BIG DATA

TOO BIG TO IGNORE

SÜMEYYE KAYNAK

OUTLINE

Big data storage technology

BIG DATA STORAGE TECHNOLOGY

- On-Disk Storage Devices
- In-Memory Storage Devices

ON-DISK STORAGE DEVICES

- Low cost hard-disk drivers
- Long-term storage
- Implemented via a distributed file system or a database.

DISTRIBUTED FILE SYSTEMS

- Distributed File Systems (DDS) today provide shared use of disks and storage resources.
- It can store large volumes of data that are not related in nature, such as semi-structured and unstructured data.
- It is not suitable in large number of small files.

DISTRIBUTED FILE SYSTEMS

- DFS must be able to give access control and storage management controls to the client system in a centralized way.
- Transparency is also of the core processes in DFS.
- DFS is suitable to storing large datasets of raw data

BENEFITS OF A DISTRIBUTED FILE SYSTEM?

A distributed file system for storage provides:

- A DFS makes it possible to restrict access to the file system, depending on access lists or capabilities on both the servers and the clients, depending on how the protocol is designed.
- Improved file availability, access time and network efficiency.
- Enhanced scalability and interoperability.
- Data access transparency and location independence.
- A unified view of shared folders and data resources.
- More efficient load-balancing.

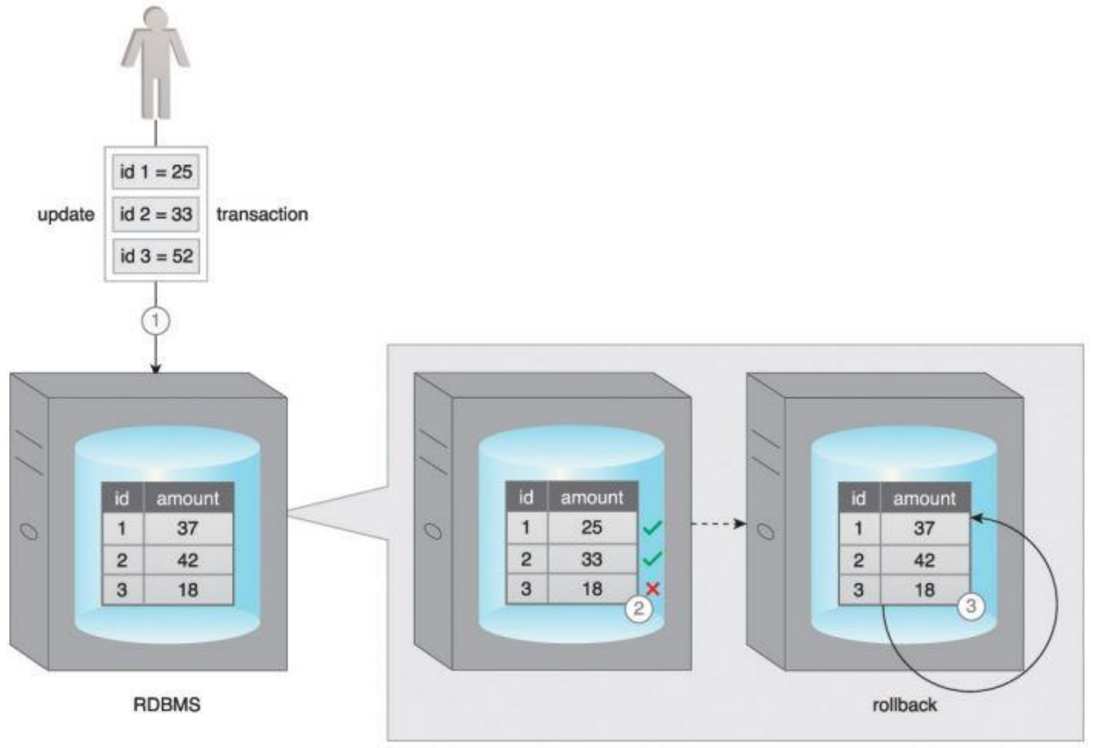
RDBMS DATABASES

- Relational DataBase Management Systems(RDBMSs)
- A relational database is a type of database that stores and provides access to data points that are related to one another.
- Relational databases are based on the relational model.
- In a relational database, each row in the table is a record with a unique ID called the key.
- The columns of the table hold attributes of the data, and each record usually has a value for each attribute, making it easy to establish the relationships among data points.

ACID PROPERTIES AND RDBMS DATABASES

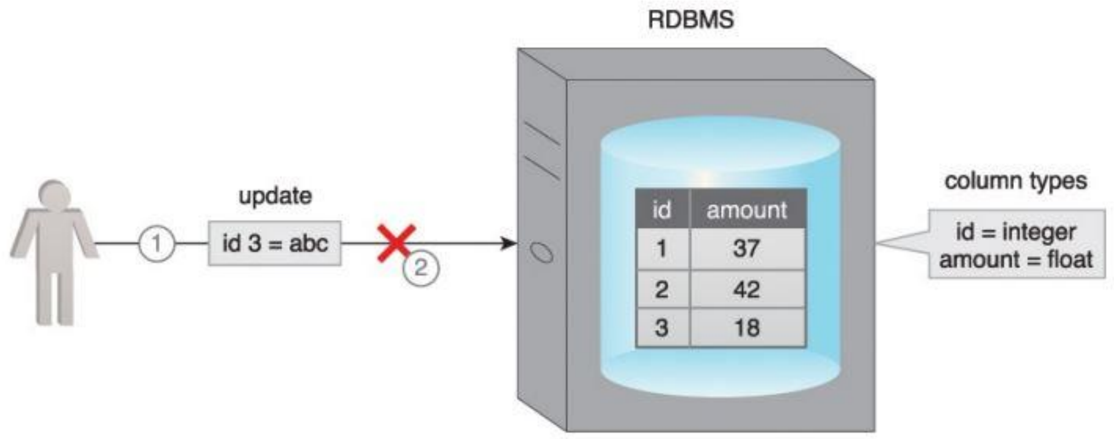
- Atomicity: ensures that all operations will always succeed or fail completely. In other words, there are no partial transactions.
- Consistency: Defines the rules for maintaining data points in a correct state after a transaction.
- Isolation: Keeps the effect of a transaction invisible to others until it is committed, to avoid confusion.
- Durability: ensures that data changes become permanent once the transaction is committed.

ATOMICITY



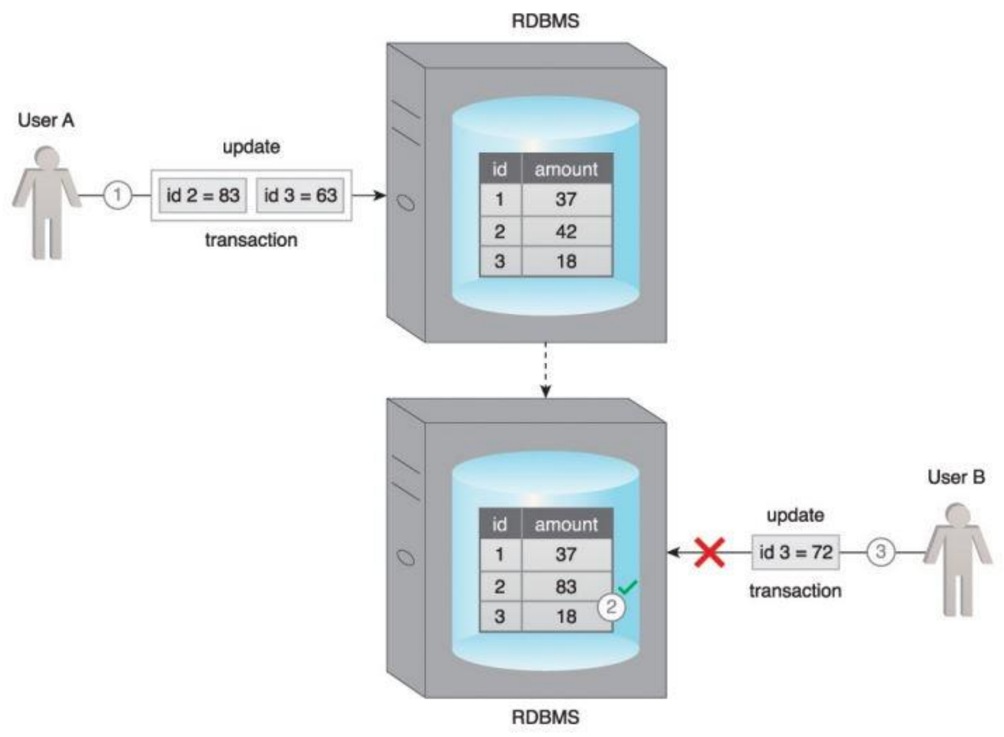
- A user attempts to update three records as a part of a transaction.
- Two records are successfully updated before the occurrence of an error.
- As a result, the database roll backs any partial effects of the transaction and puts the system back to its prior state.

CONSISTENCY

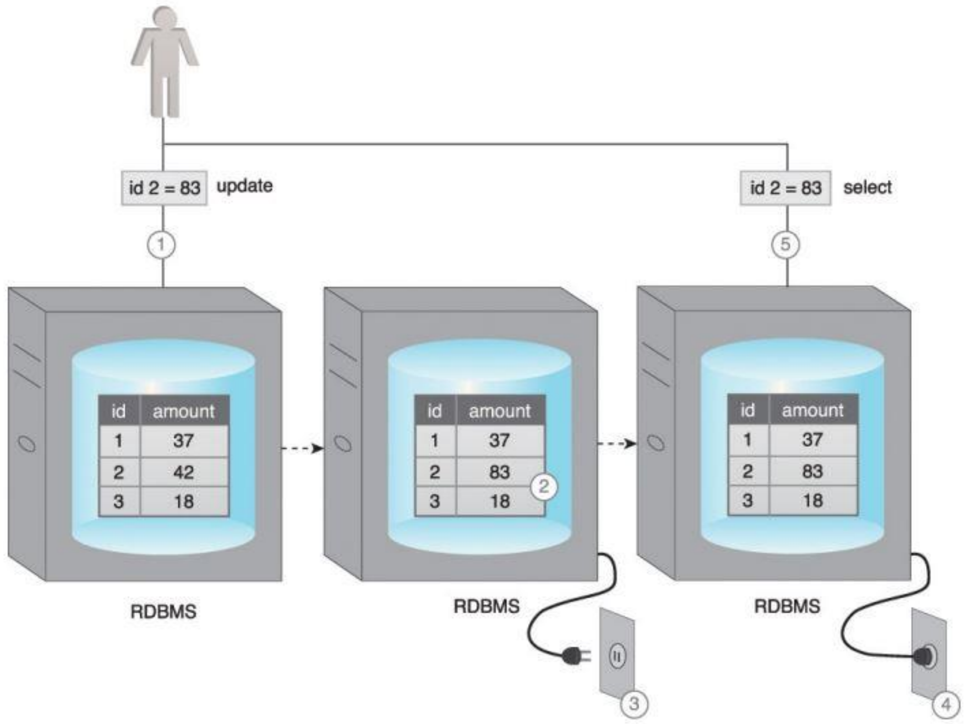


- A user attempts to update the amount column of the table that is of type float with a varchar value.
- The database applies its validation check and rejects this update because the value violates the constraint checks for the amount column.

