Final-Evaluation-Questions

1. Zipped function

The `zip()` function in Python **combines multiple lists (or other iterables) together element by element**.

It **"zips" them up like a zipper** — side-by-side — into pairs or tuples.

```
names = ["Alice", "Bob", "Charlie"]
scores = [85, 92, 78]

zipped = zip(names, scores)
print(list(zipped))
```

You're telling Python:
"Please pair the first name with the first score, second name with second score, and so on…"

So what happens:

names	scores
"Alice"	85
"Bob"	92
"Charlie"	78

And Python combines them into:

```
[("Alice", 85), ("Bob", 92), ("Charlie", 78)]
```

2. What is the difference between UNION and UNION ALL?

Feature	`UNION`	`UNION ALL`
Removes duplicates	Yes	No

Feature	`UNION`	`UNION ALL`
Performance	Slower	Faster
Use case	Unique combined data	All combined data

3. DDL,DML,DCL,TCL Syntax

1. DDL – Data Definition Language

Explanation:
DDL stands for Data Definition Language. It includes SQL commands that are used to define and modify the structure of database objects like tables, schemas, or indexes. These commands **change the schema** of the database.

Common DDL commands:

- `CREATE`
- `ALTER`
- `DROP`
- `TRUNCATE`

Syntax Examples:

- **CREATE**
  Creates a new table.

```
CREATE TABLE employees (
    id INT PRIMARY KEY,
    name VARCHAR(100),
    salary DECIMAL(10, 2)
);
```

- **ALTER**
  Modifies the structure of an existing table.

```
ALTER TABLE employees ADD department VARCHAR(50);
```

- **DROP**
  Deletes a table permanently.

```
DROP TABLE employees;
```

- **TRUNCATE**
  Deletes all records from a table, but keeps the structure.

```
TRUNCATE TABLE employees;
```

2. DML – Data Manipulation Language

Explanation:
DML stands for Data Manipulation Language. These statements are used to **insert, update, delete, or retrieve data** from existing tables. DML does not affect the structure of the table, only the content.

Common DML commands:

- `SELECT`
- `INSERT`
- `UPDATE`
- `DELETE`

Syntax Examples:

- **SELECT**
  Retrieves data.

```
SELECT * FROM employees;
```

- **INSERT**
  Adds new data.

```
INSERT INTO employees (id, name, salary) VALUES (1, 'Alice', 50000.00);
```

- **UPDATE**
  Modifies existing data.

```
UPDATE employees SET salary = 55000.00 WHERE id = 1;
```

- **DELETE**
  Removes data.

```
DELETE FROM employees WHERE id = 1;
```

3. DCL – Data Control Language

Explanation:
DCL stands for Data Control Language. These commands are used to **control access to data** in the database. They are mainly used to give or take away user privileges.

Common DCL commands:

- `GRANT`
- `REVOKE`

Syntax Examples:

- **GRANT**
  Gives permissions to a user.

```
GRANT SELECT, INSERT ON employees TO user1;
```

- **REVOKE**
  Removes permissions from a user.

```
REVOKE INSERT ON employees FROM user1;
```

4. TCL (Transaction control language)

TCL commands manage the execution of transactions, ensuring data consistency and integrity when multiple operations happen together.

Key Commands:

1. **COMMIT**

- Saves all changes made in the current transaction permanently to the database.
- After `COMMIT`, changes cannot be undone.

Example:

```
UPDATE employees SET salary = 60000 WHERE id = 1;
COMMIT;
```

2. **ROLLBACK**

- Undoes all changes made since the last commit or the start of the transaction.
- Used to cancel operations if something goes wrong.

Example:

```
UPDATE employees SET salary = 60000 WHERE id = 1;
ROLLBACK;
```

3. **SAVEPOINT**

- Creates a checkpoint within a transaction.
- Allows partial rollback to this point without undoing the entire transaction.
  Example:

```
SAVEPOINT sp1;
-- some operations
ROLLBACK TO sp1;
```

4. Primary, foreign, composite

1. Primary Key

- A **Primary Key** is a column (or a set of columns) in a table that **uniquely identifies each row**.
- It **cannot contain NULL values** and must be unique for every record.
- Each table can have **only one primary key**.

Example:

```
CREATE TABLE employees (
    emp_id INT PRIMARY KEY,
    name VARCHAR(100),
    salary DECIMAL(10,2)
);
```

2. Foreign Key

- A **Foreign Key** is a column (or columns) in one table that **refers to the primary key in another table**.

Example:

```
CREATE TABLE orders (
    order_id INT PRIMARY KEY,
    emp_id INT,
    FOREIGN KEY (emp_id) REFERENCES employees(emp_id)
);
```

3. Composite Key

- A **Composite Key** is a **primary key made up of two or more columns** together that uniquely identify a row.
- Used when a single column isn't enough to guarantee uniqueness.

Example:

```
CREATE TABLE order_items (
    order_id INT,
    product_id INT,
    quantity INT,
```

```
    PRIMARY KEY (order_id, product_id)
);
```

5. What is normalization and its types?

Organizing the tables and columns in a way that minimizes redundancy and dependency

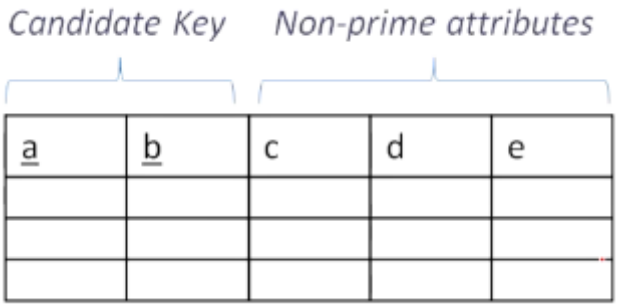
Anomalies

- Insert
  - Unable to insert customer before invoice is created
- Update
  - Customer contact details with each invoice, need to u
- Delete
  - customer details are lost when invoice is deleted

First Normal Form(1NF)

- Attributes/Columns are uniquely named
- No duplicate rows

Second Normal Form (2NF)



- Every non-prime attribute attribute of the table is dependent on the complete key (Whole of the candidate key)
- Non prime attribute is an attribute that is not part of any candidate key of the table

Example

PARTS SUPPLY

Supplier Id	Part Id	Part Name	Quantity
supplier1	BA	Base Assembly	10
supplier1	CS	Crank Shaft	5
supplier2	BA	Base Assembly	40
supplier2	BU	Battery Unit	35
supplier3	BA	Base Assembly	8
supplier3	CS	Crank Shaft	2
supplier3	BU	Battery Unit	12

- Here there are 2 candidate keys (underlined)
  - supplier id
  - part id
- The non prime attributes here are
  - Part name
  - Quantity
- 2nf says that every non prime attribute is dependent on the complete key (both the candidate keys)
- Lets check each non prime attribute one by one
  - **Part name**
    - Part name is determined by Part ID
    - But its not determined by supplier ID as well
    - So its partially dependent on the complete key
  - **Quantity**
    - Quantity is determined by the combination of supplier ID and quantity
- Transforming it into 2NF
  - We need to split into 2 tables where 2nf is satisfied

