ALGONQUIN COLLEGE

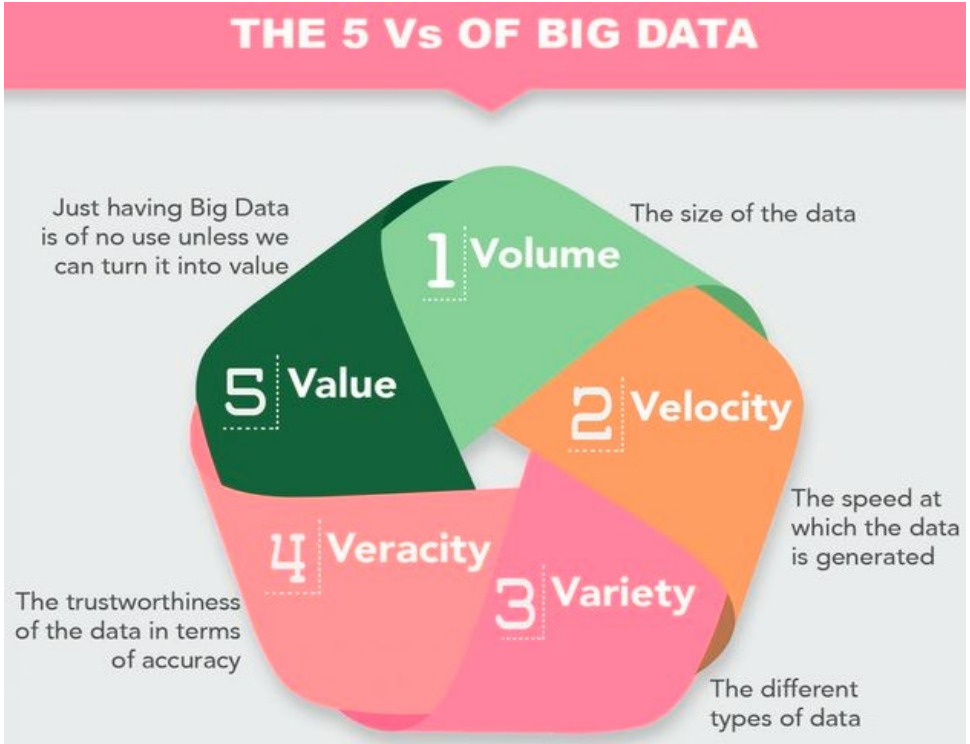
CST8390 BUSINESS INTELLIGENCE & DATA ANALYTICS

Week 10

Big Data

Characteristics of Big Data (Five Vs)

- Volume
- Velocity
- Variety
- Veracity
- Value



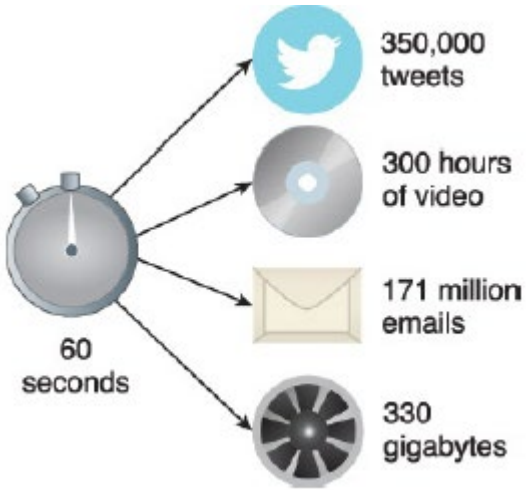
Taken from: http://bigdata.black/featured/what-is-big-data/

Volume

- refers to the vast amount of data that is generated every second/minute/hour/day in the digitized world
- Examples of data sources:
  - Online transactions such as point-of-sale and banking
  - Sensors such as GPS sensors, accelerometer, gyroscope etc.
  - Social media such as Facebook and Twitter

Velocity

- refers to the speed at which data is being generated and the pace at which data moves from one point to the next



Variety

- refers to the ever-increasing different forms of data that can come in
- Brings challenges in terms of data integration, transformation, processing and storage