ISOLATION



- User A attempts to update two records as part of a transaction.
- The database successfully updates the first record.
- However, before it can update the second record, User B attempts to update the same record. The database does not permit User B’s update until User A’s update succeeds or fails in full. This occurs because the record with id3 is locked by the database until the transaction is complete

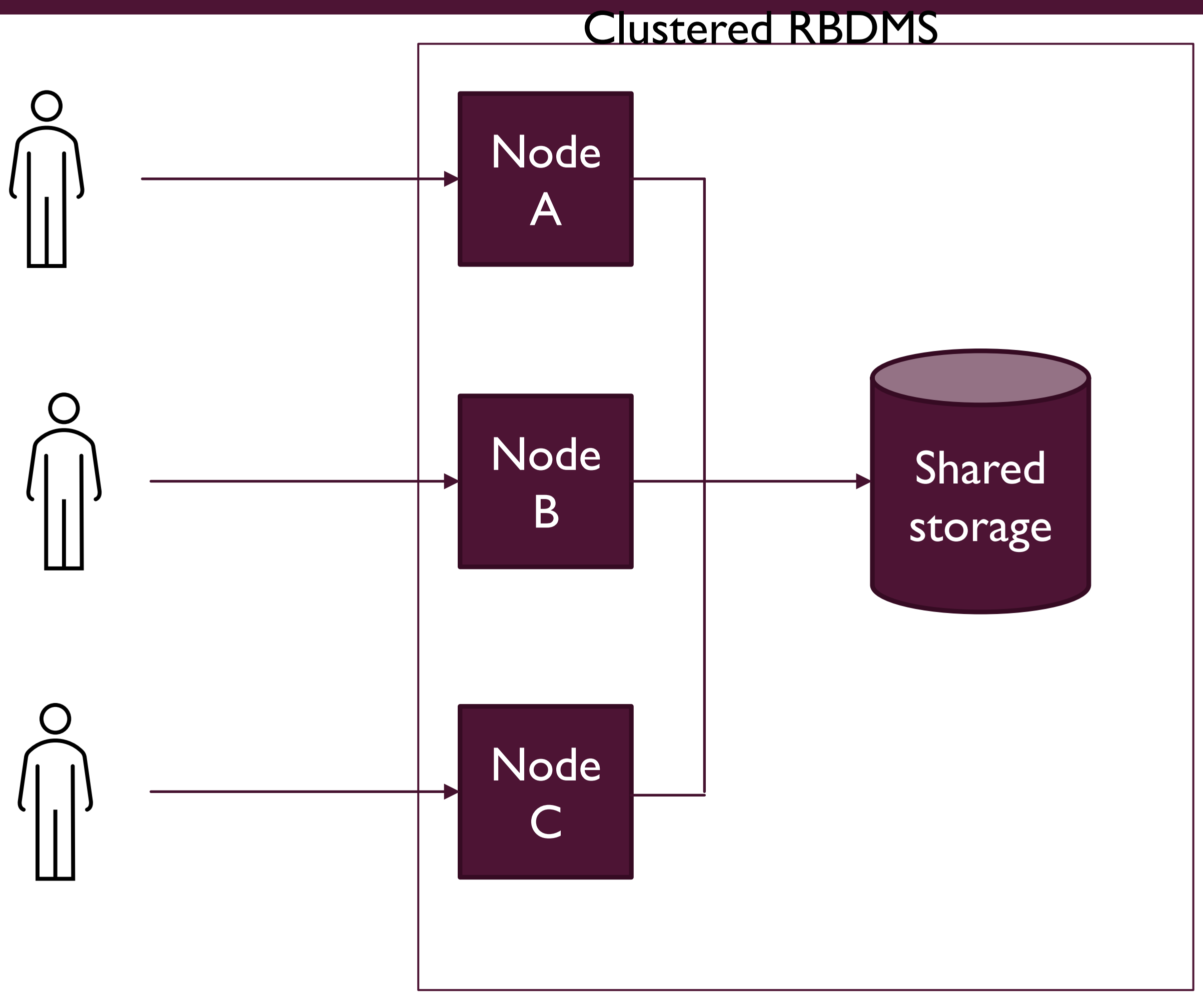


- A user updates a record as part of a transaction.
- The database successfully updates the record.
- Right after this update, a power failure occurs. The database maintains its state while there is no power.
- The power is resumed.
- The database serves the record as per last update when requested by the user

RDBMS DATABASES

- In RDBMS Databases are designed to run on a single server in order to maintain the integrity of the table mappings and avoid the problems of distributed computing.
- So relational databases are not designed for scale.

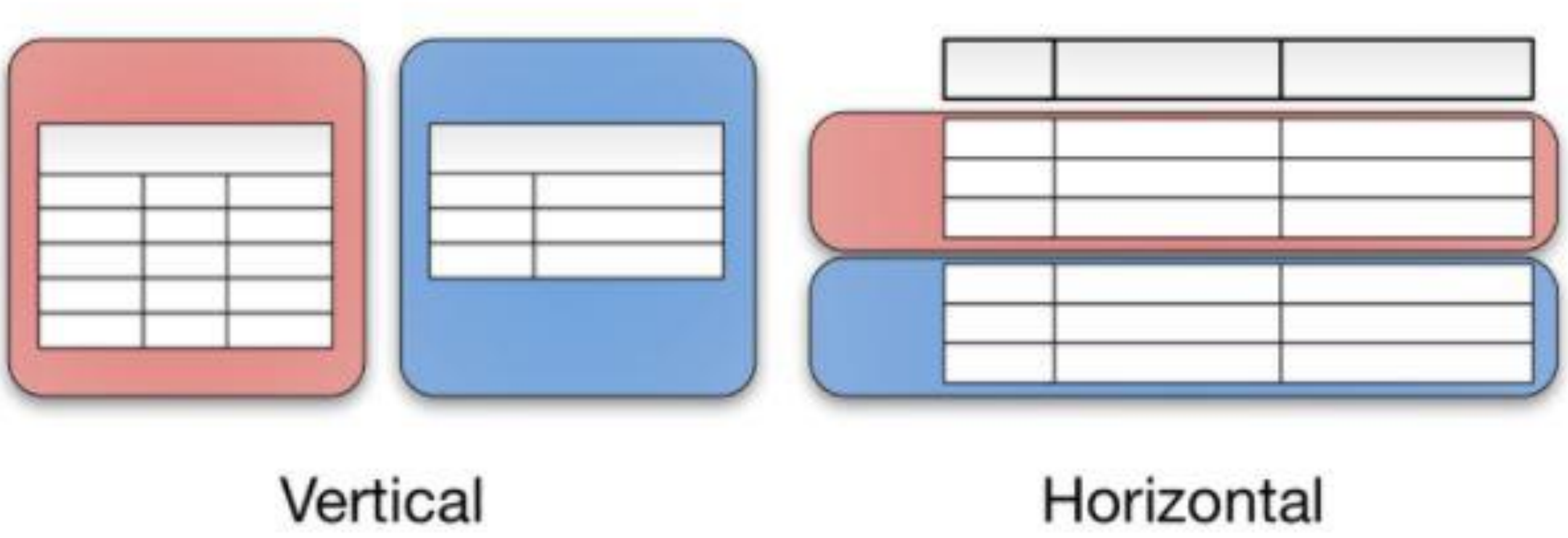
CLUSTERED RDBMS



SHARD IN RDBMS DATABASES

- Sharding is the process of breaking up large tables into smaller chunks called shards that are spread across multiple servers.
- A shard is essentially a horizontal data partition that contains a subset of the total data set and hence is responsible for serving a portion of the overall workload.
- The idea is to distribute data that can’t fit on a single node onto a cluster of database nodes.
- Sharding is also referred to as horizontal partitioning.

