Getting the Most from MarkLogic Semantics

John Snelson, Principal Engineer, MarkLogic
Stephen Buxton, Sr. Director, Product Management, MarkLogic
Introduction
Semantics Is: A New Way to Organize Data

Data is stored in RDF triples expressed as:

- **Subject**: John Smith : livesIn : London
- **Object**: London : isIn : England

Query with SPARQL, gives us simple lookup .. and more!

Find people who live in (a place that's in) England
Triples Alongside Documents

- Senior Manager
- Compliance Officer
- User1
- rank
- role
- basedIn
- runs
- runsOn
- requires
- accesses
- Cluster1
- TopSecret
- Database1
- Geneva
- High risk person
- App1
- Cluster1 runsOn TopSecret requires Database1 runs Cluster1 requires Cluster1
Triples About Documents – Extended Metadata

Senior Manager

rank

basedIn

Geneva

User1

role

User1 runs

App1 requires

Cluster1

runsOn

accesses

TopSecret

requires

Database1

accesses

Cluster1

runsOn

Compliance Officer

2016-12-31 expires

Delaware jurisdiction

Thread and Cs Ts and Cs

language

English

format

JSON

App1 runs

order

User1 basedIn

Compliance Officer rank
Triples About Documents - Integration

Vendor equivalent Seller equivalent Provider

App1 order format

XML

License

Computer

Asset
<order id="12345">
   <VENDOR>Acme Corp</VENDOR>
   <payment>
      <amount>3427</amount>
      <unit>USD</unit>
      <period>annual</period>
   </payment>
   <sem:triple>
      <sem:object>2016-12-31</sem:object>
   </sem:triple>
   <sem:triple>
      <sem:predicate>http://youruri.com/predicates/TsAndCs</sem:predicate>
      <sem:object>http://youruri.com/terms/34567</sem:object>
   </sem:triple>
   <description> .... </description>
</order>
Set of Triples with XML [JSON] annotation

```xml
<userInfo>
  <source>myApp44</source>
  <confidence>100</confidence>
  <location>37.52 -122.25</location>
  <icd9-proc-code>1111</icd9-proc-code>
  <temporal>
    <systemStart/> <systemEnd/>
    <validStart>2014-04-03T11:00:00</validStart>
    <validEnd>2014-04-03T16:00:00</validEnd>
  </temporal>
  ...
  <sem:triple>
    <sem:subject>http://youruri.com/users/11111</sem:subject>
    <sem:object>http://youruri.com/applications/1111</sem:object>
  </sem:triple>
  <sem:triple>
    <sem:subject>http://youruri.com/users/11111</sem:subject>
    <sem:object>http://youruri.com/applications/3333</sem:object>
  </sem:triple>
</userInfo>
```
Semantics Performance at Scale

CREDITS
Balvinder Dang
Ed Thomas
James Kerr
Tom Ternquist
Semantics Performance at Scale

- Note: not all Semantics use cases are “at scale”
- Some uses require only a small number of triples with simple queries:
  - Semantic Search – expand search terms, concept search
  - Semantic Integration – expand location of search via a Semantic Model
- Here, we can be more generous with joins, inference, and so on
Tip: Use MarkLogic!

- Use MarkLogic capabilities:
  - Security
  - Partition: e.g. Collections
  - Search
    - Filtering
    - Projection

- DON’T: pull all possibly-relevant data into the mid-tier
- DO: use the power of MarkLogic
  - distributed operations / Map-Reduce
  - close to the data
Tip: Scope the query

- Trim results sets early
- Compare with SELECT * FROM [table]
  - It’s worse than that with a triple store or document store
    - SELECT * FROM [the whole database]

- See “Use MarkLogic”: partitioning
- Advanced: get smart about keeping like-triples in the same document
Tip: Documents for Entities; Triples for Facts, Relationships

- It’s better that way!
- It’s more efficient
  - Keep entity information together
  - Reduce joins on query AND retrieval
- … some clever tricks if you know where/how triples are stored
Tip: Documents for Entities; Triples for Facts, Relationships

