[Kernel] Remove contextualization-related APIs

Venki Korukanti Tathagata "TD" Das

Context

Currently Delta Kernel has `FileHandler.contextualize` and `FileReadContext` API/interfaces to allow the connectors to split the scan files into chunks. However, Kernel is the one initiating these calls. It should be in the control of the connector. By making the connectors control the contextualization/splitting, we can get rid of a few interfaces/APIs from Kernel which simplifies Kernel interfaces.

One of the major feedback from the RUST-Kernel community is that having FileContext on TableClient causes too many template parameters. By removing file context related APIs, the number of template params is reduced significantly.

Proposed changes to remove the contextualize APIs.

Current	Proposed

Kernel APIs
 	
`class Scan {`
`  static CloseableIterator<FilteredColumnarBatch> readData(`
`  TableClient tableClient,`
`  Row scanState,`
`  CloseableIterator<Row> scanFileRowIter,`
`  Optional<Predicate> predicate`
`  ) {`
`  // What does this method do?`
`  1. Convert the logical read schema to physical read schema`
`  2. Remove the partition columns from the physical read schema`
`  3. Add metadata (ROW_INDEX) columns to the read physical schema`
`  4. Contextualize the scan file rows`
`  5. Call ParquetHandler with FileContextes and read physical schema`
`  a. For each data batch returned by the ParquetHandler`
`  b. Generated vectors for partition columns and attached them to the data batch.`
`  c. Remove the metadata (ROW_INDEX) columns`
	Kernel APIs
	
	`class Scan {`
	`  // Earlier it used by a `readData`, but now it just transforms`
	`  // the physical data read from the data file.`
	`  static CloseableIterator<FilteredColumnarBatch> transformData(`
	`  TableClient tableClient,`
	`  Row scanState,`
	`  Row scanFile,`
	`  // Data that belongs to the scanFile. The connector opens`
	`  // the Parquet file and gives Kernel the data from the file`
	`  // The data is of required physical schema + additional`
	`  // required metadata columns (row_index)`
	`  CloseableIterator<ColumnarBatch> dataIter`
	`  ) {`
	`  // What does this method do?`
	`  // For each data batch returned by the ParquetHandler`
	`  1. Generated vectors for partition columns and attached them to the data batch.`

```
         d. Generate DV selection vector
         e. Make the schema of the batch be same as the logical read schema
   6. Return the FilteredColumnarBatch.
  }
}

// FileDataReadResult contains a `ColumnarBatch` and scan file row from where
// it is read.
// This is used by both the LogReplay and data reading path in Kernel.
class ParquetHandler {
  CloseableIterator<FileDataReadResult> readParquetFiles(
    CloseableIterator<FileReadContext> fileIter,
    StructType physicalSchema
  ) throws IOException;
}
```

```
   2. Remove the metadata (ROW_INDEX) columns
   3. Materialize the Row Ids
   4. Generate DV selection vector
   5. Make the schema of the batch be same as the logical schema
   6. Create a FilteredColumnarBatch containing
         a. Data read from the Parquet file
         b. Partition column vectors created using the expression
            handler
         c. (minus) metadata columns
         d. Selection vector
  }
}

// Now this will be used only by the LogReplay code to read the
// checkpoint files. In the current code, the ParquetHandler is used both
// but the LogReplay and data reading path in Kernel.
// One difference between the current API and this API is that the new
// API just returns a ColumnarBatch and it doesn't associate with the
// scan file row from where the batch came from. There isn't a need for
// this now. We needed it for associating the DV info in data read path.
class ParquetHandler {
  CloseableIterator<ColumnarBatch> readParquetFiles(
    CloseableIterator<Row> scanFiles,
    StructType readPhysicalSchema,
   // predicate to prune data (not mandatory that all data
   // returned satisfies the predicate.
    Optional<Predicate> predicate
  ) throws IOException;
}

// Utility method
StructType getReadPhysicalSchema(Row scanState)
```

Current APIs shown to the left will be removed. Also the FileHandler. contextualize and FileReadContext.

Connector Code

```
Scan scan = …

CloseableIterator<ColumnarBatch> scanFilesIter = scan.getScanFiles();

scanFileItera.foreach(scanFileBatch ->
  CloseableIterator<FilteredColumnarBatch> data =
    Scan.readData(
```

Connector Code

```
Scan scan = …

CloseableIterator<ColumnarBatch> scanFilesIter = scan.getScanFiles();

StructType readPhysicalSchema =
  ScanStateUtils.getReadPhysicalSchema(scan.getScanState)
```

```
      tableClient,
      scan.getScanState(),
      scanFileBatch.getRows(),
      predicate
  )

  ... consume the data ...
)
```

```
scanFileIter.foreach(scanFileBatch ->
  scanFileBatch.getRows().foreach(scanFileRow ->

    // This is connector-specific implementation where it can
    // choose the split the file based on the connector requirements.
    CloseableIterator<EngineScanFileChunk> chunksIter =
       contextualize/split(scanFileRow);

    chunksIter.foreach(chunk ->
      // Open the Parquet file for reading the specific chunk
      CloseableIterarator<ColumnarBatch> parquetDataIter =
         EngineParquetHandler.readParquetFile(
           Chunk,
           readPhysicalSchema,
           predicate
        )

      CloseableIterator<FilteredColumnarBatch> data =
        Scan.transformData(
          tableClient,
          scan.getScanState(),
          scanFileRow,
          parquetDataIter
        )
      ... consume the data ...
    )
  )
)
```