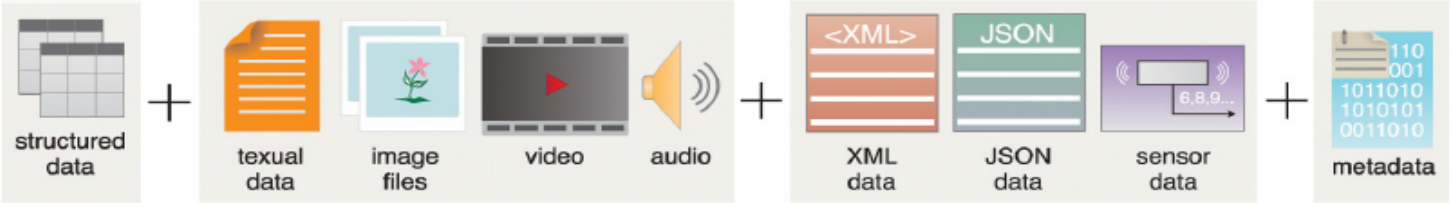


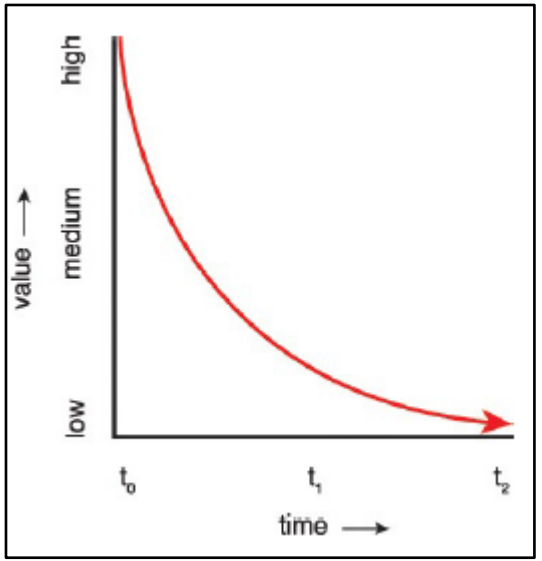
Figure 1.14 Examples of high-variety Big Data datasets include structured, textual, image, video, audio, XML, JSON, sensor data and metadata.

Veracity

- refers to the quality of the data, which can vary greatly.
- Noise from the data to be removed

Value

- Refers to the usefulness of data for an enterprise
- Value and time are inversely related. The longer it takes for data to be turned into meaningful information, the less value it has for a business.



Big Data Analytics Lifecycle

1. Business Case Evaluation
2. Data Identification
3. Data Acquisition & Filtering
4. Data Extraction
5. Data Validation & Cleansing
6. Data Aggregation & Representation
7. Data Analysis
8. Data Visualization
9. Utilization of Analysis Results

Big Data Storage Concepts

- Clusters
- Distributed File Systems
- Sharding
- Replication
  - Master-Slave
  - Peer-to-Peer

Data Wrangling

- Data acquired from external sources is often not in a format or structure that can be directly processed. To overcome these incompatibilities and prepare data for storage and processing, data wrangling is necessary.

- **Data wrangling includes steps to filter, cleanse and otherwise prepare the data for downstream analysis.**

- From a storage perspective, a copy of the data is first stored in its acquired format, and, after wrangling, the prepared data needs to be stored again.

https://tdwi.org/articles/2017/02/10/data-wrangling-and-etl-differences.aspx

https://www.talend.com/resources/data-wrangling-vs-etl/

Clusters

- A cluster is a tightly coupled collection of servers, or nodes. These servers usually have the same hardware specifications and are connected together via a network to work as a single unit.
- Each node in the cluster has its own dedicated resources, such as memory, a processor, and a hard drive.
- A cluster can execute a task by splitting it into small pieces and distributing their execution onto different computers that belong to the cluster.

Distributed File Systems

- A file system is the method of storing and organizing data on a storage device, such as flash drives, DVDs and hard drives.
- A file system provides a logical view of the data stored on the storage device and presents it as a tree structure of directories and files
- A distributed file system is a file system that can store large files spread across the nodes of a cluster. To the client, files appear to be local; however, this is only a logical view as physically the files are distributed throughout the cluster. This local view is presented via the distributed file system and it enables the files to be accessed from multiple locations.

Sharding

- Process of horizontally partitioning a large dataset into a collection of smaller, more manageable datasets called *shards*. The shards are distributed across multiple nodes, where a node is a server or a machine.
- Each shard is stored on a separate node and each node is responsible for only the data stored on it.
- Each shard shares the same schema, and all shards collectively represent the complete dataset.

