Unit-4 : Storing Data in BD context

Big Data (CS-3032)

Kalinga Institute of Industrial Technology
Deemed to be University
Bhubaneswar-751024

School of Computer Engineering

Strictly for internal circulation (within KIIT) and reference only. Not for outside circulation without permission

3 Credit

Lecture Note

Course Contents

Sr #	Major and Detailed Coverage Area	Hrs
4	**Storing Data in Big Data context**	8
	Data Models, RDBMS and Hadoop, Non-Relational Database, Introduction to NoSQL, Types of NoSQL, Polyglot Persistence, Sharding	

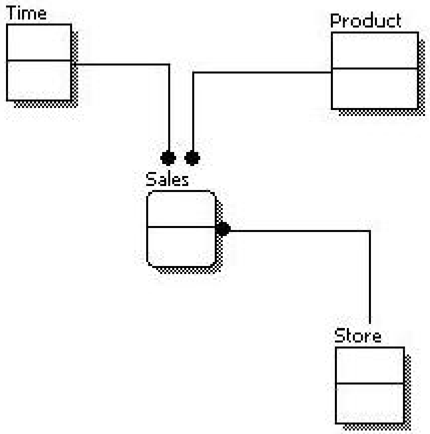
Data Model

In information technology, data architecture is composed of models, policies, rules or standards that govern which data is collected, and how it is stored, arranged, integrated, and put to use in data systems and in organizations. A few basic concepts in data architecture:

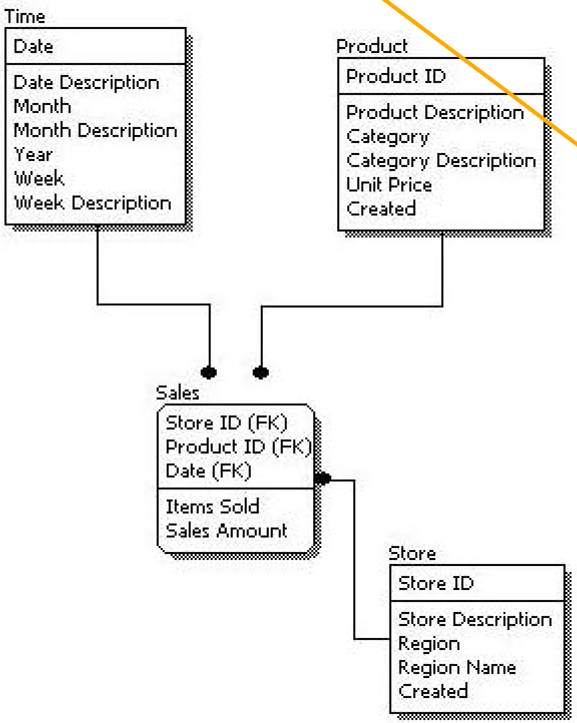
- **Conceptual data model** - shows data entities such as customer, product and transaction, and their semantics.
- **Logical model** - defines the data in as much detail as possible, including relations between data elements, but without considering how data is stored or managed.
- **Physical data model** - defines how the data is represented and stored, for example in a flat file, database, data warehouse, key-value store