SHARD IN RDBMS DATABASES



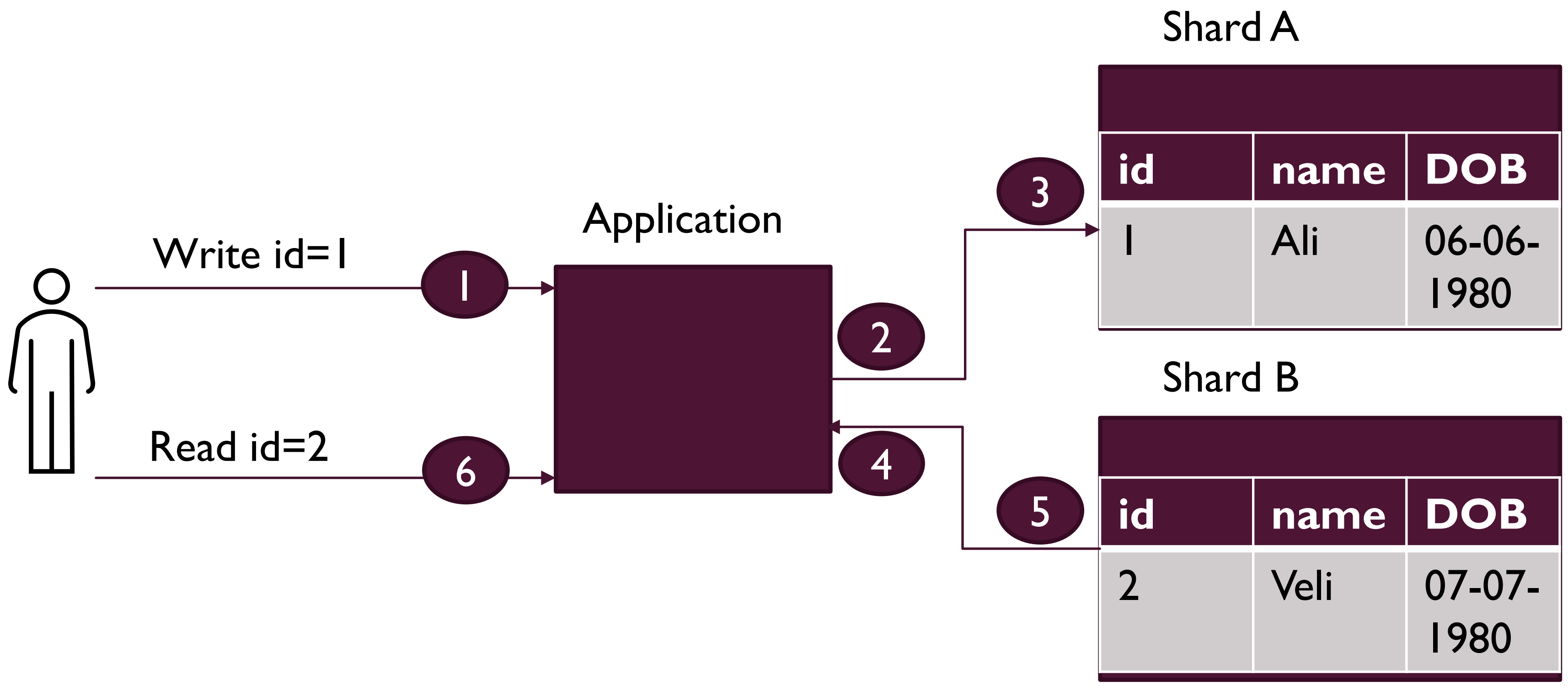
WHY SHARD A DATABASE?

- RDBMS hit bottlenecks as they grow. With limited CPU, storage capacity, and memory, query throughput and response times are bound to suffer.
- Horizontally partitioning a table means more compute capacity to serve incoming queries, and therefore you end up with faster query response times.
- By continuously balancing the load and data set over additional nodes, sharding also enables usage of additional capacity.
- A network of smaller, cheaper servers may be more cost effective in the long term than maintaining one big server.

MANUAL SHARDING

- Databases like Oracle, PostgreSQL, MySQL, and even newer distributed SQL databases like Amazon Aurora do not support automatic sharding. This means manual sharding at the application layer has to be performed if you want to continue to use these databases.

MANUAL SHARDING



Shard A

id	name	DOB
1	Ali	06-06-1980

Shard B

id	name	DOB
2	Veli	07-07-1980

NOSQL DATABASES-CHARACTERISTICS

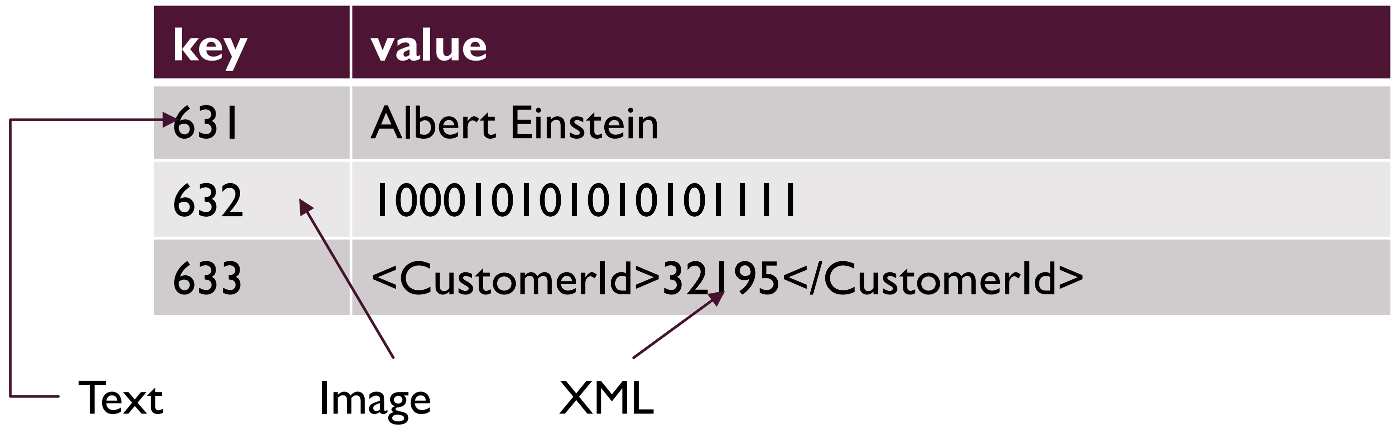
- Data can exist in its raw form.
- They can process both unstructured and semi-structured data.
- They are cost effective
- Complex-free working
- Independent of Schema
- Better Scalability
- Flexible to accommodate
- Durable

TYPES OF NOSQL DATABASES

- Key-Value Database
- Document Database
- Column-oriented Database
- Graph Database

KEY-VALUE DATABASE

- A data point is categorized as a key to which a value (another data point) is allotted.
- The data is fetched by a unique key.
- There is no schema, and the value of the data can be just about anything. Values can be numbers, strings, counters, JSON, XML, HTML, binaries, images, videos, and more.



key	value
631	Albert Einstein
632	10001010101010101111
633	<CustomerId>32195</CustomerId>

Text
Image
XML

KEY-VALUE DATABASE

- It is the most flexible NoSQL model because there are no restrictions on what is stored in the value field.
- Key-value stores have no query language, but they do provide a way to add and remove key-value pairs.
- Values cannot be queried or searched upon. Only the key can be queried.

KEY-VALUE DATABASE

- Within a key-value database, only three functions can be executed (put, get, delete).
- Put: Adds a new key-value pair and updates the value if the key is already present.
  Get: Returns the value for any given key.
  Delete: Removes a key-value pair.

KEY-VALUE DATABASE

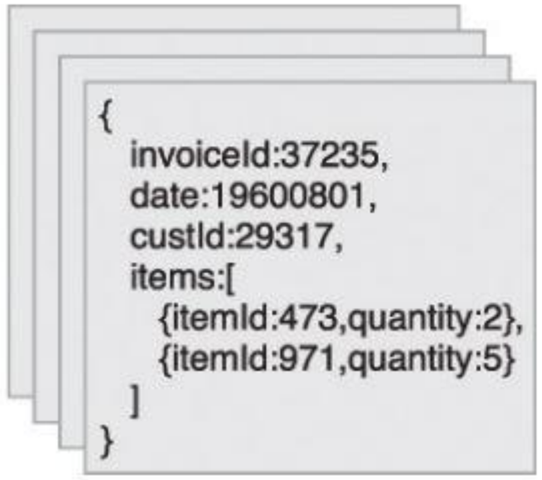
- Value is standalone entity that is not dependent on other values. Rapid retrieval of values can occur regardless of the number of records / items within the database.
- Scalability is achieved through the sharding (a.k.a. partitioning) of data across nodes.
- Values have a comparatively simple structure or are binary.

DOCUMENT DATABASE

- Data and metadata are stored hierarchically in JSON-based documents inside the database.
- Document databases maintain sets of key-value pairs within a document. However, unlike key-value storage devices, the stored value is a document that can be queried by the database.
- Like key-value storage devices, most document storage devices provide collections or buckets (like tables) into which key-value pairs can be organized.

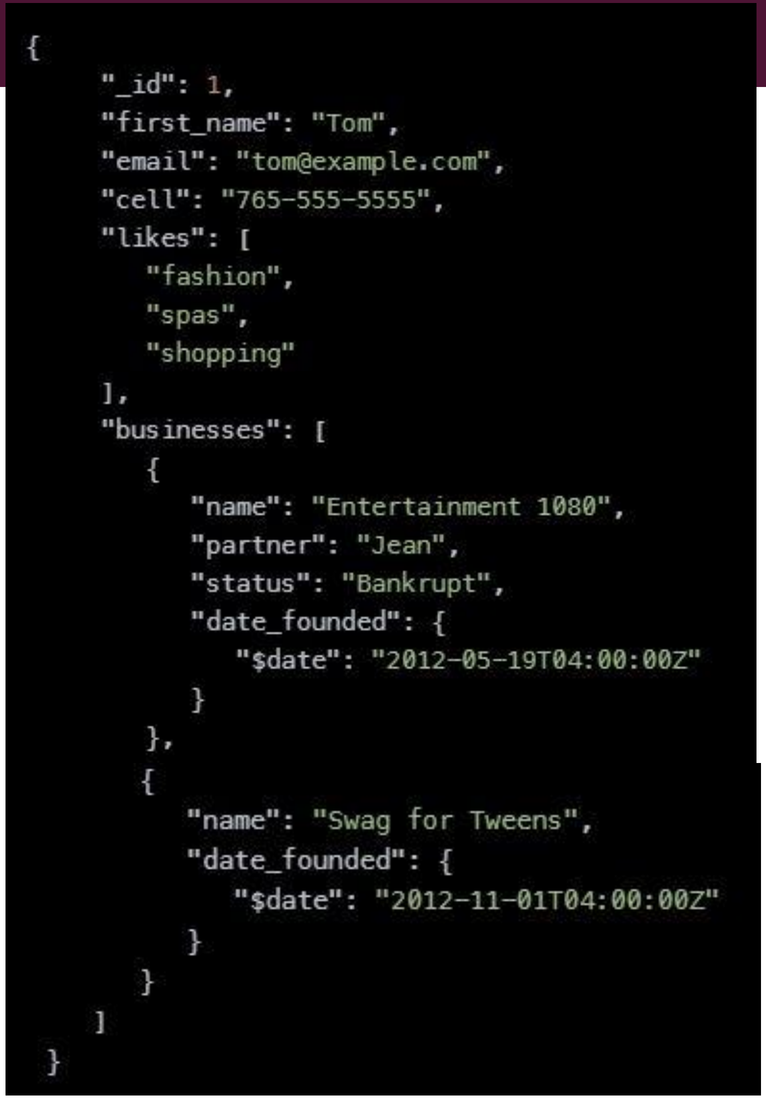
DOCUMENT DATABASE

- The values can be a variety of types and structures, including strings, numbers, dates, arrays, or objects. Documents can be stored in formats like JSON, BSON, and XML.



```
{
  invoiceId:37235,
  date:19600801,
  custId:29317,
  items:[
    {itemId:473,quantity:2},
    {itemId:971,quantity:5}
  ]
}
```

DOCUMENT DATABASE



```
{
    "_id": 1,
    "first_name": "Tom",
    "email": "tom@example.com",
    "cell": "765-555-5555",
    "likes": [
       "fashion",
       "spas",
       "shopping"
    ],
    "businesses": [
       {
          "name": "Entertainment 1080",
          "partner": "Jean",
          "status": "Bankrupt",
          "date_founded": {
             "$date": "2012-05-19T04:00:00Z"
          }
       },
       {
          "name": "Swag for Tweens",
          "date_founded": {
             "$date": "2012-11-01T04:00:00Z"
          }
       }
    ]
 }
```

Collection:

- A collection is a group of documents. Collections typically store documents that have similar contents.
- Not all documents in a collection are required to have the same fields, because document databases have a flexible schema.

