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

Jaydeep Sen[1], Chuan Lei[2], Abdul Quamar[2], Fatma Özcan[2], Vasilis Efthymiou[2],Ayushi Dalmia[1], Greg Stager[3], Ashish Mittal[1], Diptikalyan Saha[1],Karthik Sankaranarayanan[1].

[1]IBM Research - India, [2]IBM Research - Almaden, [3]IBM Canada

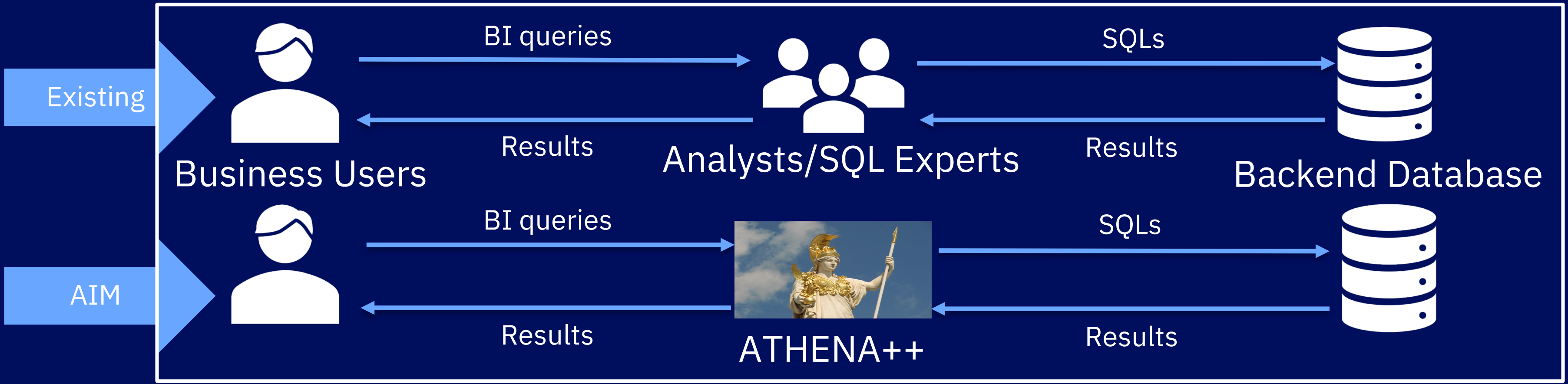
Introduction

Problem:

- Natural language querying for nested queries.

Motivation:

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



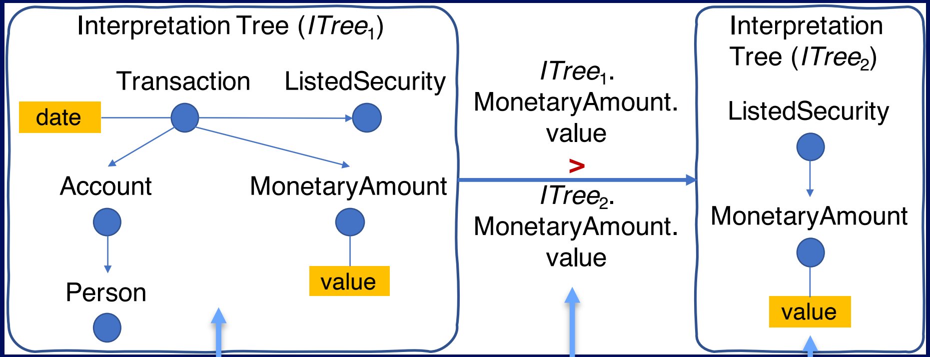
- Aim is to **<u>democratize access to BI insights</u>** for business users.
- Without depending on SQL experts/analysts or the need to know the schema or SQL language.

Nested Queries - Challenges

Transaction.type Transaction.time MonetaryAmount.value

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

Person, Customer, Account Manager ListedSecurity Operator: ‘>’