Sharding - Example

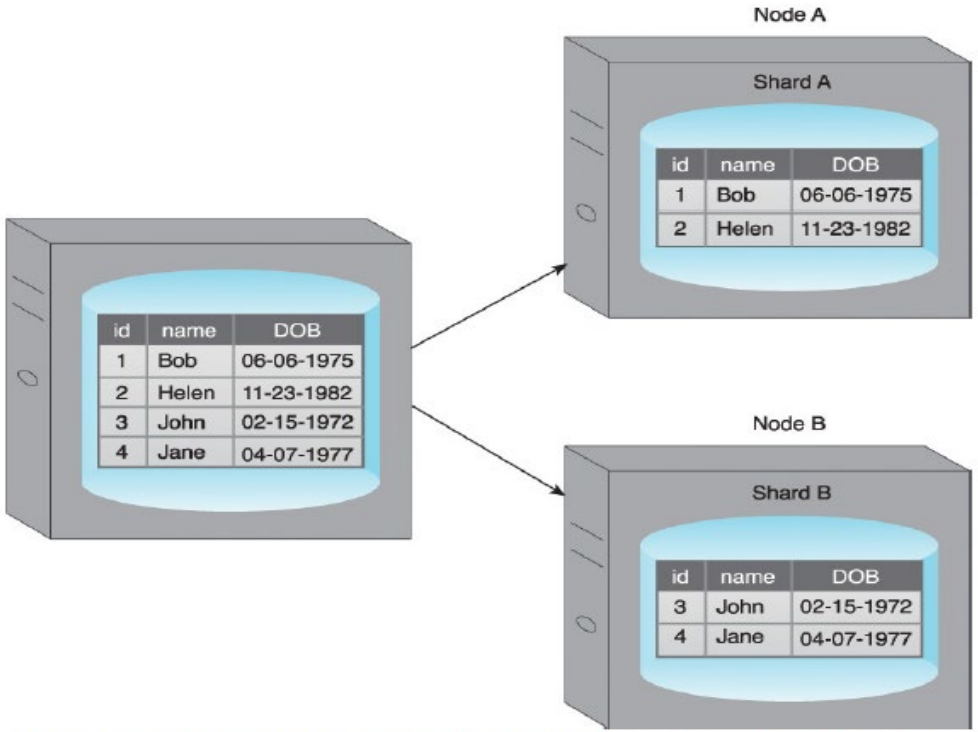


Figure 5.5 An example of sharding where a dataset is spread across Node A and Node B, resulting in Shard A and Shard B, respectively.