Additional Connector APIs

```
CloseableIterator<ColumnarBatch> readParquetFile(
      EngineScanFileChunk chunk,
      StructType readPhysicalSchema,
      predicate) throws IOException
```

Pros:

- Slightly simpler connector code

Cons:

- Connector has no control over splits during the query task scheduling time. It can only split during a task execution
- Additional interfaces: contextualize and FileReadContext

Pros:

- Fewer interfaces
- Connector has control over the splits during the query task scheduling time.

Cons:

- Connector code is a bit complicated as it has to deal with opening the Parquet file.

Native Support for ID column mapping mode in the Parquet Handler

The current Parquet reader can be extended to support the field-based schema mapping. Currently we pass a StructType (in case of name based mapping, it contains the physical names) as read schema. The Parquet reader reads the schema from footer and searches for the columns given by Kernel using the physical names in the Parquet schema. This can be extended to lookup by the `fileId`. Only thing is, Kernel needs to pass the metadata in the `StructField` for the fieldId.

https://github.com/delta-io/delta/blob/master/kernel/kernel-defaults/src/main/java/io/delta/kernel/defaults/internal/DefaultKernelUtils.java#L60

PR to support the id column mapping mode within the ParquetHandler. Supporting native column mapping id in ParquetHandler is a must in order to make the above proposed API changes.

========== STOP READING - NOTES ==========

```
None
class Scan {

       CloseableIterator<FilteredColumnarBatch> readData(
              TableClient tableClient,
              Row scanState,
              CloseableIterator<Row> scanFileRowIter,
              Optional<Predicate> predicate
       )
}

class ParquetHandler {
    CloseableIterator<FileDataReadResult> readParquetFiles(
        CloseableIterator<FileReadContext> fileIter,
        StructType physicalSchema) throws IOException;
}
```

New

```
None

class Scan {
       CloseableIterator<FilteredColumnarBatch> transformData(
              TableClient tableClient,
              Row scanState,
```

```
            // From which scan file row is the data coming from
            Row scanFileRow,
            CloseableIterator<ColumnarBatch> dataFromParquetReader,
      ) {
            // Add partition columns
            // Remove ROW_INDEX columns
            // Generate DV selection vector based on the `scanFileRow`
            // physical schema comes from the Parquet batches
            //   Get the logicl schema from the scanState and convert
            //   the physical schema to logical schema.
            //   TODO: can't just put the required logical read schema
            //   directly in scan State and use it directly here?
            // Return a new columnar batch with the logical schema
      }
}


// Before calling the `Scan.readData` the connector has to do the following
steps

For each scan file Row
      - See if it wants to divide the file into chunks. If yes, create one
      - or more `FileReadContext` for each scan file row
      - From the ScanState
            - get the readPhysicalSchema - Kernel should add the required
              physical columns including the ROW_INDEX columns as part of
              the ScanState row generation
      - Make a call to ParquetHandler.readParquetFiles(FileReadContext)
            - Here we are making the call with one Parquet chunk read
            - at a time. This is not helpful with pre-fetching
      - Call Scan.transform(
            tableClient,
            scanState,
            scanFileRow,
            dataIteratorFromTheParquetFile
      )


Parquet Handler read code need to be changes as follows:


// Executor code to read the table data

CloseableIterator<EngineReadContext> contextsFromDriver = ....
```

```
Row scanStateRow = ...

contextsFromDriver.flatMap { context =>

       Row scanFileRow = context.getScanFileRow;
       StructType readPhysicalSchema =
              RowUtils.getReadPhysicalReadSchema(scanState)

       CloseableIterator<ColumnarBatch> readBatches =
              EngineParquetReader.readParquetFile(context, readPhysicalSchema)

       CloseableIterator<FilteredColumnarBatch> finalData = Scan.transformData(
              tableClient,
              scanState,
              scanFileRow,
              EngineParquetReader.readParquetFile(context, readPhysicalSchema)

       finaldata
}
```

```
// engine specific implementation of ParquetHandler Kernel interface

class ParquetHandler {
       CloseableIterator<ColumnarBatch> readParquetFiles(
              CloseableIterator<Row> scanFileRowIter,
              StructType readPhysicalSchema) throws IOException) {

              scanFileRowIter.map {row =>
                     EngineReadContext context =
                            ...  // from row

                     ParquetReadUtils.readParquetFiles(
                            context, readPhysicalSchema)
              }
       }
}
```

```
// engine internal class

class EngineParquetReader {
   CloseableIterator<ColumnarBatch> readParquetFile(
              EngineReadContext contexts,
              StructType readPhysicalSchema) throws IOException) {
              ....
   }
}
```