UNIT -IV
The End of Disk?
SSD and In-Memory Databases

The End of Disk? SSD and In-Memory Databases: The End of Disk?, Solid State Disk, The Economics of Disk, SSD-Enabled Databases, In-Memory Databases, TimesTen, Redis, SAP HANA, VoltDB, Oracle 12c "in-Memory Database, Berkeley Analytics Data Stack and Spark, Spark Architecture

PYQ : (Previous Year Mumbai University Question)

Nov – 18

1. Explain Times Ten Architecture with the help of neat diagram

Apr -19

1. Diagrammatically explain the spark architecture
2. Explain the concept of Oracle 12c “in memory databases”.
3. Discuss the concept of Disk Economics

Nov – 19

1. Define In Memory Database. What are the techniques used in In Memory Database to ensure that data is not lost.
2. Explain how does Redis uses disk files for persistence.
3. What is Berkeley Analytics Data Stack ? Explain its components

Nov – 22

1. Explain Times Ten Architecture with the help of neat diagram
2. Explain Solid State Disk
3. Discuss the oracle 12c In Memory Database architecture with a neat diagram

Dec – 23

1. With a suitable diagram Explain Times Ten Architecture with the help of neat diagram explain architecture of Times Ten .

Nov – 24

1. Write a short note on Redis Architecture.
2. Explain the concept of Solid State Disks in detail

The End of Disk? SSD and In-Memory Databases:

The End of Disk?

Traditionally, computers used hard disk drives (HDDs) to store data. HDDs have spinning disks and read/write heads, which can be slower because they have moving parts.

SSDs (Solid-State Drives)

SSDs are a newer type of storage that use flash memory. Instead of spinning disks, they store data electronically. This makes SSDs much faster and more reliable because there are no moving parts. They can access data almost instantly, which speeds up your computer and makes everything feel quicker.

In-Memory Databases

In-memory databases are another big leap in speed. Instead of storing data on disk or SSD, they keep all the data in the computer's RAM (memory). RAM is extremely fast compared to both HDDs and SSDs. So, if a database is in memory, it can retrieve and process data almost instantly, making applications that rely on these databases perform very quickly.

In a Nutshell

- **HDDs:** Older, slower storage with moving parts.
- **SSDs:** Newer, faster storage with no moving parts.
- **In-Memory Databases:** Store data in RAM for even faster access, bypassing the need for disk storage altogether.

With SSDs and in-memory databases, we're moving away from older, slower disk-based storage systems toward much quicker, more efficient ways of handling and accessing data.

Magnetic disks have been used in computers since the 1950s. They work by storing data on spinning metal platters with magnetic charges. An arm moves across the disk to read or write data, but this process is slow because it relies on moving parts and spinning disks. Unlike computer processors, which have gotten much faster over time (thanks to Moore's law), the speed of these disks hasn't improved much. As a result, magnetic disks are now a bottleneck, slowing down how quickly data can be accessed and processed in databases.

Figure 7-1. Disk devices over the years

Solid State Disk:

Question 1 : Explain Solid State Disk
Question 2 : Explain the concept of Solid State Disks in detail

- **SSDs vs. Magnetic Disks:**
  - **SSDs**: No moving parts, so they work much faster.
  - **Magnetic Disks**: Have moving parts, so they are slower.

- **Types of SSDs:**
  - **DDR RAM SSDs**: Uses special RAM that needs a battery to keep data when the power is off.
  - **NAND Flash SSDs**: Common type that keeps data without power and is used in most SSDs.

- **Speed:**
  - **SSDs**: Can read data super-fast (in 25 microseconds).
  - **Magnetic Disks**: Much slower (up to 4 milliseconds).

- ****Writing Data**:**
  - **SSDs**: Fast for reading, but writing or changing data is slower.
  - **How SSDs Store Data**:
    - **Single-Level Cell (SLC)**: Stores 1 bit of data per cell.
    - **Multi-Level Cell (MLC)**: Stores more data per cell (usually 2 or 3 bits).

- **Writing Process:**

- **Reading/Writing**: Reading and starting a write are fast (one page at a time).
- **Changing Data**: Slower because it has to erase and rewrite an entire block of data (takes around 2 milliseconds).

- **Wear and Performance:**
- **Wear**: SSDs can wear out as they are used.
- **Performance Tricks**: Manufacturers use special techniques to make SSDs last longer and work better, but these can make writing data involve more actions than expected.

The Economics of Disk:

Question 1 : Discuss the concept of Disk Economics

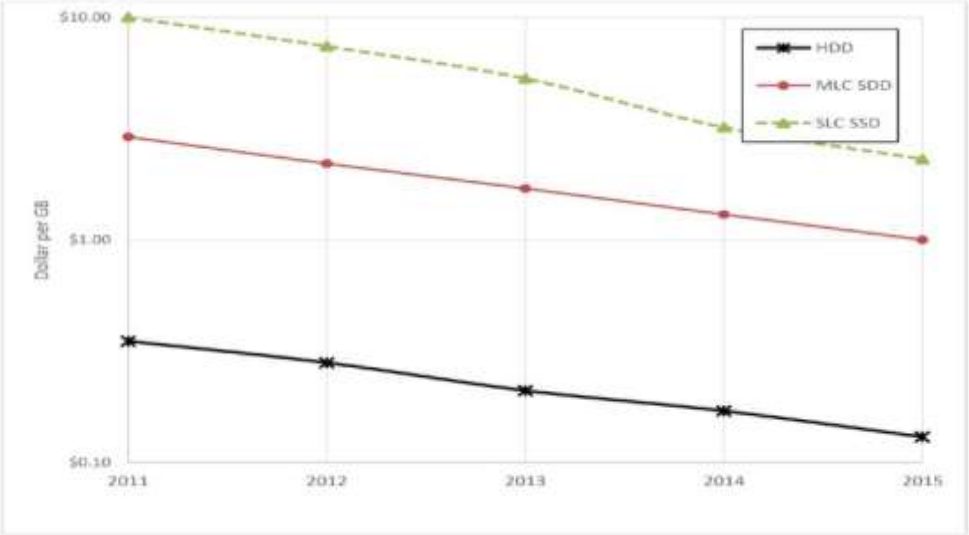

***Figure 7-3.** Trends in SSD and HDD storage costs (note logarithmic scale)*

The text discusses the future of SSDs (Solid State Disks) compared to traditional magnetic hard disks (HDDs).
1. **Future of SSD**: Many believe that one day SSDs might fully replace magnetic disks (HDDs).

2. **Near Future Reality**: This might happen eventually, but for now, the cost of storage and speed (I/O performance) is different for SSDs and HDDs.
   - **Magnetic Disks (HDDs)**: Are cheaper per unit of storage.
   - **SSDs**: Are better for high-speed data access and low wait times (latencies).
3. **Cost Trends**: The price of SSDs is decreasing quickly, but HDD prices are also dropping.
4. **Practical Use**: SSDs are becoming more affordable for small databases or systems needing speed, but they are not likely to replace HDDs for huge databases or for storing data that isn't accessed often.
5. **Future Storage Mix**: We'll likely see a combination of SSDs, HDDs, and memory used together in future database systems to balance performance and cost.

SSD-Enabled Databases :

1. **Next-gen databases & SSDs:**
   - Some modern databases are specifically built to make the most out of SSDs' unique characteristics.

2. **Issue with traditional databases on SSD:**
   - Traditional relational databases often don't perform well on SSDs because they depend on sequential write operations (good for HDDs but bad for SSDs).
   - In an ACID-compliant database, when a transaction happens, it writes data sequentially to a transaction log.
   - Moving this log to SSD may not speed things up as expected because of lower SSD write performance and something called write amplification (explained earlier).

3. **User disappointment:**
   - Users might expect better performance when moving a write-heavy traditional database to SSD, but it often doesn't improve as much as hoped.

4. **Non-relational databases (Cassandra):**

- Some non-relational databases (e.g., Cassandra) use log-structured storage (LSM trees), which don't update old blocks but instead batch-write data in a way that's better suited for SSDs.

5. **Aerospike's unique approach:**
- Aerospike, a NoSQL database, uses a log-structured file system where updates are written as new data, and old data is marked invalid. Later, the old data is cleaned up in the background.

6. **Aerospike's memory management:**
- Instead of using main memory as a cache like traditional databases, Aerospike stores indexes in memory while keeping the actual data on SSD. This is because SSDs are fast enough that constantly avoiding IO (disk reads/writes) isn't necessary anymore.

In-Memory Databases :

Question 1 : Define In Memory Database. What are the techniques used in In Memory Database to ensure that data is not lost.

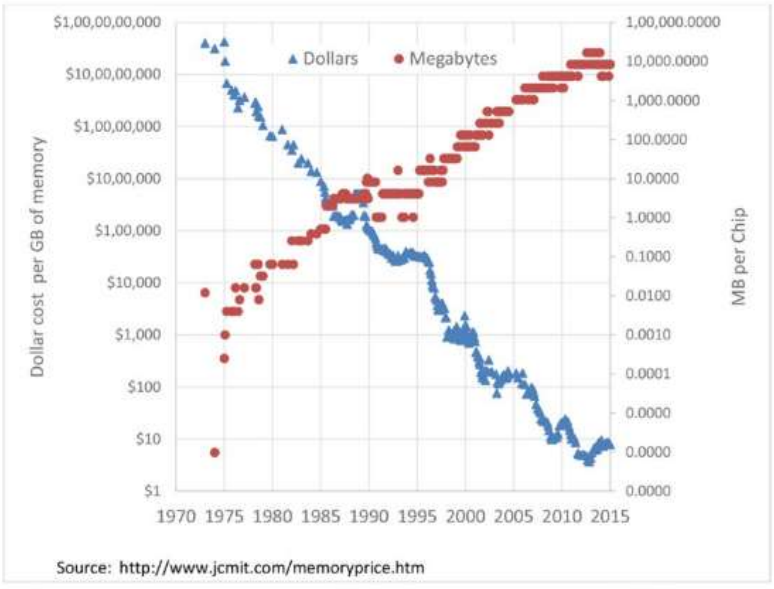

Figure 7-4. Trends for memory cost and capacity for the last 40 years (note logarithmic scale)

1. **Impact on Database Performance:**

- Solid State Disks (SSDs) have improved how fast databases work but haven't changed the overall structure of most databases much.
- A bigger shift is happening with storing entire databases directly in the computer's main memory (RAM) instead of on disks.

2. **Cost and Memory Capacity:**
- The cost of memory and how much can be stored on a server has improved a lot over the years.
- Memory has become cheaper and can hold more data than before, leading to the possibility of in-memory databases.

3. **Traditional vs. In-Memory Databases:**
- In traditional databases, data is cached in memory to speed up access from disks.
- In in-memory databases, caching in memory is unnecessary because the data is already stored in memory, making it faster and more efficient.