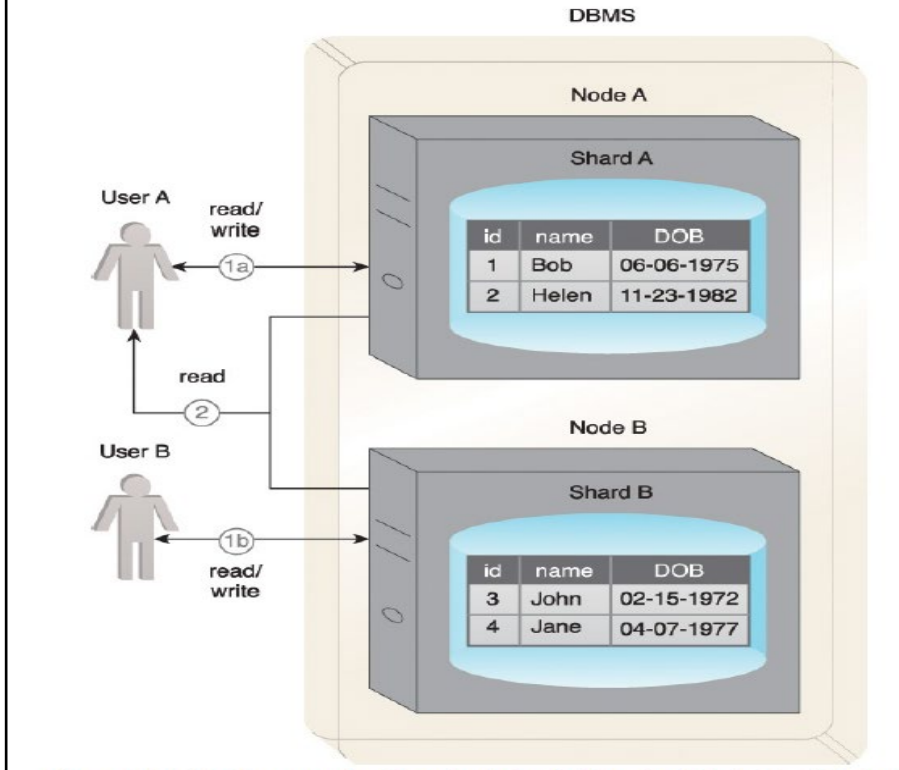


Figure 5.6 A sharding example where data is fetched from both Node A and Node B.

Sharding

- Sharding allows the distribution of processing loads across multiple nodes to achieve horizontal scalability.
- Horizontal scaling is a method for increasing a system's capacity by adding similar or higher capacity resources alongside existing resources.
- Since each node is responsible for only a part of the whole dataset, read/write times are greatly improved.

Replication

- Stores multiple copies of a dataset, known as *replicas*, on multiple nodes
- Replication provides scalability and availability due to the fact that the same data is replicated on various nodes.
- Fault tolerance is achieved since data redundancy ensures that data is not lost when an individual node fails.
- There are two different methods that are used to implement replication:
  - master-slave
  - peer-to-peer

Replication

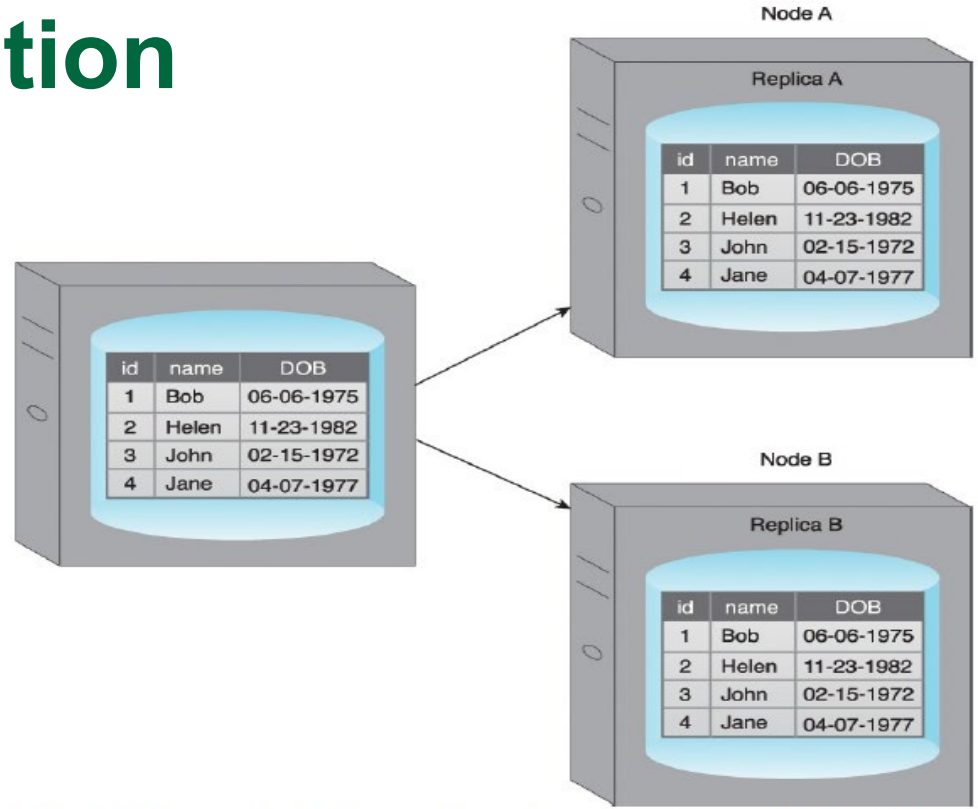


Figure 5.7 An example of replication where a dataset is replicated to Node A and Node B, resulting in Replica A and Replica B.

Master-Slave Replication

- nodes are arranged in a master-slave configuration, and all data is written to a master node. Once saved, the data is replicated over to multiple slave nodes.

