1. How many types of join strategies are there in Spark?

In Spark, there are four main types of join strategies:

1. **Broadcast Join**: Used when one of the DataFrames is small enough to be broadcasted to all executor nodes.
2. **Sort Merge Join**: Used when both DataFrames are large and can be sorted and merged efficiently.
3. **Shuffle Hash Join**: Used for joining large DataFrames when the join keys are evenly distributed.
4. **Cartesian Join**: Used when you want to join every row of one DataFrame with every row of another DataFrame.

2. When using Shuffle Sort Merge Join, does the shuffling occur on the driver node or the executor node?

Shuffling occurs on the executor nodes. The driver node initiates the job, but the actual shuffling and sorting of data happen on the executors.

3. What optimization techniques have you used in Spark?

Common Spark optimization techniques include:

- **Persisting/Caching Data**: To avoid recomputation of the same data.

- **Using the `repartition` and `coalesce` methods**: For efficient partition management.
- **Broadcasting small DataFrames**: To avoid shuffling.
- **Predicate Pushdown**: Filtering data as early as possible.
- **Using proper file formats**: Such as Parquet or ORC for efficient storage and query performance.
- **Tuning Spark configurations**: Like executor memory, cores, and shuffle partitions.

4. What is a DAG in Spark, and what is its purpose?

A DAG (Directed Acyclic Graph) in Spark is a sequence of computation stages where each stage consists of a set of tasks that can be executed in parallel. The DAG scheduler in Spark is responsible for breaking down a job into stages of tasks based on shuffle boundaries and executing these stages in the right order to compute the final result. The purpose of the DAG is to optimize the execution plan for performance and fault tolerance.

5. If data can be spilled to disk, why do we encounter OOM (Out Of Memory) errors?

OOM (Out Of Memory) errors can occur in Spark even if data can be spilled to disk because:

- **Executor memory limits**: If intermediate data structures (like RDDs) are too large to fit in memory.
- **Memory leaks**: Due to inefficient code or bugs.
- **Improperly tuned memory settings**: Such as insufficient memory allocated to executors or drivers.
- **Skewed data**: Where certain partitions are significantly larger than others, causing uneven memory usage.
- **Complex transformations**: That require a large amount of intermediate data to be held in memory simultaneously.

6. How does Spark work internally?

Spark works internally using the following steps:

1. **Job Submission**: The user submits a Spark job via a SparkContext.
2. **DAG Creation**: The SparkContext creates a logical plan in the form of a DAG.
3. **Job Scheduling**: The DAG scheduler splits the job into stages based on shuffle boundaries.
4. **Task Assignment**: The stages are further divided into tasks and assigned to executor nodes.
5. **Task Execution**: Executors run the tasks, processing data and storing intermediate results in memory or disk.
6. **Shuffle Operations**: Data is shuffled between executors as required by operations like joins.
7. **Result Collection**: The final results are collected back to the driver node or written to storage.

7. What are the different phases of the SQL Engine?

The different phases of the SQL engine in Spark include:

1. **Parsing**: Converts the SQL query into an unresolved logical plan.
2. **Analysis**: Resolves the logical plan by determining the correct attributes and data types using the catalog.
3. **Optimization**: Optimizes the logical plan using Catalyst optimizers.
4. **Physical Planning**: Converts the optimized logical plan into a physical plan with specific execution strategies.
5. **Execution**: Executes the physical plan and returns the result.

8. Why do we get an Analysis Exception error?

Analysis Exception errors occur in Spark for several reasons:

- **Unresolved attributes**: When column names in the query don't match the schema.
- **Missing tables**: When referenced tables or DataFrames are not available.
- **Schema mismatch**: When the data types of the columns don't match the expected types.
- **Unsupported operations**: When trying to perform operations that are not supported by Spark SQL.

9. Explain in detail the different types of transformations in Spark.

Transformations in Spark are of two types:

1. **Narrow Transformations**: These transformations do not require shuffling of data across partitions and can be executed without data movement. Examples include:

   – **map()**: Applies a function to each element.
   – **filter()**: Selects elements that satisfy a condition.
   – **flatMap()**: Similar to map but can return multiple elements for each input element.
2. **Wide Transformations**: These transformations require shuffling of data across partitions and involve data movement. Examples include:

   – **reduceByKey()**: Aggregates data across keys and requires shuffling.
   – **groupByKey()**: Groups data by key, resulting in shuffling of data.
   – **join()**: Joins two RDDs based on a key, involving shuffling.

10. How many partitions are created when we invoke a wide dependency transformation?

The number of partitions created during a wide dependency transformation is determined by the `spark.sql.shuffle.partitions` configuration, which defaults to 200. However, it can be adjusted based on the size of the data and the specific requirements of the job.