Data Model cont'd …

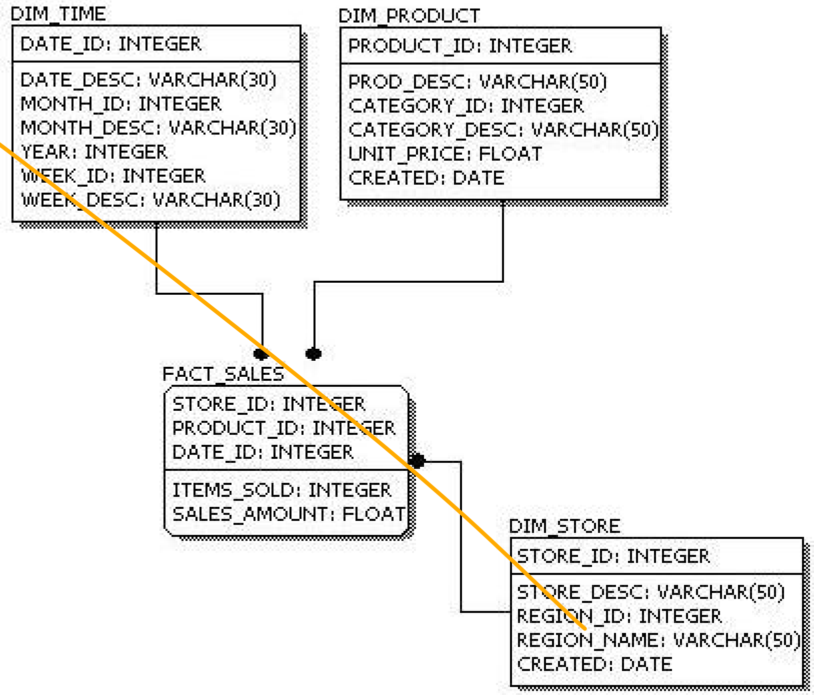
Conceptual Data Model



Logical Data Model

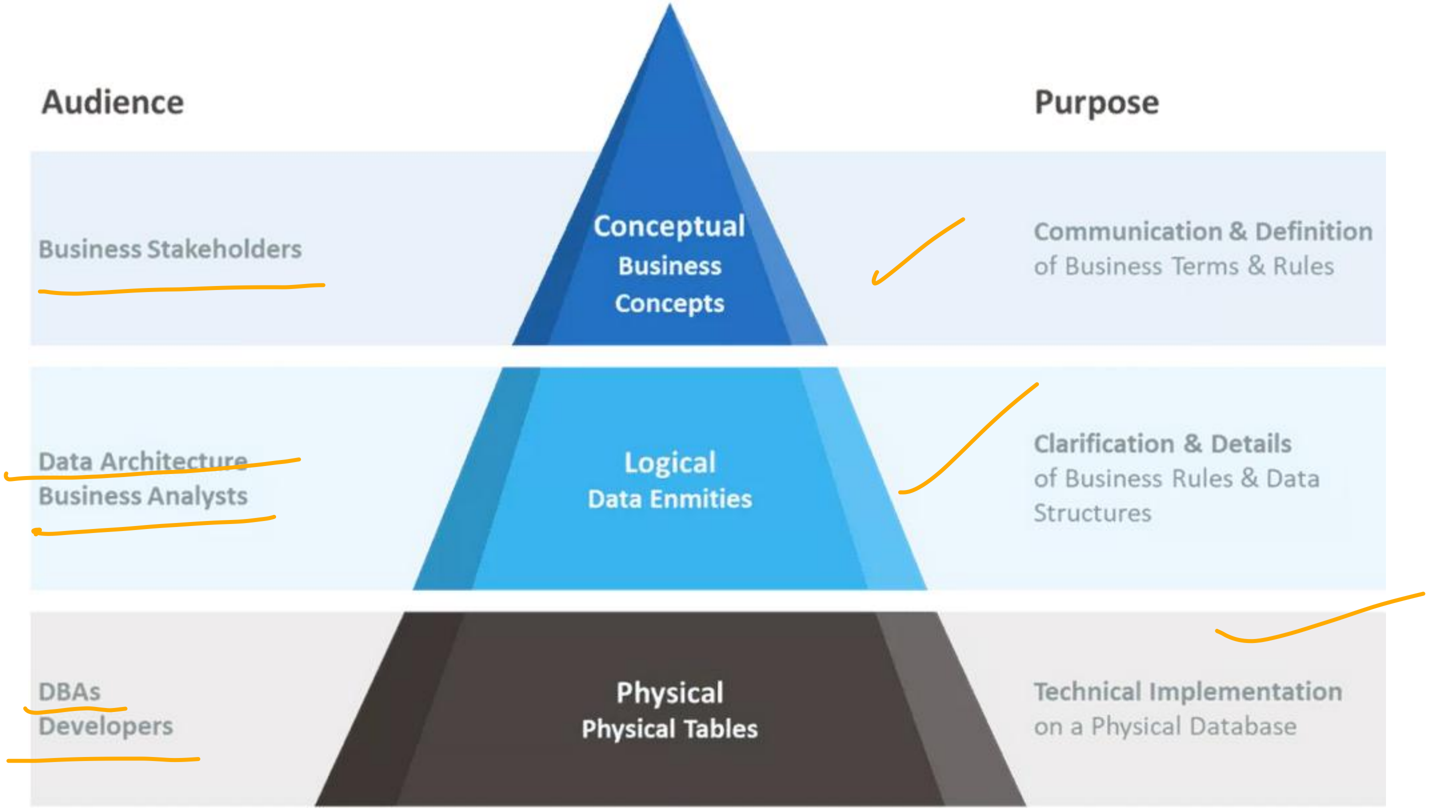


Physical Data Model



Source: 1keydata

Data Model cont'd …



Audience

Purpose

Business Stakeholders

Conceptual Business Concepts

Communication & Definition of Business Terms & Rules

Data Architecture
Business Analysts

Logical Data Enmities

Clarification & Details of Business Rules & Data Structures

DBAs
Developers

Physical Physical Tables

Technical Implementation on a Physical Database

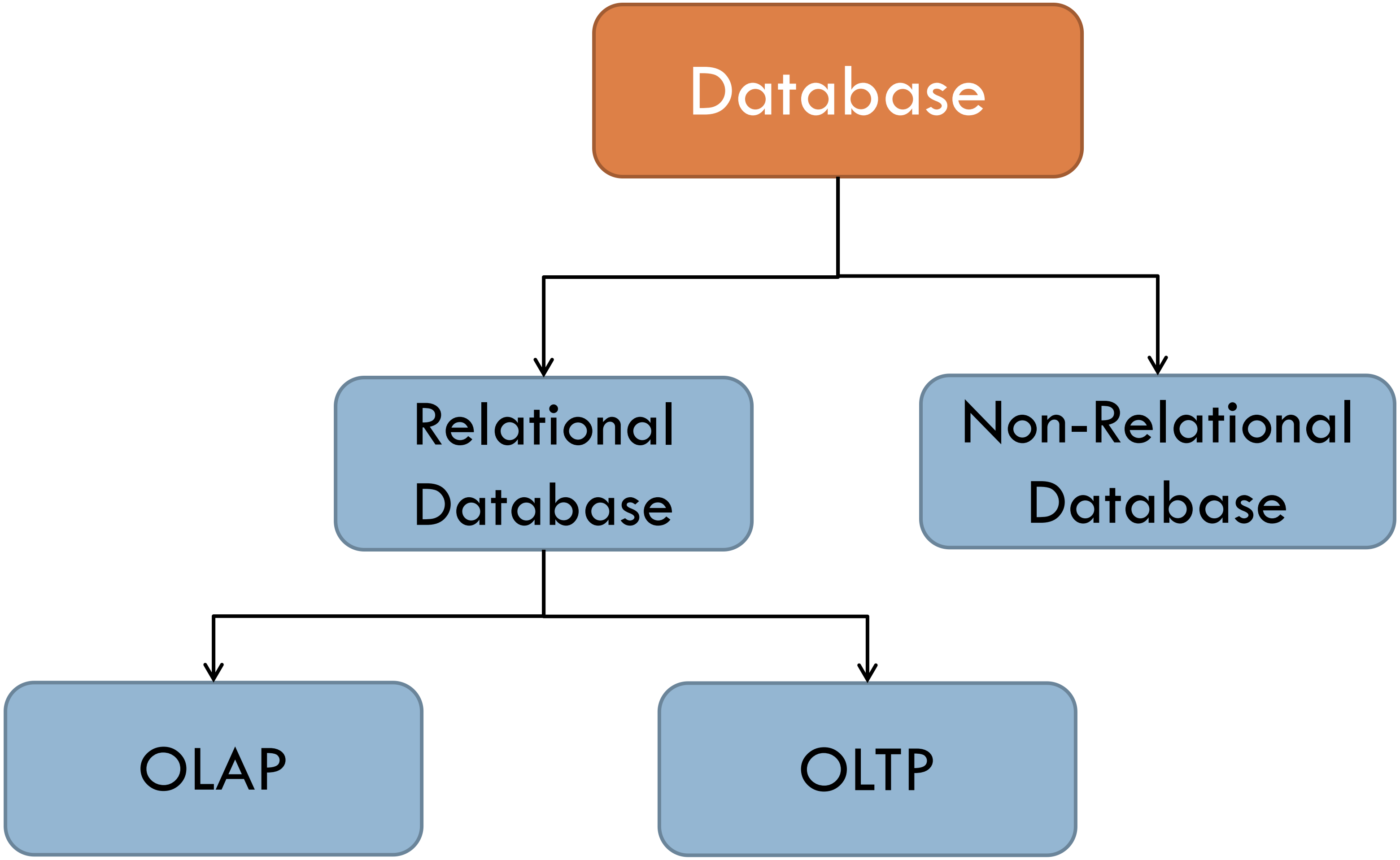
RDBMS and Hadoop

RDBMS	Hadoop
RDBMS is an data/information management system. In RDBMS tables are used for data/information storage. Each row of the table represents a record and column represents an attribute of data. Organization of data and their manipulation processes are different in RDBMS from other databases. RDBMS ensures ACID (atomicity, consistency, integrity, durability) properties required for designing a database. The purpose of RDBMS is to store, manage, and retrieve data as quickly and reliably as possible.	It is an open-source software framework used for storing data and running applications on a group of commodity hardware. It has large storage capacity and high processing power. It can manage multiple concurrent processes at the same time. It is used in predictive analysis, data mining and machine learning. It can handle both structured and unstructured form of data. It is more flexible in storing, processing, and managing data than traditional RDBMS. Unlike traditional systems, Hadoop enables multiple analytical processes on the same data at the same time. It supports scalability very flexibly.

RDBMS and Hadoop difference

RDBMS	Hadoop
Traditional row-column based databases, basically used for data storage, manipulation and retrieval.	An open-source software used for storing data and running applications or processes concurrently.
It is best suited for OLTP environment.	It is best suited for BIG data.
In this, structured data is mostly processed.	In this, both structured and unstructured data is processed.
Data normalization is required.	Data normalization is not required.
The data schema is static type.	The data schema is dynamic type.
Cost is applicable for licensed software.	Free of cost, as it is an open source software.
High data integrity available.	Low data integrity available than RDBMS.

Database

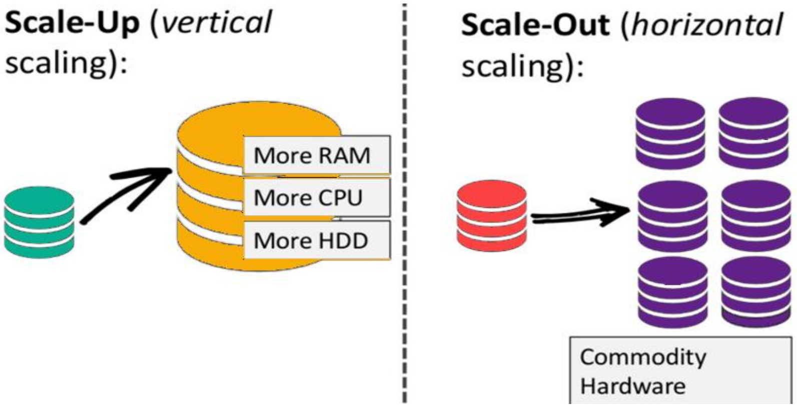

Database
Relational Database
Non-Relational Database
OLAP
OLTP

OLTP vs. OLAP

OLTP (Transaction)	OLAP (Analytical)
Many short transactions	Long transactions, complex queries
Example: - Update account balance - Add book to shopping cart - Enroll in course	Example: - Count the classes with fewer than 10 classes - Report total sales for each dept in each month
Queries touch small amounts of data (few records)	Queries touch large amounts of data
Updates are frequent	Updates are infrequent
Concurrency is biggest performance problem	Individual queries can require lots of resources

Non-relational databases

- The non-relational database does not require a fixed schema, and avoids joins. There are no tables, rows, primary keys or foreign keys.
- It is used for distributed data stores and specifically targeted for big data, for example Google or Facebook which collects terabytes of data every day for their users.
- Traditional relational database uses SQL syntax to store and retrieve data for further insights. Instead, a non-relational database encompasses a wide range of database technologies that can store structured, semi-structured, and unstructured data.
- It adhere to Brewer's CAP theorem.
- The tables are stored as ASCII files and usually, each field is separated by tabs
- The data scale horizontally.


Scale-Up (*vertical* scaling):
More RAM
More CPU
More HDD
Scale-Out (*horizontal* scaling):
Commodity Hardware

NoSQL

NoSQL

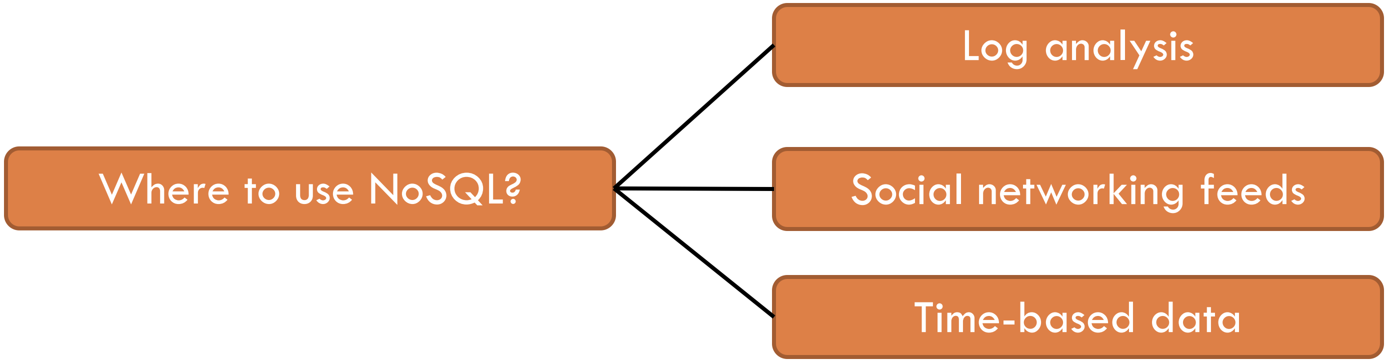
Not Only SQL

Non-relational database

Why and Uses of NoSQL

Why: In today's time data is becoming easier to access and capture through third parties such as Facebook, Google+ and others. Personal user information, social graphs, geo location data, user-generated content and machine logging data are just a few examples where the data has been increasing exponentially. To avail the above service properly, it is required to process huge amount of data which SQL databases were never designed. The evolution of NoSql databases is to handle these huge data properly.

Uses:



Where to use NoSQL?