4. **Alternative Persistence Models:**
- Memory is volatile, meaning data is lost if power is turned off.
- In-memory databases must use other methods to prevent data loss, such as:
  - **Replication**: Copying data to other servers in a cluster.
  - **Snapshots/Checkpoints**: Saving a full copy of the database to disk at certain points.
  - **Transaction Logs**: Recording every change to the database in a log file on disk to recover if needed.

Times Ten :

Question 1: Explain Times Ten Architecture with the help of neat diagram

Question 2 : Explain Times Ten Architecture with the help of neat diagram

Question 3 : With a suitable diagram Explain Times Ten Architecture with the help of neat diagram explain architecture of Times Ten

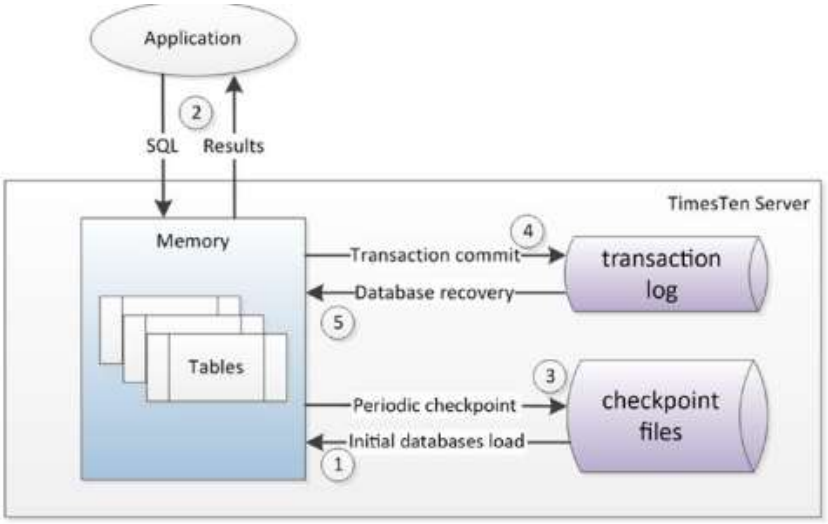

Figure 7-5. TimesTen Architecture

Figure 7-5 illustrates the TimesTen architecture.

- When the database is started, all data is loaded from checkpoint files into main memory (1).
- The application interacts with TimesTen via SQL requests that are guaranteed to find all relevant data inside that main memory (2).
- Periodically or when required database data is written to checkpoint files (3).
- An application commit triggers a write to the transaction log (4), though by default this write will be asynchronous so that the application will not need to wait on disk.
- The transaction log can be used to recover the database in the event of failure (5).

1. **Early In-Memory Database:**

- TimesTen is one of the earlier in-memory databases designed for fast performance, similar to traditional relational databases but faster.

2. **Founded and Acquired:**

- Founded in 1995 and acquired by Oracle in 2005.

3. **Standalone or Cache:**

- Offered by Oracle as a standalone in-memory database or as a cache to enhance the performance of traditional disk-based Oracle databases.

4. **SQL Support:**

- Uses a SQL-based relational model and later adopted ANSI standard SQL. It also supports Oracle's PL/SQL language.

5. **Memory Resident Data:**
- All data in TimesTen is stored in memory (RAM) for fast access.

6. **Persistence:**
- Data is saved (persisted) by periodically writing memory snapshots to disk and logging transactions to a disk-based log.

7. **Asynchronous Disk Writes:**
- By default, disk writes are asynchronous, meaning they don't wait for disk operations to complete, resulting in faster performance.

8. **Data Loss Risk:**
- If power is lost before the transaction log is written, data could be lost, meaning the system is not fully ACID compliant by default (the "D" in ACID stands for "Durability").

9. **Synchronous Writes Option:**
- Users can configure synchronous writes, which ensures ACID compliance but may slow down some operations due to waiting on disk writes.

10. **TimesTen Architecture:**
- When the database starts, it loads data from disk into memory.
- The application accesses the in-memory data using SQL.
- Data is periodically saved to disk.
- Transaction logs are written on commit, which can be asynchronous by default or synchronous if configured.
- The transaction log can recover the database after a failure.

11. **Enterprise Use:**
- Today, Times Ten is important within Oracle's enterprise solutions but remains an example of an early in-memory relational database.

Redis :

Question 1 : Explain how does Redis uses disk files for persistence

Question 2 : Write a short note on Redis Architecture.

