Q1. Demonstrate how to implement a custom Writable class in Hadoop to serialize and deserialize an object containing multiple fields, such as an employee record

Ans: To implement a custom Writable class in Hadoop for serializing and deserializing an object with multiple fields (like an employee record), you would follow these steps:

1. Create the Custom Writable Class

Define a class (e.g., EmployeeRecord) that implements the Writable and WritableComparable interfaces.

2. Add Fields

Declare the fields for the object, such as employeeId, name, and salary.

3. Implement Methods

Implement the following methods:

- **Default Constructor:** Create a no-argument constructor to ensure the class is instantiable by Hadoop.
- **Getter and Setter Methods:** Provide methods for accessing and modifying the fields.
- **write(DataOutput out):** Serialize each field using DataOutput methods.
- **readFields(DataInput in):** Deserialize each field using DataInput methods.
- **compareTo(Object o):** If the object is comparable, implement the comparison logic for sorting.

Q2. Show the differences between the local and distributed modes of running Pig scripts. When would you use each mode?

Ans: **Differences Between Local and Distributed Modes of Running Pig Scripts**

Aspect	**Local Mode**	**Distributed Mode**
Environment	Runs on a single machine without requiring Hadoop cluster.	Runs on a Hadoop cluster with HDFS integration.
Input/Output	Uses local file system for data storage.	Uses HDFS for input and output data.
Execution	Suitable for smaller datasets.	Designed for processing large-scale data.
Configuration	Requires minimal configuration, often standalone.	Requires Hadoop setup and Pig configured for the cluster.
Performance	Slower for larger datasets due to limited resources.	Leverages distributed processing for efficiency.

When to Use Each Mode

1. **Local Mode:**
    - When developing or debugging Pig scripts.
    - For small datasets that can fit into the memory of a single machine.
    - When a Hadoop cluster is unavailable or unnecessary for the task.
2. **Distributed Mode:**
    - For processing large datasets stored in HDFS.
    - When running production-level Pig scripts.
    - To utilize the distributed nature of Hadoop for scalability and performance.

Q3. Analyze how Hive data types differ from traditional SQL data types. Provide examples of where specific Hive data types are most useful

Ans: **Differences Between Hive and Traditional SQL Data Types**

1. **Data Type Coverage**:
   Hive provides specialized data types to handle semi-structured and unstructured data, which are not typically supported in traditional SQL.
2. **Custom Data Types**:
   Hive includes **complex types** (like ARRAY, MAP, and STRUCT) and **specialized types** (e.g., TIMESTAMP, DECIMAL) tailored for distributed processing and big data needs. Traditional SQL mostly focuses on scalar types (e.g., INTEGER, CHAR, VARCHAR).
3. **Null Handling**:
   Hive treats nulls explicitly in computations, whereas traditional SQL implementations may vary in their handling of nulls.

Key Hive Data Types and Their Usage

Hive Data Type	Traditional SQL Equivalent	Use Case
STRING	VARCHAR, CHAR	Storing textual data; ideal for large, variable-length text such as logs or user comments.
BIGINT	BIGINT	Handling large integers, like IDs or large counters in big data.
FLOAT, **DOUBLE**	FLOAT, DOUBLE	Storing precise decimal values, e.g., financial or scientific data.
DECIMAL	DECIMAL, NUMERIC	High-precision calculations like currency values.

Hive Data Type	Traditional SQL Equivalent	Use Case
BOOLEAN	BOOLEAN	Flags or binary states, e.g., is_active or is_deleted fields.
ARRAY<TYPE>	Not directly supported in SQL	Storing lists, e.g., a user's multiple email addresses (ARRAY<STRING>).
MAP<KEY, VALUE>	Not directly supported in SQL	Key-value pairs, e.g., storing product attributes like MAP<STRING, STRING>.
STRUCT	ROW (in some SQL implementations)	Grouping multiple related fields, e.g., user profile with nested fields.
TIMESTAMP	DATETIME, TIMESTAMP	Storing date and time, useful in time-series data.
BINARY	BLOB, VARBINARY	Storing binary data, e.g., image files or encoded strings.