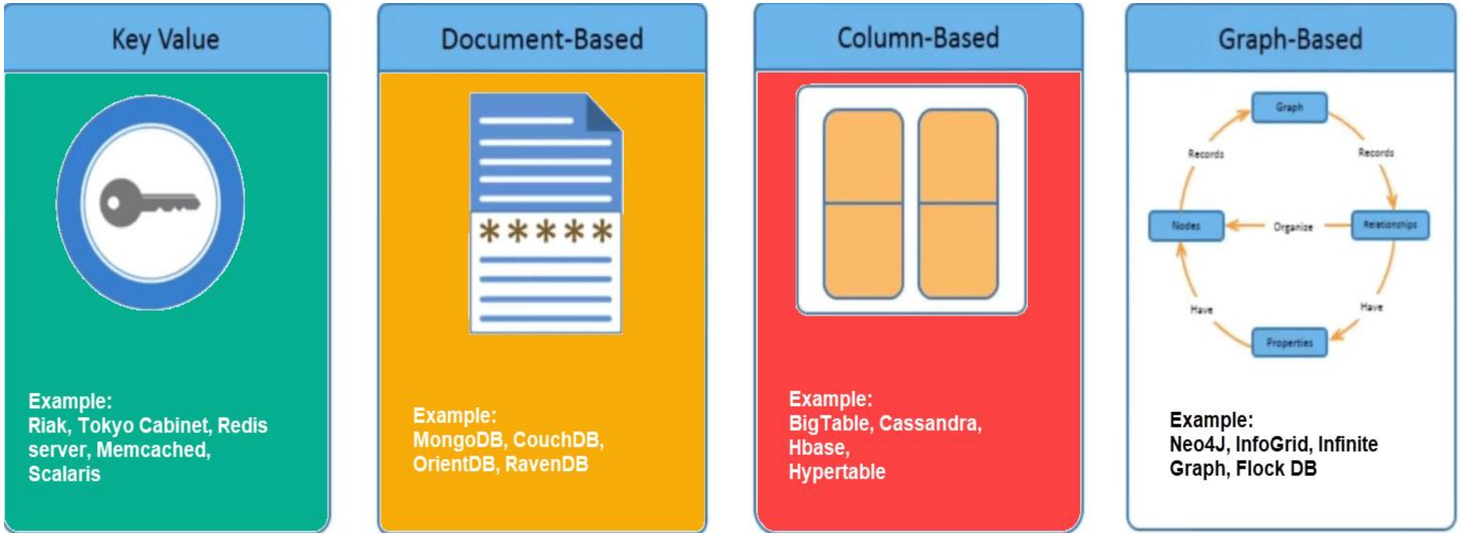
Log analysis

Social networking feeds

Time-based data

Types of NoSQL Database

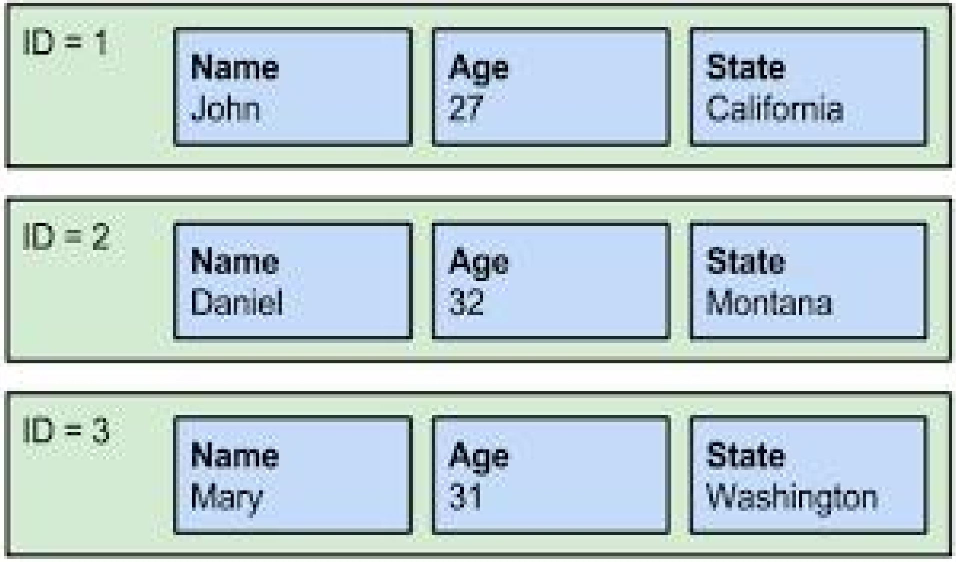
There are mainly four categories of NoSQL databases. Each of these categories has its unique attributes and limitations.


Key Value
Example:
Riak, Tokyo Cabinet, Redis server, Memcached, Scalaris
Document-Based
*****
Example:
MongoDB, CouchDB, OrientDB, RavenDB
Column-Based
Example:
BigTable, Cassandra, Hbase, Hypertable
Graph-Based
Graph
Records
Records
Nodes
Organize
Relationships
Have
Have
Properties
Example:
Neo4J, InfoGrid, Infinite Graph, Flock DB

Performance	High	High	High	Variable
Scalability	High	High	Moderate	Minimal
Flexibility	High	Variable (high)	High	Variable (low)
Functionality	Variable	Variable	Variable	Graph Theory

Key Value

Data is stored in key/value pairs. It is designed in such a way to handle lots of data and heavy load. Key-value pair storage databases store data as a hash table where each key is unique, and the value can be a JSON, BLOB, string, etc. It is one of the most basic types of NoSQL databases. This kind of NoSQL database is used as a collection, dictionaries, associative arrays, etc. Key value stores help the developer to store schema-less data. They work best for shopping cart contents. Redis, Dynamo, Riak are some examples of key-value store.



ID = 1

Name John | Age 27 | State California

ID = 2

Name Daniel | Age 32 | State Montana

ID = 3

Name Mary | Age 31 | State Washington

SQL

ID	Name	Age	State
1	John	27	California

NoSQL – Key Value

Key (i.e. ID)	Values
1	Name: John Age:27 State: California

Document-Based

Document-Oriented NoSQL DB stores and retrieves data as a key value pair but the value part is stored as a document. The document is stored in JSON or XML formats. The document type is mostly used for CMS (Content Management Systems), blogging platforms, real-time analytics & e-commerce applications. It should not use for complex transactions which require multiple operations or queries against varying aggregate structures.

SQL

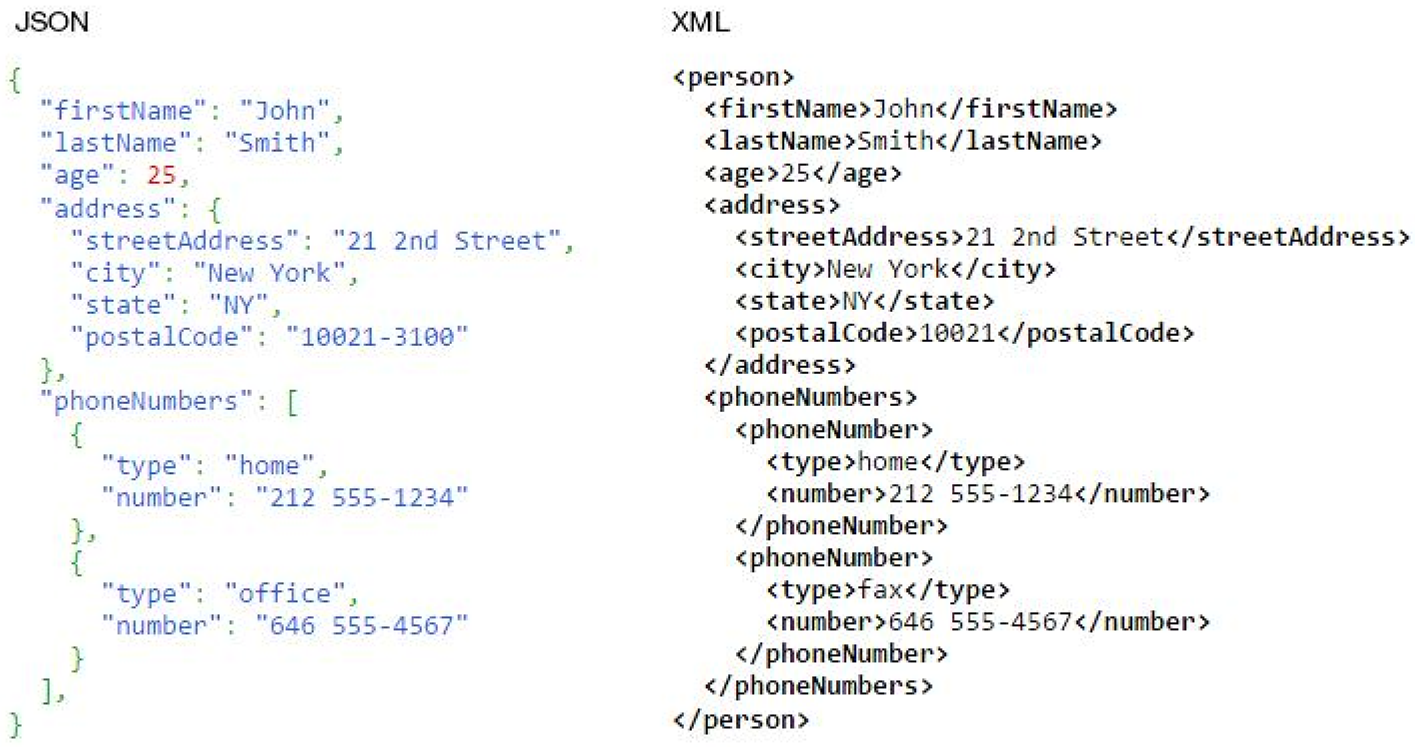
ID	Name	Age	State
1	John	27	California

NoSQL – Document-Based

Key (ID)	Value (JSON)

1	{
	"Name": John
	"Age":27
	"State": California
	}

JSON vs. XML format



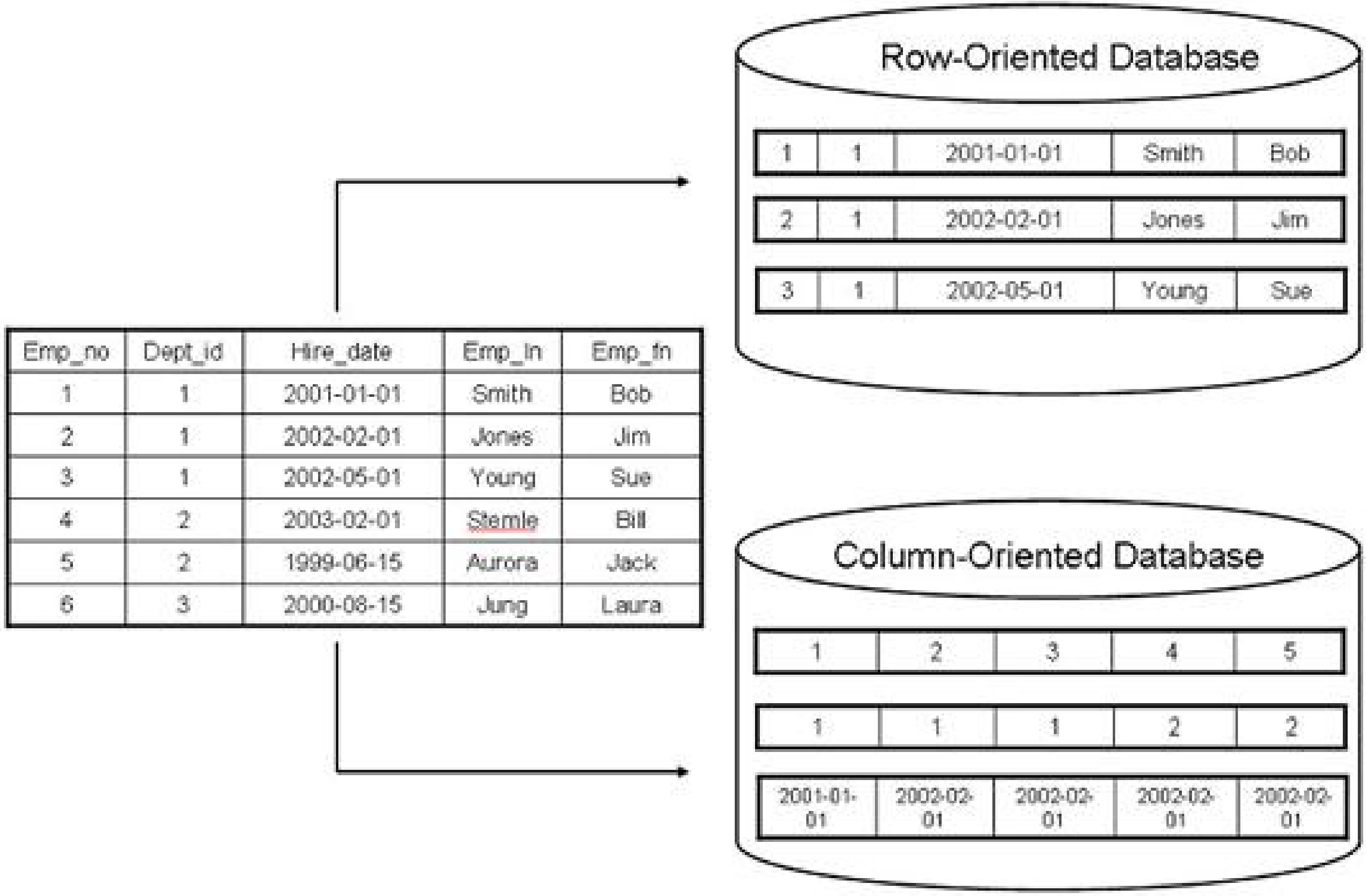
JSON

```json
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 25,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
}
```