- **S. Shady** is member of **D12**
- **Blue Van** is rented by S. Shady
- S. Shady claims credit for activity associated with **Noise Comp.**
- Activity is in the vicinity of **Blue Van**
- Activity is a subclass of **Suspicious Activity**
- Activity occurred on **31st**

---

© COPYRIGHT 2017 MARKLOGIC CORPORATION. ALL RIGHTS RESERVED.
Tip: Documents for Entities; Triples for Facts, Relationships

- Person
  - is member of Organization
  - seen around Event

- Organization
  - suspected in Event
  - claims credit for Person

- Asset
  - owns Person
  - material support Event

- Event
  - seen around Asset

Entities

Relationships
Tip: Inference

Inference is *powerful* and *convenient*, but can be *expensive*

- Scope the query
- Consider SPARQL-based inference
SPARQL Based Inference

- Inference ruleset:
  - Automatically expands rdf:type

- SPARQL Based Inference:
  - Use a property path query

```sql
// With SPARQL Based Inference
sem.sparql(""
   SELECT ?uri {
   ?s rdf:type/\rdfs:subClassOf* fs:Trade.
   ?s rdfs:isDefinedBy ?uri
   } LIMIT 10");
```

```sql
// With SPARQL Based Inference
sem.sparql(""
   SELECT ?uri {
   ?s rdf:type fs:Trade .
   ?s rdfs:isDefinedBy ?uri
   } LIMIT 10",
   [],[],
   sem.rulesetStore(
      "subClassOf.rules",sem.store())
   )
```
Tip: Inference

Inference is powerful and convenient, but can be expensive

- Scope the query
- Consider SPARQL-based inference
- Consider materialization (SPARQL CONSTRUCT)
Tips: detailed [1]

- Use MarkLogic indexes to scope a query
  - Collection query (or SPARQL FROM) to partition the RDF space
  - Put ontologies and other lookup/mapping triples into their own graphs/collections
  - Consider pushing-down some SPARQL FILTERs to the document
- Look for joins that can be eliminated by materializing those relationships at load/update time
  - Think of this as *denormalizing* triples
  - “joins are free … conceptually”
- “Materialize” often-queried elements (in documents) consistently for better indexing
Tips: detailed [2]

- Project the result:
  - From SPARQL for small results sets:
    - Get the set of documents that match your query using search
    - Return the relevant triples *directly from the index*
  - From documents for large results sets:
    - “get me customers/orders/contracts that …”
    - Fetch document in a single read, no joins
Tips: detailed [3]

- Use the latest version of MarkLogic: perf improvements on minor releases
- Add more memory: allows the optimizer to choose faster plans
- Add more hardware: cluster for parallelization
- Re-use queries: query plan is cached for 5 minutes; use bind variables
- Use MarkLogic built-in functions in SPARQL
- Consider dedup-off option to sem:sparql() [ML9]
  - Avoid dedup processing
  - No effect on results if you have no duplicate triples and/or you use DISTINCT
  - Can make a big difference
Case Studies

- Company names have been obscured, but these are real projects
- Query timings are given for comparison only
Case Study: Educational Publisher
Case Study: Educational Publisher

Central metadata repository to store metadata, product mapping and central rights management using all-RDF

- Semantic enrichment of content: provide bespoke products using intelligent/smart search.
- Easy discovery and re-use of content
- Central rights management
- Use (and extend) standard RDF vocabularies to share metadata, e.g. Dublin Core.
- RDF Multilingual support

- Before: some SPARQL queries were very slow
- Resolution: 4-week exercise to identify and improve slowest queries
- After: performance improvements of up to 100x
Query: find triples where object matches mat?s

Dataset: 6 Million triples

Query: find triples where object matches mat?s

- Regex term filter
- Language filter
- Searchable filter
- UNION Blank Nodes
- Authorization filter based on SHACL
SELECT DISTINCT ?id
WHERE
{

  ?id a ?__type.