PARTS SUPPLY

Supplier Id	Part Id	Quantity
supplier1	BA	10
supplier1	CS	5
supplier2	BA	40
supplier2	BU	35
supplier3	BA	8
supplier3	CS	2
supplier3	BU	12

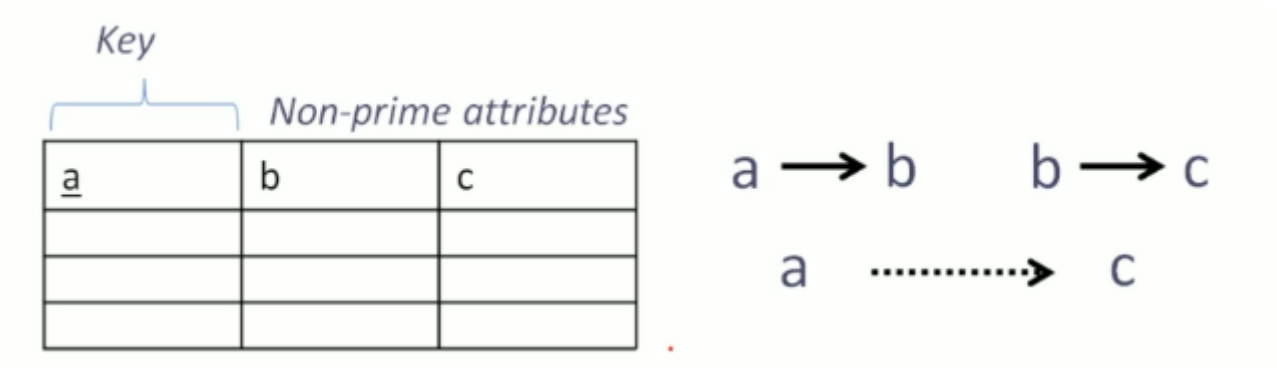
PARTS

Part Id	Part Name
BA	Base Assembly
CS	Crank Shaft
BU	Battery Unit

-
- Now quantity is determined by combination of supplier ID and part ID
- Part name is determined by Part ID
- Both tables follow 2nf

Third Normal Form (3NF)

- Every Non-Prime Attribute of the table is directly dependent on the key
- Non transitively dependent on the key



Example

SALES

Order No	Customer	Product	Coupon Code	Discount(%)	Net Amount
ORD1001	Alice Cooper	Video Camera	NEWYEAR40	40	600.00
ORD1002	Barbara Park	Keyboard	NEWYEAR40	40	30.00
ORD1003	Cynthia Nixon	LCD Monitor	SUMMER30	30	70.00
ORD1004	Mohan Lal	Mobile phone	SUMMER30	30	350.00
ORD1005	Zoya Afrose	Hard Disk	SPRING25	25	450
ORD1006	Steve Smith	Printer	SPRING25	25	150.00
ORD1007	Barbara Park	Network Router	NOCOUPON	00	45.00

- Here our key is Order No, other columns are non prime attributes

- Here Discount depends on Coupon Code
- And coupon code depends on Order number
- So this is Transitively dependent
    - Discount -> Coupon Code -> Order Number
    - This is not allowed in 3NF
- We need to split the tables such that the transitive dependencies are removed
- We need a separate table for the columns in the transitive dependency
    - The columns are Coupon code and discount

COUPON

Coupon Code	Discount(%)
NEWYEAR40	40
SUMMER30	30
SPRING25	25
NOCOUPON	00

SALES

Order No	Customer	Product	Coupon Code	Net Amount
ORD1001	Alice Cooper	Video Camera	NEWYEAR40	600.00
ORD1002	Barbara Park	Keyboard	NEWYEAR40	30.00
ORD1003	Cynthia Nixon	LCD Monitor	SUMMER30	70.00
ORD1004	Mohan Lal	Mobile phone	SUMMER30	350.00
ORD1005	Zoya Afrose	Hard Disk	SPRING25	450
ORD1006	Steve Smith	Printer	SPRING25	150.00
ORD1007	Barbara Park	Network Router	NOCOUPON	45.00

- Now the Tables follow 3NF

BCNF

Boyce-Codd Normal Form(BCNF)

- Developed by Raymond F. Boyce and E.F. Codd
- A table is in BCNF if every determinant is a candidate key

- Determinant is the value which can determine other attributes

PRODUCT SPECIALIST ASSIGNMENT

Customer Id	Product Id	Specialist Id	Date	Status
C10001	LED Television	Peter	01-Dec-2014	Active
C10002	Personal Computer	Reshmi	03-Jan-2014	Active
C10002	Network Router	Mallik	11-Feb-2014	Active
C10003	Mobile Phone	Roonie	07-Nov-2014	Inactive
C10005	LED Television	Peter	17-Oct-2014	Inactive
C10006	Network Router	Mallik	19-Mar-2014	Active

- Here the determinants are
- CustomerID, SpecialistID,ProductId are deteminants
- Combination of customerid, productid and customerid,specialist ID can be keys
- We need to split the tables

PRODUCT SPECIALİST

Specialist Id	Product Id	Specialization Level
Peter	LED Television	L1
Reshmi	Personal Computer	L1
Mallik	Network Router	L2
Roonie	Mobile Phone	L1

PRODUCT SPECIALIST ASSIGNMENT

Customer Id	Specialist Id	Date	Status
C10001	Peter	01-Dec-2014	Active
C10002	Reshmi	03-Jan-2014	Active
C10002	Mallik	11-Feb-2014	Active
C10003	Roonie	07-Nov-2014	Inactive
C10005	Peter	17-Oct-2014	Inactive
C10006	Mallik	19-Mar-2014	Active

1. **Students (ID, Name, Email)**
2. **BorrowedBooks (StudentID, ISBN, BorrowDate, ReturnDate)**

This way, the **BorrowedBooks** table links students and books without storing duplicate data.

6. What is COALESCE function

```
SELECT COALESCE(NULL, NULL, 'apple', 'banana');
```

- Returns the first value that is **not NULL** among the arguments.
- If all values are NULL, it returns NULL.

7. What is Correlated and Non Correlated subqueries

A **subquery** is a query inside another query (usually inside `WHERE`, `FROM`, or `SELECT`).

1. Non-Correlated Subquery

- The subquery **can run independently** on its own.
- It does **not depend on the outer query**.
- The outer query uses the result of the subquery.

Example:

```
SELECT emp_name
FROM employees
WHERE dept_id IN (
    SELECT dept_id FROM departments WHERE location = 'Mumbai'
);
```

- The inner query runs once, returns a list of `dept_id`s.
- Outer query uses that list to filter employees.

