A Scalable Scheme for Bulk Loading Large RDF Graphs into Oracle

Souripriya Das, Eugene Inseok Chong, Zhe Wu, Melliyal Annamalai, Jagannathan Srinivasan
Outline

• Storage Schema
• Bulk-load Scheme
• Performance Study
• Summary
• Future Work
Semantic Technology Stack

User Interface & Applications

Trust

Proof

Unifying Logic

Ontology: OWL

Rule: RIF

Data interchange: RDF

Query: SPARQL

RDFS

XML

URI/IRI

Standards based

Presented by Souri Das at ICDE 2008

http://www.w3.org/2007/03/layerCake.svg
Form of Data to be loaded
Form of Data to be loaded

• RDF triple represents a *directed labeled* edge
Form of Data to be loaded

- RDF triple represents a **directed labeled** edge

<http://purl.uniprot.org/uniprot/Q4U9M9>
Form of Data to be loaded

- RDF triple represents a *directed labeled* edge

<http://purl.uniprot.org/uniprot/Q4U9M9>

<http://purl.uniprot.org/core/Protein>
Form of Data to be loaded

• RDF triple represents a **directed labeled** edge

<http://purl.uniprot.org/uniprot/Q4U9M9>

<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>

<http://purl.uniprot.org/core/Protein>
Form of Data to be loaded

• RDF triple represents a *directed labeled* edge

```
<http://purl.uniprot.org/uniprot/Q4U9M9>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://purl.uniprot.org/core/Protein>
```

• RDF triple *set* represents a directed labeled graph

```
# uniprot:... rdf:type :Protein
<http://purl.uniprot.org/uniprot/P15711> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://purl.uniprot.org/core/Protein>.
<http://purl.uniprot.org/uniprot/P18646> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://purl.uniprot.org/core/Protein>.
<http://purl.uniprot.org/uniprot/P13813> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://purl.uniprot.org/core/Protein>.
# other attributes
```
**Simple schema for RDF storage**

<table>
<thead>
<tr>
<th>SUBJECT</th>
<th>PREDICATE</th>
<th>OBJECT</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://purl.uniprot.org/uniprot/Q4U9M9">http://purl.uniprot.org/uniprot/Q4U9M9</a></td>
<td><a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#type">http://www.w3.org/1999/02/22-rdf-syntax-ns#type</a></td>
<td><a href="http://purl.uniprot.org/core/Protein">http://purl.uniprot.org/core/Protein</a></td>
</tr>
<tr>
<td><a href="http://purl.uniprot.org/uniprot/P15711">http://purl.uniprot.org/uniprot/P15711</a></td>
<td><a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#type">http://www.w3.org/1999/02/22-rdf-syntax-ns#type</a></td>
<td><a href="http://purl.uniprot.org/core/Protein">http://purl.uniprot.org/core/Protein</a></td>
</tr>
<tr>
<td><a href="http://purl.uniprot.org/uniprot/Q43495">http://purl.uniprot.org/uniprot/Q43495</a></td>
<td><a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#type">http://www.w3.org/1999/02/22-rdf-syntax-ns#type</a></td>
<td><a href="http://purl.uniprot.org/core/Protein">http://purl.uniprot.org/core/Protein</a></td>
</tr>
<tr>
<td><a href="http://purl.uniprot.org/uniprot/P18646">http://purl.uniprot.org/uniprot/P18646</a></td>
<td><a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#type">http://www.w3.org/1999/02/22-rdf-syntax-ns#type</a></td>
<td><a href="http://purl.uniprot.org/core/Protein">http://purl.uniprot.org/core/Protein</a></td>
</tr>
<tr>
<td><a href="http://purl.uniprot.org/uniprot/P13813">http://purl.uniprot.org/uniprot/P13813</a></td>
<td><a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#type">http://www.w3.org/1999/02/22-rdf-syntax-ns#type</a></td>
<td><a href="http://purl.uniprot.org/core/Protein">http://purl.uniprot.org/core/Protein</a></td>
</tr>
<tr>
<td><a href="http://purl.uniprot.org/uniprot/Q4U9M9">http://purl.uniprot.org/uniprot/Q4U9M9</a></td>
<td><a href="http://www.w3.org/2000/01/rdf-schema#seeAlso">http://www.w3.org/2000/01/rdf-schema#seeAlso</a></td>
<td><a href="http://purl.uniprot.org/interpro/IPR007480">http://purl.uniprot.org/interpro/IPR007480</a></td>
</tr>
</tbody>
</table>