DOCUMENT DATABASE-CRUD OPERATION

- **Create**: Documents can be created in the database. Each document has a unique identifier.
- **Read**: Documents can be read from the database. The API or query language allows developers to query for documents using their unique identifiers or field values. Indexes can be added to the database in order to increase read performance.
- **Update**: Existing documents can be updated — either in whole or in part.
- **Delete**: Documents can be deleted from the database.

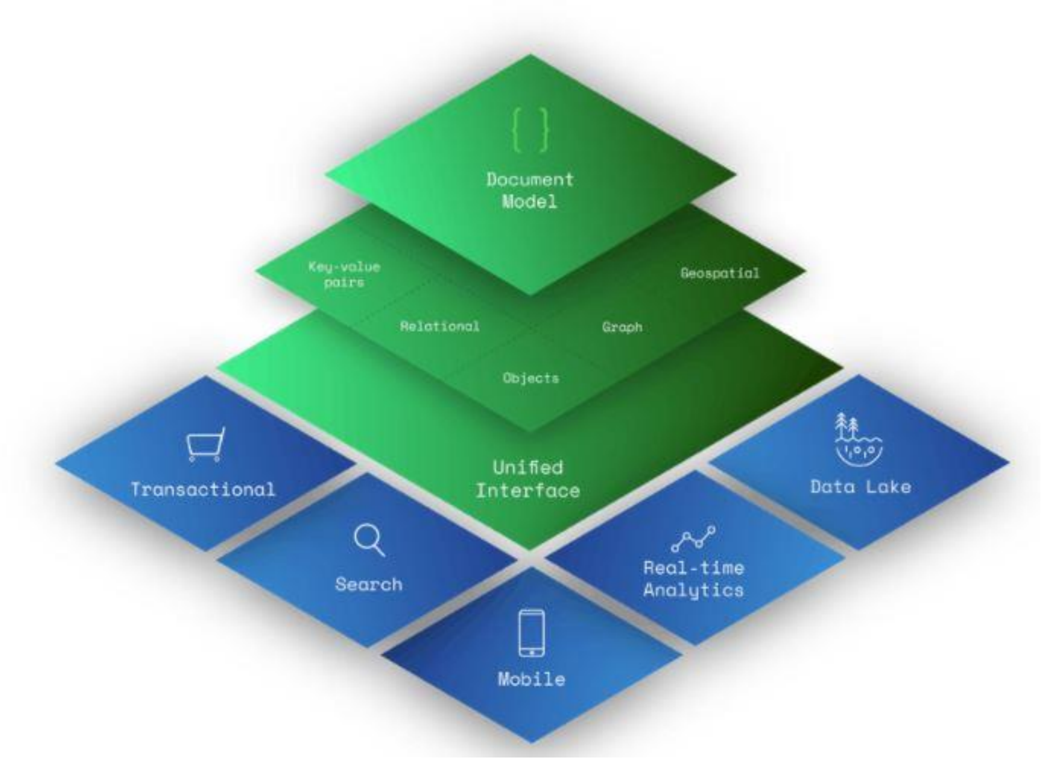
DOCUMENT DATABASE-KEY FEATURES

- **Document model:** Data is stored in documents (unlike other databases that store data in structures like tables or graphs).
- **Flexible schema**: Document databases have a flexible schema, meaning that not all documents in a collection need to have the same fields.
- **Distributed and resilient:** Document databases are distributed, which allows for horizontal scaling and data distribution. Document databases provide resiliency through replication.
- **Querying through an API or query language:** Document databases have an API or query language that allows developers to execute the CRUD operations on the database.

WHAT MAKES DOCUMENT DATABASES DIFFERENT FROM RELATIONAL DATABASES?

1. **The intuitiveness of the data model**
2. **The ubiquity of JSON documents**
3. **The flexibility of the schema**

DOCUMENT DATABASE



- Key-value pairs can be modeled with fields and values in a document.
- Relational data can be modeled differently by keeping related data together in a single document using embedded documents and arrays.
- Documents map to objects in most popular programming languages.
- Graph nodes and/or edges can be modeled as documents.
- Geospatial data can be modeled as arrays in documents.

WHAT ARE THE STRENGTHS OF DOCUMENT DATABASES?

Document databases have many strengths:

- The document model is ubiquitous, intuitive, and enables rapid software development.
- The flexible schema allows for the data model to change as an application's requirements change.
- Document databases have rich APIs and query languages that allow developers to easily interact with their data.
- Document databases are distributed (allowing for horizontal scaling as well as global data distribution) and resilient.
- Faster creation and care: Minimal maintenance is required once you create the document, which can be as simple as adding your complex object once.

WHAT ARE THE STRENGTHS OF DOCUMENT DATABASES?

Document databases have many strengths:

- No foreign keys. With the absence of this relationship dynamic, documents can be independent of one another.

WHAT ARE THE USE CASES FOR DOCUMENT DATABASES?

- Single view or data hub
- Customer data management and personalization
- Internet of Things (IoT) and time-series data
- Product catalogs and content management
- Payment processing
- Mobile apps
- Real-time analytics

COLUMN-ORIENTED DATABASE

- Group related columns together in a row
- Each column can be collection of related columns itself, referred to as a super-column
- Each row have many column-families and can have a different set of columns
- Each row has a key
- Different column-families can be stored in separate files.
- Such as Cassandra, Hbase, Hypertable, Amazon SimpleDB.

COLUMN-ORIENTED DATABASE

- Data represents a tabular structure; each row consists of a large number of columns and nested groups of interrelated data exist.
- Column families can be added or removed without any system downtime.

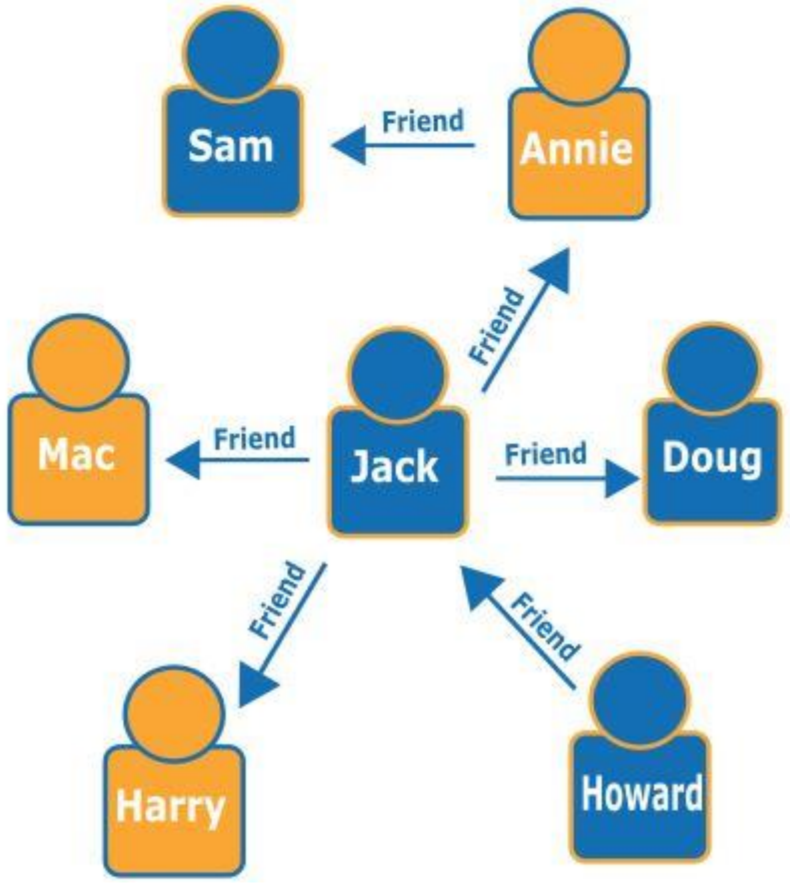
Column-families

Rows

Student_ID	Personal_details	address
101	First_name: :Ahmet Last_name:Veli DOB: 12. 07.2017 Gender: Male Ethnicity: Turkish	Street: 123 New Ave City: Porland State: Oregon Zipcode: 1234 Country: USA
201	First_name: : George Last_name: ABC DOB: 12. 07.2007 Gender: Male Ethnicity: USA	Street: 456 Old Ave City: Los Angeles Country: USA

GRAPH DATABASE

- used for inter-connected entities.
- Graph database (GDB) is a database that uses graph structures with nodes, edges, and properties to represent and store data.
- The edges representing the relationships between the nodes.
- Entities stored as nodes
- Linkages stored as edges.
- Multiple edges can be(like multiple foreign key in RDBMS)



GRAPH DATABASE-USE CASE

Querying entities based on the type of relationship with each other rather than the attributes of the entities.

- Fraud detection
- Finding groups of interconnected entities.
- Finding distances between entities in terms of the node traversal distance.
- Mining data toward finding patterns.

For example: Neo4J is graph database.