XML

```xml
<person>
  <firstName>John</firstName>
  <lastName>Smith</lastName>
  <age>25</age>
  <address>
    <streetAddress>21 2nd Street</streetAddress>
    <city>New York</city>
    <state>NY</state>
    <postalCode>10021</postalCode>
  </address>
  <phoneNumbers>
    <phoneNumber>
      <type>home</type>
      <number>212 555-1234</number>
    </phoneNumber>
    <phoneNumber>
      <type>fax</type>
      <number>646 555-4567</number>
    </phoneNumber>
  </phoneNumbers>
</person>
```

Column-Oriented vs. Row-Oriented Database



Emp_no	Dept_id	Hire_date	Emp_ln	Emp_fn
1	1	2001-01-01	Smith	Bob
2	1	2002-02-01	Jones	Jim
3	1	2002-05-01	Young	Sue
4	2	2003-02-01	Sternle	Bill
5	2	1999-06-15	Aurora	Jack
6	3	2000-08-15	Jung	Laura

Row-Oriented Database

1	1	2001-01-01	Smith	Bob
2	1	2002-02-01	Jones	Jim
3	1	2002-05-01	Young	Sue

Column-Oriented Database

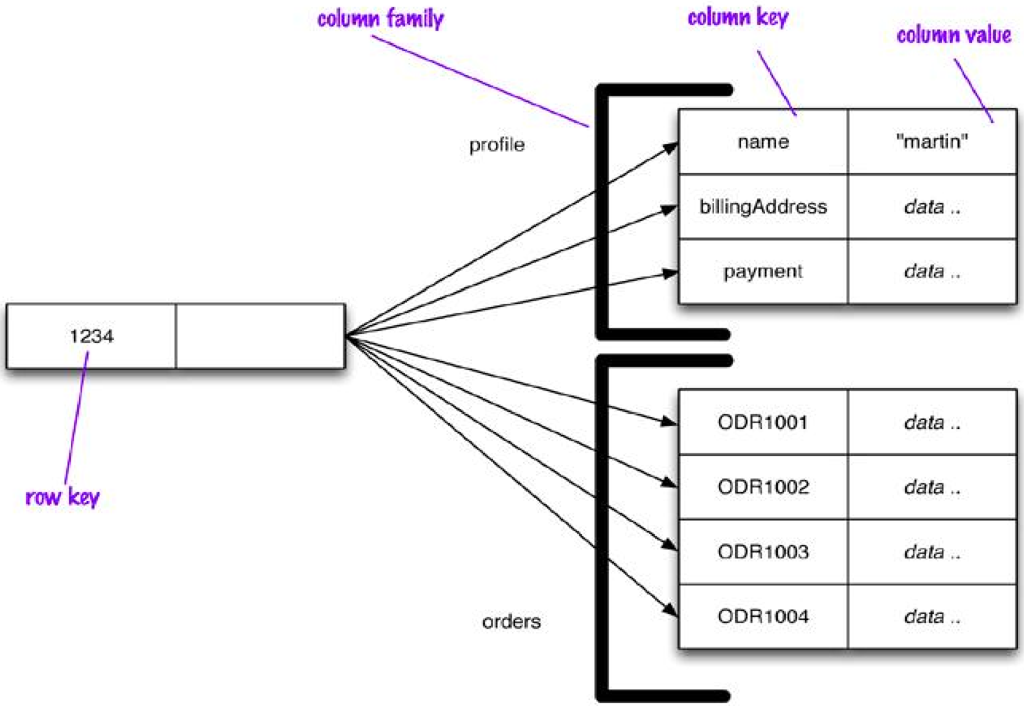
1	2	3	4	5
1	1	1	2	2
2001-01-01	2002-02-01	2002-02-01	2002-02-01	2002-02-01

Column-Oriented vs. Row-Oriented Database cont'd

Row Oriented	Column Oriented
Data is stored and retrieved one row at a time and hence could read unnecessary data if some of the data in a row are required.	In this type of data stores, data are stored and retrieve in columns and hence it can only able to read only the relevant data if required.
Records in Row Oriented Data stores are easy to read and write.	In this type of data stores, read and write operations are slower as compared to row-oriented.
Row-oriented data stores are best suited for online transaction system.	Column-oriented stores are best suited for online analytical processing.
These are not efficient in performing operations applicable to the entire datasets and hence aggregation in row-oriented is an expensive job or operations.	These are efficient in performing operations applicable to the entire dataset and hence enables aggregation over many rows and columns.

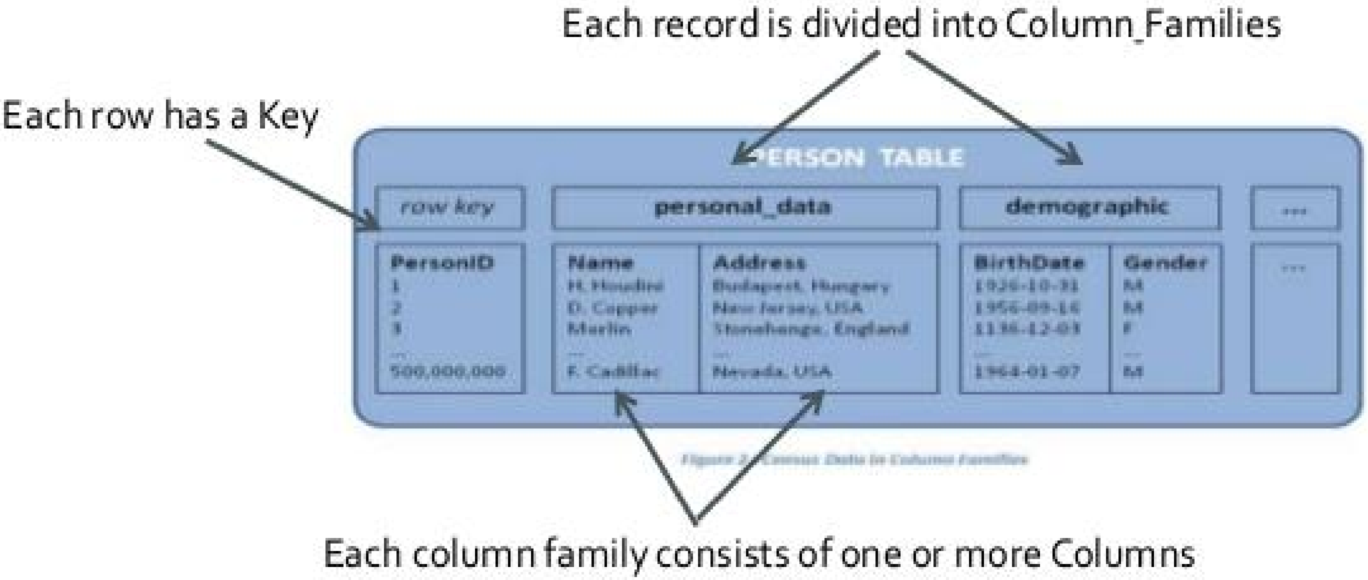
Column-Based Database

Column-oriented databases work on column family and based on BigTable paper by Google. Every column is treated separately. Values of single column databases are stored contiguously. They deliver high performance on aggregation queries like SUM, COUNT, AVG, MIN etc. as the data is readily available in a column. Such NoSQL databases are widely used to manage data warehouses, business intelligence, CRM, Library card catalogs etc.



