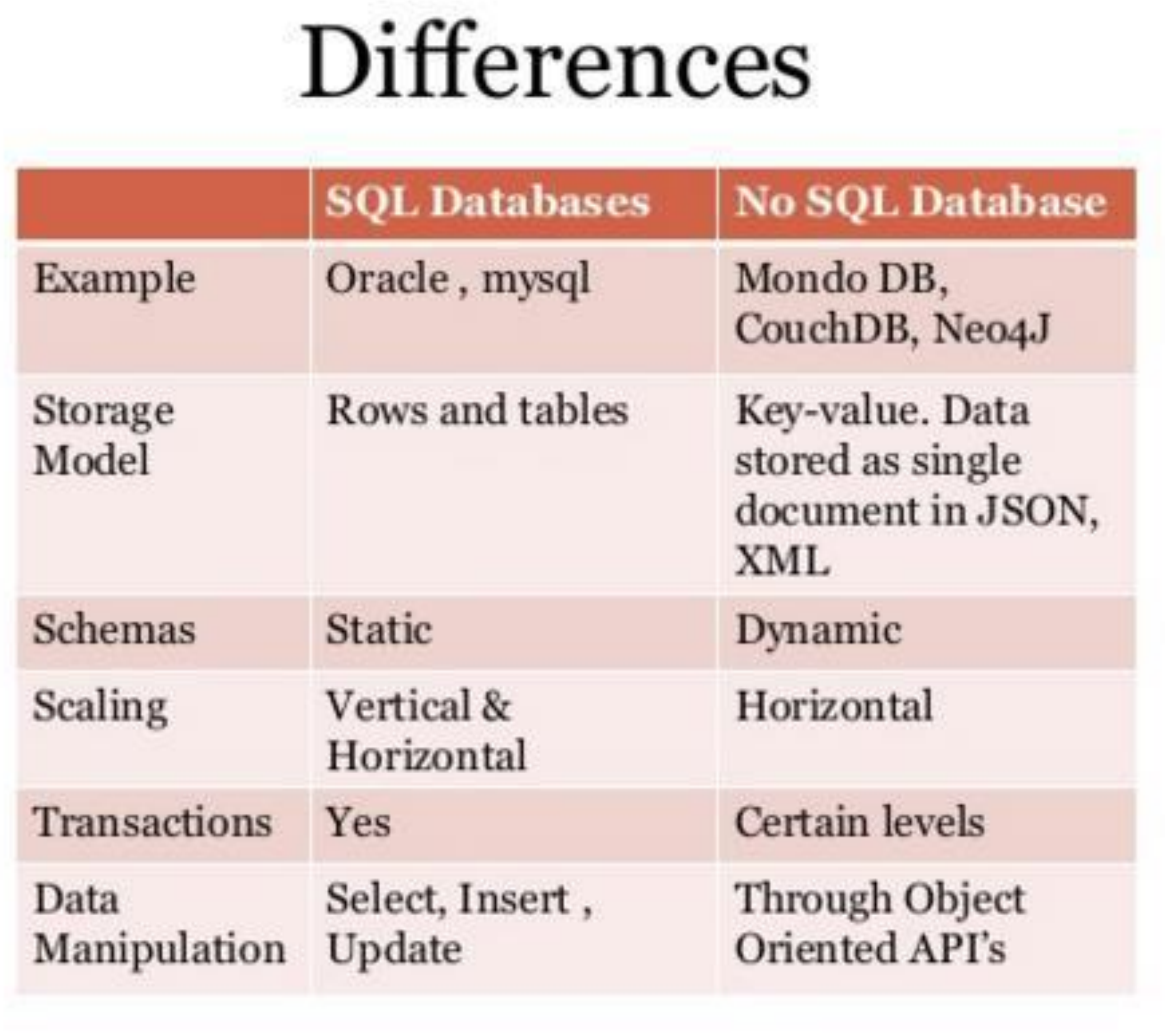
```
SELECT  name
FROM    friendlist
WHERE   owner = me()
```

```
SELECT  name, pic
FROM    user
WHERE   online_presence = "active"
        AND
        uid IN ( SELECT  uid2
                 FROM    friend
                 WHERE   uid1 = me() )
```

```
SELECT  name
FROM    group
WHERE   gid IN ( SELECT  gid
                 FROM    group_member
                 WHERE   uid = me() )
```

https://developers.facebook.com/docs/reference/fql/

SQL vs NOSQL



Differences

	SQL Databases	No SQL Database
Example	Oracle , mysql	Mondo DB, CouchDB, Neo4J
Storage Model	Rows and tables	Key-value. Data stored as single document in JSON, XML
Schemas	Static	Dynamic
Scaling	Vertical & Horizontal	Horizontal
Transactions	Yes	Certain levels
Data Manipulation	Select, Insert , Update	Through Object Oriented API's

Features of Semi-Structured Data Models

- **Multivalued data types**
  - **Sets**, **multisets**
    - E.g.,: set of interests {‘basketball, ‘La Liga’, ‘cooking’, ‘anime’, ‘jazz’}
  - **Key-value map** (or just **map** for short)
    - Store a set of key-value pairs
    - E.g., {(brand, Apple), (ID, MacBook Air), (size, 13), (color, silver)}
    - Operations on maps: *put*(key, value), *get*(key), *delete*(key)
  - , **Arrays**
    - Widely used for scientific and monitoring applications

Features of Semi-Structured Data Models

- **Arrays**
  - Widely used for scientific and monitoring applications
  - E.g., readings taken at regular intervals can be represented as array of values instead of (time, value) pairs
    - [5, 8, 9, 11] instead of {(1,5), (2, 8), (3, 9), (4, 11)}
- Multi-valued attribute types
  - Modeled using *non first-normal-form* (*NFNF*) data model
  - Supported by most database systems today
- **Array database**: a database that provides specialized support for arrays
  - E.g., compressed storage, query language extensions etc
  - Oracle GeoRaster, PostGIS, SciDB, etc

Nested Data Types

- Hierarchical data is common in many applications
- JSON: JavaScript Object Notation
    - Widely used today
- XML: Extensible Markup Language
    - Earlier generation notation, still used extensively

JSON

- Textual representation widely used for data exchange

- Since objects do not have to adhere to any fixed schema, they are basically the same as key-value maps, with the attribute names as keys and the attribute values as the associated values.
- Data is in name/value pairs
- Data is separated by commas
- Curly braces hold objects
- Square brackets hold arrays

- In JSON, values must be one of the following data types:

a string, a number, an object (JSON object), an array, a Boolean, null

JSON

- Example of JSON data

```
{
        "ID": "22222",
        "name": {
                "firstname: "Albert",
                "lastname: "Einstein"
        },
        "deptname": "Physics",
        "children": [
                {"firstname": "Hans", "lastname": "Einstein" },
                {"firstname": "Eduard", "lastname": "Einstein" }
        ]
}
```

Question 29-1: Write the relational representation of this JSON data

- Types: integer, real, string, and
  - *Objects: are* key-value maps, i.e. sets of (attribute name, value) pairs
  - Arrays are also key-value maps (from offset to value)