  {{
    ?id _propVar1 ?_o2.
    FILTER regex (?_o2, "mat?s"^^<http://www.w3.org/2001/XMLSchema#string>, "i")
    FILTER (langmatches(lang(?_o2), "en") || lang(?_o2) = "")
    FILTER NOT EXISTS { graph <nonsearchable> {?id _propVar1 ?obj } } }
  UNION {
    ?id ?bnodeProp _:b0 .
    _:b0 _propVar1 ?_o2.
    FILTER regex (?_o2, "mat?s"^^<http://www.w3.org/2001/XMLSchema#string>, "i")
    FILTER (langmatches(lang(?_o2), "en") || lang(?_o2) = "")
    FILTER NOT EXISTS { graph <nonsearchable> {_:b0 _propVar1 ?obj } } }
  }

  OPTIONAL { ?id raf:retrievableBy ?__irole } 
  FILTER(!BOUND(?__irole) || ?__irole IN ("metadataReader"))
  OPTIONAL {
    SELECT ?__badShape ?__type {
      ?__badShape sh:scopeClass ?__type.
      MINUS {
        ?__badShape sh:scopeClass ?__type.
        ?__badShape raf:retrievableBy ?userRole
        VALUES ?userRole {"metadataReader"}
      }
      MINUS {
        ?__badShape sh:scopeClass ?__type.
        ?__badShape raf:sorRole ??sorRole
      }
    }
    FILTER (!BOUND(?__badShape))
  }
}
LIMIT 20
Query: find triples where object matches mat?s

Dataset: 6 Million triples

Query: find triples where object matches mat?s

**Timings:**
- Initial: 20 secs
- Use cts:contains instead of regex() in SPARQL: 7 secs
- Use collection query to partition by collections/graphs: 3 secs
- Use a cts:query to partition data further: 0.4 secs
- Overall improvement: 100x
Query: find triples where object matches mat?s

Dataset: 6 Million triples

Query: find triples where object matches mat?s

Next steps:

- Replace UNION Blank Nodes with property path (new in MarkLogic 8)
- Look at using MarkLogic security (index-based)
  - Replace SHACL constraints in each query
  - Remove `FILTER NOT EXISTS { graph <nonsearchable> ...`
Query: GET Description

Dataset: 6 Million triples
Query: Fetch everything you know about X

Timings:
- Initial: 6 secs
- Use named graph/collection and collection-lexicon
- Use cts:triple-range-query to scope by subjects
- Final: 0.2 secs
- Overall improvement: 30x

Next step:
- Consider documents for “everything you know about X”
- Use TDE to index (parts of) documents as triples [ML9]
Performance Exercise

Search Improvements

Other Requests

© COPYRIGHT 2017 MARKLOGIC CORPORATION. ALL RIGHTS RESERVED.
Case Study: Data Store for Clinical Data
Case Study: Data Store for Clinical Data

Data store for clinical data - Organizations, patients, encounters, conditions, medications, etc.

- Data comes in in the form of FHIR messages
- Need to track provenance of every data element in the system
- Enrich clinical data
- Need to be able to query based on one or more ontologies
  - Connect concepts, traverse relationships, etc.
- Role-based access to PHI
- Encrypt and audit access of PHI
- Archive older, less used data
Initial Approach

- All data represented as triples
  - Limited access to document features: security, tiered storage, bitemporal
- Provenance and other metadata about triples via “instantiated predicates”
  - One way to do “reification” in a pure-triples world
  - Every query requires inference OR re-write
# Initial Approach

- All data represented as triples
- Provenance and other metadata about triples via "instantiated predicates"

**Issues:**

- Inferencing required for every query against the ADR
- Limited use of MarkLogic’s powerful multi-model query and analytic capabilities
- Limited use of MarkLogic’s data management capabilities
  - Security, Tiered Storage, Bitemporal
- Many joins to retrieve back a single resource/record (e.g. Patient)
- Complex ETL from entities to triples
<Sidetrack: Metadata about triples>
Metadata about a triple[1]: Reification

- Triple:
  :John :livesIn :London

- Reified triple:
  :triple1234 rdf:type rdf:Statement .
  :triple1234 rdf:subject :John
  :triple1234 rdf:predicate :livesIn
  :triple1234 rdf:object :London

- Now you can say things about this triple:
  :triple1234 :source :patient-record-42

- Downside:
  - 4x triples
  - more complex queries OR inference
Metadata about a triple[2]: Instantiated Predicates