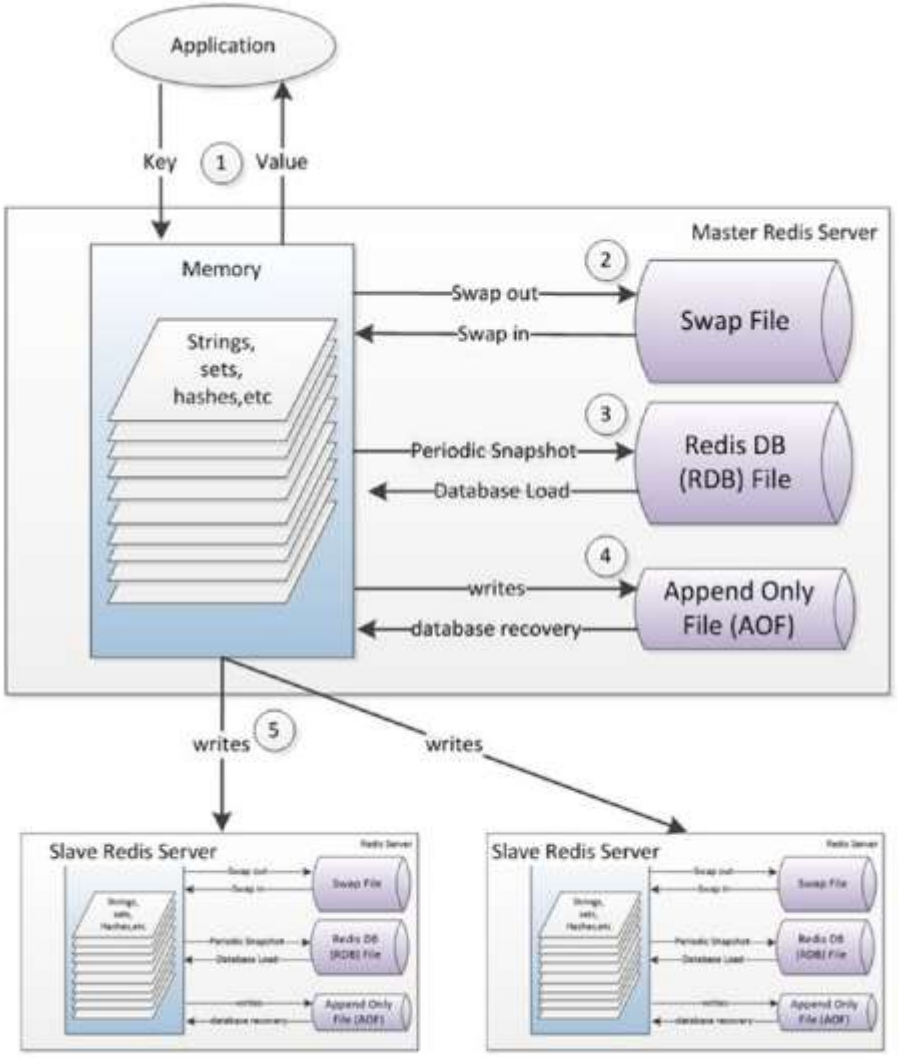

Figure 7-6. *Redis architecture*

Figure 7-6 illustrates these architectural components.

- The application interacts with Redis through primary key lookups that return "values"—strings, sets of strings, hashes of strings, and so on (1).
- The key values will almost always be in memory, though it is possible to configure Redis with a virtual memory system, in which case key values may have to be swapped in or out (2).
- Periodically, Redis may dump a copy of the entire memory space to disk (3).
- Additionally, Redis can be configured to write changes to an append-only journal file either at short intervals or after every operation (4).
- Finally, Redis may replicate the state of the master database asynchronously to slave Redis servers (5)

1. **TimesTen vs Redis:**
- **TimesTen**: An attempt to build a relational in-memory database (RDBMS-compatible).
- **Redis**: An in-memory key-value store, built for high-speed transactions on less powerful systems (like virtual machines).

2. **Redis Origins:**
- Created by **Salvatore Sanfilippo** in 2009.
- Sponsored by **VMware** in 2010 and later by **Pivotal Software** (a Big Data company).

3. **Redis Structure:**
- Uses a **key-value store architecture**: keys point to data objects like strings or collections (lists, sets, hash maps).
- Only supports **primary key lookups** (no secondary indexing).

4. **Redis and Memory:**
- Originally designed to keep all data in memory.
- Can handle datasets larger than available memory by using **virtual memory**, swapping out old data to disk, which slows down performance.

5. **Persistence in Redis:**
- **Snapshot Files**: Capture the entire Redis state at specific times or after a set number of changes.
- **Append Only File (AOF)**: Logs every change to the database, which helps recover data after a failure. You can configure when these writes happen (after each operation, every second, or as per the system).

6. **Replication:**
- Redis supports **asynchronous master/slave replication**: the master database can replicate data to slave servers for backup, even though the slave might lag behind during high loads.

7. **Redis IO Delays:**
- Operations can be delayed in Redis if:
  - The AOF is set to write after every operation (waiting for the disk write to finish).
  - Virtual memory is enabled (waiting for data to be swapped into memory).

8. **Performance:**

- Redis is simple, fast, and doesn't require expensive hardware. It works best when data fits into memory or as a **caching layer** in front of a traditional disk-based database.

In summary:
- Redis is a fast, in-memory database that uses key-value storage, designed for high performance even on lower-powered systems.
- It can persist data with snapshot files and AOF, supports replication, but may experience delays if configured to use disk operations.

SAP HANA :

