III. SQL 读取链路实现

读取链路

本节，我们尝试读取写入的数据。

```
select id, val + 10 from t where val < 10;
```

我们以这条 SQL 为例。首先它是一条 Select 语句，Select 会通过 `executor/table_reader.go` 中的 `TableReaderExecutor` 执行，但是这里并不是直接返回读取到的结果，需要将 val + 10 后的结果返回给客户端，因此还需要使用到 `executor/projection.go` 中的 `ProjectionExec` 来对 Select 的结果做一次运算处理。

- 1. `executor/builder.go`，因为数据处理的顺序是先通过 `SelectionExec` 获取数据再使用 `ProjectionExec` 进行计算处理，所以最外层的是 `ProjectionExec`，内层是 `TableReaderExecutor`。在 build 阶段，首先会执行 `executorBuilder.build` 中调用到 `executorBuilder.buildProjection` 函数，`ProjectionExec` 一定会对下层的结果进行处理，所以有 children，这里会递归的调用 `executorBuilder.build` 函数来 build 子 Executor。
- 2. `executor/table_reader.go`，`TableReaderExecutor` 的数据源是 `TableReaderExecutor.resultHandler`，最后会通过 `distsql/select_result.go` 中的 `SelectResult` 来执行。`SelectResult` 仅会从 TiKV 中获取所需要的数据来减少数据的传输量。具体的调用链路是 `TableReaderExecutor.Next` 调用 `tableResultHandler.nextChunk`，其中通过 `selectResult.Next` 方法（定义在 `SelectResult` 接口中）填充 Chunk。
- 3. `executor/projection.go`，我们来看一看 `ProjectionExec` 的 `ProjectionExec.parallelExecute` 是怎么运行的，可以结合 lab0 的 Map-Reduce 来理解。下面所描述的流程在 `ProjectionExec.Next` 的注释中有示意图。
    - 3.1 外部线程不停的调用 `ProjectionExec.Next` 获取处理完成的数据，在并行处理时会调用 `ProjectionExec.parallelExecute`。`ProjectionExec.parallelExecute` 函数中会从 `ProjectionExec.outputCh` 中拿到数据并且通过 `Chunk.SwapColumns` 将数据写入外部传入的 `Chunk` 中。
    - 3.2 `fetcher` 线程负责从内部的 Executor 获取读到的数据，这里是从

`projectionInputFetcher.inputCh` 拿到 `projectionInput`，然后把 `TableReaderExecutor` 中读数据通过 `projectionInput.chk.SetRequiredRows` 写入，最后将带有数据的 `projectionInput` 发送到 `input.targetWorker.inputCh` 当中。从 `projectionInputFetcher.outputCh` 读到的数据是 `worker` 线程处理完的结果，将结果发送给 `ProjectionExec.outputCh`（也是 `projectionInputFetcher.globalOutputCh`），同时也会发送到 `input.targetWorker.outputCh`。

- 3.3 `worker` 线程会把 `fetcher` 写入到 `projectionWorker.inputCh` 当中的内部 Executor 结果数据取出，把 `projectionWorker.outputCh` 的结果写入用的 `projectionOutput` 取出，计算后写入从 `projectionOutput.chk`。在处理之后，只需要将 `projectionInput` 从 `projectionWorker.inputGiveBackCh`（3.2 中的 `projectionInputFetcher.inputCh`）还给 `fetcher`。

以上是读取数据的关键路径，这个调用链路中的关键函数也被移除了，你需要根据调用链路的描述进行填充。

测试

由于数据库的初始化需要依赖 `lab4a`，`lab4b`，`lab4c` 的逻辑，所以我们需要在完成 lab4 的所有代码之后再进行测试。

运行 `make lab4`，通过所有测试用例。

可以使用下面的命令测试指定的单个或多个用例。

```
go test {package path} -check.f ^{regex}$
# example
go test -timeout 5s ./store/tikv -check.f ^TestFailAfterPrimary$
```