ATHENA++: Natural Language Querying for Complex Nested SQL Queries

Jaydeep Sen¹, Chuan Lei², Abdul Quamar², Fatma Özcan², Vasilis Efthymiou², Ayushi Dalmia¹, Greg Stager³, Ashish Mittal¹, Diptikalyan Saha¹, Karthik Sankaranarayanan¹.

¹IBM Research - India, ²IBM Research - Almaden, ³IBM Canada
Introduction

Problem:
- Natural language querying for nested queries.

Motivation:
- Existing NLIDB systems do not focus on BI queries with nesting.

Aim is to **democratize access to BI insights** for business users.
- Without depending on SQL experts/analysts or the need to know the schema or SQL language.
Nested Queries - Challenges

Example: “Show me everyone who bought stocks in 2019 that have gone up in value”

• Nested Query Detection:
  • How to detect nesting?

• Subquery Formation:
  • How to divide the query into subqueries?

• Subquery Joining:
  • How to join subquery results?

Outer query graph

Inner query graph

Join condition
Question:
Show me everyone who bought stocks in 2019 that have gone up in value.

Token Annotations:
- everyone: [Person, Customer, Manager]
- bought: [Transaction.type]
- stocks: [ListedSecurity]
- value: [MonetaryAmount.value]
- gone up: [Operator (>)]
- in 2019: [Time Comparison]

Nested Query Detected

Outer Query
{everyone, bought, stocks, 2019, value}

Inner Query
{stocks, 2019, value}
Detection: Intuition and Examples

- Annotate tokens based on their semantic role
- Detect if query belongs to one of the 4 nesting types: A, N, J, JA.

---

**Annotate tokens based on their semantic role**

<table>
<thead>
<tr>
<th>Entity</th>
<th>IKEA, Sony, etc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance</td>
<td>IBM, Apple, etc.</td>
</tr>
<tr>
<td>Time</td>
<td>since 2010, in 2019, from 2010 to 2019, etc.</td>
</tr>
<tr>
<td>Numeric</td>
<td>16.8, sixty eight, etc.</td>
</tr>
<tr>
<td>Measure</td>
<td>revenue, price, value, volume of trade, etc.</td>
</tr>
<tr>
<td>Count</td>
<td>count of, number of, how many, etc.</td>
</tr>
<tr>
<td>Aggregation</td>
<td>total/sum, max, min, average, etc.</td>
</tr>
<tr>
<td>Comparison</td>
<td>more/less than, gone up, etc. equal, same, also, too, etc. not equal, different, another, etc.</td>
</tr>
<tr>
<td>Negation</td>
<td>no, not, none, nothing, etc.</td>
</tr>
</tbody>
</table>

---

**Detect if query belongs to one of the 4 nesting types: A, N, J, JA.**

<table>
<thead>
<tr>
<th>Query Types</th>
<th>Aggregation</th>
<th>Correlation between Inner &amp; Outer Queries</th>
<th>Division Predicate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type-A</td>
<td>✓</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Type-N</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Type-J</td>
<td>X</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>Type-JA</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>Type-D</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

---

- Show me the customers who are also account managers. (Equality Comparison between two separate entities => Type N)

- Show me everyone who bought stocks in 2019 that have gone up in value? (Numeric Comparison between a co-ref and measure => Type J)

- Who bought Alphabet stocks with price more than his average buying price in 2019? (Numeric Comparison with an aggregation result having a co-ref => Type JA)
Subquery Formation: Intuition and Example

- Position of join token is treated as the boundary to initialize subquery tokens.
- We design heuristics on how to share tokens across outer and inner query.
- Heuristics depend on the annotations and nested type detected.

### Table: Heuristic Details

<table>
<thead>
<tr>
<th>Heuristic</th>
<th>Type-N</th>
<th>Type-A</th>
<th>Type-J</th>
<th>Type-JA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heuristic 1</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Heuristic 2</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Heuristic 3</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Heuristic 4</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Heuristic 5</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Heuristic 6</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Heuristic 1: co-referred entities to be shared.
Heuristic 2: time mentions are to be shared (if missing).
Heuristic 3: instance sharing when inner does not have aggregation.
Heuristic 4: focus sharing for non-numeric comparison queries.
Heuristic 5: comparison argument sharing across subqueries (if missing).
Heuristic 6: dependent entity/instance to be shared.

Show me everyone who bought stocks in 2019 that have gone up in value
- \{everyone, bought, stocks, in 2019\}, \{value\}
  - (Apply heuristics: Argument Sharing, Time Sharing, Dependent Entity Sharing)
- \{everyone, bought, stocks, in 2019, value\} \(\leftrightarrow\) \{stocks, in 2019, value\}
Subquery Join: Intuition and Example

- Building individual subqueries with their respective tokens including shared tokens.
- Figuring out the right join condition between subqueries.
- Hierarchical query building by joining Outer and Inner subqueries.

Example: “Show me everyone who bought stocks in 2019 that have gone up in value”

Outer: {everyone, bought, stocks, in 2019, value} | join Op: ‘>’ | Inner: {stocks, in 2019, value}

- Steiner tree-based algorithm (ATHENA-PVLDB’16) for each subquery formation
- Subquery joining depends on join types e.g., ‘>’ is a numeric comparator that can be applied on the measure 'value'
A New Benchmark: FIBEN

Schema
- Conforms to the combination of two standard finance ontologies: FIBO and FRO
- Contains information on
  - Security transactions, insider history, financial metrics, industry info, etc.
- Emulates a real data mart in finance.

Queries
- 300 pairs of <NL,SQL> queries with 237 unique SQLs, 170 of them nested.
- Specifically focus on BI queries as obtained from BI experts.
- Covers enough examples of different types of nested queries.
- Open-sourced at: https://github.com/IBM/fiben-benchmark
ATHENA++ outperforms NALIR and ATHENA on all benchmarks.
Only ATHENA++ achieves a decent accuracy for nested queries.
Accuracy gap is significant in FIBEN which includes maximum # nested queries.
Conclusion

- ATHENA++ is the first system to handle nested BI queries.
- ATHENA++ is a step towards making NLIDB systems usable for real enterprise BI applications.
- New benchmark designed for the BI queries and open-sourced at: https://github.com/IBM/fiben-benchmark
Thank You