- Triple:
  :John :livesIn :London

- Instantiated Predicate:
  :JohnLivesIn rdfs:type :livesIn
  :John :JohnLivesIn :London

- Now you can say things about this triple:
  :JohnLivesIn :source :patient-record-42
  
  *(need at least 1 more triple to constrain :JohnLivesIn to :John)*

- Downside:
  - At least 3x triples
  - More complex queries OR inference
  - Inference: infer around 100,000 new triples for each query
Metadata about a triple[3]: Embed triple in a document

- **Triple:**
  :John :livesIn :London

- **Now you can say things about this triple:**
  <doc>
  <triple>
    <subject>John</subject>
    <predicate>livesIn</predicate>
    <object>London</object>
  </triple>
  <source>patient-record-42</source>
  </doc>

- **Embedded triple:**
  <doc>
    <triple>
      <subject>John</subject>
      <predicate>livesIn</predicate>
      <object>London</object>
    </triple>
  </doc>

Any metadata: source, confidence, bitemporal, etc
Metadata can be structured
Query with combination query or Optic
<Sidetrack: Metadata about triples>
Initial Approach

- All data represented as triples
- Provenance and other metadata about triples via “instantiated predicates”

Issues:
- Inferencing required for every query against the ADR
- Limited use of MarkLogic’s powerful multi-model query and analytic capabilities
- Limited use of MarkLogic’s data management capabilities
  - Security, Tiered storage, Bitemporal
- Many joins to retrieve back a single resource/record (e.g. Patient)
- Complex ETL from entities to triples
New Approach

Documents plus triples:

- Store incoming messages as documents
- Map data elements into top-level domain entities (Patient, Practitioner, Encounter, etc.) and store these as documents
- Store the triples that go with each entity in the entity document (share management)
- Store enriched / derived triples in the entity documents that they came from
- Capture provenance in XML data structures
- Leverage bitemporal to track changes to entities over time
Unified Query View example for Patient ID

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>EntPatientID</td>
<td>owl:sameAs</td>
<td>../recordTarget/patientClinical/id</td>
<td>HL7 v3 patient ID</td>
</tr>
<tr>
<td>EntPatientID</td>
<td>owl:sameAs</td>
<td>PID-3</td>
<td>HL7 v2 patient ID</td>
</tr>
<tr>
<td>EntPatientID</td>
<td>owl:sameAs</td>
<td>../entry/resource/Patient/id</td>
<td>FHIR patient ID</td>
</tr>
</tbody>
</table>

- EntPatientID is the enterprise patient ID that the API exposes in queries
- The patient ID is labeled and located differently in HL7 v3, HL7 v2 and FHIR
- Ontology triples can be used to expand the search to all the known ID locations in the combined sources
Provenance

- Each entity has metadata / provenance / bitemporal information

```xml
<envelope>
  <metadata>
    <lastUpdatedDateTime>2015-05-25T12:00:03Z</lastUpdatedDateTime>
    <firstCreatedDateTime>2015-05-25T12:00:03Z</firstCreatedDateTime>
    <source>/fhir/message-9876.xml</source>
    <lastUpdatedDateTime>2015-05-25T12:00:03Z</lastUpdatedDateTime>
    <firstCreatedDateTime>2015-05-25T12:00:03Z</firstCreatedDateTime>
  </metadata>
  <original>
    ...
  </original>
</envelope>
```

- Note: Prior versions of entities are kept via temporal collection
Query

- Combination: MarkLogic **CTS** queries to scope **SPARQL** and **Inference**
  - Reduce the set of entities we are interested in by using CTS
  - Use SPARQL to query across entities and concepts
  - Retrieve records as single entity documents without the need for joins

- **SPARQL** for semantic search
  - Query triples (an ontology) to expand a concept search to include related concepts

- **SPARQL** for integration
  - Query triples (an ontology) to expand a search over a canonical patientID to a search over all representations of patientID
Query

- Combiantion: MarkLogic CTS queries to scope SPARQL and Inference
  - Reduce the set of entities we are interested in by using CTS
  - Use SPARQL to query across entities
  - Retrieve records as single entity documents without the need for joins

