DrillBeyond: Processing Multi-Result Open World SQL Queries

Julian Eberius*, Maik Thiele, Katrin Braunschweig and Wolfgang Lehner
Technische Universität Dresden, Germany
Motivation

DOMAINS OF...
- information retrieval and database systems...

...HAVE BEEN TRADITIONALLY KEPT SEPARATE

DIFFERENCES IN...
- type of data that is managed (structured versus unstructured)
- query language used (fully specified query versus keyword query)
- nature of the query result (exact single answer versus ranked list of possible answers)
- usage scenarios (analytic scenarios versus information gathering)

WE EXPLORE A NEW WAY...
- of merging the two paradigms for use in ad-hoc and self-service analytics
DrillBeyond

**DrillBeyond = Novel Hybrid DBMS/IR System**

- Blurring the lines between type of data managed, query language used, and nature of the query result

```sql
SELECT n_name, gdp, AVG(o_totalprice)
FROM nation, customer, orders
WHERE n_nationkey = c_nationkey
  AND c_custkey = o_custkey
  AND gdp > 10.0
GROUP BY n_name, gdp
ORDER BY gdp DESC
```

**Mix of...**

- ..relational queries on a database and...
- ..top-k keyword-based searches (= Entity Augmentation Queries)
DrillBeyond – System Architecture

**QUERY ANALYZER**
- Maps SQL query tokens to the database catalog
- For unrecognized tokens (e.g. “gdp”) we introduce transient metadata
- Query is rewritten to include an additional join with a transient relation

**ENTITY AUGMENTATION SYSTEM**
- Implements the entity augmentation processing

**WEB DATA INDEX**
- Index of Web tables
- Generic system exposing an interface for keyword-based document search
DrillBeyond – System Architecture (2)

**EXECUTOR**
- Implements the repeated execution of operator trees using the DrillBeyond operator $\omega$

**QUERY PLANNER**
- Minimize the overhead of creating multiple result variants
System Integration Challenges

**Multi-solution Query Processing**
- Top-k query result versions for a single user query
- Naïve way: process the query k times → query runtime increased by factor k
- Goals:
  - Minimizes duplicate work between query executions
  - Maximize invariant parts of multiple executions

**Query Planning**
- Web tables are not fully known at plan-time → no selectivity information or data types at plan-time
- Goal: plan queries with relations only known at run-time
DrillBeyond Operator $\omega$

**INIT()**
- Initializes state

```latex
\textbf{function} \textsc{Init} \\
\quad state \leftarrow \text{'collecting'} \\
\quad \text{tupletstore} \leftarrow \emptyset \\
\quad \text{augMap} \leftarrow \text{HashMap}() \\
\quad n \leftarrow 0
```

**NEXT()**
- Produces augmented tuples in three phases
  - Collect()
  - Augment()
  - Project()

```latex
\textbf{function} \textsc{Next} \\
\quad \textbf{if} \ state = \text{'collecting'} \ \textbf{then} \\
\quad \quad \textsc{Collect}() \\
\quad \quad \textsc{Augment}() \\
\quad \quad \text{state} \leftarrow \text{'projecting'} \\
\quad \textbf{return} \ \textsc{Project}()
```
DrillBeyond Operator $\omega$ (2)

**COLLECT()**
- Pulls and stores all tuples
- Blocking

**AUGMENT()**
- Pass all entries from the EAS into a hashtable

**function COLLECT**

```
function COLLECT
    while true do
        t ← Next(childPlan)
        if t = NULL then
            break
        tuplestore ← t
        augKey ← TEXT_ATTRS(t)
        if augKey ∉ augMap then
            augMap[augKey] ← ∅
```

**function AUGMENT**

```
function AUGMENT
    augReq ← (∀k ∈ augMap | augMap[k] = ∅)
    for all augKey, [augValues...] ∈ SEND(augReq) do
        augMap[augKey] ← [augValues...]
```
DrillBeyond Operator $\omega$ (3)

**PROJECT()**
- Produces output tuples by replaying the stored tuples and filling the augmentation attribute

**RESCAN()**
- Called when subtrees have to be re-executed

**NEXTVARIANT()**
- Produces the multi-variant query results

```plaintext
function PROJECT
    t ← NEXT(tuplestore)
    if t = NULL then return NULL
    augKey ← TEXT_ATTRS(t)
    t[augAttr] = augMap[augKey][n] return t

function RESCAN
    state ← 'collecting'
    tuplestore ← Ø

function NEXTVARIANT
    RESCAN(tuplestore)
    n ← n + 1
```
**Why Blocking?**

**Example**
- Send each tuple to the augmentation system at its own (tuple-at-a-time)
- Augmentation system may choose $d_{s1}$, $d_{s2}$ and $d_{s4}$

**More consistent: $d_{s1}$ and $d_{s3}$**

**For quality reasons the DrillBeyond operator...**
- ..needs to be a blocking operator
- Realized in the “collecting” state in function Next()
- Consumes tuples from underlying operators until these are exhausted
- Hands them over to the augmentation system
Lower Placement Bound