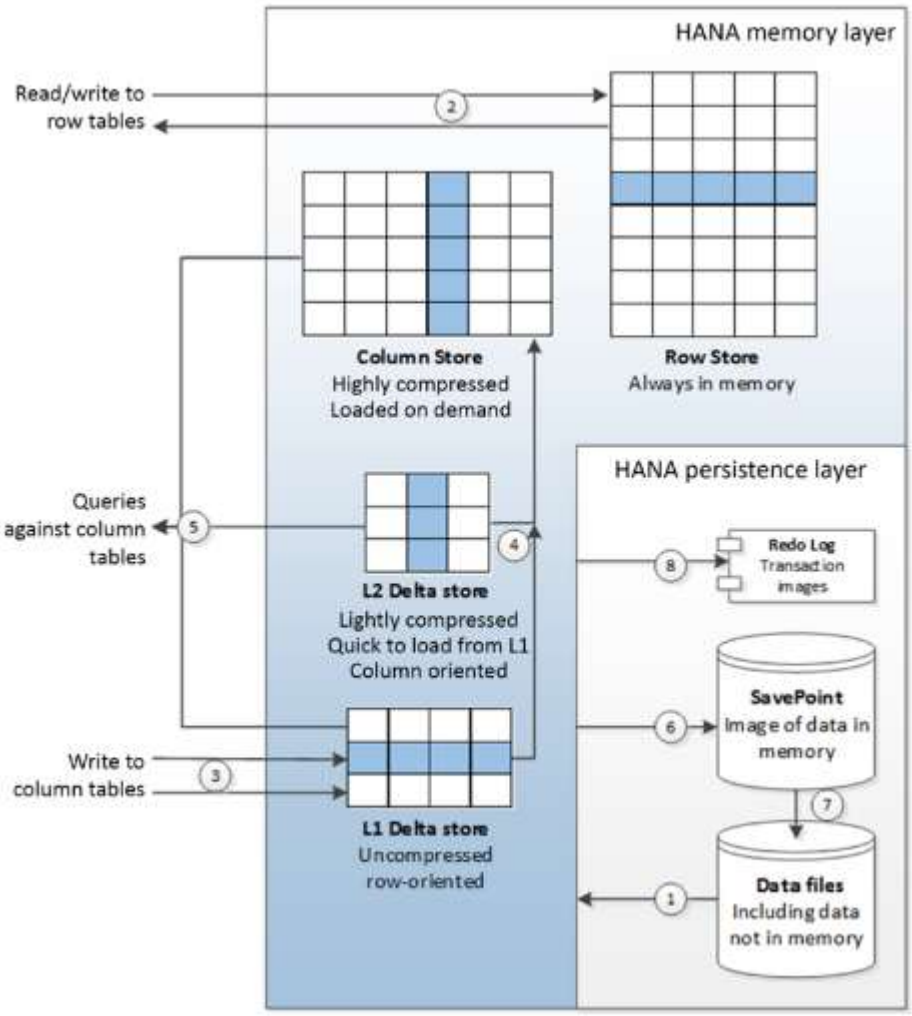

Figure 7-7. *SAP HANA architecture*

Here's a simple breakdown of the key points from the passage about HANA architecture:

1. **Start-Up Process**:
   - When HANA starts, it loads certain tables from the database files.
   - **Row-based tables** and some **column tables** are loaded immediately.
   - Other **column tables** are loaded only when needed.

2. **Reads and Writes**:
   - **Row-oriented tables** can be read from and written to directly.

3. **Column Tables Update**:
   - Updates to **column-based tables** are first stored in a temporary area called the **delta store**.
   - Initially, these updates go to an area called the **L1 row-oriented store**.

4. **Promotion of Data**:
  - Data from the **L1 store** is moved to the **L2 store** and then finally to the **main column store**.

5. **Query Process**:
  - When querying column tables, HANA reads from both the **delta store** and the **main column store**.
The passage describes how data is managed and saved in a system, specifically within the persistence architecture.

1. **Busy Little Figure 7-7**: This refers to a diagram showing how the system works.

2. **Images of Memory**: The system takes copies (snapshots) of what's currently in memory (RAM).

3. **Save Points**: These memory snapshots are saved regularly to prevent data loss.

4. **Merging with Data Files**: These saved snapshots are combined with the system's main data files over time.

5. **Commit and Transaction Record**: When a change is finalized (commit), a record of that change (transaction) is written down.

6. **Redo Log**: This transaction record is stored in a redo log, which is used to recover data in case of system failure.

7. **Fast SSD**: The redo log is saved on fast solid-state drives (SSDs) for quick access and reliability.

In short, it describes how memory data is saved, combined with main files, and recorded on SSDs for safe recovery.
This captures the main flow of how HANA handles data loading, updates, and queries.

1. **Introduction of SAP HANA (2010):**
  - SAP introduced HANA in 2010 as an in-memory database, meaning it stores data in memory (RAM) for fast access.
  - Initially designed for Business Intelligence (BI) but also supports transactional processing (OLTP).

2. **Design and Performance:**
- HANA is a relational database, known for high-speed performance.
- It uses a combination of in-memory technology and columnar data storage for efficiency.
- Installed on specialized hardware with fast SSDs (solid-state drives).

3. **Hardware Setup:**
- SAP doesn't provide the hardware itself but gives guidelines for certified servers with requirements like fast SSDs.

4. **Storage Options:**
- HANA allows data tables to be stored in two ways:
  a. **Row-oriented**: Usually used for OLTP (transactional processing).
  b. **Column-oriented**: Typically used for BI (analytics) to speed up query performance.

5. **Data in Memory:**
- Row-oriented data is always kept in memory.
- Column-oriented data is loaded into memory only when needed or can be pre-loaded at startup.

6. **Persistence and Savepoints:**
- HANA periodically saves memory data (snapshots) to disk (Savepoints).
- These Savepoints help keep the main database up-to-date.

7. **ACID Compliance:**
- HANA follows ACID principles (Atomicity, Consistency, Isolation, Durability) for reliable transactions.
- Uses a "redo" log to record transactions for recovery, stored on SSDs to reduce delays.

8. **Columnar Architecture and Delta Store:**
- **Delta store**: Temporary storage for new data before it's fully processed.
  - Data moves through stages: L1 (row-oriented), L2 (columnar but not fully processed), and then the main column store (fully compressed and organized).