Column-Based cont'd

Column Families



Each record is divided into Column_Families

Each row has a Key

PERSON TABLE

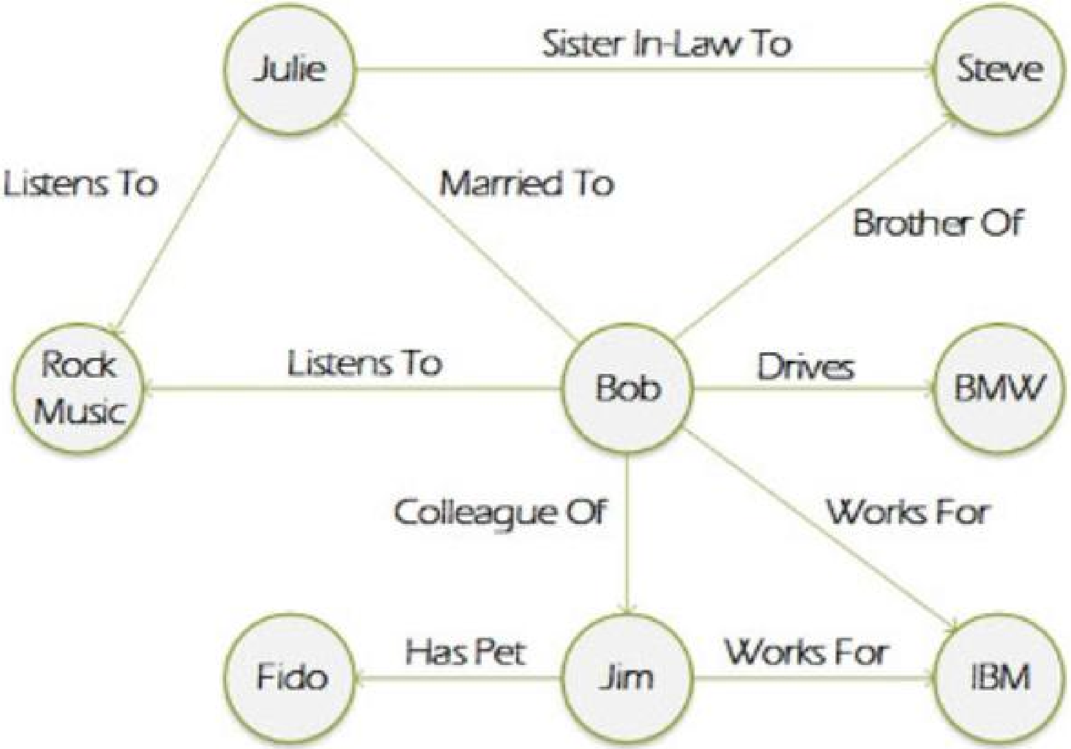
row key	personal_data		demographic		...
PersonID	Name	Address	BirthDate	Gender	...
1	H. Houdini	Budapest, Hungary	1926-10-31	M	
2	D. Copper	New Jersey, USA	1956-09-16	M	
3	Merlin	Stonehenge, England	1136-12-03	F	
...	...	...	...	...	
500,000,000	F. Cadillac	Nevada, USA	1964-01-07	M	

Figure 2: Census Data in Column Families

Each column family consists of one or more Columns

Graph-Based

A graph type database stores entities as well the relations amongst those entities. The entity is stored as a node with the relationship as edges. An edge gives a relationship between nodes. Every node and edge has a unique identifier. Graph base database mostly used for social networks, logistics, spatial data.



Advantages of NoSQL

- Can be used as Primary or Analytic Data Source
- Big Data Capability
- No Single Point of Failure
- Easy Replication
- No Need for Separate Caching Layer
- Provides fast performance and horizontal scalability.
- Can handle structured, semi-structured, and unstructured data with equal effect
- NoSQL databases don't need a dedicated high-performance server
- Support Key Developer Languages and Platforms
- Simple to implement than using RDBMS
- It can serve as the primary data source for online applications.
- Handles big data which manages data velocity, variety, volume, and complexity
- Excels at distributed database and multi-data center operations
- Eliminates the need for a specific caching layer to store data
- Offers a flexible schema design which can easily be altered without downtime or service disruption

Disadvantages of NoSQL

- ❑ No standardization rules
- ❑ Limited query capabilities
- ❑ RDBMS databases and tools are comparatively mature
- ❑ It does not offer any traditional database capabilities, like consistency when multiple transactions are performed simultaneously.
- ❑ When the volume of data increases it is difficult to maintain unique values as keys become difficult
- ❑ Doesn't work as well with relational data
- ❑ The learning curve is stiff for new developers
- ❑ Open source options so not so popular for enterprises.

SQL vs. NoSQL

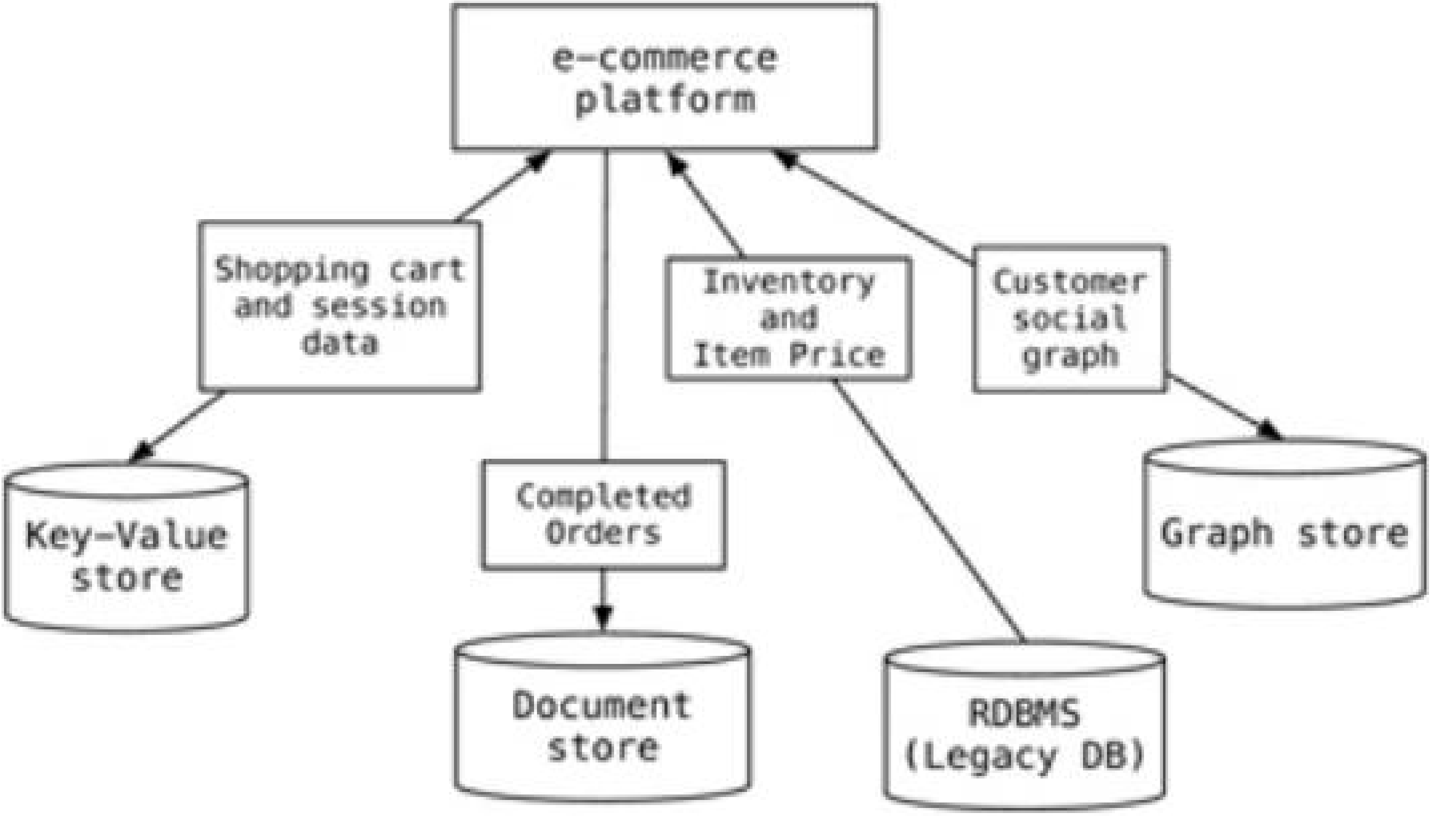
SQL	NoSQL
Relational database	Non-relational, distributed database
Relational model	Model-less approach
Pre-defined schema	Dynamic schema for unstructured data
Table based databases	Document-based or graph-based or wide column store or key-value pairs databases
Vertically scalable (by increasing system resources)	Horizontally scalable (by creating a cluster of commodity machines)
Uses SQL	Uses UnQL (Unstructured Query Language)
Not preferred for large datasets	Largely preferred for large datasets
Not a best fit for hierarchical data	Best fit for hierarchical storage as it follows the key-value pair of storing data similar to JSON
Emphasis on ACID properties	Follows Brewer's CAP theorem

SQL vs. NoSQL cont'd

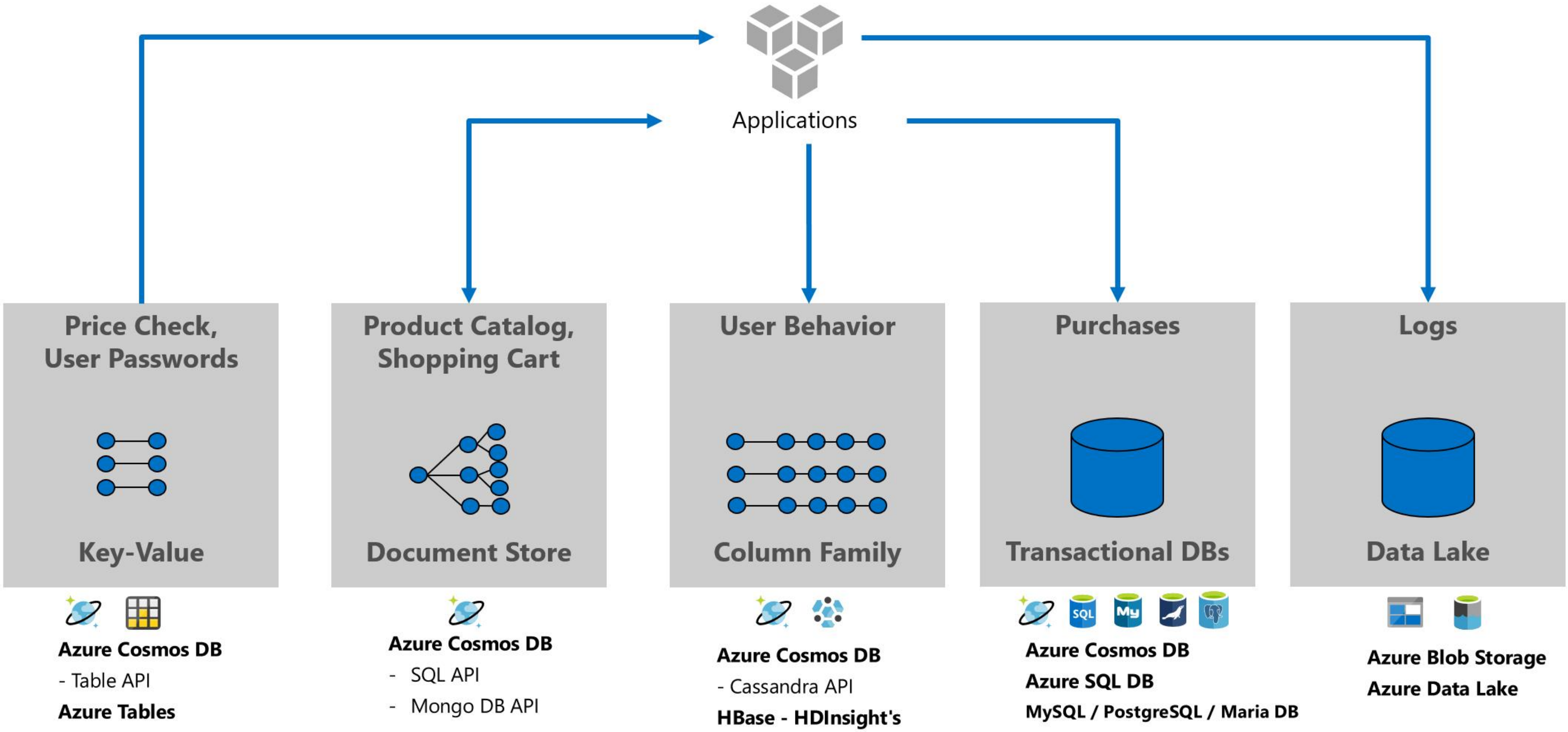
SQL	NoSQL
Excellent support from vendors	Relies heavily on community support
Supports complex querying and data keeping needs	Does not have good support for complex querying
Can be configured for strong consistency	Few support strong consistency (e.g., MongoDB), few others can be configured for eventual consistency (e.g., Cassandra)
Examples: Oracle, DB2, MySQL, MS SQL, PostgreSQL, etc.	Examples: MongoDB, HBase, Cassandra, Redis, Neo4j, CouchDB, Couchbase, Riak, etc.