**SAME QUERY, TWO DIFFERENT QUERY PLANS → TWO DIFFERENT AUGMENTATIONS**

\[
\gamma_{\text{name}, \text{sum}(\text{totalprice})}
\]

\[
\omega_{\text{nation.gdp}} \sigma_{\text{name}=\text{'EUROPE'}}
\]

\[
\gamma_{\text{name}, \text{sum}(\text{totalprice})}
\]

\[
\omega_{\text{nation.gdp}} \sigma_{\text{name}=\text{'EUROPE'}}
\]

**LOWER PLACEMENT BOUND**

- \( \omega \) can only be placed when all filters on R, e.g., joins and predicates, have been applied
- In other words: apply \( \omega \) to the minimum number of entities in R
Invariant Caching

**Example**
- Modeled after TPCH query 2

**LIMITATION:** Depending on the cost-based placement of $\omega$, most of the query may be varying.
Maximize Invariant Sub-Trees

OPTIMAL PLAN...
- ...with respect to a single query execution

OPTIMAL PLAN...
- ...with respect to multiple query executions
- Upper bound placement rule: Place $\omega$ not earlier than the augmented values need to be accessed

Lower bound placement rule

Upper bound placement rule
OBSERVATION

- We can separate the input part of $\omega$ from the actual projection of augmented values

SPLIT THE DRILLBEYOND OPERATOR INTO TWO PARTS

- $\omega$
  - Performing the augmentation (collect() and augment())
  - Projects placeholders
  - Placed at the lower placement bound

- $\Omega$
  - Performing the projection of values
  - Dereferences to the correct value array
  - Placed at the upper placement bound

→ MINIMIZING AUGMENTATION OPERATOR COST
→ MAXIMIZING SIZE OF INVARIANT QUERY PARTS

Maximize distance between $\omega$ and $\Omega$
**Dynamic Selection Pull-Up**

**Selection push-down**
- Smaller intermediate join result...
- ..but more varying nodes in the plan

**Selection pull-up**
- Place $\Omega$ as describe before
- Larger invariant subplan...
- ..but increased join results

Open questions:
- Cost model for this decision?
- Selectivity estimation of the predicate?
Dynamic Selection Pull-Up (2)

Cost Model

- Binary decision → decide for the minimum cost of the subtree above $\omega$

$$C_k(\omega, \Omega) = \begin{cases} C_1(\omega, \Omega) + \sum_{i=1}^{k} s_i \cdot C_1(\omega, \Omega) & (Pull - up) \\ \sum_{i=1}^{k} s_i \cdot C_1(\omega, \Omega) & (no \ Pull - up) \end{cases}$$

- $C(x, y)$: cost of the subplan from operator $x$ to $y$
- $s_i$
  - Selectivity of the $i^{th}$ data source for the attribute in question
  - Not known at plan-time → determined in the augment() phase
Evaluation
Experimental Setup

IMPLEMENTATION

PostgreSQL + DrillBeyond extension

http://github.com/JulianEberius/DrillBeyond

Entity Augmentation System REA

http://github.com/JulianEberius/REA

Dresden Web Table Corpus (125M web tables)

http://wwwdb.inf.tu-dresden.de/misc/dwtc

TEST DATABASE

- Variation of TPC-H replacing generic identifiers with real-world entities, e.g. real company names for the Supplier relation

TEST QUERIES

- Subset of TPC-H queries in which dimension tables are used
- Added arbitrary where-clauses to one of the dimensions: “relation.X > Y”

PARAMETERS

- Number of augmentations $k \in (1, 3, 5, 10)$
- Selectivity $s$ ranging between 0.01 and 0.99 in ten steps
# Overall Performance

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Norm. Execution Time</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Optimizations</td>
<td>2.27</td>
<td>1.82</td>
</tr>
<tr>
<td>Invariant Caching</td>
<td>1.99</td>
<td>1.59</td>
</tr>
<tr>
<td>Static $\omega/\Omega$</td>
<td>1.79</td>
<td>2.35</td>
</tr>
<tr>
<td>Dynamic $\omega/\Omega$</td>
<td>1.21</td>
<td>0.67</td>
</tr>
<tr>
<td>Dynamic $\omega/\Omega$ + Re-Opt.</td>
<td>1.13</td>
<td>0.28</td>
</tr>
</tbody>
</table>

![Graph showing normalized execution time and standard deviation for different scenarios.](image-url)
Performance of query 9, by selectivity

$k = 1$

$k = 3$

$k = 5$

$k = 10$
Summary

**DRILLBEYOND A RDBMS / IR HYBRID SYSTEM**
- Integration of top-k entity augmentation system into a RDBMS
- Multiple alternative query result solving the uncertainty and ambiguity of the automated integration

**NEW OPERATORS: DRILLBEYOND $\omega/\Omega$**
- Implemented in traditional interface functions: Init(), Next() and ReScan()
- Own cost model

**NEW OPTIMIZATION STRATEGIES**
- Invariant Caching
- Maximization of invariant sub-trees
- Selection push-down versus selection pull-up (runtime reoptimization)