Generic File Source Options

»

These generic options/configurations are effective only when using file-based sources: parquet, orc, avro, json, csv, text.

Please note that the hierarchy of directories used in examples below are:

```
dir1/
 ├── dir2/
 │    └── file2.parquet (schema: <file: string>, content: "file2.parquet")
 └── file1.parquet (schema: <file, string>, content: "file1.parquet")
 └── file3.json (schema: <file, string>, content: "{'file':'corrupt.json'}")
```

Ignore Corrupt Files

Spark allows you to use `spark.sql.files.ignoreCorruptFiles` to ignore corrupt files while reading data from files. When set to true, the Spark jobs will continue to run when encountering corrupted files and the contents that have been read will still be returned.

To ignore corrupt files while reading data files, you can use:

Scala | Java | Python | R

```java
// enable ignore corrupt files
spark.sql("set spark.sql.files.ignoreCorruptFiles=true");
// dir1/file3.json is corrupt from parquet's view
Dataset<Row> testCorruptDF = spark.read().parquet(
        "examples/src/main/resources/dir1/",
        "examples/src/main/resources/dir1/dir2/");
testCorruptDF.show();
// +-------------+
// |         file|
// +-------------+
// |file1.parquet|
// |file2.parquet|
// +-------------+
```

Find full example code at "examples/src/main/java/org/apache/spark/examples/sql/JavaSQLDataSourceExample.java" in the Spark repo.

Ignore Missing Files

Spark allows you to use `spark.sql.files.ignoreMissingFiles` to ignore missing files while reading data from files. Here, missing file really means the deleted file under directory after you construct the `DataFrame`. When set to true, the Spark jobs will continue to run when encountering missing files and the contents that have been read will still be returned.

Path Global Filter

`pathGlobFilter` is used to only include files with file names matching the pattern. The syntax follows `org.apache.hadoop.fs.GlobFilter`. It does not change the behavior of partition discovery.

To load files with paths matching a given glob pattern while keeping the behavior of partition discovery, you can use:

Scala | Java | Python | R

```java
Dataset<Row> testGlobFilterDF = spark.read().format("parquet")
        .option("pathGlobFilter", "*.parquet") // json file should be filtered out
        .load("examples/src/main/resources/dir1");
testGlobFilterDF.show();
// +-------------+
// |         file|
// +-------------+
// |file1.parquet|
// +-------------+
```

Find full example code at "examples/src/main/java/org/apache/spark/examples/sql/JavaSQLDataSourceExample.java" in the Spark repo.

Recursive File Lookup

`recursiveFileLookup` is used to recursively load files and it disables partition inferring. Its default value is `false`. If data source explicitly specifies the `partitionSpec` when `recursiveFileLookup` is true, exception will be thrown.

To load all files recursively, you can use:

» Scala | Java | Python | R

```
Dataset<Row> recursiveLoadedDF = spark.read().format("parquet")
        .option("recursiveFileLookup", "true")
        .load("examples/src/main/resources/dir1");
recursiveLoadedDF.show();
// +-------------+
// |         file|
// +-------------+
// |file1.parquet|
// |file2.parquet|
// +-------------+
```

Find full example code at "examples/src/main/java/org/apache/spark/examples/sql/JavaSQLDataSourceExample.java" in the Spark repo.

Modification Time Path Filters

`modifiedBefore` and `modifiedAfter` are options that can be applied together or separately in order to achieve greater granularity over which files may load during a Spark batch query. (Note that Structured Streaming file sources don't support these options.)

- `modifiedBefore`: an optional timestamp to only include files with modification times occurring before the specified time. The provided timestamp must be in the following format: YYYY-MM-DDTHH:mm:ss (e.g. 2020-06-01T13:00:00)
- `modifiedAfter`: an optional timestamp to only include files with modification times occurring after the specified time. The provided timestamp must be in the following format: YYYY-MM-DDTHH:mm:ss (e.g. 2020-06-01T13:00:00)

When a timezone option is not provided, the timestamps will be interpreted according to the Spark session timezone (`spark.sql.session.timeZone`).

To load files with paths matching a given modified time range, you can use:

Scala | Java | Python | R

```
Dataset<Row> beforeFilterDF = spark.read().format("parquet")
        // Only load files modified before 7/1/2020 at 05:30
        .option("modifiedBefore", "2020-07-01T05:30:00")
        // Only load files modified after 6/1/2020 at 05:30
        .option("modifiedAfter", "2020-06-01T05:30:00")
        // Interpret both times above relative to CST timezone
        .option("timeZone", "CST")
        .load("examples/src/main/resources/dir1");
beforeFilterDF.show();
// +-------------+
// |         file|
// +-------------+
// |file1.parquet|
// +-------------+
```

Find full example code at "examples/src/main/java/org/apache/spark/examples/sql/JavaSQLDataSourceExample.java" in the Spark repo.