Impala Row Batches

Note: the canonical reference for this information is row-batch.h in the Impala source base. This page is a higher-level overview of how batches of rows in Impala are represented.

Impala represents row data internally using the RowBatch class. Query operators in Impala operate on rows in a batched manner for efficiency. By default batches of 1024 rows are used.

Concepts and Definitions

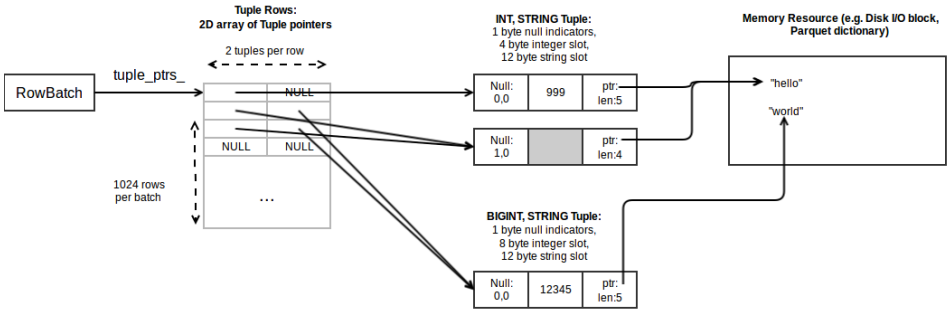
- Value: a value (e.g. int, string, array, etc). All values in Impala are nullable internally.
- Fixed-length data: the part of a value that is fixed in size (e.g. a 32-bit integer, the 32-bit length + 64-bit pointer representing a string)
- Variable-length data: parts of a value that vary in length, e.g. string data, maps, arrays
- Slot: an area of memory that holds the fixed-length part of a value (e.g. INT, STRING) if not null
- Null indicators: a fixed-length bitstring that indicates whether slots are NULL
- Tuple: an array of slots, plus null indicators
- Row: a logical row comprised of a number of values. A row is comprised of multiple tuples and represented as a fixed-length array of pointers to tuples.
- RowBatch: a batch of rows, plus information about memory resources referenced by the rows.
- Operator/ExecNode: a physical query operator, e.g. aggregation, join, scan

Memory Layout

Here is an example memory layout for an (INT, STRING, BIGINT, STRING) row that is comprised of two tuples. The data is:

int_col	string_col1	bigint_col	string_col2
999	"hello"	NULL	NULL
NULL	"hell"	12345	"world"
NULL	"hell"	12345	"world"
NULL	NULL	NULL	NULL

The memory layout purposefully uses many features to illustrate how data can be shared between rows and tuples. Most batches have simpler layouts.



Constructing a RowBatch in a Scan

Most table scanners in Impala follow a similar pattern for constructing row batches. Rows returned from all scans in Impala have only a single tuple per row, which simplifies this process.

1. A RowBatch is allocated, with 'tuple_ptrs_' allocated with tuple pointers for 1024 rows.
2. The fixed-length tuples are allocated as a large block (e.g. with RowBatch::ResizeAndAllocateTupleBuffer()) from the batch's MemPool.
3. Tuple data is filled in by the scanner.
    a. Predicates are evaluated on the row. If a row doesn't pass, the tuple memory is overwritten with data from the next tuple.
    b. Some scanners allocate variable-length data for strings from the RowBatch's MemPool.
4. Additional memory resources are attached to the RowBatch when they are finished. E.g. when the end of a disk I/O block has been reached.
5. When each RowBatch is full (has 1024 rows, or has more than 8mb of memory attached), it is returned from the scanner and a new RowBatch is allocated.

Memory Resources

All of the memory used to represent the RowBatch data needs to be managed so that memory is freed as query execution proceeds.

There are several types of memory resources that can be attached to row batches:

- MemPools: Each RowBatch has a MemPool that row data can be allocated from. Data from other MemPools can be transferred to a batch's MemPool.
- Disk IO buffers: Data is read from disk in large fixed-size buffers up to 8mb in size. RowBatches may directly reference data in these buffers (e.g. string data)
- BufferedTupleStreams/Blocks: some operators manage memory internally with in blocks and streams. Similarly to Disk IO buffers, row batches can directly reference data in these.

In the simplest case, all of the memory resources referenced by a RowBatch are directly attached to the RowBatch. However, memory resources are often shared between RowBatches for efficiency. E.g. a disk I/O buffer or a Parquet dictionary may contain string data referenced by many RowBatches. To deal with this problem, memory management for RowBatches returned by operators treats the batches returned from GetNext() as a stream. Resources attached to a batch may be referenced by previous batches in the stream. Therefore:

- If an operator is processing one batch at a time, this means that it can safely free attached resources after it processes each batch.
- If an operator is accumulating batches, this means that it must be careful not to destroy or reset a batch if previous batches are still in use, because this could release memory resources that are used by the previous batches.

Another deprecated mechanism also exists: need_to_return()/MarkNeedToReturn(). If GetNext() returns a batch with the need_to_return() flag set, it means that the batch may reference memory that will be freed on the next GetNext() call. If the caller wants to retain the batch's data, it need to copy the batch.