- SPARQL for semantic search
  - Query triples (an ontology) to expand a concept search to include related concepts

- SPARQL for integration
  - Query triples (an ontology) to expand a search over a canonical patientID to a search over all representations of patientID

Query expansion:
- Expand the value you’re querying for
- Expand the places you look for that value
Example Combination Query

- Find **patients** who have **encounters** of type “outpatient” and date after 1/1/2010 with a **physician** with last name of “ROBERT”

Done in one call to the DB

- CTS query to select encounters with type “outpatient” and date after 1/1/2010 and physicians with last name of “ROBERT”

- SPARQL against the triples from the selected entities to join physicians and encounters and return patient IRIs (as contained in the encounters)

Efficient Read

- Use the patient IRIs to retrieve complete or partial patient entities (one by one or in batches)
Query: find patient info for some condition code

Dataset: 300 MN triples => (30K documents + 22 MN triples)

Query: find patient info for some condition code

Timings:
- Initial, triples-only: **45 secs**
- Triples-only with re-written queries and some materialization: **30 secs**
- Triples + documents, SPARQL with CTS constraints: **0.7 secs**
- overall improvement: **65x perf + bitemporal**
Query: Aggregate patients by condition code, gender

Dataset: 300 MN triples => (30K documents + 22 MN triples)

Query: Aggregate patients by condition code, gender

Timings:

- Initial, triples-only: 1200 secs (8)
- Triples-only with re-written queries and some materialization: 50 secs
- Triples + documents, SPARQL with CTS constraints: 0.9 secs
- overall improvement: 55x perf + bitemporal
Case Study: Leading Global Bank
Case Study: Leading Global Bank

Inventory system for tracking technical assets

- 2 Billion triples
- Queries are heavily graph-traversal
- Documents for triples enrichment, provenance
- Bitemporal audit trail
Performance Summary

- Documents: entities
- Triples: relationships, facts, graphs
- You don’t have to choose just one!
- They go together like ….
Understanding SPARQL Execution
Well-Behaved Query

```prefix : <http://example.org/kennedy/>
select * {
  ?person :last-name ?last .
  filter(?birthPlace = 'Wien')
}
order by ?first ?last```
Executing a Query

1. Parse
2. Initial query plan
3. Cost-based optimization
4. Execution plan
5. Run the plan
Query Plan

- Trace flag "SPARQL AST" [ML7,8,9]
- Trace flag "Optic Plan" [ML9]
- Trace option ("trace=XXXX") to sem:sparql() and xdmp:sql() [ML9]
Query Plan

- Trace flag "SPARQL AST" [ML7,8,9]
- Trace flag "Optic Plan" [ML9]
- Trace option ("trace=XXXX") to sem:sparql() and xdmp:sql() [ML9]
Query Plan
Constraining Condition

```
select * {
    ?person :last-name ?last .
    filter(?birthPlace = 'Wien')
}

order by ?first ?last
```
Statistics

- Trace flag "SPARQL Value Frequencies" [ML7,8,9]
- Trace flag "Optic Statistics" [ML9]
- Trace option ("trace=XXXX") to sem:sparql() and xdmp:sql() [ML9]
- cts:triple-value-statistics() [ML8,9]
Statistics

<triple-value-statistics count="1618" unique-subjects="144" unique-predicates="19" unique-objects="510" xmlns="cts:triple-value-statistics">
<triple-value-entries>
<triple-value-entry count="75">
<triple-value>http://example.org/kennedy/last-name</triple-value>
<subject-statistics count="0" unique-predicates="0" unique-objects="0"/>
<predicate-statistics count="75" unique-subjects="75" unique-objects="36"/>
<object-statistics count="0" unique-subjects="0" unique-predicates="0"/>
</triple-value-entry>
<triple-value-entry count="76">
<triple-value>http://example.org/kennedy/birth-place</triple-value>
<subject-statistics count="0" unique-predicates="0" unique-objects="0"/>
<predicate-statistics count="76" unique-subjects="76" unique-objects="28"/>
<object-statistics count="0" unique-subjects="0" unique-predicates="0"/>
</triple-value-entry>
<triple-value-entry count="51">
<triple-value>http://example.org/kennedy/name</triple-value>
<subject-statistics count="0" unique-predicates="0" unique-objects="0"/>
<predicate-statistics count="51" unique-subjects="51" unique-objects="51"/>
<object-statistics count="0" unique-subjects="0" unique-predicates="0"/>
</triple-value-entry>
<triple-value-entry count="1">
<triple-value datatype="http://www.w3.org/2001/XMLSchema#string">Wien</triple-value>
<subject-statistics count="0" unique-predicates="0" unique-objects="0"/>
<predicate-statistics count="0" unique-subjects="0" unique-objects="0"/>
<object-statistics count="1" unique-subjects="1" unique-predicates="1"/>
</triple-value-entry>
...</triple-value-entries>
</triple-value-statistics>
Query Execution

- Trace flag "SPARQL Execution" [ML7,8,9]
- Trace flag "Optic Execution" [ML9]
- Trace option ("trace=XXXX") to sem:sparql() and xdmp:sql() [ML9]
Query Execution