2. Correlated Subquery

- The subquery **depends on each row of the outer query**.
- It runs **once for every row** processed by the outer query.
- References columns from the outer query inside the subquery.

Example:

```
SELECT emp_name, salary
FROM employees e1
WHERE salary > (
    SELECT AVG(salary)
    FROM employees e2
```

```
    WHERE e1.dept_id = e2.dept_id
);
```

- For each employee (`e1`), the subquery calculates the average salary of their department (`e2`).
- So subquery is executed repeatedly, once per outer row.

8. What are the ACID Properties

1. Atomicity

- Means **“all or nothing”**.
- A transaction must be completed fully, or if any part fails, **everything is rolled back**.
- No partial changes are saved.
  Example: If you transfer money between accounts, either both debit and credit happen or none at all.

2. Consistency

- Ensures that a transaction brings the database from one **valid state** to another.
- All rules, constraints, and validations must be satisfied after the transaction.
  Example: After a transfer, total money in accounts should remain the same.

3. Isolation

- Each transaction runs **independently**, without interference from other concurrent transactions.

4. Durability

- Once a transaction is **committed**, its changes are permanent.
- Changes survive system crashes or failures.
- *Example:* After you commit your changes, they will remain saved even if the power goes out.

9. What is Sharding? What is Partitioning? Whats the difference between both?

Partitioning in Big Data

- Partitioning means **dividing large datasets into smaller chunks**, often based on a key or range, to enable **parallel processing** and **faster queries**.
- In systems like Hadoop or Spark, data is partitioned across different blocks or nodes for distributed storage and computation.
- Partitioning helps reduce the data scanned for queries by focusing on relevant partitions.
  Example:
  In Spark, a large dataset can be partitioned by date, so operations run on only the needed date partitions instead of the entire dataset.

Sharding in Big Data

- Sharding is **horizontal splitting of data** across **multiple independent database systems or clusters**.
- It distributes the workload and storage across different servers or clusters to handle very large-scale data and high throughput
- In Big Data, sharding helps in scaling out systems beyond the capacity of a single cluster.
  Example:
  A user database in a big application might shard users by region or user ID ranges across different database clusters to balance load and storage.

Key Difference

Aspect	Partitioning	Sharding
Data Division	Splitting data into partitions within a cluster or storage system	Splitting data across multiple independent clusters or database systems
Purpose	Optimize processing and query speed within a cluster	Scale out data storage and workload across many servers/clusters
System Level	Managed within distributed computing frameworks like Hadoop, Spark	Managed at the database or cluster orchestration level
Example	Partitioning HDFS files by date or category	Sharding user data across multiple database clusters

10. What is Data bricks

- Databricks is a cloud-based platform built on Apache Spark that helps teams process big data and build machine learning models efficiently.
- It offers a collaborative workspace, handles cluster management automatically, and supports multiple languages like Python and SQL. It also integrates Delta Lake to ensure data reliability and performance.
- Delta Lake is a storage layer that makes data lakes more reliable and consistent by adding ACID transaction support and schema enforcement.
- It lets you handle large-scale data with the ability to safely read and write concurrently, and also provides time travel to access historical data versions. This improves data quality and pipeline reliability compared to traditional data lakes.

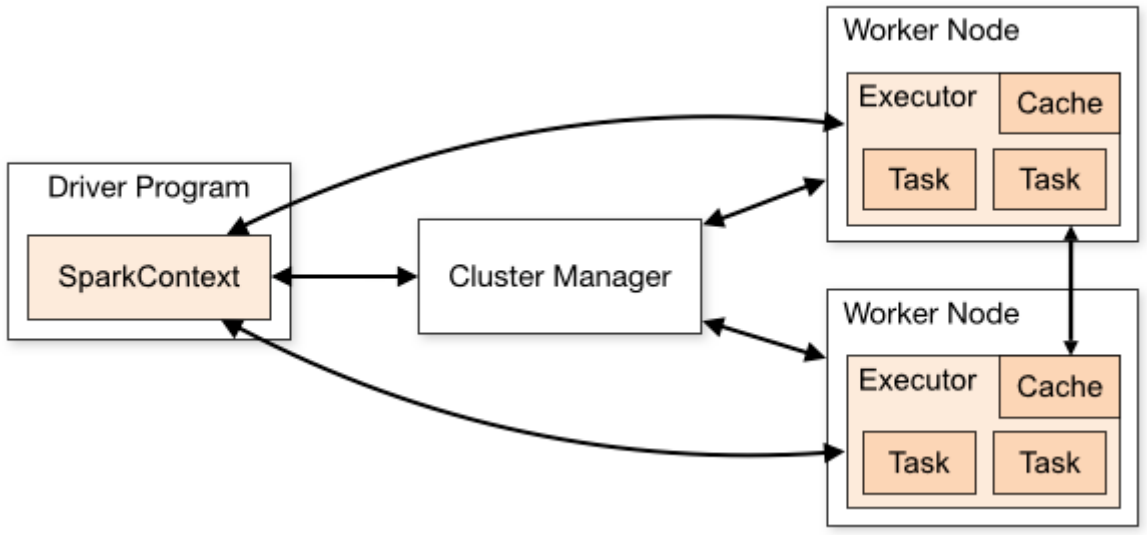
11. What is Apache Spark? How is it different from hadoop mapreduce?

- Apache Spark is an open-source big data processing engine. It helps you process large amounts of data really fast by distributing the work across many computers (called a cluster).
- It lets you do **data analysis, machine learning, streaming data processing**, and more.
- Spark works mainly in-memory, which means it keeps data in the computer's RAM while working. This makes it much faster than reading/writing data from disk repeatedly.

Aspect	Apache Spark	Hadoop MapReduce
Speed	Much faster because it uses in-memory processing (RAM)	Slower because it reads/writes data to disk after every step
Ease of use	Has easy-to-use APIs in Java, Scala, Python, and R	Programming is more complex, mostly Java-based
Processing types	Supports batch, streaming, interactive, and machine learning workloads	Mostly batch processing
Data caching	Can cache data in memory to reuse across operations	No caching, writes intermediate results to disk
Fault tolerance	Uses lineage info to recompute lost data	Writes data to disk after each step for fault recovery

Aspect	Apache Spark	Hadoop MapReduce
Ecosystem	Has built-in libraries for SQL, ML, graph processing	Part of Hadoop ecosystem but limited libraries

12. Explain the architecture of apache spark.



1. Driver Program

- This is the main program that you write and run.
- It **creates a SparkContext** which is the entry point to Spark.
- The driver is responsible for:
  - Converting your code into tasks.
  - Scheduling these tasks on worker nodes.
  - Collecting results back.

  Think of the driver as the **master brain** controlling the whole job.

2. Cluster Manager

- It manages the cluster of worker machines.
- Spark can use different cluster managers like:
  - **Standalone** (Spark's own manager),
  - **YARN** (used in Hadoop clusters),
  - **Mesos**,

    - or **Kubernetes**.
- The cluster manager allocates resources (CPU, memory) to Spark applications.