Desktop Loading speed (without any indexes) = 100 to 400 million/hr
RDF Data Characteristics

[Based upon the benchmark datasets we have seen]

• RDF terms are long
• Same RDF term gets used in many triples
• Distinct predicate count in an RDF graph is small
• Many RDF terms share common long prefixes
• Distinct RDF terms is less than $\frac{1}{4}$ of number of triples
RDF Data Characteristics

Based upon the benchmark datasets we have seen

- RDF terms are long
- Same RDF term gets used in many triples
- Distinct predicate count in an RDF graph is small
- Many RDF terms share common long prefixes
- Distinct RDF terms is less than $\frac{1}{4}$ of number of triples

<table>
<thead>
<tr>
<th>Component of Triples</th>
<th>LUBM50: #triples=6.9m</th>
<th>UniProt: #triples=6.9m</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#Distinct</td>
<td>Avg Length</td>
</tr>
<tr>
<td>SUBJECT</td>
<td>1.1 million</td>
<td>64</td>
</tr>
<tr>
<td>PREDICATE</td>
<td>18</td>
<td>62</td>
</tr>
<tr>
<td>OBJECT</td>
<td>0.8 million</td>
<td>46</td>
</tr>
</tbody>
</table>
Drawbacks of Simple Schema

- Table and index storage size is high
- Access to data is slow
- Query and inference is slow
**Normalized schema for RDF storage**

<table>
<thead>
<tr>
<th>LexVal</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://purl.uniprot.org/uniprot/Q4U9M9">http://purl.uniprot.org/uniprot/Q4U9M9</a></td>
<td>100</td>
</tr>
<tr>
<td><a href="http://purl.uniprot.org/uniprot/P15711">http://purl.uniprot.org/uniprot/P15711</a></td>
<td>200</td>
</tr>
<tr>
<td><a href="http://purl.uniprot.org/uniprot/P15711">http://purl.uniprot.org/uniprot/P15711</a></td>
<td>300</td>
</tr>
<tr>
<td><a href="http://purl.uniprot.org/uniprot/P18646">http://purl.uniprot.org/uniprot/P18646</a></td>
<td>400</td>
</tr>
<tr>
<td><a href="http://purl.uniprot.org/uniprot/P13813">http://purl.uniprot.org/uniprot/P13813</a></td>
<td>500</td>
</tr>
<tr>
<td><a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#type">http://www.w3.org/1999/02/22-rdf-syntax-ns#type</a></td>
<td>600</td>
</tr>
<tr>
<td><a href="http://www.w3.org/2000/01/rdf-schema#seeAlso">http://www.w3.org/2000/01/rdf-schema#seeAlso</a></td>
<td>700</td>
</tr>
<tr>
<td><a href="http://purl.uniprot.org/core/Protein">http://purl.uniprot.org/core/Protein</a></td>
<td>800</td>
</tr>
<tr>
<td><a href="http://purl.uniprot.org/interpro/IPR007480">http://purl.uniprot.org/interpro/IPR007480</a></td>
<td>900</td>
</tr>
</tbody>
</table>