```xml
<plan:call locking="read-write" xmlns:plan="http://marklogic.com/plan">
  <plan:triple permutation="POS">
    <plan:subject column-index="4"/>
    <plan:predicate operator="=">
      <plan:value>http://example.org/kennedy/name</plan:value>
    </plan:predicate>
    <plan:object column-index="3" operator="=">
      <plan:value datatype="http://www.w3.org/2001/XMLSchema#string">Wien</plan:value>
    </plan:object>
  </plan:triple>
  <plan:ordered-nodup-result>
    <plan:order-spec column-index="3" descending="false"/>
    <plan:order-spec column-index="4" descending="false"/>
  </plan:ordered-nodup-result>
</plan:call>
```
Query Execution

```xml
<plan:call locking="read-write" xmlns:plan="http://marklogic.com/plan">
  <plan:column-constraint>
    <plan:constraint column-index="4" operator="=">
      <plan:value>http://example.org/kennedy/place51</plan:value>
    </plan:constraint>
    <plan:triple permutation="OPS">
      <plan:subject column-index="0"/>
      <plan:predicate operator="=">
        <plan:value>http://example.org/kennedy/birth-place</plan:value>
      </plan:predicate>
      <plan:object column-index="4"/>
    </plan:triple>
  </plan:column-constraint>
  <plan:ordered-nodup-result>
    <plan:order-spec column-index="4" descending="false"/>
    <plan:order-spec column-index="0" descending="false"/>
  </plan:ordered-nodup-result>
</plan:call>
```
Query Execution

```xml
<plan:call locking="read-write" xmlns:plan="http://marklogic.com/plan">
  <plan:column-constraint>
    <plan:constraint column-index="0" operator="=">
      <plan:value>http://example.org/kennedy/person27</plan:value>
    </plan:constraint>
  </plan:column-constraint>
  <plan:triple permutation="PSO">
    <plan:subject column-index="0"/>
    <plan:predicate operator="=">
      <plan:value>http://example.org/kennedy/last-name</plan:value>
    </plan:predicate>
    <plan:object column-index="2"/>
  </plan:triple>
  <plan:column-constraint>
    <plan:ordered-nodup-result>
      <plan:order-spec column-index="0" descending="false"/>
      <plan:order-spec column-index="2" descending="false"/>
    </plan:ordered-nodup-result>
  </plan:column-constraint>
</plan:call>
```
Query Execution

```
<plan:call locking="read-write" xmlns:plan="http://marklogic.com/plan">
  <plan:column-constraint>
    <plan:constraint column-index="0" operator="=">
      <plan:value>http://example.org/kennedy/person27</plan:value>
    </plan:constraint>
  </plan:column-constraint>
  <plan:triple permutation="POS">
    <plan:subject column-index="0"/>
    <plan:predicate operator="=">
      <plan:value>http://example.org/kennedy/first-name</plan:value>
    </plan:predicate>
    <plan:object column-index="1"/>
  </plan:triple>
  <plan:ordered-nodup-result>
    <plan:order-spec column-index="1" descending="false"/>
    <plan:order-spec column-index="0" descending="false"/>
  </plan:ordered-nodup-result>
</plan:call>
```
Problem Query