NOSQL DATABASES-NEWSQL DATABASES

- NoSQL provides scability and fault tolerance. But do not provide the same transaction and consistency support as exhibited by ACID compliant RDBMSs. Thus, not suitable for large transactional systems.
- NewSQL which combines the functionality of traditional RDBMS solutions, including ACID compliance, while also delivering the scalability and performance that NoSQL databases are known for.
- For example; VoltDB, InnoDB

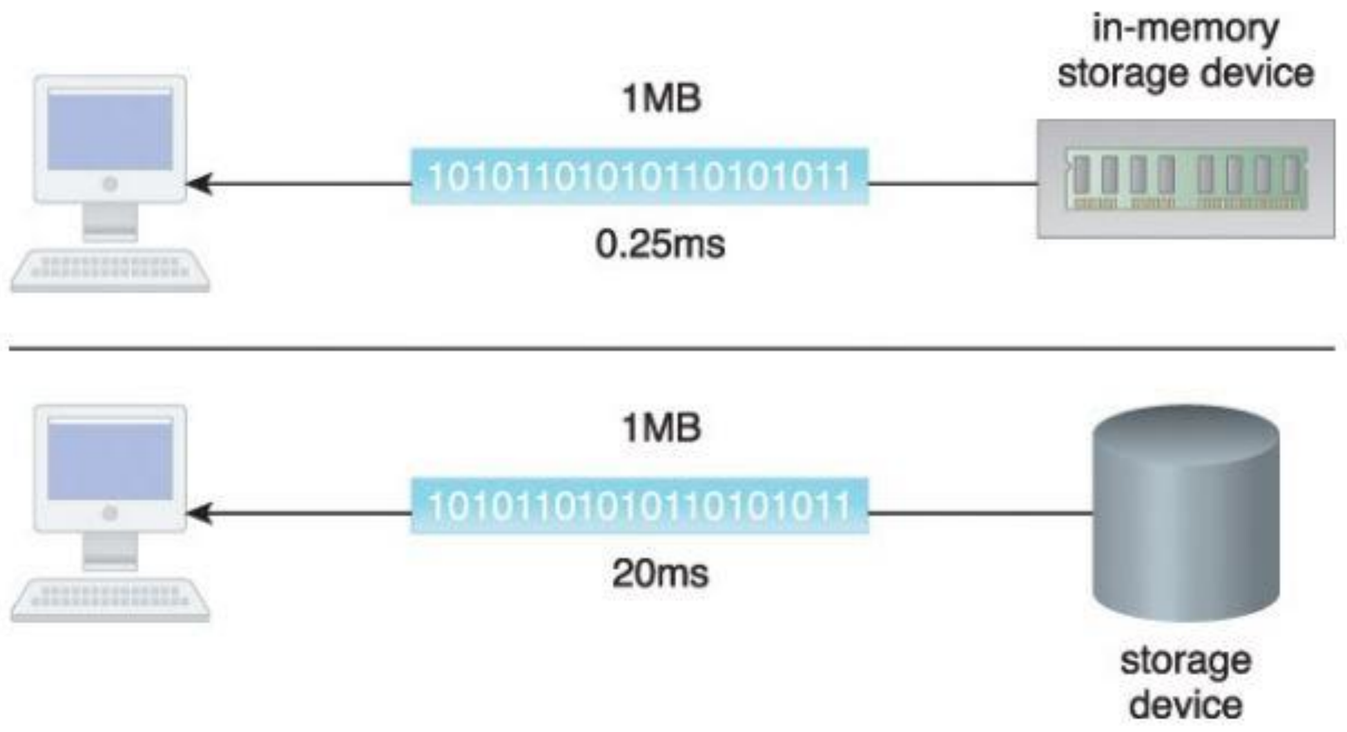
COMPARING NOSQL-NEWSQL

1. Scalability
2. Availability
3. Consistency
4. Performance

IN-MEMORY STORAGE DEVICES

- An in-memory storage device generally utilizes RAM, the main memory of a computer, as its storage medium to provide fast data access.
- Storage of data in memory eliminates the latency of disk I/O and the data transfer time between the main memory and the hard drive.
- In-memory storage device capacity can be increased massively by horizontally scaling the cluster that is hosting the in-memory storage device.
- Cluster- based memory enables real-time Big Data analytics.

IN-MEMORY STORAGE DEVICES



AN IN-MEMORY STORAGE DEVICE IS APPROPRIATE WHEN:

- data arrives at a fast pace and requires realtime analytics or event stream processing
- interactive query processing and realtime data visualization needs to be performed, including what-if analysis and drill-down operations
- the same dataset is required by multiple data processing jobs
- performing exploratory data analysis, as the same dataset does not need to be reloaded from disk if the algorithm changes
- data processing involves iterative access to the same dataset, such as executing graph-based algorithms
- developing low latency Big Data solutions with ACID transaction support

AN IN-MEMORY STORAGE DEVICE IS INAPPROPRIATE WHEN:

- data processing consists of batch processing
- very large amounts of data need to be persisted in-memory for a long time in order to perform in-depth data analysis
- datasets are extremely large and do not fit into the available memory
- an enterprise has a limited budget, as setting up an in-memory storage device may require upgrading nodes, which could either be done by node replacement or by adding more RAM

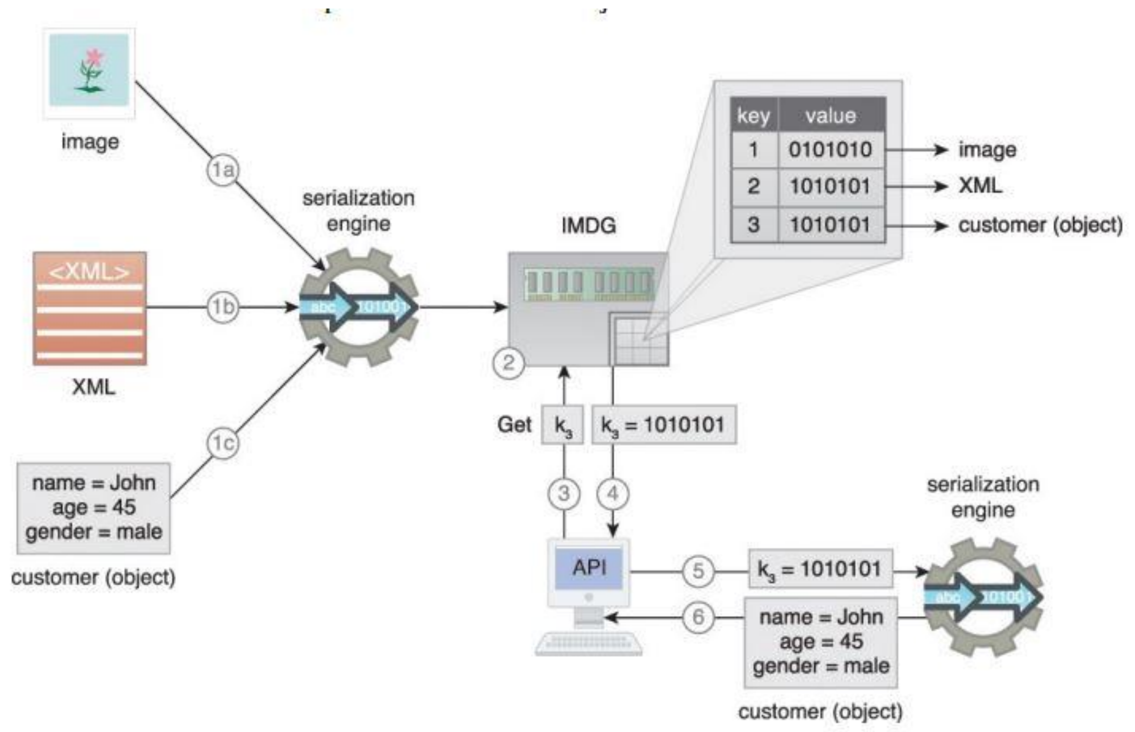
IN-MEMORY STORAGE DEVICES

- In-memory database is expensive than disk-storage.
- In-memory storage devices can be implemented as
  - In-memory data grid (IMDG)
  - In-memory database (IMDB)

IN-MEMORY DATA GRIDS

- IMDGs store data in memory as key-value pairs across multiple nodes
- supports schema-less data storage through storage of semi/unstructured data.
- Data access is typically provided via APIs.

IN-MEMORY DATA GRIDS

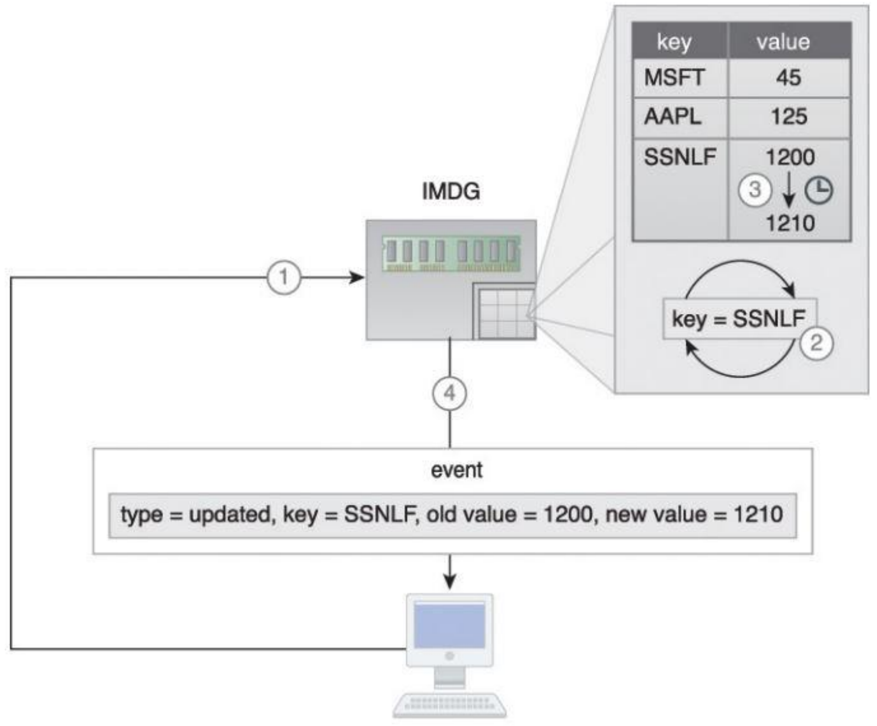


1. An image (a), XML data (b) and a customer object (c) are first serialized using a serialization engine.
2. They are then stored as key-value pairs in an IMDG.
3. A client requests the customer object via its key.
4. The value is then returned by the IMDG in serialized form
5. The client then utilizes a serialization engine to deserialize the value to obtain the customer object.
6. The client gets in order to manipulate the customer object.