Presented by Souri Das at ICDE 2008
Normalized schema for RDF storage

<table>
<thead>
<tr>
<th>LexVal</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://purl.uniprot.org/uniprot/Q4U9M9">http://purl.uniprot.org/uniprot/Q4U9M9</a></td>
<td>100</td>
</tr>
<tr>
<td><a href="http://purl.uniprot.org/uniprot/P15711">http://purl.uniprot.org/uniprot/P15711</a></td>
<td>200</td>
</tr>
<tr>
<td><a href="http://purl.uniprot.org/uniprot/P15711">http://purl.uniprot.org/uniprot/P15711</a></td>
<td>300</td>
</tr>
<tr>
<td><a href="http://purl.uniprot.org/uniprot/P18646">http://purl.uniprot.org/uniprot/P18646</a></td>
<td>400</td>
</tr>
<tr>
<td><a href="http://purl.uniprot.org/uniprot/P13813">http://purl.uniprot.org/uniprot/P13813</a></td>
<td>500</td>
</tr>
<tr>
<td><a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#type">http://www.w3.org/1999/02/22-rdf-syntax-ns#type</a></td>
<td>600</td>
</tr>
<tr>
<td><a href="http://www.w3.org/2000/01/rdf-schema#seeAlso">http://www.w3.org/2000/01/rdf-schema#seeAlso</a></td>
<td>700</td>
</tr>
<tr>
<td><a href="http://purl.uniprot.org/core/Protein">http://purl.uniprot.org/core/Protein</a></td>
<td>800</td>
</tr>
<tr>
<td><a href="http://purl.uniprot.org/interpro/IPR007480">http://purl.uniprot.org/interpro/IPR007480</a></td>
<td>900</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Subj_id</th>
<th>Pred_id</th>
<th>Obj_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>600</td>
<td>800</td>
</tr>
<tr>
<td>200</td>
<td>600</td>
<td>800</td>
</tr>
<tr>
<td>300</td>
<td>600</td>
<td>800</td>
</tr>
<tr>
<td>400</td>
<td>600</td>
<td>800</td>
</tr>
<tr>
<td>500</td>
<td>600</td>
<td>800</td>
</tr>
<tr>
<td>100</td>
<td>700</td>
<td>900</td>
</tr>
</tbody>
</table>
ID Generation: Seq-based or hash-based?
ID Generation: Seq-based or hash-based?

- Advantages of hash-based scheme
  - Virtually no history-dependence (which makes it easier to combine RDF data from multiple stores)
  - No need to access globally unique IDs from a sequence
  - IDs can be computed just by use of hashing function and consulting the (rarely populated) collisions list
  - Allows overlapping LexValues processing and IdTriples processing (with adjustments in case of collision)
ID Generation: Seq-based or hash-based?

- Advantages of hash-based scheme
  - Virtually no history-dependence (which makes it easier to combine RDF data from multiple stores)
  - No need to access globally unique Ids from a sequence
  - Ids can be computed just by use of hashing function and consulting the (rarely populated) collisions list
  - Allows overlapping LexValues processing and IdTriples processing (with adjustments in case of collision)

- We chose the hash-based scheme
  - involving a powerful but efficient (native) hash function
  - included collision detection and resolution
RDF Data Model
RDF Data Model

• RDF terms (values) can be of three types
  • URIs ➔ <…>
  • Blank Nodes ➔ _:x
RDF Data Model

• RDF terms (values) can be of three types
  • URIs ➔ <…>
  • Blank Nodes ➔ _:x

• Positional constraints
  • Predicate ➔ must be a URI
  • Subject ➔ must be a URI or a blank node
  • Object ➔ must be a URI, blank node, or literal
RDF Data Model

- RDF terms (values) can be of three types
  - URIs ➞ `<…>`
  - Blank Nodes ➞ `_:x`

- Positional constraints
  - Predicate ➞ must be a URI
  - Subject ➞ must be a URI or a blank node
  - Object ➞ must be a URI, blank node, or literal

- Set property
RDF Data Model

- RDF terms (values) can be of three types
  - URIs $\rightarrow$ `<…>`
  - Blank Nodes $\rightarrow$ `_:x`

- Positional constraints
  - Predicate $\rightarrow$ must be a URI
  - Subject $\rightarrow$ must be a URI or a blank node
  - Object $\rightarrow$ must be a URI, blank node, or literal

- Set property
- Value-point based equivalence
## Functionality Choices

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Oracle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value point equivalence (via Canonicalization)</td>
<td>YES</td>
</tr>
<tr>
<td>Duplicate triples elimination</td>
<td>YES</td>
</tr>
<tr>
<td>Fidelity (preserving user-specified form of data)</td>
<td>YES</td>
</tr>
<tr>
<td>Graph level access control</td>
<td>YES</td>
</tr>
<tr>
<td>Triple level ancillary information</td>
<td>YES</td>
</tr>
<tr>
<td>Compression of data and indexes</td>
<td>YES</td>
</tr>
</tbody>
</table>
Design choices
Design choices

- *Generic storage structure* avoids dependency on specific RDF schema
Design choices

- **Generic storage structure** avoids dependency on specific RDF schema
- **Normalized representation** reduces space needed for storing triples and indexes and helps query and inference performance
Design choices

