[DRAFT] Partition level column stats in Apache Iceberg

Background & Motivation:

For query engines that use cost-based optimizer along with rule-based optimizer (like Hive), at the planning time, it is good to know the partition level column stats like

- Histograms (KLL sketch), particularly useful for skewed data and range predicates;
- NDV (Number of Distinct Values);
- Min and Max values, estimate the selectivity of range filters;
- Number of Null / True values, play a role in selectivities, used for estimating IS NULL, IS NOT NULL, IS TRUE, IS NOT TRUE predicates;

Most if not all are used to estimate row counts and help CBO decide on the join reordering and join type, identifying the parallelism.

Current State

Iceberg currently does not support partition-level column statistics, required for cost-based optimization (CBO).

Additionally, table-level column statistics are constrained by the THETA sketch, which is the only standard Blob type defined in Iceberg's Puffin specification.

The THETA sketch provides the number of distinct values (NDVs) and is stored as a Blob per column within a Puffin file.

```
private static Blob toBlob(Types.NestedField field, Sketch sketch, Snapshot
snapshot) {
 return new Blob(
     StandardBlobTypes.APACHE_DATASKETCHES_THETA_V1,
     ImmutableList.of(field.fieldId()),
     snapshot.snapshotId(),
     snapshot.sequenceNumber(),
     ByteBuffer.wrap(sketch.toByteArray()),
     PuffinCompressionCodec.ZSTD,
     ImmutableMap.of(
         APACHE_DATASKETCHES_THETA_V1_NDV_PROPERTY,
         String.valueOf((long) sketch.getEstimate())));
}
```

```
"statistics": [
 {
   "snapshot-id": 3055729675574597004,
   "statistics-path": "s3://a/b/stats.puffin",
```

```
  "file-size-in-bytes": 413,
  "file-footer-size-in-bytes": 42,
  "blob-metadata": [
    {
      "type": "ndv",
      "snapshot-id": 3055729675574597004,
      "sequence-number": 1,
      "fields": [1]
    }
  ]
 }
]
```

Iceberg also supports tracking column lower bounds, upper bounds, column counts, null value counts, and NaN counts per column in table metadata. These are kept at the file level.

In version 1.5.0, Iceberg introduced *updatePartitionStatistics()* API that tracks partition statistics within TableMetadata. Additionally, in version 1.7.0, a utility method was added to compute basic partition statistics.

```
PartitionStatisticsFile statsFile =
PartitionStatsHandler.computeAndWriteStatsFile(table);

table.updatePartitionStatistics()
   .setPartitionStatistics(statsFile)
   .commit();
```

Proposal

A. What to store in partition-level column statistics?

- ☐ *count distinct,*
- ☐ *lower / upper bounds*, https://github.com/apache/iceberg/issues/11083
- ☐ *avg column length*,
- ☐ *null value counts*,
- ☐ *NaN counts,*
- ☐ *numTrue / numFalse* ???*,*
- ☐ *NDV sketch,*
- ☐ *histogram* (`KllFloatsSketch` / `KllDoublesSketch` for numeric columns, and `KllItemsSketch` for string/binary columns)

Q: Could we add numTrue/numFalse sparse bitVector in the table metadata? Applicable only for the Boolean column type.

Extend Iceberg's puffin spec with KLL sketch Blob type for table-level column stats (*apache-datasketches-kll-sketch-v1*).

B. How to store partition-level statistics?

Extend existing Iceberg's PartitionStats with partition-level column stats.

Option 1:
Write sketches into the same stats partition file.

```
PartitionStats {

  ColumnStats {
     ,Long ndv, // https://github.com/apache/iceberg/pull/10549
      Long numNulls,
      Long numNaNs,
      Double avgColLen,
      byte[] lowerBound,
      byte[] upperBound,
      Long numTrues,
      Long numFalses,
      byte[] ndvSketch,
      byte[] histogram;
  }

  Map<int fieldId, ColumnStats> colStats; // optional field
}
```

Option 2:
Store NDVs and KLL sketches within the single Puffin file at the partition level and reference it from the root partition stats file footer.

Histograms are generally more costly to build and maintain, so it could make sense to keep them separated.

This approach would require 1 Puffin file per table snapshot, containing a single Blob per partition value.

Each Blob represents a Map where *fieldId* serves as a secondary index and Struct of NDVs and KLL sketches as the value;

Cons:

1. Performance ???: partition values could grow to millions of rows; Option 3 ???

2. This implementation requires modifications to the Iceberg's puffin spec. In particular, the registration of a Blob type representing Map<fieldId, [NDVs, KLL]>.

Option 3:
Use batching: write column stats for multiple partitions within a single Puffin blob. Partitions within a batch should be globally sorted. Might require changing Puffin's *inputFields* type to String/Object.

```
Map<String, Map<fieldId, [NDVs, KLL]>> partitionStats;
Blob#inputFields - list of partition values in a batch
```