IN-MEMORY DATA GRIDS

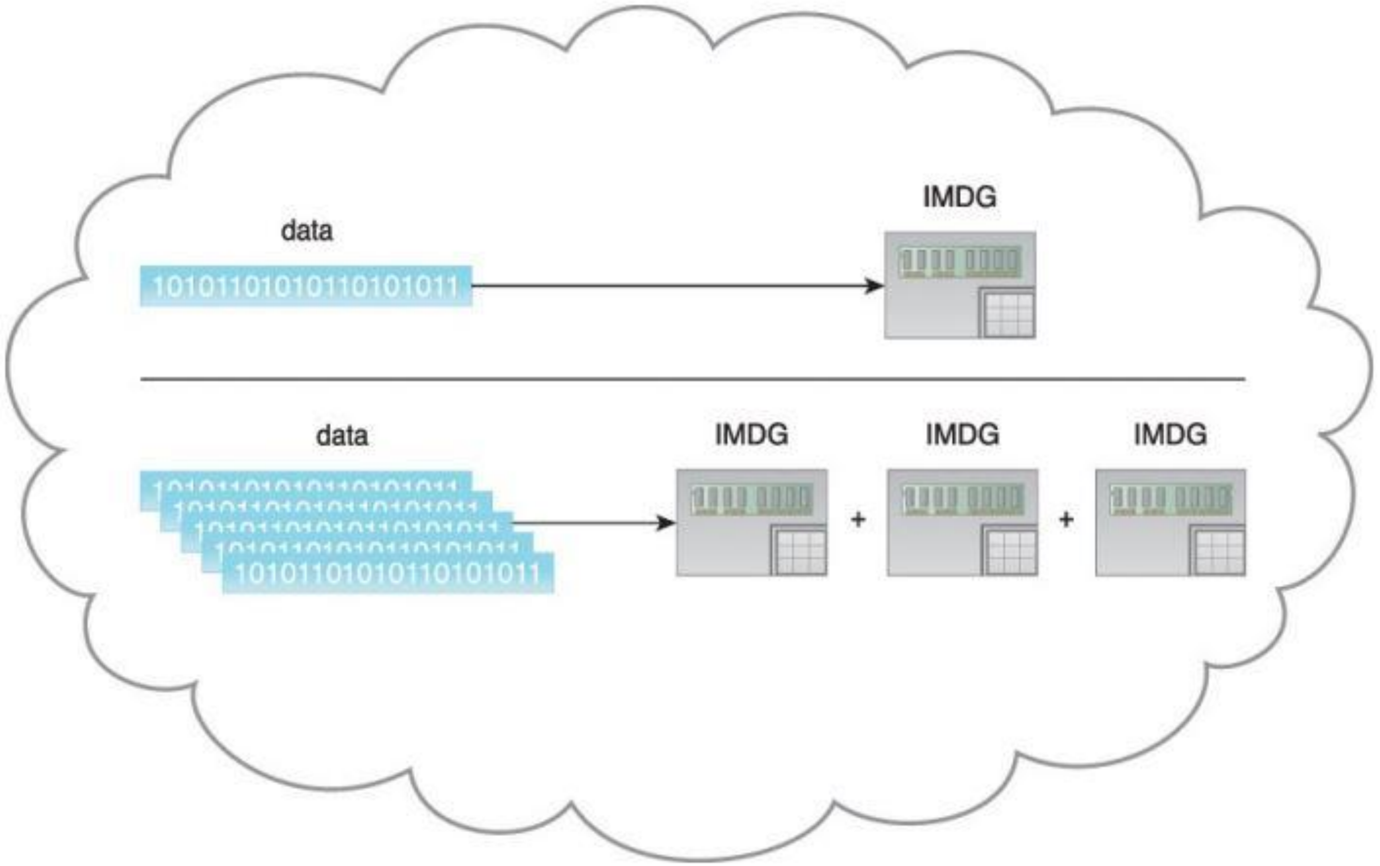
- Nodes in IMDGs keep themselves synchronized and collectively provide high availability, fault tolerance and consistency.
- IMDGs scale horizontally by implementing data partitioning and data replication and further support reliability by replicating data to at least one extra node.
- In case of a machine failure, IMDGs automatically re-create lost copies of data from replicas as part of the recovery process.

IN-MEMORY DATA GRIDS


IMDG
key
value
MSFT
45
AAPL
125
SSNLF
1200
3
1210
key = SSNLF
2
1
4
event
type = updated, key = SSNLF, old value = 1200, new value = 1210

- An IMDG stores stock prices where the key is the stock symbol, and the value is the stock price (shown as text for readability).
- A client issues a continuous query (key=SSNLF) (1) which is registered in the IMDG (2).
- When the stock price for SSNLF stock changes (3), an updated event is sent to the subscribing client that contains various details about the event (4).

IN-MEMORY DATA GRIDS



An IMDG deployed in a cloud scales out automatically as the demand for data storage increases.

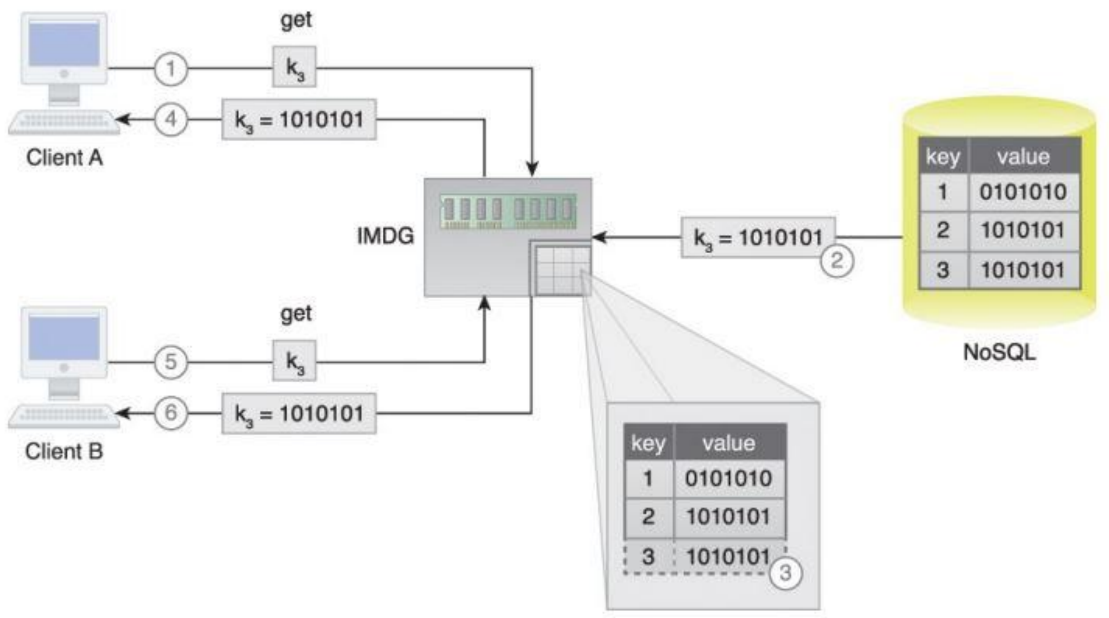
IN-MEMORY DATA GRIDS

- IMDG implementations may also provide limited or full SQL support.
- Examples include In-Memory Data Fabric, Hazelcast and Oracle Coherence

IN-MEMORY DATA GRIDS

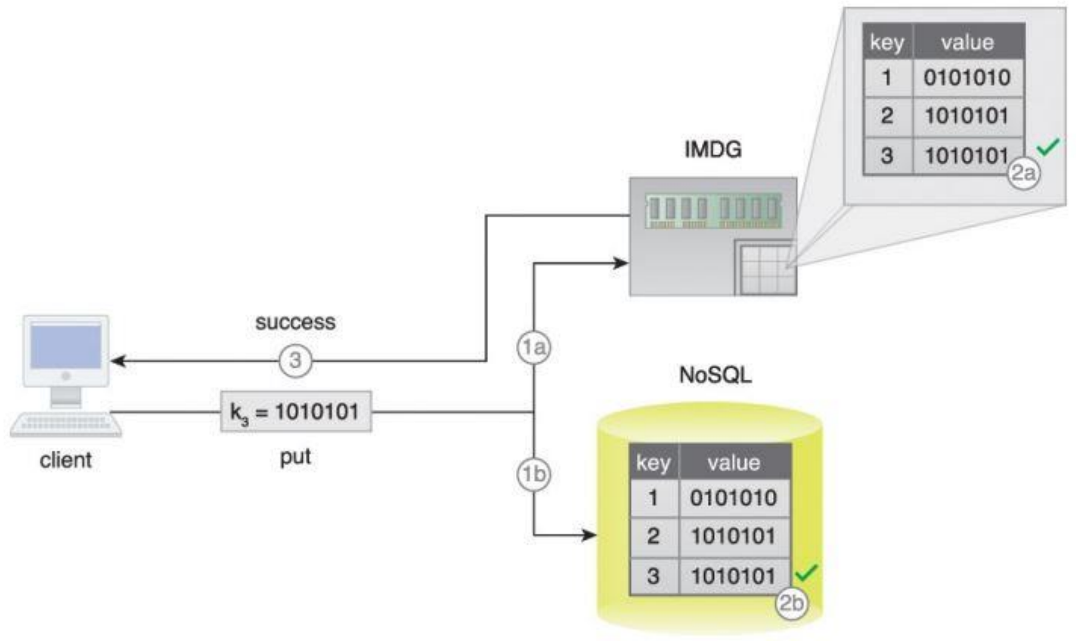
- In a Big Data solution environment, IMDGs are often deployed together with on-disk storage devices that act as the backend storage.
- This is achieved via the following approaches that can be combined as necessary to support read/write performance, consistency and simplicity requirements:
  - read-through
  - write-through
  - write-behind
  - refresh-ahead