3. Worker Nodes (Executors)

- These are the actual computers/nodes where your tasks run.
- Each worker node runs one or more **Executors**.
- Executors:
    - Run the tasks assigned by the driver.
    - Perform computation and data storage.
    - Keep data in memory if needed (for fast processing).
    - Send results back to the driver.

4. Tasks

- Tasks are small units of work created by splitting your job.
- The driver sends tasks to executors on worker nodes to execute in parallel.

How It Works — Step by Step:

1. You write a Spark program and start the **Driver**.
2. Driver asks the **Cluster Manager** to get resources.
3. Cluster Manager assigns **Executors** on **Worker Nodes**.
4. Driver breaks your job into **Tasks**.
5. Executors run these tasks in parallel.
6. Results are sent back to the Driver.
7. Driver combines results and shows the final output.

13. What is an RDD in spark? What are its features?

- It’s the **core data structure** in Apache Spark.
- An **RDD** is a way for Spark to store and work with big data by **splitting it into smaller parts**, **spreading it across multiple computers**, and **processing it in parallel**.
- It is **fault-tolerant**, **read-only**, and supports operations like **map**, **filter**, and **reduce** to transform or get results from data.

- **Resilient** → If something fails, Spark can rebuild the data.
- **Distributed** → Data is split and spread across many computers.
- **Dataset** → It's a collection of data (like a big list or table).

14. Differentiate between RDD, Dataframe, Dataset

Type	Definition
RDD	A **low-level**, **immutable**, distributed collection of objects. Good for fine-grained control over data.
DataFrame	A **distributed table** with rows and named columns, like a database or Excel sheet. Higher-level API built on top of RDD.
Dataset	A combination of **RDD + DataFrame**. It provides the **structure of DataFrame** and **type safety** like RDD. (Available only in Scala & Java, not Python.)

15. What is spark context and spark session?

- `SparkContext` is the **entry point to Spark's low-level API** (RDDs).
  - It connects your program to the **Spark Cluster**, handles resource allocation, and lets you create RDDs.
- `SparkSession` is the **entry point to Spark's high-level APIs** (DataFrame, SQL, etc).
  - It replaces older components like `SQLContext` and `HiveContext`.

```
from pyspark.sql import SparkSession

# Create a Spark session (recommended)
spark = SparkSession.builder.appName("MyApp").getOrCreate()

# Get SparkContext from it
sc = spark.sparkContext

# Now you can use both
rdd = sc.parallelize([1, 2, 3])
```

```
df = spark.createDataFrame([(1, "A"), (2, "B")], ["id", "name"])
```

16. Explain in memory computing in spark?

- **In-memory computing** means Spark stores data in **RAM (memory)** instead of reading/writing from the **hard disk** again and again.
- This makes data processing **much faster** compared to traditional systems like Hadoop MapReduce, which rely on disk at every step.
- Reading from **RAM** is **10–100x faster** than reading from **disk**.
- Big data jobs involve many steps. If Spark kept going to disk each time, it would be slow.

17. What are broadcast variables and accumulators in spark?

1. Broadcast Variables

- A **broadcast variable** is used to **share read-only data** (like a big lookup table or a config) **across all worker nodes**, **without sending it again and again**.
- When you need to use the **same large data** in multiple tasks, sending it repeatedly to each executor is wasteful.
- **Broadcast variables solve this** by sending the data **just once** and storing it **on each worker**.

Example in PySpark:

```
broadcast_var = sc.broadcast({"IN": "India", "US": "USA"})

rdd = sc.parallelize(["IN", "US", "IN"])
rdd.map(lambda code: broadcast_var.value[code]).collect()
# Output: ['India', 'USA', 'India']
```

2. Accumulators

- An **accumulator** is a variable used to **count or sum values across tasks**, **write-only** from workers and **readable only by the driver**.

- When you want to **track metrics** like:
  - Number of blank lines in a file,
  - Total error count in a log file, etc.
- You **can't use normal variables**, because each task runs separately.
- **Accumulators** allow **safe, parallel aggregation**.

Example in PySpark:

```
acc = sc.accumulator(0)

def count_blank_lines(line):
    if line == "":
        acc.add(1)

rdd = sc.parallelize(["", "Hello", "", "World"])
rdd.foreach(count_blank_lines)

print("Blank lines:", acc.value)  # Output: 2
```

18. What is the role of driver and executor in spark?

- Driver
  - The **Driver** is the **main controller** of your Spark program.
  - What it does
    - Runs your **main program** (the one you write).
    - Creates the **SparkSession** or **SparkContext**.
    - **Plans the work**: divides your job into tasks.
    - **Sends tasks** to the Executors.
    - **Collects the results** from Executors.
- Executor
  - An **Executor** is a **worker process** that runs on each machine in the cluster.
  - **Executes the tasks** assigned by the Driver.
  - **Processes the data** (e.g., maps, filters, reduces).
  - Sends **results back** to the Driver.

- Workflow
  - You run a Spark program → Driver starts.
  - Driver talks to the **Cluster Manager** to get machines.
  - Executors are launched on those machines.
  - Driver splits your code into **tasks**.
  - **Executors run the tasks** and return results to Driver.

19. Explain sparks cluster modes: Local, standalone, YARN, and Mesos

- Spark can run on **different environments (cluster managers)** that manage **resources and execution**.
- These modes decide **where** and **how** Spark runs your application.

1. Local Mode

- Spark runs on **a single machine**, like your laptop.
- Best for **learning**, **testing**, and **small datasets**
- Doesn't need a cluster manager

2. Standalone Mode

- Works across multiple machines
- Requires setting up **master** and **worker** nodes

3. YARN Mode

- Can access **HDFS**, Hive, and Hadoop data easily
- Two types: **client mode** and **cluster mode**

4. Mesos Mode

- Spark runs on **Apache Mesos**, a general-purpose cluster manager.
- More **flexible** and **multi-tenant**
- Harder to set up than YARN or Standalone
- Can run different types of apps (not just Spark)

20. What is Spark SQL? What are its benefits?

- **Spark SQL** is a module in Apache Spark that lets you work with **structured data** using **SQL queries** or **DataFrame API**.
- It allows you to query data using **SQL language** just like in traditional databases.
- You can work with data from various sources like JSON, Parquet, Hive tables, and more.
- It integrates SQL queries with Spark's distributed processing.
- Benefits
  - You can use familiar SQL syntax to query big data.
  - Works seamlessly with Spark programs (RDDs, DataFrames).
  - Allows mixing SQL queries with complex Spark programs easily.