9. **Startup Process:**
- When HANA starts, row-oriented data and some column data are loaded into memory.
- Other column data is loaded when needed (on demand).

10. **Updates and Queries:**
- Row-based data is updated directly in memory.
- Column-based data updates go through the delta store before reaching the main column store.
- Queries on column data check both delta store and main column store.

11. **Snapshots and Savepoints:**
- HANA regularly creates memory images (snapshots) and saves them to disk (Savepoints) to keep the database synchronized.

12. **Redo Log on SSD:**
- When a transaction is committed, a record is written to the redo log, which is stored on a fast SSD to prevent slowdowns.

13. **In-memory with Disk IO:**
- Even though HANA is an in-memory database, disk operations are needed during data commits and when column tables are loaded into memory.

VoltDB :

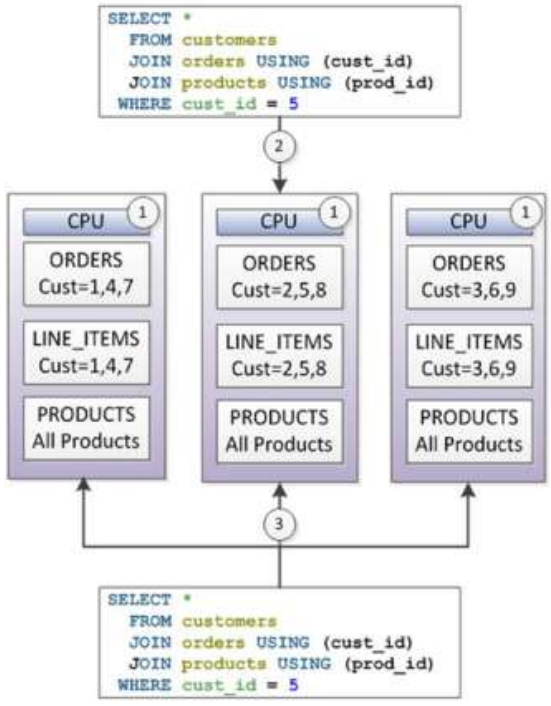

Figure 7-8. VoltDB partitioning

Figure 7.8 indicates:

1. **Partitions and CPU cores:**
- Each partition (a section of the database) is linked to one CPU core.
- Only one SQL statement is active in a partition at a time.

2. **Partitioning of tables:**
- The *ORDERS* and *CUSTOMERS* tables are divided (partitioned) based on customer ID.
- The *PRODUCTS* table is not partitioned and is copied (replicated) fully to all partitions.

3. **Query types:**
- **Single customer query**: If the query involves just one customer's data, only one partition is used.
- **Multiple customers query**: If the query involves multiple customers, all partitions are used for a short time.

In short, partitioning helps distribute data, and replication helps ensure all partitions have the necessary data. Depending on the query, it may involve one or all partitions.

In-Memory Databases

- **Redis, HANA, TimesTen:**
- These are in-memory databases, meaning they store data primarily in memory (RAM), not on disk.
- Even though they use memory, they sometimes still need to wait for disk access, especially when committing a transaction to a journal or log for safety.

VoltDB Overview

- **VoltDB:**
- A commercial database based on **H-store**, which is designed to avoid disk usage during normal transactions and operate entirely in memory.
- It supports **ACID** (Atomicity, Consistency, Isolation, Durability) properties like traditional databases.
- Instead of using a disk to ensure data safety, it uses **replication** across multiple machines.

- The number of machines required depends on the **K-safety level**, which determines how many machine failures can occur without losing data (e.g., K=2 means up to two machines can fail without data loss).

Logging in VoltDB

- **Command Log**:**
- VoltDB can use a ****command log**** to record transactions. This log can be stored on disk, either synchronously (at each commit) or asynchronously (after several transactions).
- This reduces overhead since only the transaction commands are logged, not the actual data blocks.

Data Partitioning and Replication

- **Partitioning**:**

- Data in VoltDB can be **partitioned** or **replicated**.
- **Partitioning** means splitting data across different machines and CPU cores, while **replication** means duplicating data across machines.
- Partitioned tables handle large amounts of transactional data, while replicated tables usually hold smaller reference data.

VoltDB Performance

- **Concurrency**:**

- Each **partition** in VoltDB is handled by a single CPU core, allowing fast transactions with minimal locking.
- Transactions are processed as stored procedures, minimizing delays and improving performance.

By summarizing these points, we get a simplified overview of how VoltDB, Redis, HANA, and TimesTen work as in-memory databases, focusing on speed, memory use, and how they ensure data safety without traditional disk use.

Oracle 12c “in-Memory Database” :

Question 1: Explain the concept of Oracle 12c “in memory databases”.
Question 2 : Discuss the oracle 12c In Memory Database architecture with a neat diagram