- **Generic storage structure** avoids dependency on specific RDF schema
- **Normalized representation** reduces space needed for storing triples and indexes and helps query and inference performance
  - LexValues (lexval, id)
Design choices

- **Generic storage structure** avoids dependency on specific RDF schema

- **Normalized representation** reduces space needed for storing triples and indexes and helps query and inference performance
  - LexValues (lexval, id)
  - IdTriples (subj_id, pred_id, obj_id)
Design choices

• **Generic storage structure** avoids dependency on specific RDF schema

• **Normalized representation** reduces space needed for storing triples and indexes and helps query and inference performance
  - LexValues (lexval, id)
  - IdTriples (subj_id, pred_id, obj_id)

• **Hash-based id generation scheme** allows translation of Triples to IdTriples using the hash function and consulting a (almost always empty) collisions list
Design choices

- **Generic storage structure** avoids dependency on specific RDF schema
- **Normalized representation** reduces space needed for storing triples and indexes and helps query and inference performance
  - LexValues (lexval, id)
  - IdTriples (subj_id, pred_id, obj_id)
- **Hash-based id generation scheme** allows translation of Triples to IdTriples using the hash function and consulting a (almost always empty) collisions list
  - LexValues (lexval, id, lexval_ext)
Design choices

- **Generic storage structure** avoids dependency on specific RDF schema
- **Normalized representation** reduces space needed for storing triples and indexes and helps query and inference performance
  - LexValues (lexval, id)
  - IdTriples (subj_id, pred_id, obj_id)
- **Hash-based id generation scheme** allows translation of Triples to IdTriples using the hash function and consulting a (almost always empty) collisions list
  - LexValues (lexval, id, lexval_ext)
- **Canonical values and Ids** are added to facilitate value-point based equivalence handling
Design choices

- **Generic storage structure** avoids dependency on specific RDF schema
- **Normalized representation** reduces space needed for storing triples and indexes and helps query and inference performance
  - LexValues (lexval, id)
  - IdTriples (subj_id, pred_id, obj_id)
- **Hash-based id generation scheme** allows translation of Triples to IdTriples using the hash function and consulting a (almost always empty) collisions list
  - LexValues (lexval, id, lexval_ext)
- **Canonical values and Ids** are added to facilitate value-point based equivalence handling
  - LexValues (lexval, id, canon_id, lexval_ext, canon_ext)
Design choices

- **Generic storage structure** avoids dependency on specific RDF schema
- **Normalized representation** reduces space needed for storing triples and indexes and helps query and inference performance
  - LexValues (lexval, id)
  - IdTriples (subj_id, pred_id, obj_id)
- **Hash-based id generation scheme** allows translation of Triples to IdTriples using the hash function and consulting a (almost always empty) collisions list
  - LexValues (lexval, id, lexval_ext)
- **Canonical values and Ids** are added to facilitate value-point based equivalence handling
  - LexValues (lexval, id, canon_id, lexval_ext, canon_ext)
  - IdTriples (subj_id, pred_id, obj_id, canon_obj_id)
Design choices (continued)
Design choices (continued)

• **Duplicate elimination via Unique constraint** (enforces RDF set property)
  • UNIQUE (subj_id, pred_id, canon_obj_id)
Design choices (continued)

• Duplicate elimination via Unique constraint (enforces RDF set property)
  • UNIQUE (subj_id, pred_id, canon_obj_id)

• Break URIs into prefix and suffix to compress out long common prefixes of distinct URIs (via use of table compression in Oracle)
  • LexValues (lexval_prefix, lexval_suffix, …)
Design choices (continued)

- Duplicate elimination via Unique constraint (enforces RDF set property)
  - UNIQUE (subj_id, pred_id, canon_obj_id)
- Break URIs into prefix and suffix to compress out long common prefixes of distinct URIs (via use of table compression in Oracle)
  - LexValues (lexval_prefix, lexval_suffix, ...)
- Index key-prefix compression compacts multiple occurrences of same key prefix
Design choices (continued)

- Duplicate elimination via Unique constraint (enforces RDF set property)
  - UNIQUE (subj_id, pred_id, canon_obj_id)

- Break URIs into prefix and suffix to compress out long common prefixes of distinct URIs (via use of table compression in Oracle)
  - LexValues (lexval_prefix, lexval_suffix, ...)