21. What are the different file formats supported by spark?

Format	Description	Use Case / Notes
Text	Plain text files	Simple data, logs, CSV without header
CSV	Comma-separated values	Most common tabular data format
JSON	JavaScript Object Notation	Semi-structured data, easy to parse
Parquet	Columnar storage format	Fast, efficient for big data analytics
ORC	Optimized Row Columnar (Hadoop format)	Efficient columnar format, good for Hive
Avro	Row-based serialization system	Schema evolution support
SequenceFile	Hadoop binary key-value pairs	Older Hadoop format
XML	eXtensible Markup Language	Used in some enterprise data systems

22. What is caching and persistence in spark?

- **Caching** means **storing an RDD or DataFrame in memory** so that it can be reused quickly without recomputing or reading from disk again.

    - When you cache data, Spark keeps it in RAM by default.
    - Useful when you **reuse the same dataset multiple times** (e.g., in iterative algorithms).
- **Persistence** is like caching but more flexible — you can choose **where and how to store the data** (memory, disk, or both).
    - You specify a **storage level** when persisting:
        - Memory only
        - Memory and disk
        - Disk only

23. Explain how spark handles fault tolerance

- Fault tolerance means Spark can **recover and keep working even if some machines fail or data is lost** during processing.
- Spark uses a concept called **RDD lineage** (or **lineage graph**) to handle failures.
- What is RDD Lineage
    - Spark **keeps track of the sequence of operations (transformations)** used to create an RDD.
    - This record of transformations is called the **lineage**.
    - If a partition (chunk) of data is lost (due to worker failure), Spark **recomputes it by reapplying those transformations on the original data**.

24. What are narrow and wide transformations in Spark?

What are Transformations in Spark?

Transformations are operations that **create a new RDD/DataFrame** from an existing one, like `map()`, `filter()`, or `reduceByKey()`.

Narrow Transformation Example (like `map`):

Imagine your data is split into 3 parts (partitions):

Partition 1	Partition 2	Partition 3
1, 2	3, 4	5, 6

- Operation: Multiply each number by 2
- How it works:
  - Partition 1 processes 1, 2 → 2, 4
  - Partition 2 processes 3, 4 → 6, 8
  - Partition 3 processes 5, 6 → 10, 12
- **No data moves between partitions** — each partition works independently.

Wide Transformation Example (like `reduceByKey`):

Partition 1	Partition 2	Partition 3
(A,1), (B,2)	(A,3), (B,4)	(A,5), (B,6)

- Operation: Sum values by key (A, B)
- Spark needs to bring all `(A, *)` pairs to one partition and all `(B, *)` pairs to another partition.
- This causes **data to be shuffled across partitions**.
- So, data moves around the cluster to group keys together before summing.

Narrow Transformation	Wide Transformation
Each partition works on its own data (no data transfer)	Data moves between partitions (shuffle happens)
Faster and simpler	Slower because of data movement
map(),filter()	reduceByKey(), join()

25. What is the role of partitioning in spark? How does it affect performance?

- Partitioning means **dividing your data into smaller chunks called partitions**.

- Each partition is a subset of the data that Spark processes **in parallel** on different machines or CPU cores.
- It helps Spark **process data faster** by working on many partitions at the same time.
- Spark distributes partitions across executors in the cluster.
- More partitions → more parallelism (up to a limit).
- How Partitioning Affects Performance
  - Not enough parallelism → slower processing
  - Overhead from managing too many tasks → slower job and more scheduling time
  - Balanced size → maximizes parallelism & minimizes overhead
  - Choosing the right key or method reduces data shuffle during operations like join or reduceByKey, improving speed

26. What is shuffle in spark and why does it occur?

- **Shuffle** is the process where **data is redistributed across different partitions or nodes** in the cluster
- It involves **moving data over the network** between executors.
- Spark writes data to disk, transfers it across the network, and reads it again.
- Why Does Shuffle Occur?
  - Shuffle happens during **wide transformations** that require data from multiple partitions to be grouped or combined differently.
  - `reduceByKey()`
  - `groupByKey()`
  - `join()`
  - `distinct()`

27. What is the difference between map, flatmap, reducebykey and groupbykey?

1. map()

- Applies a function to **each element** in the RDD.

- Each input element produces **exactly one output element**.

Example: Multiply each number by 2.

```
rdd = sc.parallelize([1, 2, 3])
rdd2 = rdd.map(lambda x: x * 2)  # Output: [2, 4, 6]
```

2. flatMap()

- Applies a function to **each element**, but the function returns a **list (or multiple) of elements**.
- The results are **flattened** into a single list.
  Example: Split sentences into words.

```
rdd = sc.parallelize(["hello world", "how are you"])
rdd2 = rdd.flatMap(lambda x: x.split(" "))  # Output: ['hello', 'world',
'how', 'are', 'you']
```

3. reduceByKey()

- Works on **key-value pairs**.
- Combines **values with the same key** using a reduce function (like sum).
- Efficient because it **combines data locally before shuffling**.
  Example: Sum counts by key.

```
rdd = sc.parallelize([("a", 1), ("b", 1), ("a", 2)])
rdd2 = rdd.reduceByKey(lambda x, y: x + y)  # Output: [('a', 3), ('b', 1)]
```

4. groupByKey()

- Works on **key-value pairs**.
- Groups **all values with the same key** into a list.
- Less efficient because it **shuffles all values directly without local combining**.

Example: Group values by key.

```
rdd = sc.parallelize([("a", 1), ("b", 1), ("a", 2)])
rdd2 = rdd.groupByKey().mapValues(list)  # Output: [('a', [1, 2]), ('b',
```

```
[1])]
```

Quick Summary

Function	Input Type	Output	Notes
map()	Any RDD	One output per input	One-to-one transformation
flatMap()	Any RDD	Zero or more outputs per input	Flattens the output lists
reduceByKey()	Pair RDD (key, value)	Combined values by key	More efficient aggregation
groupByKey()	Pair RDD (key, value)	List of values per key	Less efficient, causes more shuffle

28. What is big data

- **Big Data** refers to **extremely large and complex datasets** that are difficult or impossible to process using traditional data processing tools.
- These datasets are so big, fast, or varied that regular databases and software can’t handle them efficiently.

29. What are the 5 V's of Big Data

1. **Volume**
   - The **amount** of data generated is huge — from terabytes to petabytes and beyond.
2. **Velocity**
   - The speed at which data is **generated and processed** — often in real-time or near real-time.
3. **Variety**
   - Different **types and formats** of data — structured (tables), semi-structured (JSON, XML), and unstructured (images, videos, text).
4. **Veracity**

- The **trustworthiness and quality** of the data — data can be messy, incomplete, or uncertain.

5. **Value**
    - The **usefulness** of data — turning raw data into meaningful insights and business value.