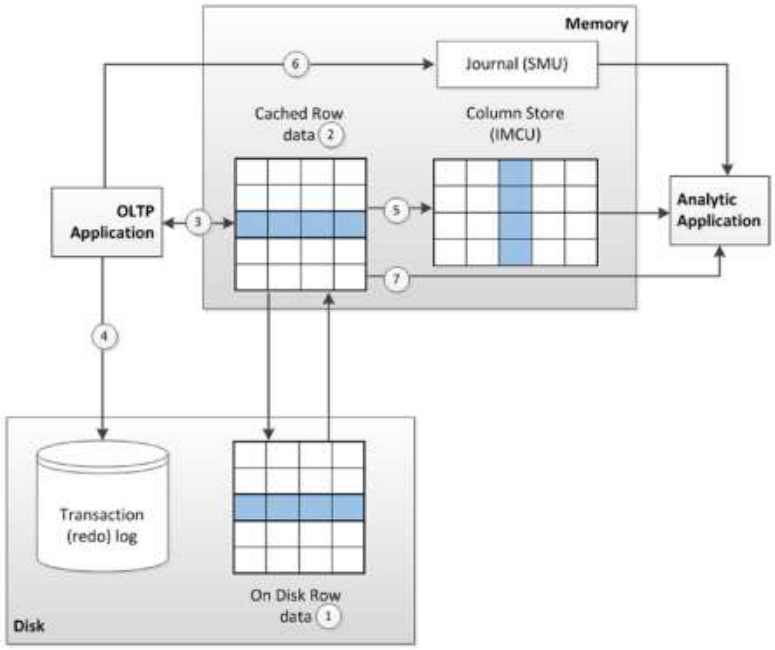

Figure 7-9. Oracle 12c "in-memory" architecture

Oracle RDBMS Version 12.1: In-Memory Feature Simplified

Key Points:
1. **Introduction of In-Memory Feature:** Oracle 12.1 introduced a feature called "Oracle database in-memory," but this doesn't mean the whole database is stored in memory.
2. **Two Data Stores**:
- **Row Store**: Data is stored in the usual disk-based row format.
- **In-Memory Column Store**: A special column-based format of the data is kept in memory to speed up certain types of queries (especially analytics).
3. **How It Works:**
- **Disk Storage (1)**: Data is saved on disk as usual.
- **Memory Cache (2)**: Data is also cached in memory for faster access.
- **OLTP Applications (3)**: Applications that perform transactions read and write from the memory, making things faster.
- **Transaction Log (4)**: Any changes made in memory are immediately saved in a log on disk.

4. **Analytics with Column Store:**

- **Column Format (5)**: Data can be converted into a column format for faster analytics when needed.
- **Journal for Updates (6)**: If new transactions happen after the column format is created, they are tracked in a journal.
- **Querying Data (7)**: When running analytics, Oracle checks if any updates exist and may adjust the column data accordingly or rebuild it.

In simple terms, Oracle 12.1 keeps data both on disk and in memory to improve speed, particularly for analytic queries, by using a column-based format stored in memory.

Berkeley Analytics Data Stack and Spark :

Question 1: What is Berkeley Analytics Data Stack ? Explain its components

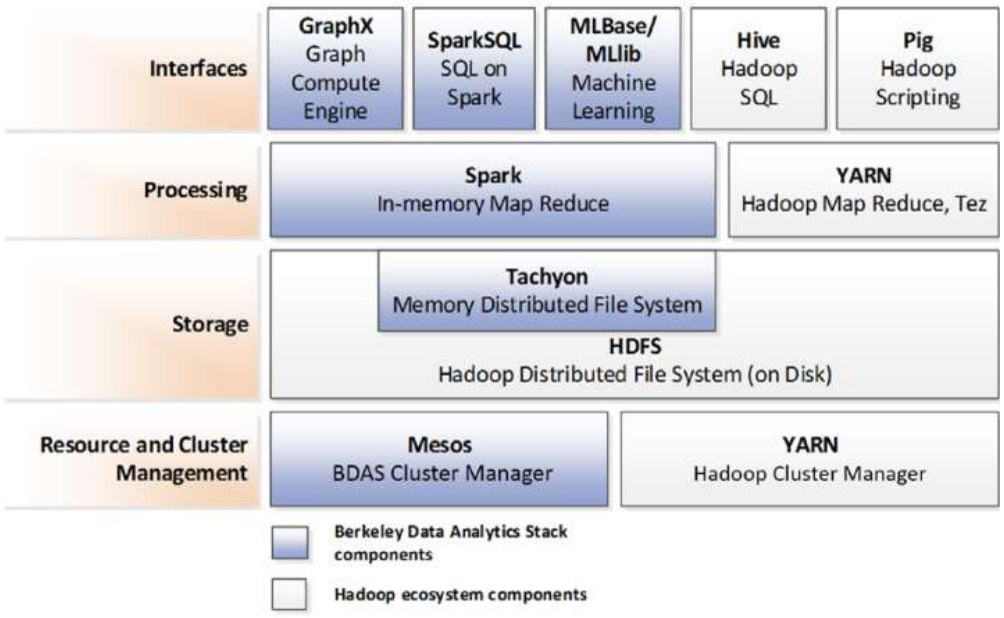

Figure 7-10. *Spark, Hadoop, and the Berkeley Data Analytics Stack*

1. **In-memory Databases Comparison:**

- **SAP HANA** and **Oracle TimesTen** are in-memory versions of traditional databases (relational).
- **Redis** is an in-memory key-value store.

- **Spark** is like an in-memory version of Hadoop, which is a system for processing large amounts of data.

2. **Hadoop's Role:**
- Hadoop became the main system for **Big Data** because it can handle large amounts of both structured and unstructured data.
- **MapReduce** (a part of Hadoop 1.0) was a simple way to process data in parallel across many servers. However, it is not very fast for real-time tasks or some complex algorithms.

3. **Limitations of MapReduce:**
- MapReduce is great for batch processing but not ideal for real-time data processing.
- It has a slow startup time, and for some machine learning tasks, it is too slow.