- All external write requests, including insert, update and delete, occur on the master node, whereas read requests can be fulfilled by any slave node.

Master-Slave Replication

- Ideal for read intensive loads rather than write intensive loads since growing read demands can be managed by horizontal scaling to add more slave nodes.
- Writes are consistent, as all writes are coordinated by the master node. The implication is that write performance will suffer as the amount of writes increases.
- If the master node fails, reads are still possible via any of the slave nodes.
- A slave node can be configured as a backup node for the master node. In the event that the master node fails, writes are not supported until a master node is reestablished.
- The master node is either resurrected from a backup of the master node, or a new master node is chosen from the slave nodes.

Peer-to-Peer

- With peer-to-peer replication, all nodes operate at the same level.
- Each node (peer) is equally capable of handling reads and writes.
- Each write is copied to all peers

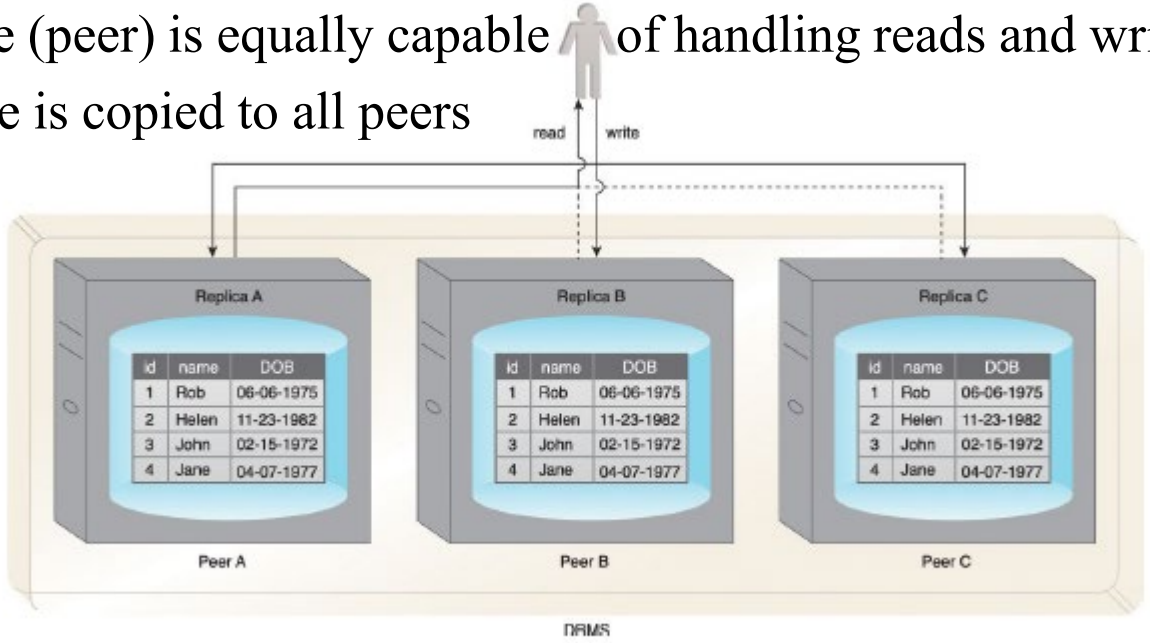


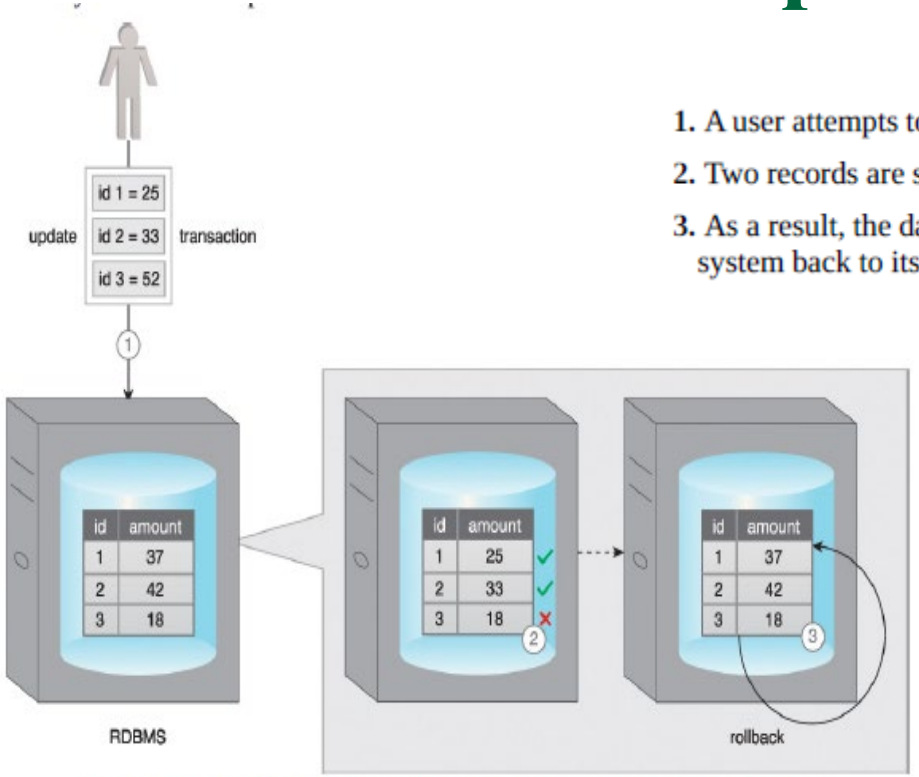
Figure 5.10 Writes are copied to Peers A, B and C simultaneously. Data is read from Peer A, but it can also be read from Peers B or C.

ACID

Database design principle related to transaction management

- Atomicity – ensures that all operations will always succeed or fail completely (no partial transactions)
- Consistency - ensures that the database will always remain in a consistent state by ensuring that only data that conforms to the constraints of the database schema can be written to the database.
- Isolation - ensures that the results of a transaction are not visible to other operations until it is complete.
- Durability - ensures that the results of an operation are permanent. In other words, once a transaction has been committed, it cannot be rolled back. This is irrespective of any system failure.

ACID – Example of Atomicity



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

Figure 5.18 An example of the atomicity property of ACID is evident here.