30. How is big data different from traditional data

Feature	Traditional Data	Big Data
Volume	GB or TB	TB, PB, and beyond
Velocity	Slow, batch processing	Real-time / streaming
Variety	Structured data only	Structured, semi-structured, unstructured
Tools	RDBMS (Relational Databases)	Hadoop, Spark, Hive, Kafka, and others

31. Main challenges of handling big data

Challenge	Explanation
Storage and Scalability	Storing huge amounts of data and scaling systems as data grows
Speed	Processing data quickly, especially in real-time
Data Variety and Integration	Managing different types of data and combining them effectively
Data Veracity	Ensuring data quality, accuracy, and reliability
Data Security	Protecting data from unauthorized access and breaches

32. What is Hadoop

Hadoop is an open-source framework that allows you to **store and process huge amounts of data** across many computers (a cluster) using simple programming models.

- **Distributed Storage:** Uses **HDFS (Hadoop Distributed File System)** to store big data across multiple machines.
- **Distributed Processing:** Uses **MapReduce** to process data in parallel across the cluster.
- **Scalable and Fault-Tolerant:** Can easily add more machines and recover from hardware failures.

33. What are the Hadoop components

1. **HDFS (Hadoop Distributed File System)**
    1. HDFS is the storage system in Hadoop. It splits large files into smaller blocks and stores them across many machines in the cluster.
    2. It also keeps multiple copies (replicas) of each block to prevent data loss if a machine fails.
2. **YARN (Yet Another Resource Negotiator)**
    1. YARN is the resource manager. It controls and allocates cluster resources like CPU and memory to various applications running on the cluster.
    2. It decides how much resource each job gets and manages job scheduling.
3. **MapReduce**
    1. MapReduce is the processing framework. It breaks down big data jobs into two phases:
    2. **Map** (which processes and filters data)
    3. **Reduce** (which aggregates and summarizes results). This lets Hadoop process data in parallel across many machines.

34. What is HDFS?

- **HDFS (Hadoop Distributed File System)** is the storage system used by Hadoop to **store big data across multiple machines**.
- **Distributed:** Splits files into blocks and stores them on different machines in the cluster.
- **Fault-Tolerant:** Keeps multiple copies (replicas) of each block. If one copy is lost, another is used.
- **Scalable:** Can easily add more machines to store more data.

- Example
  - If you upload a 1 GB file to HDFS, it will:
  - Split it into blocks (say, 128 MB each),
  - Store the blocks across different machines,
  - And make 3 copies of each block (by default) to avoid data loss.

35. How does HDFS work?

- **File is split into blocks**
  - When you store a file in HDFS, it gets **split into large blocks** (usually 128 MB or 256 MB).
- **Blocks are stored across machines**
  - These blocks are **distributed across multiple machines** in the cluster, called **DataNodes**.
- **Each block is replicated**
  - Every block is **copied (replicated)** 3 times by default, so if one machine fails, your data is still safe.
- **NameNode keeps track**
  - There's a special machine called the **NameNode** that **keeps a record of which block is stored on which DataNode**. It acts like a **file manager** or **index**.
- **Client reads/writes data**
  - When you **read a file**, the client contacts the NameNode to find out where the blocks are, and then reads them directly from the DataNodes.
  - When you **write a file**, the client sends blocks to DataNodes, and the NameNode keeps track of where they're stored.

36. What is Mapreduce?

MapReduce is a programming model used in Hadoop to **process large amounts of data** by **splitting the work into two steps: Map and Reduce**.

- **Map Step**

    - Takes raw input data.
    - Processes it and converts it into key-value pairs.
- **Shuffle and Sort (in the background)**
    - Groups all similar keys together (e.g., all data related to the word "apple").
- **Reduce Step**
    - Takes each group of key-value pairs.
    - Combines or aggregates them to produce the final result.

Example

Let’s say you want to count how many times each word appears in a big document.

- **Map step**:
    - Takes each line → splits it into words → emits (word, 1)
        - Example: `"hello world"` → `("hello", 1), ("world", 1)`
- **Reduce step**:
    - Combines all values for each word
    - `("hello", [1, 1, 1])` → `("hello", 3)`

37. What is YARN in Hadoop

- **YARN** stands for **Yet Another Resource Negotiator**.
- It is the **resource manager** in the Hadoop ecosystem.
- YARN manages and allocates **CPU, memory, and other resources** across the cluster for different big data applications (like MapReduce, Spark, Hive, etc.).
- What Does YARN Do?
    - **Allocates resources** (CPU, RAM) to applications running on the cluster.
    - **Schedules jobs** and decides when and where they should run.
    - **Monitors applications** to keep them running smoothly.
    - Supports **multiple big data tools** to run side-by-side.

38. Difference between Hadoop 1 and Hadoop 2

Hadoop 1.x

- Uses **MapReduce** as **both processing and resource manager**.
- The **JobTracker** handles everything: job scheduling, resource allocation, and monitoring.
- Can run **only MapReduce jobs**.
- **Less scalable** and more prone to bottlenecks.

Hadoop 2.x

- Introduces **YARN** (Yet Another Resource Negotiator) as a **separate resource manager**.
- **YARN replaces JobTracker** for managing cluster resources.
- Can run **multiple types of applications**, not just MapReduce (like Spark, Hive, etc.).
- **More scalable, flexible, and efficient**.

39. What tools are used for data ingestion in bigdata

1. Batch Ingestion

- Data is collected and loaded in **bulk at scheduled intervals** (e.g., hourly, daily).
- **Tool: Apache Sqoop**
  - Used to **import data from relational databases** (like MySQL, Oracle) into Hadoop.
  - Helps move structured data into HDFS, Hive, or HBase.
  - Ideal for **static or historical data**.

2. Streaming Ingestion

- Data is collected and ingested **in real-time or near real-time**, as it is generated.
- **Tool: Apache Kafka**
  - A **distributed streaming platform**.
  - Handles high-throughput, fault-tolerant real-time data pipelines.
  - Ideal for data from sensors, web clicks, logs, etc.
- **Tool: Apache Flume**
  - Specially designed to **collect and transport log data**.
  - Good at aggregating logs from multiple sources and sending them to HDFS or HBase.

40. Compare apache flume and apache kafka?

Apache Flume

- Designed mainly for **collecting, aggregating, and moving log data**.
- Supports **streaming log files** from many sources like web servers.
- Built with **HDFS integration** in mind.
- Has built-in **sources, channels, and sinks** (plug-and-play components).
- Best when your main use case is **log data ingestion into Hadoop**.

Apache Kafka

- A **general-purpose distributed messaging system**.
- Handles **high-throughput** data streams in real time.
- Suitable for **many types of data** — logs, events, metrics, app data, etc.
- More **scalable, fault-tolerant**, and used widely in big data pipelines.
- Supports multiple consumers, durable storage, and message replay.

Feature	Apache Flume	Apache Kafka
Use Case	Log data ingestion	General-purpose data streaming
Sources	Mostly logs	Logs, app data, sensors, anything
Scalability	Moderate	Very high
Durability	Limited (in-memory channels may lose data)	High (writes to disk, supports replay)
Flexibility	Predefined architecture (source-channel-sink)	More flexible and extensible
Data Storage	Temporary (until sent to sink)	Persistent (can retain data for days)

- **Flume** = Great for **logs → Hadoop**.
- **Kafka** = Great for **all kinds of streaming data**, with better performance and flexibility.