- Index key-prefix compression compacts multiple occurrences of same key prefix

- Fidelity maintained by storing user-specified lexical forms in LexValues and references to them from Application Table rows
Design choices (continued)

- **Duplicate elimination via Unique constraint** (enforces RDF set property)
  - UNIQUE (subj_id, pred_id, canon_obj_id)

- **Break URIs into prefix and suffix** to compress out long common prefixes of distinct URIs (via use of table compression in Oracle)
  - LexValues (lexval_prefix, lexval_suffix, …)

- **Index key-prefix compression** compacts multiple occurrences of same key prefix

- **Fidelity** maintained by storing user-specified lexical forms in LexValues and references to them from Application Table rows

- **Triple-level ancillary information** stored in Application Table
DB Schema for RDF storage
### DB Schema for RDF storage

**Unique key**

<table>
<thead>
<tr>
<th>lexval_prefix</th>
<th>lexval_suffix</th>
<th>id</th>
<th>canon_id</th>
<th>lexval_ext</th>
<th>canon_ext</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://xyz.com/">http://xyz.com/</a></td>
<td>John</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><a href="http://xyz.com/">http://xyz.com/</a></td>
<td>Mary</td>
<td>200</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><a href="http://xyz.com/">http://xyz.com/</a></td>
<td>managerOf</td>
<td>300</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**LexValues Table**
DB Schema for RDF storage

<table>
<thead>
<tr>
<th>lexval_prefix</th>
<th>lexval_suffix</th>
<th>id</th>
<th>canon_id</th>
<th>lexval_ext</th>
<th>canon_ext</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://xyz.com/">http://xyz.com/</a></td>
<td></td>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><a href="http://xyz.com/">http://xyz.com/</a></td>
<td>John</td>
<td>200</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><a href="http://xyz.com/">http://xyz.com/</a></td>
<td>Mary</td>
<td>300</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><a href="http://xyz.com/">http://xyz.com/</a></td>
<td>managerOf</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**LexValues Table**

<table>
<thead>
<tr>
<th>subj_id</th>
<th>pred_id</th>
<th>canon_id</th>
<th>model_id</th>
<th>obj_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>300</td>
<td>200</td>
<td>1</td>
<td>200</td>
</tr>
</tbody>
</table>

**IdTriples Table**

**Unique key**

Presented by Souri Das at ICDE 2008
Storage: Showing Application Table
Storage: Showing Application Table

Application Tables with RDF object type columns

Model 1
RDF/OWL data and ontologies

Model 2

Inferred Triple Set 1

Inferred Triple Set 2

Inferred Triple Set p

Vocabularies and Rulebases

OWL subset
RDF/RDFS
Rulebase m

RDF/OWL data and ontologies

Oracle DB Semantic Network (inside MDSYS)

Rules Indexes (Derived data)

A_1
A_2
A_n

scott

herman

Presented by Souri Das at ICDE 2008
Storage: Showing Application Table

Application Tables with RDF object type columns

RDF/OWL data and ontologies

Model 1

Model 2

Model n

Oracle DB Semantic Network (inside MDSYS)

Vocabularies and Rulebases

OWL subset

RDF / RDFS

Rulebase m

Inferred Triple Set 1

Inferred Triple Set 2

Inferred Triple Set p

Application Tables with RDF object type columns

A₁

A₂

Aₙ

Presented by Souri Das at ICDE 2008
## Bulk-Load Functionality Choices

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Oracle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strict parse with error reporting (optional)</td>
<td>YES</td>
</tr>
<tr>
<td>Value Point equivalence</td>
<td>YES</td>
</tr>
<tr>
<td>Elimination of duplicate triples</td>
<td>YES</td>
</tr>
<tr>
<td>Collision detection and resolution</td>
<td>YES</td>
</tr>
<tr>
<td>Fidelity (preserving user-specified form of data)</td>
<td>YES</td>
</tr>
<tr>
<td>Triple level ancillary information</td>
<td>YES</td>
</tr>
<tr>
<td>Long literal values (&gt; 4000 bytes)</td>
<td>NO</td>
</tr>
<tr>
<td>Inference as part of Load</td>
<td>NO</td>
</tr>
<tr>
<td>Multiple input file formats</td>
<td>NO</td>
</tr>
</tbody>
</table>
Bulk-load: transformation steps