ACID – Example of Consistency

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

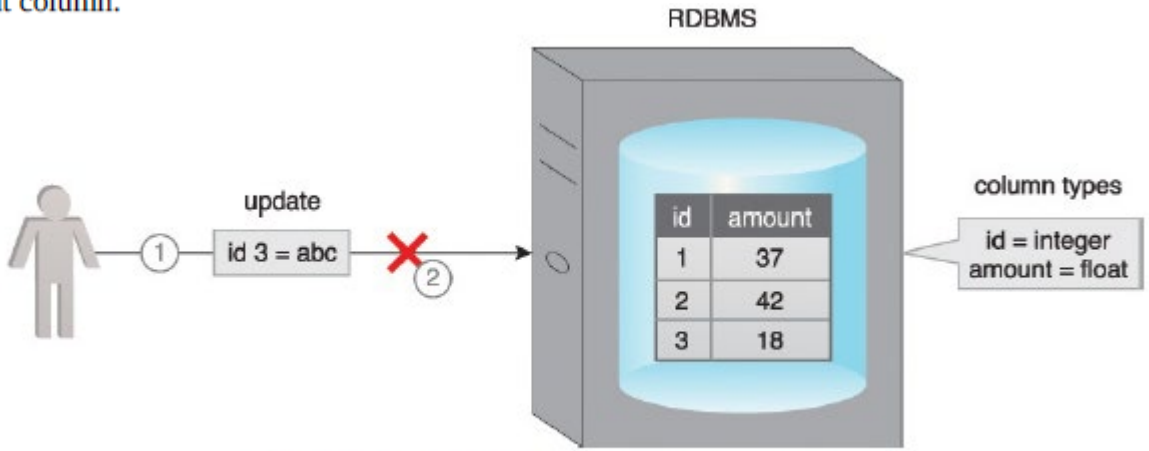
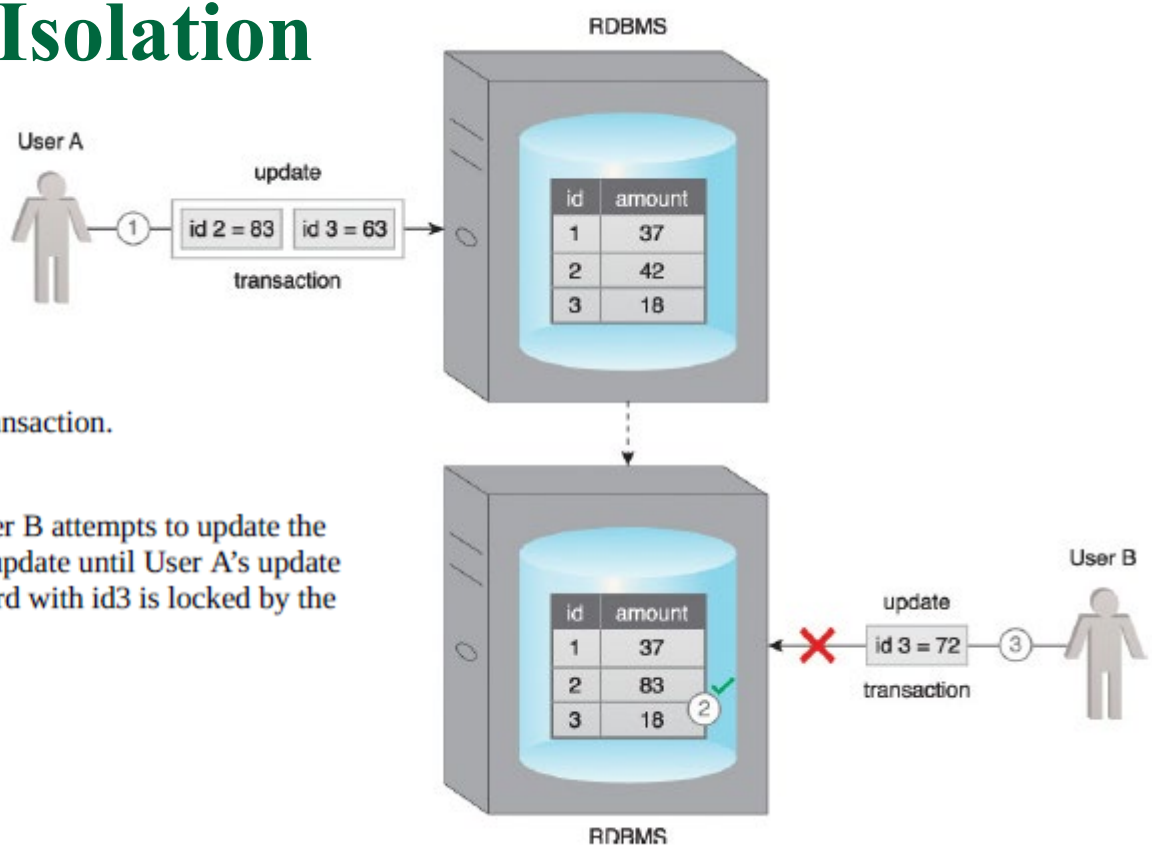


Figure 5.19 An example of the consistency of ACID.

ACID – Example of Isolation



1. User A attempts to update two records as part of a transaction.
2. The database successfully updates the first record.
3. 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.

Figure 5.20 An example of the isolation property of ACID.

