Balancing Act to improve RDF Query Performance in Oracle Database

Eugene I. Chong
Agenda

• RDF Query processing Issues
• RDF Order-By and Filter Processing
• RDF In-Memory Processing
• RDF In-Memory Virtual Columns
• Conclusion
Oracle RDF

- **RDF_LINK$ table (triples)**
  - normalized
  - subject, predicate, object IDs

- **RDF_VALUE$ table (ID to value mapping)**
  - value, type, etc.

- **Issues**
  - frequent joins with RDF_VALUE$ table to present results, process filters and order-by queries
  - complete de-normalization incurs large storage requirements
  - self-joins: large intermediate join results
Oracle RDF Filters and Order-By Processing

• SPARQL order-by semantics
  – order: no values, blank nodes, IRIs, literals
  – case statement: value type, numeric value, date value, string value
  – ORDER BY CASE WHEN (V4.VALUE_TYPE IS NULL) THEN 0
    WHEN (V4.VALUE_TYPE IN ('BLN','BN')) THEN 1
    WHEN (V4.VALUE_TYPE IN ('URI','UR')) THEN 2
    WHEN (V4.VALUE_TYPE IN ('PL', 'PLL', 'CPLL', 'PL@', 'PLL@', 'CPLL@', 'TL', 'TLL', 'CTLL', 'LIT'))
    THEN (CASE WHEN (V4.LANGUAGE_TYPE IS NOT NULL) THEN 5
    ........
Oracle RDF Filters and Order-By Processing

- literal type - numeric: `TO_NUMBER()`
- literal type - date/time: `TO_TIMESTAMP_TZ ( )`, `DECODE( )`
- use function calls to generate SQL for order-by
- case statements executed for every row at runtime
- same problem for filters

• Solution
  - materialize value type and values in RDF_VALUE$ table
  - stored as `ORDER_TYPE`, `ORDER_NUM`, `ORDER_DATE`
  - filled in at load time
  - generate SQL: `ORDER BY order_type, order_num, order_date, value_name`
  - filter clause: `WHERE order_num < to_number(89)`
Oracle RDF Order-By and Filter Performance using BSBM Benchmark Queries (in secs)
Oracle RDF In-Memory Processing

• Utilize Oracle IMC
  – load frequently accessed columns in memory
    • RDF_LINK$ table: subject, predicate, object IDs
    • RDF_VALUE$: id, value
  – fast full scan of the table: good for hash join

• Experiment
  – 32GB memory, 2TB disk space
  – LUBM benchmark queries (8,763,829 rows including entailment)
  – varying the size of the memory: 6G(100%), 4G(56%), 2G(27%), 1G(12%)
Oracle RDF In-Memory Query Times (in sec) for LUBM Benchmark Queries

- 100% : 4x – 6x gain

- 56%

[Graph showing query times for Q1 to Q14 with comparisons of No IM and IM (100%) for 100% and No IM and IM (56%) for 56% gains.]
Oracle RDF In-Memory Query Times (in sec) for LUBM Benchmark Queries

- 27%

- 12%
Oracle RDF In-Memory Full Scan Performance (in sec)

- Fetching 3 IDs from RDF_LINK$ table
- 100% - 190x gain
Oracle RDF In-Memory Virtual Columns

• In-memory complete de-normalization without incurring disk storage requirements
  – define virtual columns in RDF_LINK$ table for values, types, etc.: VALUE_NAME_S, VALUE_NAME_P, VALUE_NAME_O, etc.
  – useful for fully populated data in memory: virtual model

Virtual column in-memory performance (in min) – fetching 3 IDs & 3 VCs
Oracle RDF In-Memory Virtual Columns

- remove joins with RDF_VALUE$ table
- queries are processed on RDF_LINK$ table only
- compression, smart scans (in-memory storage index), dictionary code for values, SIMD vector processing
Oracle RDF In-memory Virtual Column Performance using LUBM Benchmark Queries (in secs)

• Up to 8x gain

• As the number of joins increases, a bigger gain is achievable
Oracle RDF In-Memory Virtual Columns

• Can apply to data mart/data warehousing star/snowflake schema
  – remove joins with dimension tables

• Can apply to any applications where joined tables have one-to-one mapping on their join keys
Conclusion

• Significant performance improvement
  – use order columns in place of complex logic in the query for RDF filter and order-by processing
  – improve hash joins by in-memory processing of frequently accessed columns
  – remove costly joins using in-memory virtual columns by complete de-normalization for fully populated data
Your Questions
Hardware and Software
Engineered to Work Together