41. What is SQOOP? When would you use it?

- **Sqoop** (SQL-to-Hadoop) is a tool used to **transfer data between relational databases and Hadoop**.
- It helps you move **structured data** (like from MySQL, Oracle, PostgreSQL) into Hadoop's ecosystem — HDFS, Hive, or HBase — and also **export data back** from Hadoop to databases.
- When Would You Use Sqoop?
  - When you need to **import data** from a traditional **RDBMS** (like MySQL, Oracle) into:
    - HDFS for storage
    - Hive for querying
    - HBase for NoSQL access

Examples

Let's say you have sales data in a MySQL database.
You can use Sqoop to:

1. Import it into Hadoop to analyze it using Hive or Spark.
2. After processing, export the results back to MySQL for business dashboards.

42. What is Hive? How does it work?

- **Apache Hive** is a **data warehouse tool** built on top of Hadoop.
- It lets you **query and analyze big data using SQL-like language** called **HiveQL**.
- How does it work?
  - **You write a HiveQL query** (like `SELECT * FROM sales WHERE amount > 1000`).
  - Hive **translates** your query into **MapReduce jobs** (or Spark/Tez depending on the engine).
  - These jobs run on the **Hadoop cluster**, process the data, and return the result.
  - Data is usually stored in **HDFS** in table-like formats (like CSV, Parquet, ORC).
- When to use Hive?
  - When you want to **query big data with familiar SQL syntax**.
  - When you need to do **batch processing or reporting** on large datasets.
  - When working with **data stored in HDFS**.

43. Where is Metadata stored in Hive

- **Hive stores metadata in a special database called the *Metastore***.
- The **Metastore** is a central place where Hive keeps information about:
  - Tables and their names
  - Columns and data types
  - Table locations in HDFS
  - Partitions
  - SerDes (serialization/deserialization details)
  - File formats (like ORC, Parquet)
- By default, Hive uses **Apache Derby** (a simple embedded database) for the metastore.
- In production, it's common to use **MySQL**, **PostgreSQL**, or **other RDBMS** for better performance and multi-user access.

44. What is Hive Indexing

- **Hive Indexing** is a technique to **speed up query processing** by creating an index on one or more columns of a Hive table.
- An index helps Hive **quickly locate rows** matching query conditions without scanning the entire table.
- How Hive Indexing works?
  - You create an index on a column (or set of columns) in a table.
  - Hive builds a separate **index table** that stores the values of the indexed column along with pointers to the original data.
  - When you run a query with a filter on the indexed column, Hive can use the index to find matching rows faster.

45. What is OLAP and OLTP

OLTP (Online Transaction Processing)

- Focuses on **managing daily transactional data**.
- Examples: Bank transactions, order entries, booking systems.

- Handles **many short, simple queries** (insert, update, delete).
- Requires **fast query processing** and high concurrency.
- Data is **highly normalized** to avoid redundancy.
- Used for **real-time operations**.

OLAP (Online Analytical Processing)

- Focuses on **analyzing large volumes of historical data** for decision making.
- Examples: Sales analysis, business intelligence reports, data mining.
- Handles **complex queries** involving aggregations and summarizations.
- Usually **read-heavy** with fewer writes.
- Data is often **denormalized** for faster query performance.
- Used for **strategic analysis and reporting**.

46. What is view? What is the purpose of creating view?

Whats a view?

A **view** is a **virtual table** in a database that is created by saving a **SQL query**.
It **does not store data physically**, but shows data stored in one or more tables.

Purpose of a view

- **Simplify complex queries:** You can save complicated joins or filters as a view and use it like a table.
- **Data security:** Restrict access by showing only specific columns or rows to users.
- **Reusability:** Use the same query logic multiple times without rewriting it.
- **Abstraction:** Hide the underlying table structure and complexity from users.

47. What is ER Model?

- **ER Model (Entity-Relationship Model)** is a way to **visually represent data and its relationships** in a database.
- It helps design the **database structure** by showing:

- **Entities** (things or objects, like Student, Car, Employee)
- **Attributes** (properties of entities, like name, age, salary)
- **Relationships** (how entities are connected, like a Student *enrolls in* a Course)

Why Use ER Model?

- Helps in **planning and designing databases** before implementation.
- Makes it easier to understand the data and connections.
- Acts as a blueprint for creating tables and relations.

48. In DBMS how many types of relationships you know? Give examples of each of these

Types of Relationships in DBMS

1. **One-to-One (1:1)**
   - Each entity in Table A is related to **only one** entity in Table B, and vice versa.
   - **Example:**
     - Each **person** has one **passport**.
     - Each passport belongs to one person.
2. **One-to-Many (1:N)**
   - One entity in Table A relates to **many** entities in Table B, but each entity in Table B relates to only one in Table A.
   - **Example:**
     - One **teacher** teaches many **students**.
     - Each student is taught by only one teacher (in this example).
3. **Many-to-Many (M:N)**
   - Entities in Table A relate to **many** entities in Table B and vice versa.
   - **Example:**
     - **Students** enroll in many **courses**.
     - Each course has many students.

49. What is SCD and its types?

1. **SCD Type 1 – Overwrite**
   - When data changes, **overwrite the old data** with new data.
   - No history is kept.
   - Example: Fixing a typo in a customer's name.
2. **SCD Type 2 – Add New Row**
   - When data changes, **add a new row** with the new data.
   - Keep old rows as history with versioning or timestamps.
   - Example: Customer changes address → old address row kept, new row added.
3. **SCD Type 3 – Add New Column**
   - Keep limited history by adding **new columns** for old values.
   - Only tracks a fixed number of changes.
   - Example: Store previous address in a separate column.
4. **SCD Type 4 – History Table**
   - Keep current data in main table, and **history in a separate table**.
   - Queries need to join both tables to get full history.
5. **SCD Type 6 (Hybrid)**
   - Combination of Types 1, 2, and 3 to track changes with both overwrite and history columns.

SCD Type	How it Works	History Maintained?	Use Case Example
Type 0	No change (data stays fixed)	No	Static data like product or country codes
Type 1	Overwrite old data	No	Fixing typos or corrections in customer info
Type 2	Add new row for each change	Yes	Track customer address or status changes over time
Type 3	Add new columns to keep limited history	Limited (few versions)	Store previous and current address in different columns
Type 4	Use separate history table	Yes	Maintain current data and history separately
Type 6	Hybrid (combination of Type 1, 2, and 3)	Yes	Complex tracking combining overwrite and history

50. Suppose you have 2 tables

- Your store is having both online and offline customers
- cust_offline: cust_name, phone_no, amount
- cust_online: cust_name, phone_no,amount
- Suppose this month end, i want to know who are the customers both online and offline, and the total amount they have purchased

```
SELECT
    cust_name,
    SUM(amount) AS total_amount
FROM (
    SELECT cust_name, amount FROM cust_online
    UNION ALL
    SELECT cust_name, amount FROM cust_offline
) AS combined
GROUP BY cust_name
```

51. Suppose in python you have a list like this

```
a = [2,3,4,5]
```

Write python code which gives square of each of these elements

```
squared = map(lambda x: x**2,a)
```

52. What is break, continue and pass in python

`break`

- Used to **exit a loop immediately** (for or while loop).