Q4. Apply a HiveQL query to create a table for storing customer orders, including columns for order ID,

customer name, product name, and order date. Insert at least two sample records

Ans: **HiveQL Query to Create and Populate a Table for Customer Orders**

Step 1: Create the Table

```
CREATE TABLE customer_orders (
    order_id INT,
    customer_name STRING,
    product_name STRING,
    order_date DATE
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
```

Step 2: Insert Sample Records

```
INSERT INTO TABLE customer_orders VALUES
    (101, 'Alice Johnson', 'Smartphone', '2024-12-01'),
    (102, 'Bob Smith', 'Laptop', '2024-12-02');
```

Step 3: Query the Data (Optional)

To verify the data insertion, run:

SELECT * FROM customer_orders;

Q5. Compare the Pig Latin application flow with traditional MapReduce programming. What are the main advantages of using Pig?

Ans: **Comparison of Pig Latin Application Flow with Traditional MapReduce**

Aspect	**Pig Latin**	**Traditional MapReduce**
Abstraction Level	High-level scripting language; abstracts complexities.	Low-level programming; requires detailed implementation.
Ease of Use	Simple syntax, reducing the need for Java programming.	Requires writing extensive Java code for mappers and reducers.
Development Time	Faster, as logic is written in concise Pig scripts.	Slower, as detailed code must be written and debugged.
Data Flow	Describes the data flow declaratively (e.g., filtering, joining).	Explicitly implements data flow in code (manual control).
Reusability	Easy to reuse and modify scripts.	Reusing code is cumbersome and often requires rewriting.

Advantages of Using Pig Over Traditional MapReduce

1. **Simplified Programming**:
   Pig Latin is a high-level language, making it easier to write and understand compared to the verbose Java code needed for MapReduce.
2. **Reduced Development Time**:
   Pig significantly cuts down on development effort by offering a declarative syntax for common operations like filtering, grouping, and joining.
3. **Automatic Optimization**:
   Pig's execution engine optimizes scripts into efficient MapReduce jobs, relieving developers from manually tuning performance.
4. **Data Flow Transparency**:
   Pig scripts explicitly show the data flow, making them easier to read and debug.

Q6. Illustrate a raw comparator in Hadoop for a Writable object that sorts employee records first by department and then by salary in descending order. Explain your implementation steps

Ans: **Raw Comparator in Hadoop: Sorting Employee Records by Department and Salary**

A **raw comparator** in Hadoop is used to compare keys directly in their serialized (binary) form, avoiding the overhead of deserialization. Here's how it works conceptually to sort employee records first by department (ascending) and then by salary (descending):

Implementation Steps

1. **Understand the Writable Object**:
    - The Writable object contains fields for department (String) and salary (double).
    - These fields are serialized into bytes for storage and transmission in Hadoop.
2. **Comparator Logic**:
    - The comparator reads serialized byte arrays for the fields directly.
    - First, it extracts the department field from both byte arrays and compares them lexicographically (ascending order).
    - If the departments are identical, it extracts the salary field (as double values) and compares them numerically in descending order.
3. **Serialization Knowledge**:
    - The raw comparator uses serialization methods to understand how the Writable object stores fields as bytes. For example, WritableUtils is used to decode strings, and standard methods are used for numeric types.
4. **Order of Comparison**:
    - Compare department: If different, return the result immediately.
    - Compare salary: If departments are the same, reverse the comparison to achieve descending order for salaries.
5. **Advantages of Raw Comparators**:
    - **Performance**: Raw comparators work on binary data directly, eliminating the need to deserialize objects, making them faster.
    - **Custom Logic**: Provides precise control over sorting logic for specific requirements.