READ-THROUGH



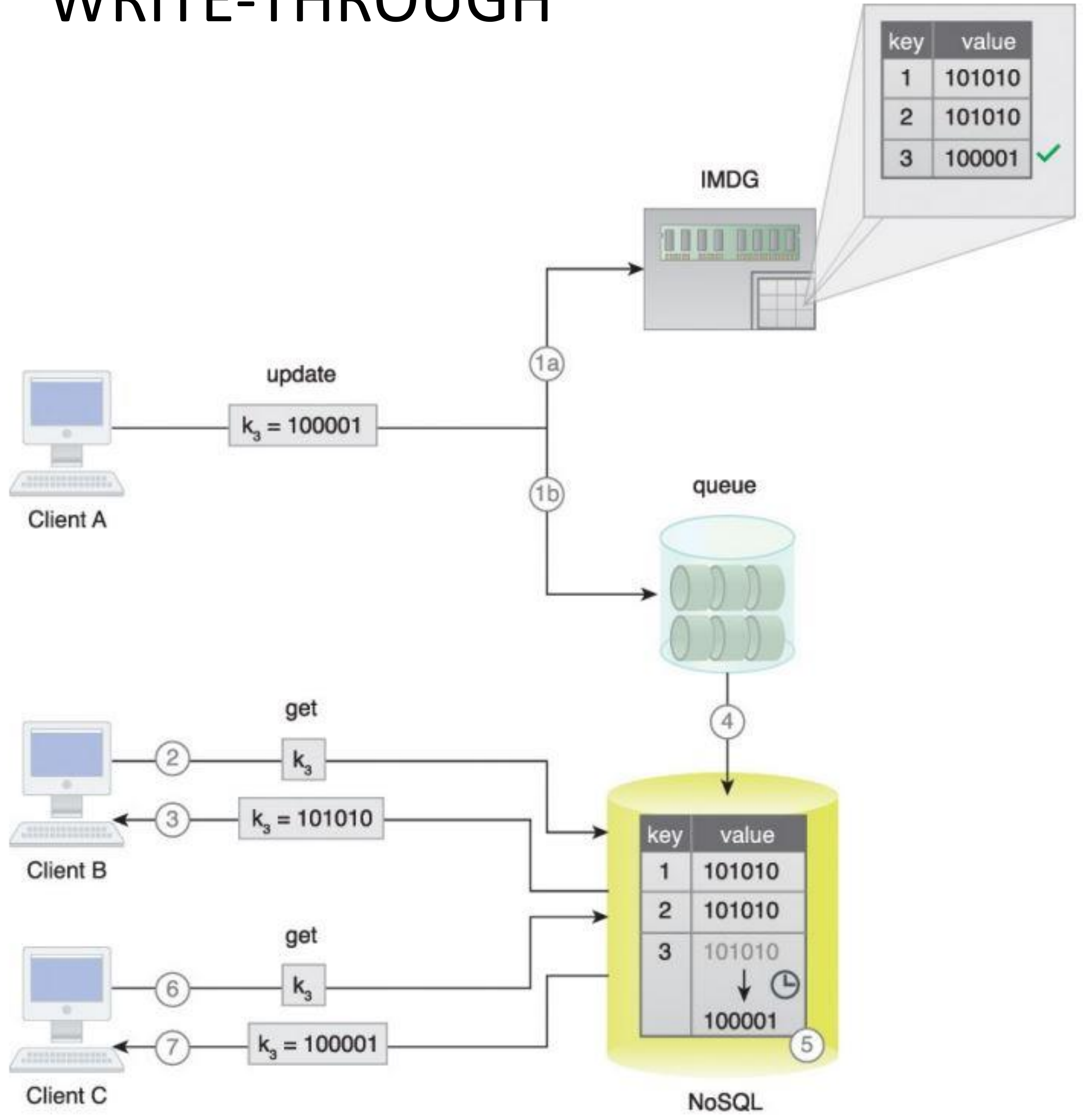
- Client A tries to read key K3 (1) which does not currently exist in the IMDG.
- Consequently, it is read from the backend storage (2) and inserted into the IMDG (3) before being sent to Client A (4).
- A subsequent request for the same key by Client B (5) is then served directly by the IMDG (6).

WRITE-THROUGH



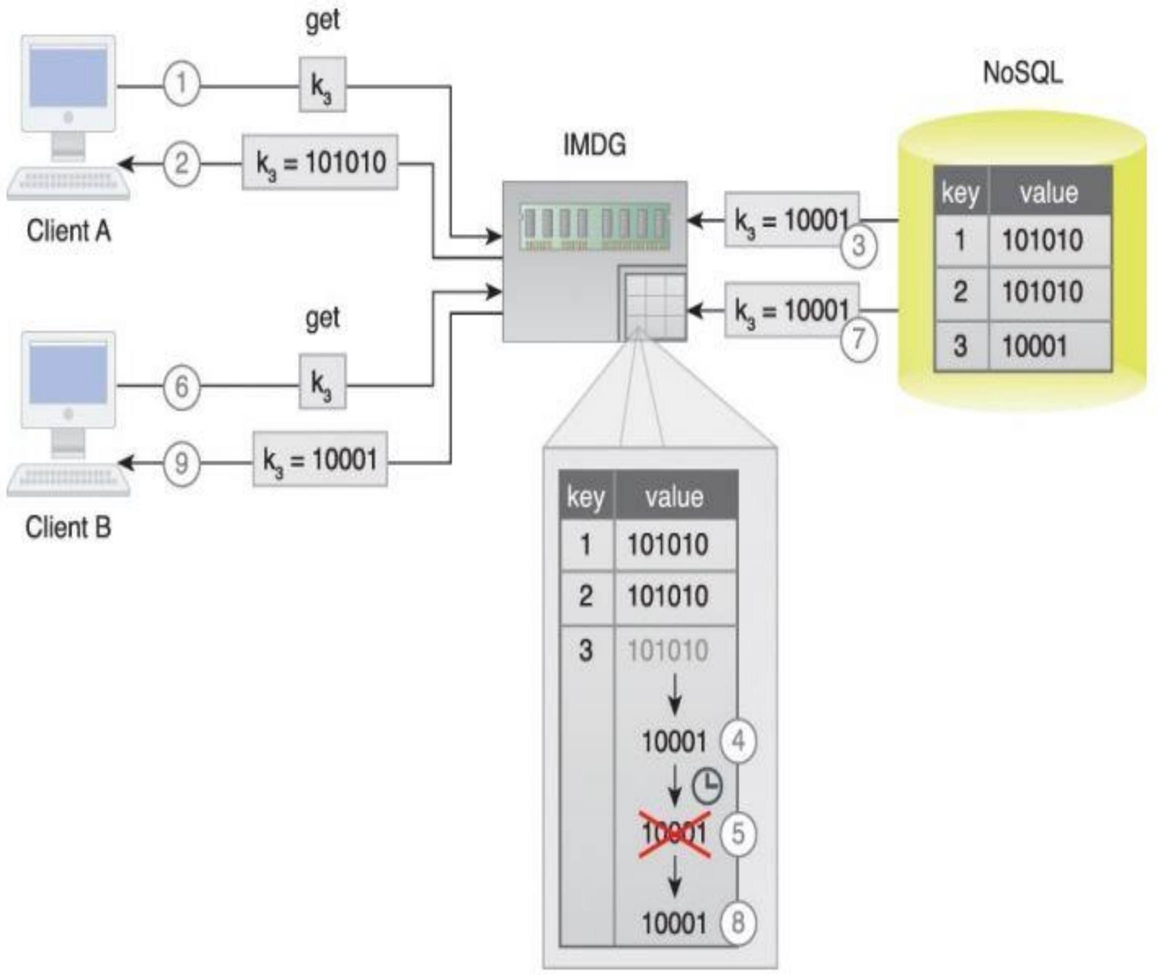
- A client inserts a new key-value pair (K3,V3) which is inserted into both the IMDG (1a) and the backend storage (1b) in a transactional manner.
- Upon successful insertion of data into the IMDG (2a) and the backend storage (2b), the client is informed that data has been successfully inserted (3).

WRITE-THROUGH



1. Client A updates value of K3, which is updated in the IMDG (a) and is also sent to a queue (b).
2. However, before the backend storage is updated, Client B makes a request for the same key.
3. The old value is sent.
4. After the configured interval…
5. … the backend storage is eventually updated.
6. Client C makes a request for the same key.
7. This time, the updated value is sent.