4. **Spark's Development:**
- **Spark** was created by the **AMPlab** at UC Berkeley to handle advanced analytics and machine learning on Big Data.
- Spark improves on Hadoop by using memory to store data that is accessed often, which makes processing faster.

5. **Spark's Features:**
- **Spark SQL** allows SQL-like queries on Spark.
- **Spark Streaming** processes data in real-time.
- **GraphX** is for graph-based computations.
- **MLBase** provides machine learning tools on Spark.

6. **Integration with Hadoop:**
- Spark is not a replacement for Hadoop but works with it. Many systems use Spark along with Hadoop’s **HDFS** (for storage) and **YARN** (for resource management).
- Spark is used in major distributions of Hadoop like **Cloudera**, **Hortonworks**, and **MapR**.
- It is also part of **Datastax's Cassandra** distribution.

Summary:
- Hadoop and Spark are both tools for handling Big Data, but Spark focuses on faster, in-memory processing. It works alongside Hadoop's storage systems and has features for SQL, streaming, graph processing, and machine learning.

Spark Architecture :

Question 1 : Diagrammatically explain the spark architecture

1. **Data Representation:**

- Spark uses something called **Resilient Distributed Datasets (RDDs)** to represent data.
- RDDs are like collections of data (objects) that can be split across many computers in a cluster.

2. **Automatic Handling:**

- The **Spark framework** automatically manages how data is divided and processed across different computers in the cluster.

3. **Immutable RDDs:**

- RDDs cannot be changed once created; instead, new RDDs are created when operations are performed.
- For example, sorting an RDD makes a new sorted RDD instead of changing the original.

4. **High-Level Operations:**

- Spark has simple methods for common tasks like joining, filtering, and aggregating data.
- These tasks, which take many lines of code in traditional systems like **MapReduce**, are much simpler in Spark.

5. **Flexible Data Types:**

- Data in RDDs can be simple types like strings, or more complex types like Java/Scala objects.
- Often, RDDs store key-value pairs, and Spark provides special operations for working with such data.

6. **Efficient Processing with DAG:**

- Spark uses **Directed Acyclic Graphs (DAGs)** to optimize data processing, which is more efficient than traditional methods like MapReduce.

7. **In-Memory Processing:**

- Spark works primarily with data in memory (RAM) but can handle larger datasets by storing parts of the data on disk if needed.

8. **Not OLTP-Focused:**

- Spark is not designed for online transaction processing (OLTP), so it doesn't need transaction logs or journals.

9. **Integration with External Systems:**
- Spark can read and write data from various sources, like local/distributed file systems (e.g., Hadoop HDFS) or relational databases via JDBC.
- It can also work with data from **HBase** or **Cassandra**.

10. **Data Persistence:**
- Data in RDDs can be saved to disk in various formats like text files or JSON documents.

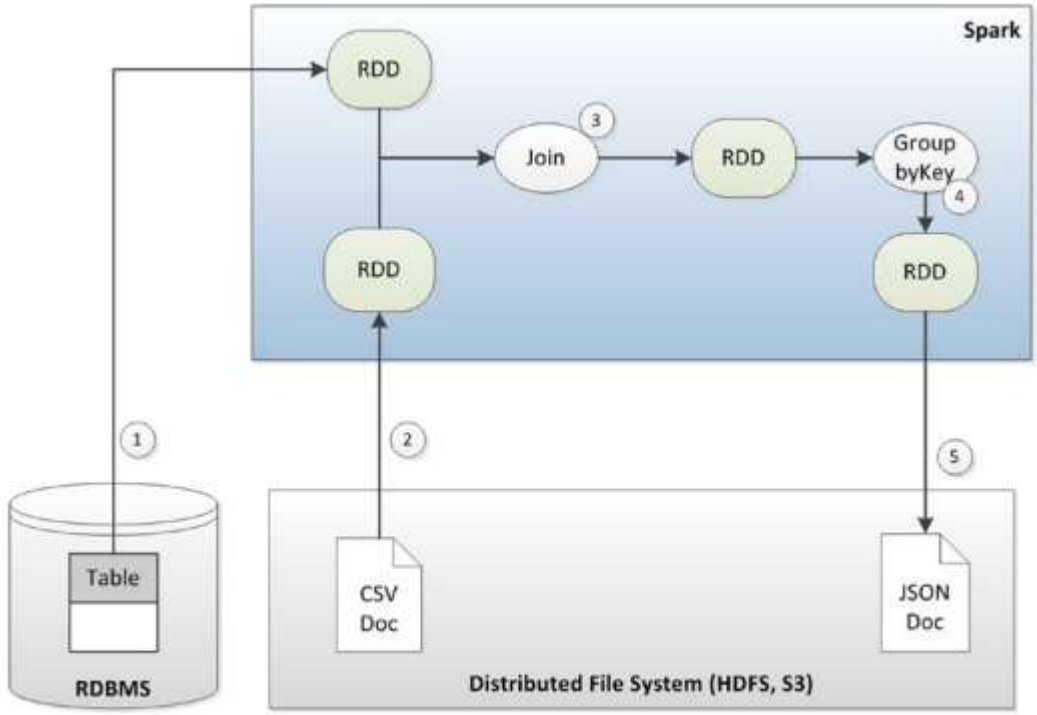

Figure 7-11. Elements of Spark processing

Summary of Figure 7-11:

1. Data can be loaded into RDDs from **databases** or **distributed file systems** (like HDFS).
2. Spark provides methods to operate on RDDs, like **joins** and **aggregations**.
3. Data can be stored back to disk in different formats.