<table>
<thead>
<tr>
<th>Global</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\leftarrow \text{LexValues Table} \rightarrow)</td>
</tr>
<tr>
<td>(\leftarrow \text{IdTriples Table} \rightarrow)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Local</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\text{External File})</td>
</tr>
</tbody>
</table>

Presented by Souri Das at ICDE 2008
Bulk-load: transformation steps

- LexValues Table
- IdTriples Table

Staging Table

External File

Load
Bulk-load: transformation steps

1. Load from External File to Staging Table
2. Select distinct values from Staging Table to BatchLexValues Table

Global: LexValues Table → IdTriples Table
Bulk-load: transformation steps

1. Load External File
2. Select distinct values from BatchLexValues Table
3a. Detect and Resolve collisions
Bulk-load: transformation steps

1. **Load**
   - **External File**

2. **Select distinct values**
   - **BatchLexValues Table**

3. **Detect and Resolve collisions**
   - **BatchLexValues Table**

4. **Merge batch values**
   - **IdTriples Table**
   - **LexValues Table**

**Global**

**Local**
Bulk-load: transformation steps

1. Load External File
2. Select distinct values
3a. Detect and Resolve collisions
3b. Merge batch values
4. Create All collisions table

- LexValues Table
- IdTriples Table
- BatchLexValues Table
- AllCollisionExt Table
- Staging Table

Global

Local
Bulk-load: transformation steps

1. Load from External File

2. Select distinct values

3a. Detect and resolve collisions

3b. Merge batch values

4. Create all collisions table

5. Generate and insert Id-based triples
Bulk-load: transformation steps

1. Load from External File

2. Select distinct values

3a. Detect and Resolve collisions

3b. Merge batch values

4. Create All collisions table

5. Generate and insert Id-based triples

6. Remove duplicate triples

Global

Local

- **LexValues** Table → **IdTriples** Table → **Global**

- **BatchLexValues** Table

- **Staging** Table

- **AllCollisionExt** Table

- **BatchIdTriples** Table
Bulk-load: transformation steps

1. Load from External File

2. Select distinct values

3. a. Detect and Resolve collisions
   b. Merge batch values

4. Create All collisions table

5. Generate and insert Id-based triples

6. Insert + index build OR Append with duplicate removal

7. Remove duplicate triples

Global

Local

- **LexValues** Table
- **IdTriples** Table
- **BatchLexValues** Table
- **AllCollisionExt** Table
- **BatchIdTriples** Table
- **Staging** Table
## Performance Factors

<table>
<thead>
<tr>
<th>Bulk-Load Performance</th>
<th>Factors affecting Performance</th>
<th>Oracle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration</td>
<td>Desktop vs. Server</td>
<td>Desktop Machine</td>
</tr>
<tr>
<td>Indexes</td>
<td>Number of Indexes</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Type of Indexes</td>
<td>B-tree indexes</td>
</tr>
<tr>
<td></td>
<td>Columns Indexed</td>
<td>PCSM and PSCO (C: canon-obj-id col)</td>
</tr>
<tr>
<td>Parallelism</td>
<td></td>
<td>Not used currently</td>
</tr>
</tbody>
</table>
## Storage Cost: normalized vs. denormalized

<table>
<thead>
<tr>
<th>Storage space for UniProt 100 million triples</th>
<th>Normalized storage</th>
<th>Denormalized storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table(s)</td>
<td>2.84 GB</td>
<td>3.83 GB</td>
</tr>
<tr>
<td>Indexes</td>
<td>7.88 GB</td>
<td>16.82 GB</td>
</tr>
<tr>
<td>TOTAL</td>
<td>10.72 GB</td>
<td>20.65 GB</td>
</tr>
</tbody>
</table>

### Notes

1. App Table storage space not included.
2. Denormalized Table storage space is 13.35 GB w/o COMPRESSION.
Desktop Performance

