Spark Fundamentals

1. RDD (Resilient Distributed Dataset)

- **Core abstraction in Spark** — the low-level API.
- Represents a **distributed collection of objects**, immutable and partitioned across the cluster.
- **Type:** Strongly-typed but unstructured (just objects).
- **Operations:** Supports **functional transformations** like map, filter, reduce.
- **Schema:** No schema or structure; you work directly with JVM objects.
- **Optimization:** No built-in optimization; transformations are executed as is.
- **Use case:** Low-level control, complex transformations, and custom code where you want fine control of data.
- **Performance:** Slower compared to DataFrames/Datasets because it lacks Catalyst optimizer and Tungsten execution.

2. DataFrame

- **Higher-level API** introduced in Spark 1.3.
- Represents data as a **distributed collection of rows with schema** (like a table in a relational database).
- **Type:** Un-typed (like a table of Rows), schema attached.
- **Operations:** SQL-like operations; you can use SQL queries or DataFrame API (select, filter, groupBy, etc.).
- **Optimization:** Uses Spark's **Catalyst optimizer** for query optimization.
- **Use case:** When you want to work with structured data easily and benefit from optimizations.
- **Performance:** Faster than RDDs due to Catalyst and Tungsten optimizations.
- **Language integration:** Available in Scala, Java, Python, R.

3. Dataset

- Introduced in Spark 1.6 (Scala/Java only).
- Combines **the best of RDD and DataFrame**:
    - Has **type safety** (like RDDs),
    - Has **schema and optimizations** (like DataFrames).
- **Type:** Strongly-typed, with compile-time type checking.
- **Operations:** Functional transformations + SQL operations.
- **Optimization:** Uses Catalyst optimizer.
- **Use case:** When you want both type safety (compile-time checks) and performance optimizations.
- **Performance:** Comparable to DataFrames, but with more type safety.
- **Language integration:** Scala and Java only (not available in Python or R).

Summary Table

Feature	RDD	DataFrame	Dataset
Type Safety	Yes (strongly typed JVM)	No (untyped Rows)	Yes (strongly typed JVM)
Schema	No	Yes	Yes
API Level	Low-level	Higher-level	Higher-level

Optimization (Catalyst)	No	Yes	Yes
Performance	Slowest	Fast	Fast
Use Case	Complex/custom logic	SQL and structured queries	Typed API with optimizations
Available Languages	Scala, Java, Python, R	Scala, Java, Python, R	Scala, Java only

Example code snippets:

```
// RDD example
val rdd = spark.sparkContext.parallelize(Seq(1, 2, 3, 4))
val rddFiltered = rdd.filter(_ > 2)
rddFiltered.collect()

// DataFrame example
import spark.implicits._
val df = Seq((1, "Amit"), (2, "Bavana")).toDF("id", "name")
val dfFiltered = df.filter($"id" > 1)
dfFiltered.collect()

// Dataset example (typed case class)
case class Person(id: Int, name: String)
val ds = Seq(Person(1, "Aleen"), Person(2, "Vandana")).toDS()
val dsFiltered = ds.filter(_.id > 1)
dfFiltered.collect()
```