Polyglot Persistence

- It is a way of storing data, in which it is best to use multiple data storage technologies, chosen based upon the way data is being used by individual applications or components of a single application.
- Polyglot persistence is a hybrid approach enabling usage of multiple databases in a single application/software.
- An e-commerce platform will deal with many types of data (i.e. shopping cart, inventory, completed orders, etc). Instead of trying to store all this data in one database, which would require a lot of data conversion to make the format of the data all the same, store the data in the database best suited for that type of data. So the e-commerce platform might look like this:


e-commerce platform
Shopping cart and session data
Inventory and Item Price
Customer social graph
Completed Orders
Key-Value store
Document store
RDBMS (Legacy DB)
Graph store

Polyglot Persistence on Azure



Applications

Price Check, User Passwords

Key-Value

Azure Cosmos DB
- Table API
Azure Tables

Product Catalog, Shopping Cart

Document Store

Azure Cosmos DB
- SQL API
- Mongo DB API

User Behavior

Column Family

Azure Cosmos DB
- Cassandra API
HBase - HDInsight's

Purchases

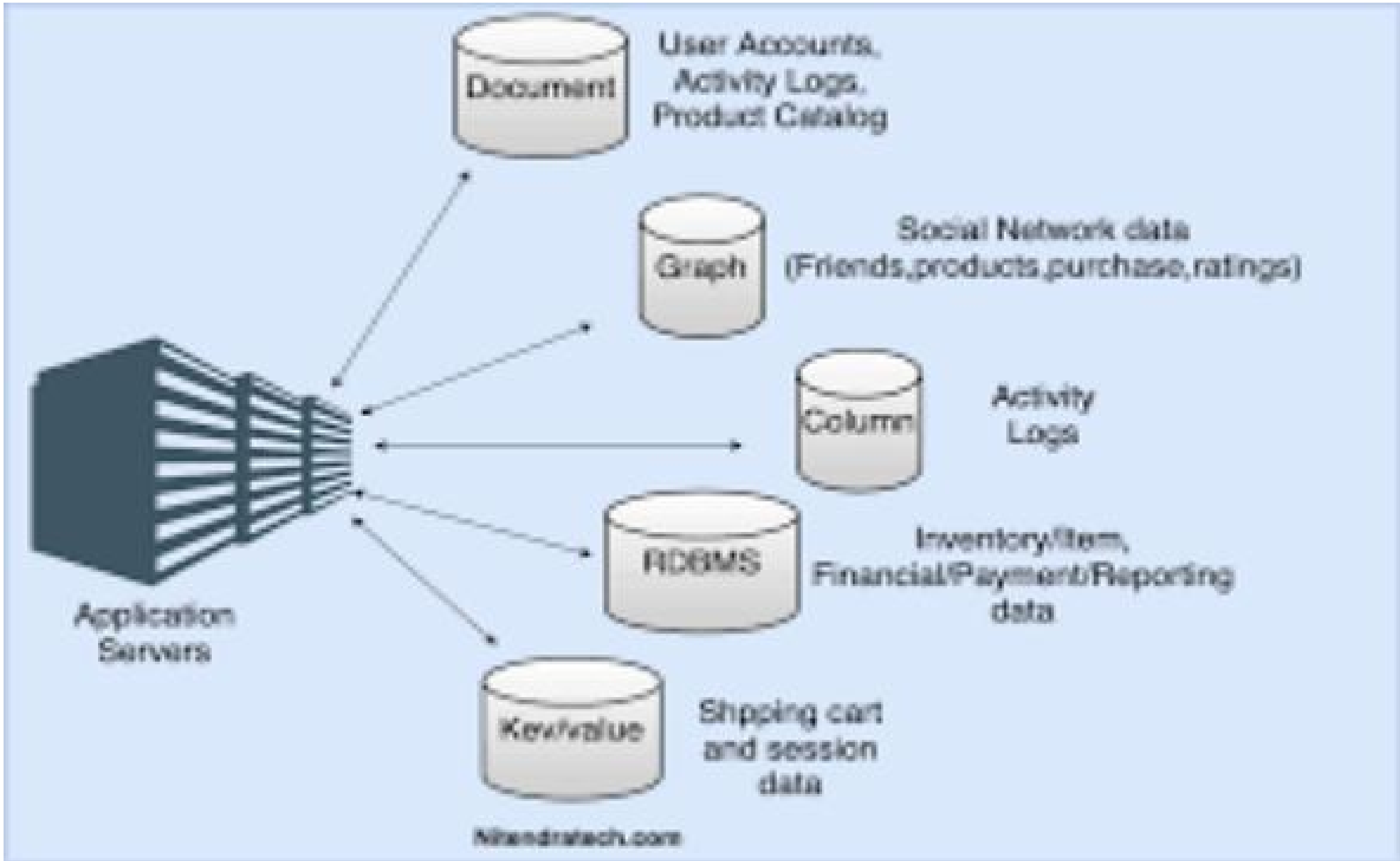
Transactional DBs

Azure Cosmos DB
Azure SQL DB
MySQL / PostgreSQL / Maria DB

Logs

Data Lake

Azure Blob Storage
Azure Data Lake



Nitendratech.com

Polyglot Persistence cont'd…

A guideline on the database type to use based on the functionality of the data:

Functionality	Considerations	Database Type
User sessions	Rapid Access for reads and writes. No need to be durable.	Key-Value
Financial	Needs transactional updates. Tabular structure fits data.	RDBMS
POS (Point of sale)	Depending on size and rate of ingest. Lots of writes, infrequent reads mostly for analytics.	RDBMS (if modest), Key Value or Document (if ingest very high) or Column if analytics is key.
Shopping Cart	High availability across multiple locations. Can merge inconsistent writes.	Document or Key Value
Recommendations	Rapidly traverse links between friends, product purchases, and ratings.	Graph, (Column if simple)

Polyglot Persistence cont…

Functionality	Considerations	Database Type
Product Catalog	Lots of reads, infrequent writes. Products make natural aggregates.	Document
Reporting	SQL interfaces well with reporting tools	RDBMS
Analytics	Large scale analytics on large cluster	Column
User activity logs, CSR logs, Social Media analysis	High volume of writes on multiple nodes	Key Value or Document

Polyglot Persistence Example

Consider a company that sells musical instruments and accessories online (and in a network of shops). At a high-level, there are a number of problems that a company needs to solve to be successful:

- ❑ Attract customers to its stores (both virtual and physical).
- ❑ Present them with relevant products.
- ❑ Once they decide to buy, process the payment and organize shipping.

To solve these problems a company might choose from a number of available technologies that were designed to solve these problems:

- ❑ Store all the products in a document-based database. There are multiple advantages of document databases: flexible schema, high availability, and replication, among others.
- ❑ Model the recommendations using a graph-based database as such databases reflect the factual and abstract relationships between customers and their preferences.
- ❑ Once a product is sold, the transaction normally has a well-structured schema. To store such data relational databases are best suited.