```sparql
select * {
  ?person :last-name ?last .
  filter(?parentBirth < '1890')
}
```
Improvement: Add Cardinality Hints

```sparql
select * {
  ?person :last-name ?last .
  { select * {
      filter(?parentBirth < '1890')
    } limit 1 }
}
```
Improvement: Add Constraining Condition

```
select * {
  ?person :last-name ?last .
  ?person :birth-place [:name 'Boston'] .
  filter(?parentBirth < '1890')
}
```
Understanding the Triple Caches
Triple Patterns use the Triple Index

- Designed to look up triple patterns
- Facilitates fast joins
- 4 triple permutations
- Not memory mapped - cached for performance
- Works seamlessly with other indexes
## Triple Index

<table>
<thead>
<tr>
<th>subject</th>
<th>predicate</th>
<th>object</th>
<th>doc ID</th>
<th>position</th>
</tr>
</thead>
<tbody>
<tr>
<td>:person4</td>
<td>:first-name</td>
<td>“John”</td>
<td>11</td>
<td>5 - 9</td>
</tr>
<tr>
<td>:person5</td>
<td>:birth-year</td>
<td>1929</td>
<td>9</td>
<td>13 - 17</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Triple Data and Triple Values

<table>
<thead>
<tr>
<th>subject</th>
<th>predicate</th>
<th>object</th>
<th>doc ID</th>
<th>position</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3</td>
<td>6</td>
<td>11</td>
<td>5 - 9</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>25 - 40</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>7</td>
<td>9</td>
<td>13 - 17</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ordinal</th>
<th>tag</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>IRI</td>
<td>:first-name</td>
</tr>
<tr>
<td>4</td>
<td>IRI</td>
<td>:person4</td>
</tr>
<tr>
<td>5</td>
<td>IRI</td>
<td>:person5</td>
</tr>
<tr>
<td>6</td>
<td>STRING</td>
<td>“John”</td>
</tr>
<tr>
<td>7</td>
<td>DECIMAL</td>
<td>1929</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
Triple and Triple Value Cache

- D-node caches
- Partitioned for lock contention
- Configurable maximum size
- Grows and shrinks
  - No up-front memory allocation
  - Trickle removed after 30s inactivity (user configurable)
- Flexibility
  - Size according to importance of triple based queries
  - Size for working set
  - Size big “in case”, but rely on it normally being small
Triple and Triple Value Cache

- xdmp:query-meters()
  - cache hits/misses for a query
- xdmp:forest-status()
  - cache hits/misses/hit rate/miss rate for each stand
- xdmp:cache-status()
  - Percentage busy/used/free by cache partition
- Admin UI database status and forest status
- Metering
Understanding Optimization
SPARQL Optimization

- Cost estimation:
  - Column cardinality estimates
  - Sort order static analysis

- Query plan mutations:
  - Multiple orders available in the triple index
  - Multiple join implementations
  - Join re-ordering

- Simulated annealing:
  - Guided randomized search for a good query plan
Optimization Levels

- Default optimization level is 1
- Larger queries may need a longer optimization process
- Optimization levels of 2, 3, 4 etc. are possible
- Optimization level of 0 only uses simple heuristic based optimization
- Trade off between planning and doing

```sparql
sem:sparql("...", ()
  "optimize=2",
  cts:directory-query("/triples/")
)
```
Improvements in MarkLogic 9
What the New Optimizations Do For You!

- Faster ORDER BY from the index with a known predicate
  - POS permutation in the triple index
- Faster descending ORDER BY
  - descending order triple index access
- Faster multi-column ORDER BY
  - partial sort uses major sort order from the triple index
- Faster range-based triple index access
  - both upper and lower bound
- Faster grouping
  - hash based grouping
- Faster disk reads
  - especially on Windows
Getting the Most from MarkLogic Semantics

- Introduction
- Performance at scale
  - Tips – general, detailed
  - Case Studies – Educational publisher, Data store for clinical data, Global bank
- Under the hood
  - Understanding SPARQL execution
  - Understanding the Triple Caches
  - Understanding optimization
  - Improvements in MarkLogic 9
Questions?