- Stops the loop completely when a certain condition is met.
  Example:

```
for i in range(5):
    if i == 3:
        break  # Exit the loop when i is 3
    print(i)
# Output: 0 1 2
```

`continue`

- Skips the **current iteration** of the loop and moves to the **next iteration**.
- Useful to skip certain values but keep looping.

Example:

```
for i in range(5):
    if i == 3:
        continue  # Skip when i is 3
    print(i)
# Output: 0 1 2 4
```

`pass`

- Does **nothing**. It's a **placeholder** when syntax requires a statement but you don't want any code to run.
- Often used when you plan to add code later.
  Example:

```
for i in range(5):
    pass  # Do nothing for now
```

53. Explain concept of datawarehouse in azure. In Azure how you are applying the concept of data warehousing

- A **Data Warehouse** is a centralized system used to **store large amounts of structured data** from multiple sources, designed for **reporting and analysis**.

- It organizes data in a way that makes it fast and easy to run complex queries and get business insights.

Data warehouse in azure

- **Azure Synapse Analytics** (formerly called Azure SQL Data Warehouse) is Microsoft's **cloud-based data warehousing solution**.
- **Data Integration:**
  - Use **Azure Data Factory** to ingest data from various sources (databases, files, streaming).
- **Storage:**
  - Store cleaned and transformed data in **Azure Synapse Analytics** or **Azure Data Lake Storage**.
- **Processing:**
  - Use **SQL pools** in Synapse to run complex analytical queries.
  - Integrate with **Spark pools** for big data processing.
- **Visualization:**
  - Connect Synapse to tools like **Power BI** for dashboards and reports.

54. Explain the process of data ingestion in azure

- Data ingestion means **collecting and importing data** from different sources into a storage or processing system.
- **Identify Data Sources**
  - Data can come from databases, files, IoT devices, applications, APIs, or streaming sources.
- **Choose Ingestion Tools**
  - **Azure Data Factory (ADF):** For batch and scheduled data movement.
  - **Azure Event Hubs:** For high-throughput streaming data ingestion.
  - **Azure IoT Hub:** For IoT device data ingestion.
  - **Azure Stream Analytics:** For real-time stream processing.
- **Ingest Data**
  - Use pipelines in Azure Data Factory to **copy and transform data** from sources to destinations.

- **Store Data**
  - Data lands in storage solutions like:
    - **Azure Data Lake Storage** (for big data files)
    - **Azure Blob Storage** (for unstructured data)
    - **Azure Synapse Analytics** (for structured data warehousing)

55. What is SLA?

- An **SLA (Service Level Agreement)** in Azure is a **commitment by Microsoft** about the **reliability and uptime** of its cloud services
- It tells you how much **availability and performance** you can expect from services like Azure Virtual Machines, Azure SQL Database, Azure Storage, etc.
- Azure SLAs usually guarantee **uptime percentages** (e.g., 99.9%, 99.95%, 99.99%).
- If Azure fails to meet the SLA, customers may get **service credits** or compensation.

56. What is cloud computing? What are the advantages of cloud computing?

- **Cloud computing** means using **remote servers on the internet** to store, manage, and process data instead of using your own local computer or server.
- It allows you to access computing resources like servers, storage, databases, and software **on-demand**, without owning physical hardware.
- Advantages
  - **Cost Savings**
    - Pay only for what you use (no upfront hardware costs).
  - **Scalability**
    - Easily scale resources up or down based on demand.
  - **Accessibility**
    - Access your data and apps from anywhere with internet.
  - **Reliability**
    - Cloud providers offer high uptime and data backup.

- **Maintenance-Free**
  - No need to manage hardware or infrastructure yourself.
- **Flexibility**
  - Wide range of services (compute, storage, AI, analytics).
- **Fast Deployment**
  - Quickly set up resources and launch applications.

57. Azure Vs AWS Vs GCP

Feature	AWS	Azure	GCP
Launch Year	2006	2010	2008
Market Share	Largest	2nd	3rd
Strengths	Most services, mature, widely used	Great for Microsoft users, hybrid solutions	Best for data, AI, and analytics
Ease of Use	Moderate	Good for Windows users	Developer-friendly
Pricing	Pay-as-you-go, sometimes expensive	Competitive	Flexible & often cheaper for data-heavy workloads
Popular Services	EC2 (virtual servers), S3 (storage), Lambda (serverless)	Virtual Machines, Azure DevOps, Active Directory	BigQuery (data analytics), Kubernetes, AI tools
Best For	All-round use	Enterprises using Microsoft tools	Startups, data projects, AI/ML workloads

58. What is Saas, Paas, Iaas

1. SaaS (Software as a Service)

- You use the software directly via browser or app.
- No need to manage servers, databases, or updates.
- Examples: Gmail, Google Docs, Zoom

2. PaaS (Platform as a Service)

- You write and deploy your code.
- The platform handles infrastructure, runtime, and scaling.
- Examples: Heroku, Google App Engine, AWS Elastic Beanstalk

3. IaaS (Infrastructure as a Service)

- You get raw virtual machines and storage.
- You manage the OS, runtime, and everything on top.
- Examples: AWS EC2, Microsoft Azure VMs, DigitalOcean

59. SAS Tokens

SAS tokens are used in **Azure Storage** to give **limited access** to resources **without sharing your storage account key**.

Key Points:

- You generate a token with **permissions**, **expiry time**, and **resource info**.
- You can give access to blobs, files, queues, or tables.
- Commonly used for secure **temporary file uploads/downloads**.
- You want to let someone upload a file to your Azure Blob Storage for 1 hour. You generate a SAS token that gives **write** permission for that specific file for 1 hour.

60. Print 1 to 10 in SQL

```sql
WITH RECURSIVE numbers AS (
    SELECT 1 AS num
    UNION ALL
    SELECT num + 1 FROM numbers WHERE num < 10
)
SELECT num FROM numbers;
```

61. Print current date in SQL

```
SELECT CURDATE();
```

62. Print Even Numbers in SQL

```
SELECT RECURSIVE numbers AS(
    SELECT 1 AS num
    UNION ALL
    SELECT num + 2 from numbers WHERE num <10
)
SELECT num FROM numbers;
```

63. Query for limit and Offset

Get first 10 rows

```
SELECT * FROM employees
LIMIT 10;
```

Get next 10 rows

```
SELECT * FROM employees
LIMIT 10 OFFSET 10;
```