Kafka

Upsert, insert, update at sink

While data transmission in Kafka, there is a source and a sink which we need to understand first.

**Source Connector**

A source connector ingests entire databases and streams table updates to Kafka topics. It can also collect metrics from all of your application servers and store these in Kafka topics, making the data available for stream processing with low latency.

**Sink Connector**

A sink connector delivers data from Kafka topics into secondary indexes such as Elasticsearch, or batch systems such as Hadoop for offline analysis.

This sink connect can be writing data into a database as for instance MongoDB.

There are different functions but here we will be understanding upsert, insert and update as per Kafka connect with MongoDB

**Insert**

Insert is the default write mode of the sink.

Kafka currently provides at least once delivery semantics. Therefore, this mode may produce errors if unique constraints have been implemented on the target tables. If the error policy has been set to NOOP then the Sink will discard the error and continue to process, however, it currently makes no attempt to distinguish violation of integrity constraints from other exceptions such as casting issues. Insert simply adds the record in the sink database as a simple write function.

**Update**

MongoDB provides the update() command to update the documents of a collection. To update only the documents you want to update, you can add a criteria to the update statement so that only selected documents are updated.

The basic parameters in the command is a condition for which document needs to be updated, and the next is the modification which needs to be performed.

**Upsert**

The Sink supports MongoDB upsert functionality which replaces the existing row if a match is found on the primary keys.

Kafka currently provides at least once delivery semantics and order is a guaranteed within partitions. This mode will, if the same record is delivered twice to the sink, result in an idempotent write. The existing record will be updated with the values of the second which are the same.

If records are delivered with the same field or group of fields that are used as the primary key on the target table, but different values, the existing record in the target table will be updated. And if the fields or groups does not exist, upsert will simply do the functionality of insert.

Since records are delivered in the order they were written per partition the write is idempotent on failure or restart. Redelivery produces the same result.

**Idempotent**

An idempotent operation can be repeated an arbitrary number of times and the result will be the same as if it had been done only once. In arithmetic, adding zero to a number is idempotent. An idempotent method is a method that can be called many times without different outcomes. It would not matter if the method is called only once, or ten times over. The result should be the same. Again, this only applies to the result, not the resource itself. This still can be manipulated (like an update-timestamp, provided this information is not shared in the (current) resource representation.

**Examples of Upsert and Insert:**

-- Select all fields from topic fx\_prices and insert into the fx collection  
INSERT INTO fx SELECT \* FROM fx\_prices

-- Select all fields from topic fx\_prices and upsert into the fx collection, The assumption is there will be a ticker field in the incoming json:  
UPSERT INTO fx SELECT \* FROM fx\_prices PK ticker

-- Select specific fields from the topic sample\_topic and insert into the sample collection:  
INSERT INTO sample SELECT field1,field2,field3 FROM sample\_topic

-- Select specific fields from the topic sample\_topic and upsert into the sample collection:  
UPSERT INTO sample SELECT field1,field2,field3 FROM sample\_fopic PK field1

-- Rename some fields while selecting all from the topic sample\_topic and insert into the sample collection:  
INSERT INTO sample SELECT \*, field1 as new\_name1,field2 as new\_name2 FROM sample\_topic

-- Rename some fields while selecting all from the topic sample\_topic and upsert into the sample collection:  
UPSERT INTO sample SELECT \*, field1 as new\_name1,field2 as new\_name2 FROM sample\_topic PK new\_name1

-- Select specific fields and rename some of them from the topic sample\_topic and insert into the sample collection:  
INSERT INTO sample SELECT field1 as new\_name1,field2, field3 as new\_name3 FROM sample\_topic

-- Select specific fields and rename some of them from the topic sample\_topic and upsert into the sample collection:  
INSERT INTO sample SELECT field1 as new\_name1,field2, field3 as new\_name3 FROM sample\_fopic PK new\_name3

**KCQL Support**

You can specify multiple KCQL statements separated by ; to have a the connector sink multiple topics.

The MongoDB sink supports [KCQL](https://docs.lenses.io/sql/connectSql.html#lsql-connect), Kafka Connect Query Language. The following support KCQL is available:

1. Field selection
2. Target collection selection
3. Insert and upset modes.

{ INSERT | UPSERT } INTO collection\_name SELECT { FIELD, ... } FROM kafka\_topic\_name