- **Hardware**
  - CPU → Single-CPU P4 (3.0GHz with Hyper Threading)
  - Memory → 2GB
  - Hard Disks → Two 500GB 7200rpm SATA 3.0G
- **OS:** Red Hat Enterprise Linux (32-bit)
- **DBMS**
  - Oracle DBMS Enterprise Server 11g Release 1
  - Settings
    - db_cache_size=832M
    - pga_aggregate_target=352M
    - sga_target=1056M
    - Db_file_multiblock_read_count=128
    - Filesystemio_options=‘SETALL’
  - Temp tablespace was allocated on a separate hard disk
Performance: Bulk-Load Transformations

1. Load Staging Table
2. Load Batch Values
3a. Delet&Res Collisions
3b. Merge Batch Values
5. Load Batch Triples
6. Remove Duplicates
7. Create Indexes and Insert batch Triples
<table>
<thead>
<tr>
<th>Ontology (size before duplicate elimination)</th>
<th>Bulk-loading into RDF model in an empty RDF network</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
</tr>
<tr>
<td></td>
<td>bulk-load API(^[1]) Time</td>
</tr>
<tr>
<td></td>
<td>Sql*loader time range low(^[2])/ high(^[3])</td>
</tr>
<tr>
<td>LUBM50 6.9 million</td>
<td>8 min</td>
</tr>
<tr>
<td></td>
<td>1 min</td>
</tr>
<tr>
<td></td>
<td>4.3 min</td>
</tr>
<tr>
<td>LUBM500 69 million</td>
<td>1 hr 45 min</td>
</tr>
<tr>
<td></td>
<td>9.5 min</td>
</tr>
<tr>
<td></td>
<td>43 min</td>
</tr>
<tr>
<td>LUBM1000 138 million</td>
<td>3 hr 25 min</td>
</tr>
<tr>
<td></td>
<td>19 min</td>
</tr>
<tr>
<td></td>
<td>1h 26m</td>
</tr>
<tr>
<td>LUBM8000 1,106 million</td>
<td>30 hr 43 min</td>
</tr>
<tr>
<td></td>
<td>2h 35m</td>
</tr>
<tr>
<td></td>
<td>11h 32m</td>
</tr>
<tr>
<td>UniProt (old) 207 million</td>
<td>4 hr 40 min</td>
</tr>
<tr>
<td></td>
<td>30m</td>
</tr>
<tr>
<td></td>
<td>1h 55m</td>
</tr>
</tbody>
</table>

\(^{[1]}\) Uses flags=>' VALUES_TABLE_INDEX_REBUILD ' arg to drop and recreate RDF_VALUES indexes.

\(^{[2]}\) Less time is needed when RDF values used in N-Triple file are checked only for length \(<= 4000\) bytes.

\(^{[3]}\) More time is needed when RDF values used in N-Triple file are checked for correctness.

\(^{[4]}\) Application table has table compression enabled.

\(^{[5]}\) Staging table has table compression enabled.
LUBM8000 1.1billion: Transformation Times

- Load Staging Table: 28%
- Load Batch Values: 11%
- Detect Collision: 12%
- Merge Batch Values: 16%
- Load Batch Triples: 15%
- Remove Duplicates: 9%
- Create Indexes and Insert Batch Triples: 8%
- Misc: 2%
Related work

- Virtuoso (OpenLink)
- BigOWLIM (OntoText)
- AllegroGraph (Franz)
- …
Summary

• Presents bulk-load of RDF data as
  • a series of transformations
  • on raw data imported into the database
  • without moving large data to non-DBMS memory

• Uses hash-based Id generation scheme
  • with full support for efficient collision detection and resolution
  • eliminates expensive joins for ID lookups
  • makes it easier to integrate data from multiple RDF stores

• Presents a detailed, fine-grained performance study
  • that shows performance of each transformation step
  • shows scalable performance
Future Work

• Strict parsing during loading into staging table via SQL*Loader needs to be re-evaluated. We are trying to see if parsing can be done more efficiently assuming errors are infrequent and still allowing robust (triple-level) error reporting.
• Merging triples into IdTriples can be slow (during bulk-append). We are working on minimizing the time needed by using optimistic merging assuming duplicate triples are infrequent.
• Similar efforts are targeting optimization of batch value merging during bulk-append.
• Exploitation of concurrency between the two tasks: processing of values and processing of triples.
• Using parallel operations for individual transformations and benchmarking using multi-CPU machines.
For More Information

search.oracle.com

or

oracle.com