REFRESH-AHEAD

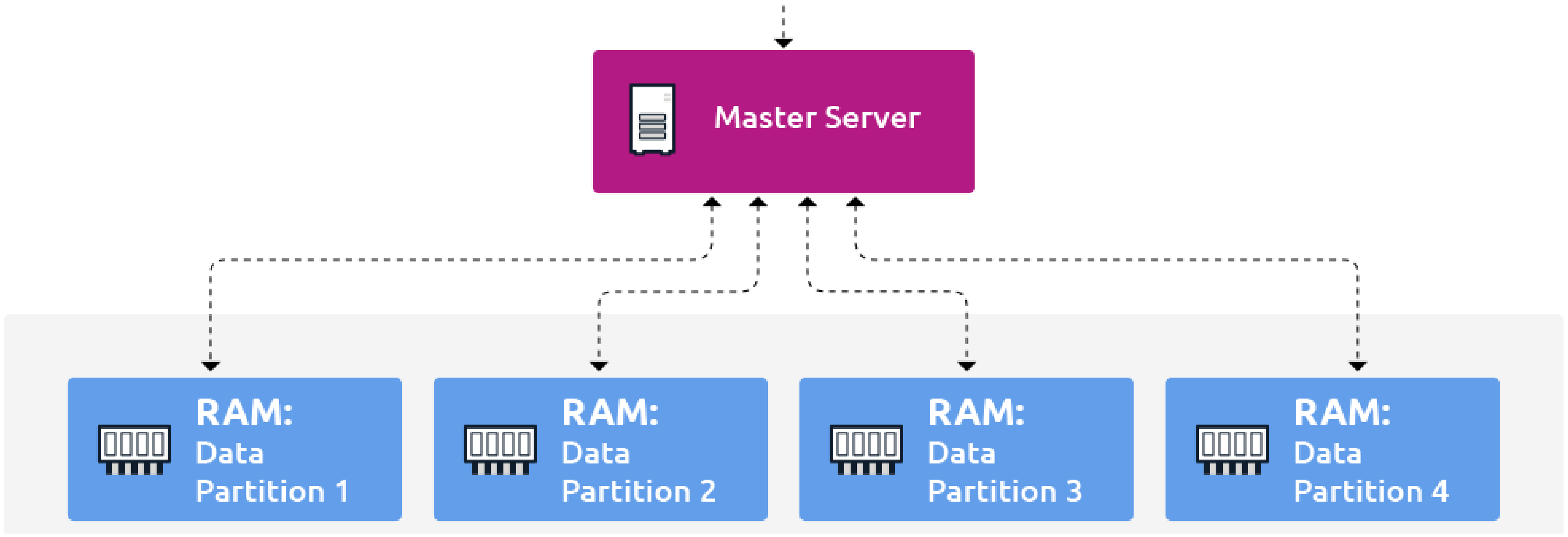


1. Client A requests K3 before its expiry time.
2. The current value is returned from the IMDG.
3. The value is refreshed from the backend storage.
4. The value is then updated in the IMDG asynchronously.
5. After the configured expiry time, the key-value pair is evicted from the IMDG.
6. Now Client B makes a request for K3.
7. As the key does not exist in the IMDG, it is synchronously requested from the backend storage…
8. …and updated
9. The value is then returned to Client B.

IN-MEMORY DATABASE

- Imports DB technology into RAM
- Can be relational or non-relational
- SQL language is used to access data.
- IMDBs are useful for when fast reads and writes of data are crucial.
- IMDBs can store relational (tabular) data, document data, key-value data, or even a combination.
- Used in analysis requiring ACID

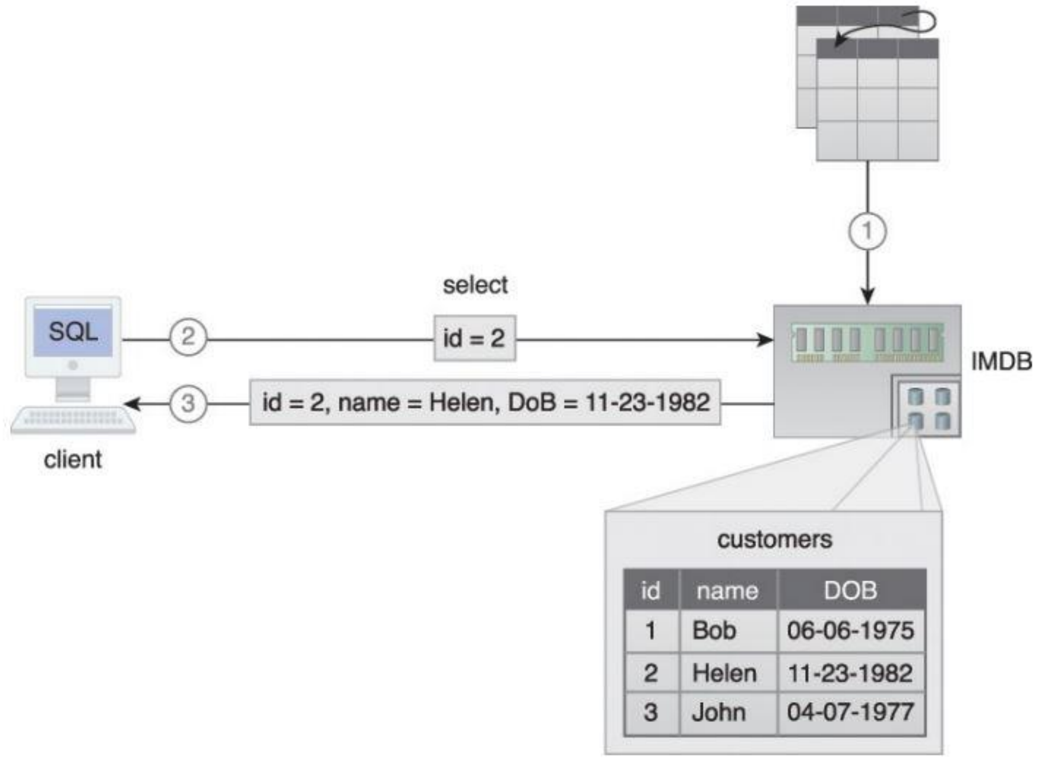
IN-MEMORY DATABASE



HOW DOES AN IN-MEMORY DATABASE WORK?

- IMDBs work by keeping all data in RAM.
- In some IMDBs, a disk-based component remains intact, but RAM is the primary storage medium.
- Most IMDBs also guard against data loss in a single data center (a capability known as “high availability”) by keeping copies (“replicas”) of all data records in multiple computers in a cluster.

IN-MEMORY DATABASE



id	name	DOB
1	Bob	06-06-1975
2	Helen	11-23-1982
3	John	04-07-1977

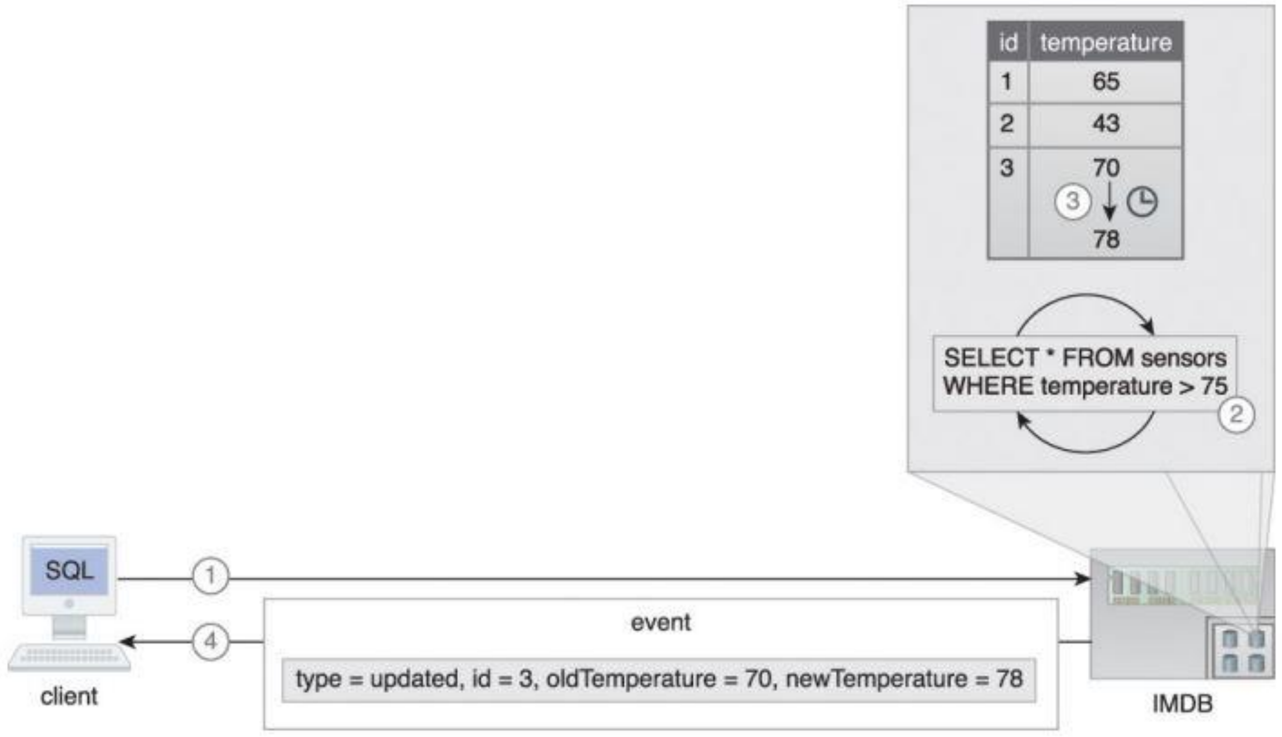
1. A relational dataset is stored into an IMDB.
2. A client requests a customer record (id = 2) via SQL.
3. The relevant customer record is then returned by the IMDB, which is directly manipulated by the client without the need for any deserialization.

IN-MEMORY DATABASE

Not all IMDB implementations directly support durability, but instead leverage various strategies for providing durability in the face of machine failures or memory corruption. These strategies include the following:

- IMDBs use of non-volatile RAM (NVRAM) for storing data permanently. (SRAM, EEPROM)
- Database transaction logs can be periodically stored to a non-volatile medium, such as disk.
- Snapshot files, which capture database state at a certain point in time, are saved to disk.
- An IMDB may leverage sharding and replication to support increasing availability and reliability as a substitute for durability.
- IMDBs can be used in conjunction with on-disk storage devices such as NoSQL databases and RDBMSs for durable storage.

IN-MEMORY DATABASE



id	temperature
1	65
2	43
3	70 → 78

```
SELECT * FROM sensors
WHERE temperature > 75
```

type = updated, id = 3, oldTemperature = 70, newTemperature = 78

1. A client issues a continuous query (select * from sensors where temperature > 75).
2. It is registered in the IMDB.
3. When the temperature for any sensor exceeds 75F
4. … an updated event is sent to the subscribing client that contains various details about the event

IN-MEMORY DATABASE

- IMDBs are heavily used in realtime analytics and can further be used for developing low latency applications requiring full ACID transaction support (relational IMDB).
- In comparison with IMDGs, IMDBs provide an easy to set up in-memory data storage option, as IMDBs do not generally require on-disk backend storage devices.
- Examples include Aerospike, MemSQL, Altibase HDB, eXtreme DB and Pivotal GemFire XD.

AN IMDB STORAGE DEVICE IS APPROPRIATE WHEN:

- relational data needs to be stored in memory with ACID support
- adding real-time support to an existing Big Data solution currently using on-disk storage
- the existing on-disk storage device can be replaced with an in-memory equivalent technology
- it is required to minimize changes to the data access layer of the application code, such as when the application consists of an SQL-based data access layer
- relational storage is more important than scalability