Handling primary keys in PySpark involves ensuring that certain columns, which are marked as primary keys in the JSON schema, have unique values and are not `null`. While PySpark's `StructType` and `StructField` don't have built-in support for primary key constraints (like relational databases), you can manually enforce these constraints by adding checks within your transformation logic.

### Steps to Handle Primary Keys in PySpark

1. **Ensure Non-Null Values for Primary Keys**

   In the JSON schema, fields like `CONTRACT_SOURCE_SYSTEM`, `CONTRACT_SOURCE_SYSTEM_ID`, and `NSE_ID` are marked as primary keys. Therefore, you need to ensure that these columns do not contain any null values before writing the data.

2. **Ensure Uniqueness for Primary Keys**

   You can enforce uniqueness by checking for duplicate values in the DataFrame.

### Enforcing Primary Key Constraints in PySpark

You can add custom checks for nulls and duplicates on the primary key fields (`CONTRACT_SOURCE_SYSTEM`, `CONTRACT_SOURCE_SYSTEM_ID`, and `NSE_ID`) as follows:

#### 1. **Check for Null Values**

You can filter the DataFrame to check for any rows where the primary key fields are `null`. If such rows exist, you can either handle them (e.g., raise an error or log them) before proceeding.

```python
# Check for null values in primary key columns
null_check_df = transactions_df.filter(
    col("CONTRACT_SOURCE_SYSTEM").isNull() |
    col("CONTRACT_SOURCE_SYSTEM_ID").isNull() |
    col("NSE_ID").isNull()
)

if null_check_df.count() > 0:
    raise ValueError("Primary key columns contain null values.")
else:
```

    print("No null values in primary key columns.")

#### 2. **Check for Duplicates**

To enforce uniqueness, you can group by the primary key columns and check if any group has more than one row.

```python
# Check for duplicates based on primary key columns
duplicate_check_df = transactions_df.groupBy(
    "CONTRACT_SOURCE_SYSTEM", "CONTRACT_SOURCE_SYSTEM_ID", "NSE_ID"
).count().filter(col("count") > 1)

if duplicate_check_df.count() > 0:
    raise ValueError("Duplicate primary key values found.")
else:
    print("No duplicates in primary key columns.")
```

#### 3. **Enforce Constraints in the Pipeline**

You can integrate both of these checks (null and duplicates) into your processing pipeline before writing the DataFrame.

### Full Example (Integrating Primary Key Validation)

Here's how you can handle primary keys by checking for nulls and duplicates in your existing code before writing the DataFrame:

```python
def validate_primary_keys(df: DataFrame):
    # Check for null values in primary key columns
    null_check_df = df.filter(
        col("CONTRACT_SOURCE_SYSTEM").isNull() |
        col("CONTRACT_SOURCE_SYSTEM_ID").isNull() |
        col("NSE_ID").isNull()
    )

    if null_check_df.count() > 0:
        raise ValueError("Primary key columns contain null values.")

    # Check for duplicate primary key combinations
    duplicate_check_df = df.groupBy(
        "CONTRACT_SOURCE_SYSTEM", "CONTRACT_SOURCE_SYSTEM_ID", "NSE_ID"
    ).count().filter(col("count") > 1)

    if duplicate_check_df.count() > 0:
        raise ValueError("Duplicate primary key values found.")

    print("Primary key validation passed.")

# Call this function before writing the data
validate_primary_keys(transactions_df)

# If validation passes, proceed to write the data
transactions_df.write.mode("overwrite").parquet("transactions_output.parquet")
```

### Summary of Key Points:

1. **Null Checks**: Ensure that the primary key columns (`CONTRACT_SOURCE_SYSTEM`, `CONTRACT_SOURCE_SYSTEM_ID`, and `NSE_ID`) do not contain `null` values.

2. **Uniqueness Checks**: Ensure that there are no duplicate values across the combination of the primary key columns.

3. **Integration in Pipeline**: Perform these checks after data transformations but before writing the output.

By adding these checks in PySpark, you simulate the enforcement of primary key constraints that are typically found in relational databases.