Outer query graph

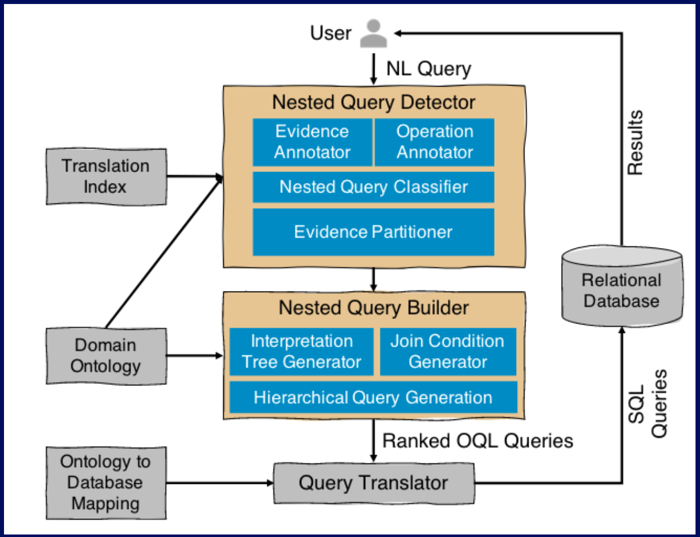
Inner query graph

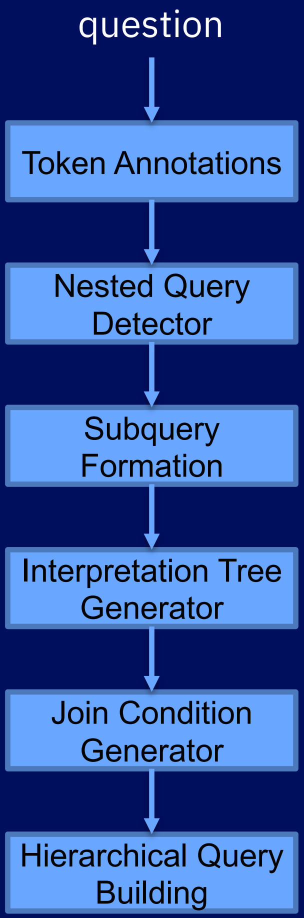
Join condition

- **Nested Query Detection:**
  - How to detect nesting?
- **Subquery Formation:**
  - How to divide the query into subqueries?
- **Subquery Joining:**
  - How to join subquery results?

Inside ATHENA++

Architecture





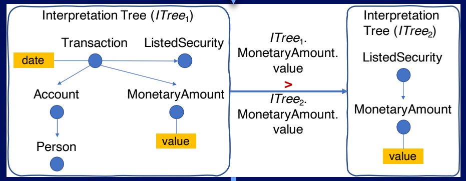
Show me everyone who bought stocks in 2019 that have gone up in value

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}*



Output SQL Query

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.

Annotations	Example Token t
Entity	customer, stocks, etc.
Instance	IBM, California, etc.
Time	since 2010, in 2019, from 2010 to 2019, etc.
Numeric	16.8, sixty eight, etc.
Measure	revenue, price, value, volume of trade, etc.
Count	count of, number of, how many, etc.
Aggregation	total/sum, max, min, average, etc.
Comparison	more/less than, gone up, etc. equal, same, also, too, etc. not equal, different, another, etc.
Negation	no, not, none, nothing, etc.

Query Types	Aggregation	Correlation between Inner & Outer Queries	Division Predicate
Type-A	✓	✗	✗
Type-N	✗	✗	✗
Type-J	✗	✓	✗
Type-JA	✓	✓	✗
Type-D	✗	✓	✓

- 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.

	Type-N	Type-A	Type-J	Type-JA
Heuristic 1			✓	✓
Heuristic 2	✓	✓	✓	✓
Heuristic 3	✓		✓	
Heuristic 4	✓		✓	
Heuristic 5	✓	✓	✓	✓
Heuristic 6		✓	✓	✓

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.

Join Token

- 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} <-> {stocks, in 2019, value}

IBM

Subquery Join: Intuition and Example

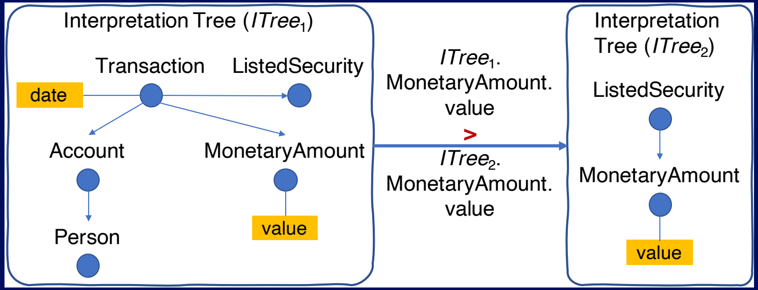
- Building individual subqueries with their respective tokens including shared tokens.
- Figuring out the right join condition between subqueries.
- Hierarchical query building by joining Outer and Inner subqueries.

Transaction.type Transaction.time MonetaryAmount.value

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

Person, Customer, Account Manager ListedSecurity Operator: ‘>’

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`

Results

Overall Accuracy %

Data Set	ATHENA++	ATHENA	NaLIR
MAS	84.61	67.03	49.08
GEO	84.25	68.20	41.04
Spider	78.82	54.93	–
FIBEN	88.33	48.00	20.66

Nested Query Accuracy %

Data Set	ATHENA++	ATHENA	NaLIR
MAS	78.37	10.81	8.10
GEO	78.57	17.14	8.57
Spider	78.26	9.93	–
FIBEN	85.88	15.29	7.05

- 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