Filtering during ingestion

Motivation

Raw source data often needs to undergo some **transformations** before it is pushed to Pinot. Transformations include extracting records from nested objects, applying simple transform functions on certain columns, filtering out unwanted columns, etc and also more advanced operations like joining between datasets. Typically, users write a **preprocessing job** to perform these operations. In streaming data sources, such transformations require users to write a **samza job, and create another intermediate topic.** Writing preprocessing jobs, especially for simple transformations, creates an additional step for user onboarding, can result in inconsistencies in the batch/stream data source, and increases the maintenance and operator overhead.
It would be nice if **Pinot started supporting transformations natively, as far as possible, so that we begin eliminating the need to write preprocessing jobs**. We recently began efforts in this direction. We added the transform functions feature, which allows users to perform simple transformations on their columns using groovy scripts/inbuilt pinot functions. Another frequent reason that makes users write a preprocessing job is to filter out unwanted records from their data source. These could be test records, or records unwanted for the specific table use case. For example, filter out records with *eventType = "ERROR"*, or filter out records where *price <= 0*.

Goal

Add support in Pinot for **simple filtering based on expressions**. The filtering phase in Pinot will decide whether to include a record in the table, or skip it.
Pinot should support filtering on either

1. any column from the data source
2. a column in the Pinot schema, which could directly map from data source or be constructed in ExpressionTransformer using transform function.
3. an expression constructed using the columns from 1 and 2.

Proposal

Config

Introducing an **ingestionConfig** section in the table config. To begin with, this ingestionConfig will only have **filterConfig**. FilterConfig can look as simple as this in version 1

```
tableConfig: {
    tableName: ...,
    tableType: ...,
    ingestionConfig: {
        filterConfig: {
            filterFunction: “<expression>”
        }
    }
}
```

The expressions allowed here will be within the scope of the transform functions support that we have in Pinot today i.e. Groovy expressions, or any inbuilt functions.

Extracting fields needed for filtering

As of today, we initialize the decoders/record readers with a **Set<String> fieldsToRead**. This Set is passed on to the RecordExtractors. The extractors only extract fields that are provided to them in the Set<String> fieldsToRead.
The Set<String> fieldsToRead is constructed by the SchemaUtils#extractSourceFields method, using the Schema, which only has destination column names and transformation expressions.
The **fieldsToRead should start including the columns needed for filtering**. This should be the responsibility of the driver of the ingestion process i.e. the **SegmentCreationDriverImpl for batch and LLReadtimeSegmentDataManager for streaming.** Before initializing the record reader/decoder, the driver should **extract source fields from the filterSpec, and add them to the Set<String> fieldsToRead.** We can write a **utility method** for this, similar to SchemaUtils#extractSourceFields.

Model the filtering phase as a RecordTransformer

We create a new RecordTransformer called FilterTransformer.
This transformer will sit in between the ExpressionTransformer and all other transformers.

This Transformer needs a TableConfig (all transformers needed only Schema so far), hence we will change the CompositeTransformer#getDefaultTransformer method to include TableConfig in the params.

The transformation steps will look like:

Decoder
Expression -> **Filter** -> Null -> DataType -> Sanitization
Indexer

Phase	**Output GenericRow contains -**
Decoder/RecordReader	All required source fields
Expression Transformer	1. All source fields 2. Destination fields
Filter Transformer	1. All source fields 2. Destination fields **3. Special field indicating filter status**
Null Value Transformer	1. All source fields 2. Destination fields **3. Special field indicating filter status** 4. Default values

Within the FilterTransformer, the filter expressions can be executed, exactly like it is done in ExpressionTransformer. **The expression must evaluate to a boolean for filtering to take effect**. If the result is true, a special field will be put into the record.

Special field indicating filter status

In order to filter the record, we will use a special key **“$FILTER_RECORD_KEY$”**. Inside the FilterTransformer, if expression evaluates to true, this key will be put into the GenericRow, with value true.
The driver (SegmentCreationDriverImpl, SegmentStatsCollector, LLRealtimeDataManager, HLRealtimeDataManager) will **check for this key in the GenericRow, at the end of the transformations, and skip the record if value is true**.

Extensions

These are out of the scope of this task directly, but this doc is a good place to discuss them due to context already built.

IngestionConfig

As part of this filtering task, we propose to introduce an IngestionConfig in the table config. For starters, this will have only FilterConfig. This is a good place to consolidate many of our configs, especially those which are inconsistently placed between batch and stream.

1) **Filter** config - as described in this doc
2) **Transform** functions - column transformations also really belong with the table config, and we could use this place to move out transform functions.
3) **Flatten** config - when we introduce flattening during ingestion, this will be a good place to put the config
4) Record reader and decoder configs - The record reader configs for offline tables are provided as part of the ingestion job spec yaml file, whereas the decoder configs for the realtime table are provided as part of the streamConfigs in the table config.The **decoder configs** can be pulled out of stream configs into ingestion config. The record reader configs can be moved from ingestion job spec to the ingestion config.
5) Batch configs - Certain configs from the ingestion job spec should be in the ingestionConfig. For example, configs such as **input file path, input file pattern, push attempts, push interval job type**. While some of it may not make sense right now (for example, typically input file path changes with every ingestion and so it cannot be in the table config), these will make a lot of sense when supporting the pull model of ingestion for batch
6) Some configs such as segmentPushFrequency, segmentPushType

Dedup

A great feature to have as part of filtering would be dedup
TBD design