Real-Time Updates

Overview

Real-time updates are fields in the AIE schema that support low-latency (high-speed) updates and can be updated without having to re-index entire documents. You can query, retrieve from, and apply facet filters to these fields as if they were normal fields in the index; however, you cannot use them for high performance join operations. Real-time updates occur in a fraction of the time required to update a full document.

Use real-time updates only when necessary, and generally, they should not contain large text values. A good rule of thumb is that an index schema should not have more than 10% of the fields in the schema defined as realtime. Making most or all fields real-time in a given index negates most performance gains you achieve with real-time updates.

The examples below use the Java Client API. Reviewing the Developing with Attivio guide prior to working with real-time updates is highly recommended.

View incoming links. **>>**

Attivio Glossary , Configure the Attivio Schema , Configuring Query Caches , Deleting Content , REST Ingest API , Simple Query Language , Updating Content , Updating Documents that Match a Query

Real-Time Update Applications

Some real-time updates applications:

- Security / ACL updates (Example)
- Click tracking
- In stock / quantity management
- Online status checks

Configuring Real-Time Updates

Configure real-time updates in the schema by changing the <field> element in the schema to a <realtimeField> element, as shown:

```
<schema name="default">
  <fields default-search-field="content">
    <!-- a normal field -->
    <field name="title" type="string" indexed="true" stored="true" facet="false" />

    <!-- a real-time field with 'realtime' enabled for all options -->
    <realtimeField name="quantity" type="string" facet="true" indexed="true" stored="true"/>

    <!-- a real-time field that is only used for searching -->
    <realtimeField name="flagged" type="string" facet="false" indexed="true" stored="false"/>
  </fields>
</schema>
```

Changing a field from <field> to <realtimeField> requires reindexing.

Performance

Query operations on real-time fields - searching, faceting, sorting and retrieving - are slightly slower than they are on regular fields.

Real-time fields also use more memory than regular fields. The amount of additional memory is proportional to the number of documents in the index. Memory usage is somewhat reduced index optimization. The reduction is proportional to the number of documents updated or deleted since the last optimization.

Developing with Real-Time Updates

Adding and Deleting Real-Time-Update Documents

As with other documents, you add and delete documents containing real-time updates by submitting them via an API with the ADD and DELETE modes.

Updating Documents

You can update documents already in the index by submitting them with the document modes described in the following sections. Use the IngestDocument .setMode() method.

Note that this API does not allow mixing "appends", "removes" and standard partial updates in the same document update.

The ADD Mode

If the submitted document is already present in the index, the ADD mode updates the existing document. It treats real-time fields somewhat differently from regular fields:

- **Regular fields**: The ADD request replaces them in their entirety. If a field is present in the existing document but not in the submitted one, it is deleted.
- **Real-time fields**: The ADD request replaces only those real-time fields that are present in the submitted document. If a field is present in the existing document but not in the submitted one, it remains in the document with its current value.

Java Client API Example:

```
// Initial Document Update
IngestDocument doc = new IngestDocument("documentId", DocumentMode.ADD);
doc.setField("tags", "Initial Value");
// feed document
```

The document will contain the value "Initial Value" in the tags field.

The PARTIAL Mode

The PARTIAL mode updates only the real-time fields in an existing document. Like the ADD request, it replaces only those real-time fields present in the submitted document. If a field is present in the existing document but not in the submitted one, the PARTIAL request leaves it in the document with its current value.

Java Client API Example:

```
// Partial Update (replace previous value)
IngestDocument doc = new IngestDocument("documentId", DocumentMode.PARTIAL);
doc.setField("tags", "A", "B", "C");
// feed document
```

The document now contains values "A", "B" and "C" for the tags field.

The PARTIAL_APPEND_VALUES Mode

Update a document, uniquely appending values to multi-value fields.

Java Client API Example:

```
 // Append values
IngestDocument doc = new IngestDocument("documentId", DocumentMode.PARTIAL_APPEND_VALUES);
doc.setField("tags", "B", "C", "D");
// Feed Document
```

Document now contains values "A", "B", "C", "D" for the tags field.

The PARTIAL_DELETE_VALUES Mode

Update a document, removing matching values from multi-value fields.

Java Client API Example:

```
 // Remove values
IngestDocument doc = new IngestDocument("documentId", DocumentMode.PARTIAL_REMOVE_VALUES);
doc.setField("tags", "B", "C");
// Feed document
```

Document now contains values "A", "D" for the tags field.

Deleting Real-Time Fields

Java Client API Example:

```
IngestDocument doc = new IngestDocument("1");
doc.setField( new StringField("flagged") ); // Remove "flagged" field from document "1" by adding field with no
values
doc.setMode(DocumentMode.PARTIAL);
myDocumentClient.feed(doc);
// ....add any other documents

// now make the changes live in the index (see explanation of REFRESH below)
myDocumentClient.refresh();
```

Refresh and Commit - Making Real-Time Updates Visible

Real-time updates do not become visible until a `COMMIT` or `REFRESH` request executes. The `COMMIT` operation makes visible all the changes made to both regular and real-time fields. The `REFRESH` operation makes visible only the changes made to real-time fields, but is significantly faster.

⚠ The `REFRESH` operation does not guarantee persistence of index changes in the event of a JVM or system crash. Any changes received since the last `COMMIT` operation may be lost in the event of a unexpected system crash.

Bulk Updates to Real-Time Fields

Use the BulkUpdate message to apply the same field value updates to all selected documents. You can use this message to update real-time fields. This message contains a template document to which you can add fields. The message also contains a list of document IDs and/or a query for selecting the documents to which you apply the update.

The query specified selects only previously committed documents.

Java Client API Example:

```
BulkUpdate update = new BulkUpdate(DocumentMode.PARTIAL);

// Add some documents to apply this update to
update.addDocumentId("doc1");
update.addDocumentId("doc2");

// Set a query to select additional documents to apply this update to
update.setQuery("table:foo", QueryLanguages.SIMPLE);
// set query workflow so query will be parsed/tokenized prior to sending to engine
update.setQueryWorkflowQueue("defaultQuery");

// Set the fields to update (these must be real-time fields)
update.getDocument().setField("flagged", "1");
update.getDocument().setField("online", "0");

// Send the update and refresh
myDocumentClient.send(update);
myDocumentClient.refresh();
```

The BulkUpdate

message supports the following document modes: DocumentMode.PARTIAL and DocumentMode.DELETE.

Bulk Updates by ID and Partition Routing

If your index has partition routing configured to operate on a field other than the document ID, you must specify a value for the routing field when performing bulk updates by document ID. Failure to include a value for the routing field will cause Attivio to route documents to index partitions based on document-ID hash values (the default non-routing behavior).

As an alternative, you can perform a bulk update by query, since the selection-by-query mechanism will ensure that updates are routed to the correct index partitions for the existing documents. You can mimic a bulk update by ID by specifying a query of this form, replacing the various `<ID_*>` placeholders with your document IDs:

```
.id:OR(<ID_1>, <ID_2>, <ID_3>, … ,<ID_N>)
```

BulkUpdate and Workflows

The BulkUpdate message, in general, need not go through the full ingest workflow. Creating a small custom workflow for sending in bulk updates may be desriable. This workflow at a minimum requires a tokenizer and indexer stage.