Some strengths of polyglot persistence include:

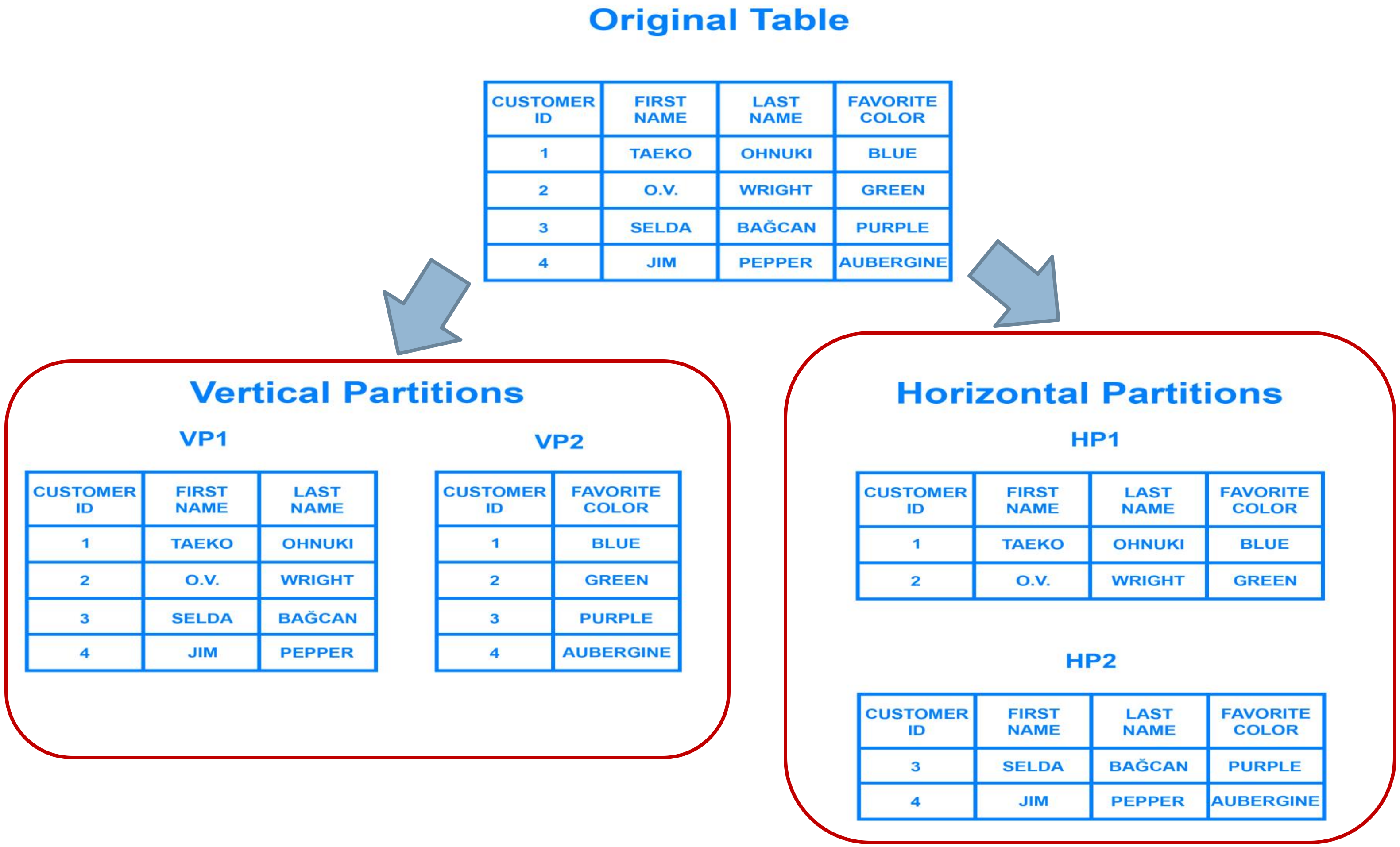
- Simplifies operations
- Eliminates fragmentation
- Creates faster response time
- Improves efficiency
- Scales extremely well
- Allows database engineers/architects to decide where data is stored

- □ Weaknesses of polyglot persistence include:
- □ Must be designed from the ground up based on the specific data architecture of an enterprise -- there are no out of the box solutions
- □ Database engineers/architects must choose where to store data
- □ Database interactions become more complicated
- □ New data stores mean increased complexity and a need for more training
- □ Maintenance and repairs take longer
- □ Database integration increases operational and engineering expenses

Sharding

- Sharding is a database architecture pattern related to horizontal partitioning i.e., the practice of separating one table's rows into multiple different tables, known as partitions. Each partition has the same schema and columns, but also entirely different rows. Likewise, the data held in each is unique and independent of the data held in other partitions.
- In a vertically-partitioned table, entire columns are separated out and put into new, distinct tables. The data held within one vertical partition is independent from the data in all the others, and each holds both distinct rows and columns.
- Sharding involves breaking up data into two or more smaller chunks, called logical shards. The logical shards are then distributed across separate database nodes, referred to as physical shards, which can hold multiple logical shards. Despite this, the data held within all the shards collectively represent an entire logical dataset.
- Database shards exemplify a **shared-nothing architecture**. This means that the shards are autonomous; they don't share any of the same data or computing resources.

Sharding Visual Illustration



Original Table

CUSTOMER ID	FIRST NAME	LAST NAME	FAVORITE COLOR
1	TAEKO	OHNUKI	BLUE
2	O.V.	WRIGHT	GREEN
3	SELDA	BAĞCAN	PURPLE
4	JIM	PEPPER	AUBERGINE

Vertical Partitions

VP1

CUSTOMER ID	FIRST NAME	LAST NAME
1	TAEKO	OHNUKI
2	O.V.	WRIGHT
3	SELDA	BAĞCAN
4	JIM	PEPPER

VP2

CUSTOMER ID	FAVORITE COLOR
1	BLUE
2	GREEN
3	PURPLE
4	AUBERGINE

Horizontal Partitions

HP1

CUSTOMER ID	FIRST NAME	LAST NAME	FAVORITE COLOR
1	TAEKO	OHNUKI	BLUE
2	O.V.	WRIGHT	GREEN

HP2

CUSTOMER ID	FIRST NAME	LAST NAME	FAVORITE COLOR
3	SELDA	BAĞCAN	PURPLE
4	JIM	PEPPER	AUBERGINE

Why Sharding?

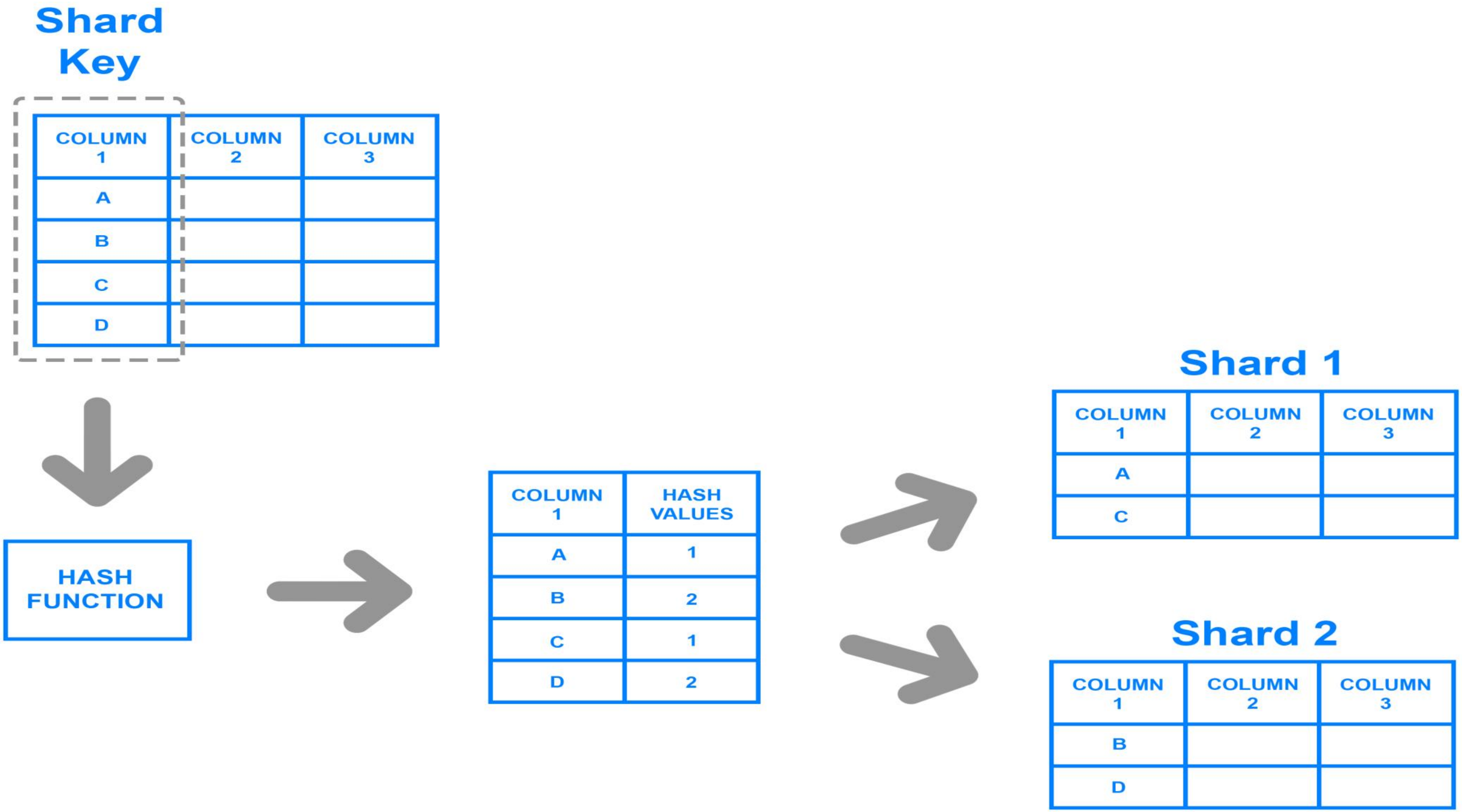
- The main appeal of sharding a database is that it can help to facilitate horizontal scaling, also known as scaling out. Horizontal scaling is the practice of adding more machines to an existing stack in order to **spread out the load and allow for more traffic and faster processing**.
- Any non-distributed database will be limited in terms of storage and compute power, so having the freedom to scale horizontally makes setup **far more flexible**.
- Choose a sharded database architecture is to **speed up query response times.** When a query is submitted on a database that hasn't been sharded, it may have to search every row in the table one is querying before it can find the result set one is looking for. For an application with a large, monolithic database, queries can become prohibitively slow. By sharding one table into multiple, though, queries have to go over fewer rows and their **result sets are returned much more quickly.**

Sharding Architecture

When running queries or distributing incoming data to sharded tables or databases, it's crucial that it goes to the correct shard. Otherwise, it could result in lost data or painfully slow queries. Therefore, sharding architectures to be defined, wherein each of which uses a slightly different process to distribute data across shards. The recommended architectural approaches are:

