# Rya Temporal Indexing

●●●

5/18/2016

A repository to store, index, and retrieve Statements based on time.
Also, a little about geospatial and full text indexing

## Temporal Queries

Enable the temporal indexer: It defaults to off, but this turns it on:

```
conf.set(ConfigUtils.USE_TEMPORAL, "true");
```

SPARQL using the temporal index:

```
PREFIX tempo: <tag:rya-rdf.org,2015:temporal#>
SELECT ?subj ?time
{
  ?subj <http://schema.org/ birthDate> ?time .
  FILTER(tempo:after(?time, '1825-01-01') ) .
  FILTER(tempo:before(?time, '2010-01-01T01:01:00Z') ) .
}
```

Equivalently, we could replace the two filters above with an interval:

```
  FILTER(tempo:insideInterval(?time, '[1825-01-01, 2010-01-01T01:01:00Z]'))
```

Typical predicates:

Owl time http://www.w3.org/2006/time#**inXSDDateTime**   http://schema.org/**startDate** and **endDate**

Timeline http://purl.org/NET/c4dm/timeline.owl#**at**

# Index Implementation

Temporal Instant is `OWL xsd:dateTime`, `xsd:dateTime` and RFC3339 a subset of ISO 8601 using the Joda-Time API, the basis for the new Java 8 java.time API

Temporal Interval is **begin-instant** + **end-instant** -- literal: "`[2001-01-01,2001-01-01]`"

Relations implemented as filters in SPARQL:

```
Instant {before, equals, after} Instant
Instant {before, after, inside} Interval
Instant {hasBeginning, hasEnd}  Interval
Interval {before, equals,  after}  Interval
```

OWL-Time also has these interval relations:

```
Interval {Meets, Overlaps, Starts, During, Finishes} Interval
```

Important classes in `rya.indexing` (source code)

```
IndexingFunctionRegistry  mvm.rya.indexing               -- Register Filters
TemporalTupleSet  mvm.rya.indexing                             -- Node for temporal expressions, delegates
RyaSecondaryIndexer                                            -- Interface for all Indexers
TemporalIndexer mvm.rya.indexing                               -- interface for all temporal indexers
NullTemporalIndexer mvm.rya.accumulo.mr                        -- Needed when indexing is turned off.
AccumuloTemporalIndexer  mvm.rya.indexing.accumulo.temporal -- the accumulo implementation.
MongoTemporalIndexer (TODO) mvm.rya.indexing.mongodb.temporal
```

# Ingest: Index Update

Summary: if Object is parsable as a DateTime, then:

Parse, normalize, hash, Serialize, and store as 3 or 5 records: **SPO**, **PO**, and **O**, **begin**, **end** in the temporal index table

Row Keys are in these two forms, where [x] denotes x is optional:

- `rowkey = constraintPrefix datetime`
- `rowkey = datetime 0x/00 uniquesuffix`
- `constraintPrefix = 0x/00 hash([subject][predicate])`
- `uniquesuffix = some bytes to make it unique, like hash(statement).`
- `datetime = normalized date time with Z timezone`

Column Qualifier is one of:

"spo" "po" "so" "o" "begin" "end"

Column Family is the named graph

# Query: Index Access

FILTER function Before()

```
  public CloseableIteration<Statement, QueryEvaluationException> queryInstantBeforeInstant(
              final TemporalInstant queryInstant, final StatementContraints constraints)
              throws QueryEvaluationException {
      // get rows where the repository time is before the given time.
      final Query query = new Query() {
              @Override
              public Range getRange(final KeyParts keyParts) {
              Text start= null;
              if (keyParts.constraintPrefix != null ) {
                      start = keyParts.constraintPrefix;   // <-- start specific logic
              } else {
                      start = new Text(KeyParts.HASH_PREFIX_FOLLOWING);
              }
              final Text endAt = keyParts.getQueryKey();              // <-- end specific logic
              return new Range(start, true, endAt, false);
              }
      };
      final ScannerBase scanner = query.doQuery(queryInstant, constraints);
      return getContextIteratorWrapper(scanner, constraints.getContext());
  }
```

## Geospatial Index

Convert from sparql to Geomesa compare functions.

## Full Text index

Uses useekm API

## Future Potential

Future extensions:

OWL specific indexing:

- beginning time + Duration = interval
- ending time + Duration = interval

TimeLine specific indexing:

- `timeline:origin` in universal timeline, convert event times from given timeline to universal timeline.