ACID – Example of Durability

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

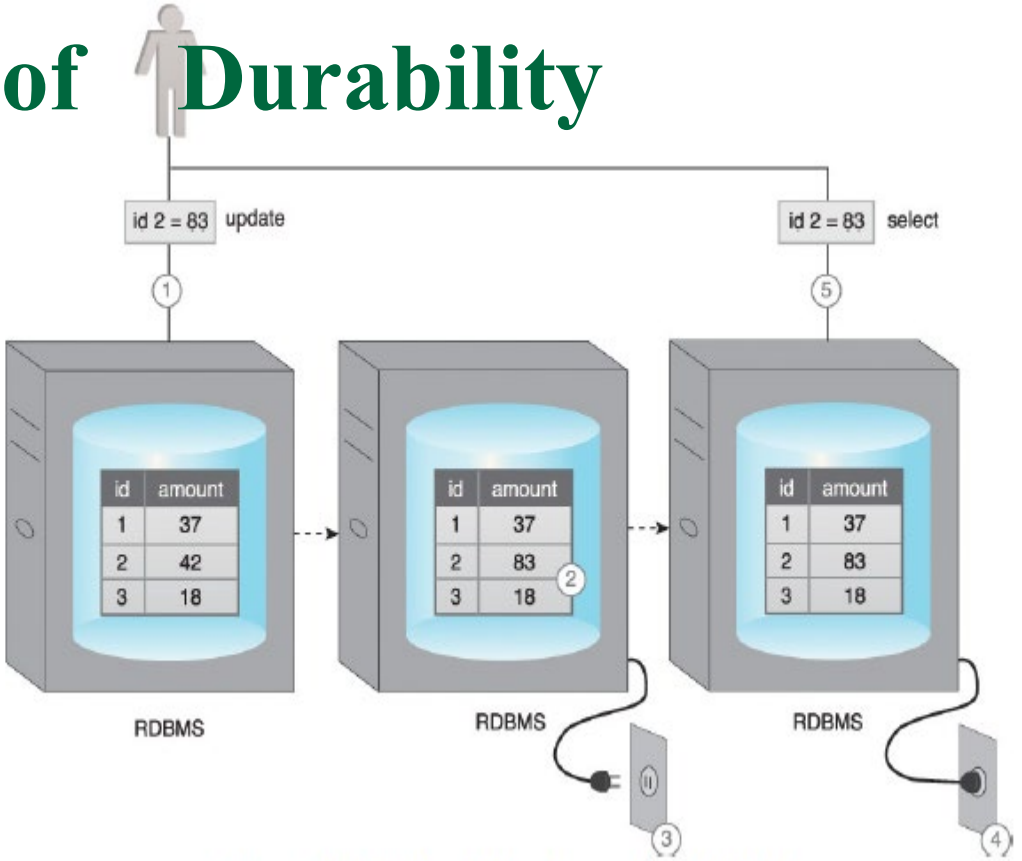


Figure 5.21 The durability characteristic of ACID.

ACID Principle - Example

1. User A attempts to update a record as part of a transaction.
2. The database validates the value and the update is successfully applied.
3. After the successful completion of the transaction, when Users B and C request the same record, the database provides the updated value to both the users.

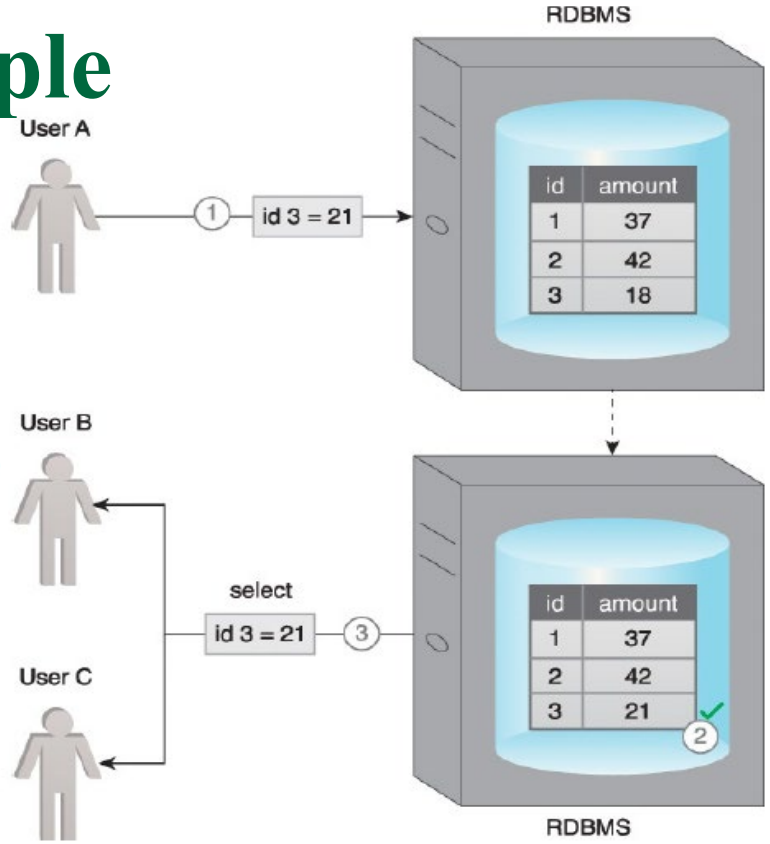


Figure 5.22 The ACID principle results in consistent database behavior.