- Key Based Sharding
- Range Based Sharding
- Directory Based Sharding

Key Based Sharding



Shard Key

COLUMN 1	COLUMN 2	COLUMN 3
A		
B		
C		
D		

HASH FUNCTION

COLUMN 1	HASH VALUES
A	1
B	2
C	1
D	2

Shard 1

COLUMN 1	COLUMN 2	COLUMN 3
A		
C		

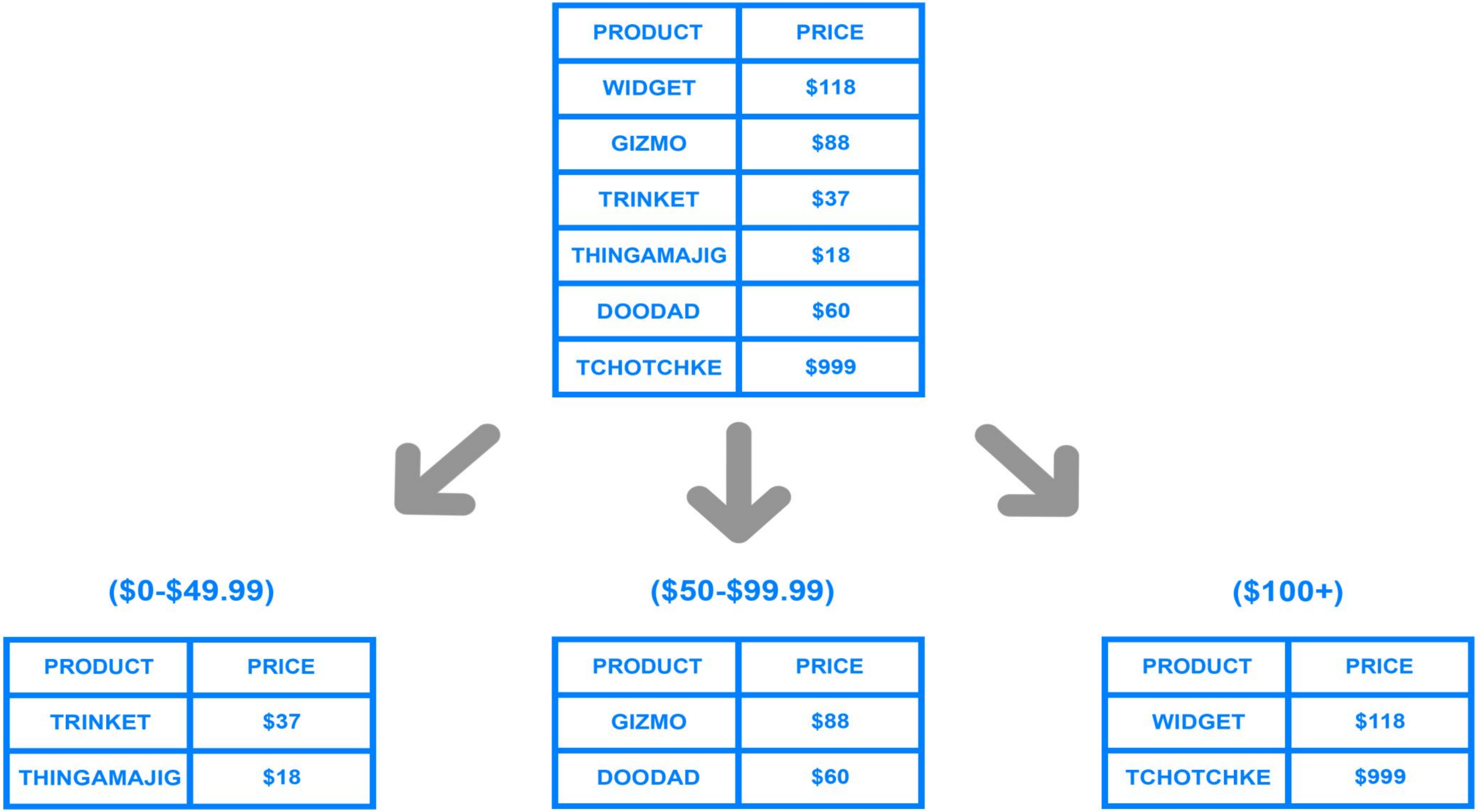
Shard 2

COLUMN 1	COLUMN 2	COLUMN 3
B		
D		

Key Based Sharding cont'd…

- Key based sharding, also known as **hash based sharding**, involves using a value taken from newly written data — such as a customer's ID, a client application's IP address, a ZIP code, etc. and plugging it into a hash function to determine which shard the data should go to. A hash function is a function that takes as input a piece of data (for example, a customer email) and outputs a discrete value, known as a hash value. In the case of sharding, the hash value is a **shard ID** used to determine which shard the incoming data will be stored on.
- To ensure that entries are placed in the correct shards and in a consistent manner, the values entered into the hash function should all come from the **same column**. **This column is known as a shard key**. In simple terms, shard keys are similar to primary keys in that both are columns which are used to establish a unique identifier for individual rows.

Range Based Sharding



PRODUCT	PRICE
WIDGET	$118
GIZMO	$88
TRINKET	$37
THINGAMAJIG	$18
DOODAD	$60
TCHOTCHKE	$999

($0-$49.99)

PRODUCT	PRICE
TRINKET	$37
THINGAMAJIG	$18

($50-$99.99)

PRODUCT	PRICE
GIZMO	$88
DOODAD	$60

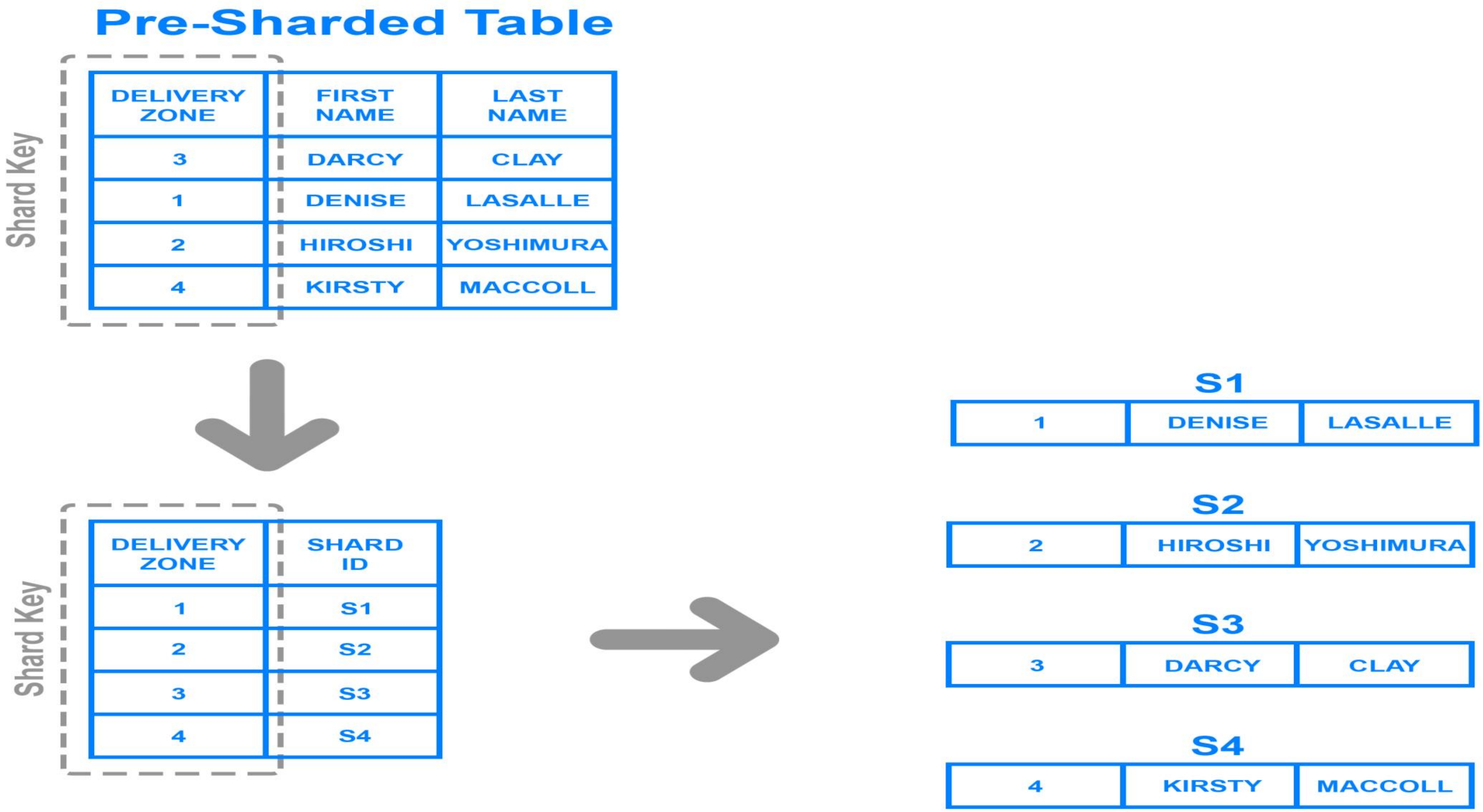
($100+)

PRODUCT	PRICE
WIDGET	$118
TCHOTCHKE	$999

Range Based Sharding cont'd…

- Range based sharding involves sharding data based on ranges of a given value.
- To illustrate, let's assume the existence of a database that stores information about all the products within a retailer's catalog. One could create a few different shards and divvy up each products' information based on which price range they fall into.
- The main benefit of range based sharding is that it's relatively simple to implement. Every shard holds a different set of data but they all have an identical schema as one another, as well as the original database. The application code reads which range the data falls into and writes it to the corresponding shard.

Directory Based Sharding



Pre-Sharded Table

DELIVERY ZONE	FIRST NAME	LAST NAME
3	DARCY	CLAY
1	DENISE	LASALLE
2	HIROSHI	YOSHIMURA
4	KIRSTY	MACCOLL

DELIVERY ZONE	SHARD ID
1	S1
2	S2
3	S3
4	S4

S1

1	DENISE	LASALLE

S2

2	HIROSHI	YOSHIMURA

S3

3	DARCY	CLAY

S4

4	KIRSTY	MACCOLL

Directory Based Sharding cont'd…

- To design a directory based sharding, one must create and maintain a lookup table that uses a shard key to keep track of which shard holds which data. A lookup table is a table that holds a static set of information about where specific data can be found.
- The Delivery Zone column is defined as a shard key. Data from the shard key is written to the lookup table along with whatever shard each respective row should be written to. This is similar to range based sharding, but instead of determining which range the shard key's data falls into, each key is tied to its own specific shard.
- The main appeal of directory based sharding is its flexibility.

THANK YOU!