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)

---

**Query Planner**
- Minimize the overhead of creating multiple result variants

**Executor**
- Implements the repeated execution of operator trees using the DrillBeyond operator \( \omega \)
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

```plaintext
function INIT
    state ← 'collecting'
tuplestore ← \emptyset
augMap ← HashMap()
n ← 0
```

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

```plaintext
function NEXT
    if state = 'collecting' then
        COLLECT()
        AUGMENT()
        state ← 'projecting'
    return PROJECT()
```
DrillBeyond Operator $\omega$ (2)

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

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

---

```python
function COLLECT
    while true do
        $t \leftarrow \text{NEXT}(\text{childPlan})$
        if $t = \text{NULL}$ then
            break
        tuplestore $\leftarrow t$
        augKey $\leftarrow \text{TEXTATRRS}(t)$
        if augKey $\notin$ augMap then
            augMap[augKey] $\leftarrow \emptyset$
    
function AUGMENT
    augReq $\leftarrow (\forall k \in \text{augMap} \mid \text{augMap}[k] = \emptyset)$
    for all augKey, [augValues...] $\in \text{SEND}(\text{augReq})$ do
        augMap[augKey] $\leftarrow$ [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 ← TEXTATTRS(t)
  t[augAttr] = augMap[augKey][n] return t

function RESCAN
  state ← 'collecting'
  tuplestore ← ∅

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 $ds_1$, $ds_2$ and $ds_4$

**MORE CONSISTENT: $ds_1$ AND $ds_3$**

**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

<table>
<thead>
<tr>
<th>country</th>
<th>gdp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Russia</td>
<td>?</td>
</tr>
<tr>
<td>UK</td>
<td>?</td>
</tr>
<tr>
<td>USA</td>
<td>?</td>
</tr>
</tbody>
</table>

$ds_1$ American Countries

<table>
<thead>
<tr>
<th>country</th>
<th>gdp m USD</th>
</tr>
</thead>
<tbody>
<tr>
<td>USA</td>
<td>X</td>
</tr>
<tr>
<td>Canada</td>
<td>Y</td>
</tr>
</tbody>
</table>

$ds_2$ European Economy

<table>
<thead>
<tr>
<th>country</th>
<th>gdp EUR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Russia</td>
<td>X</td>
</tr>
<tr>
<td>France</td>
<td>Y</td>
</tr>
</tbody>
</table>

$ds_3$ World GDPs

<table>
<thead>
<tr>
<th>country</th>
<th>gdp Mil. $</th>
</tr>
</thead>
<tbody>
<tr>
<td>UK</td>
<td>X</td>
</tr>
<tr>
<td>Russia</td>
<td>Z</td>
</tr>
</tbody>
</table>

$ds_4$ European Economy

<table>
<thead>
<tr>
<th>country</th>
<th>gdp EUR</th>
</tr>
</thead>
<tbody>
<tr>
<td>UK</td>
<td>X</td>
</tr>
</tbody>
</table>
Lower Placement Bound

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

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

\[ \begin{align*}
\omega_{\text{nation.gdp}} & \quad \sigma_{\text{name='EUROPE'}} \\
\text{Nation} & \quad \text{Region} \\
\end{align*} \]

\[ \begin{align*}
\gamma_{\text{name, sum(totalprice)}} \\
\omega_{\text{nation.gdp}} & \quad \sigma_{\text{name='EUROPE'}} \\
\text{Nation} & \quad \text{Region} \\
\end{align*} \]

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

**OPTIMAL PLAN...**
- Upper bound placement rule

![Diagram of query execution](image)
Maximize Invariant Sub-Trees (2)

**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**
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 $\rightarrow$ decide for the minimum cost of the subtree above $\omega$

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

- $C(x, y)$: 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 $\rightarrow$ determined in the augment() phase
Evaluation
**Experimental Setup**

**IMPLEMENTATION**

- **PostgreSQL + DrillBeyond extension**
  - [GitHub](http://github.com/JulianEberius/DrillBeyond)

- **Entity Augmentation System REA**
  - [GitHub](http://github.com/JulianEberius/REA)

- **Dresden Web Table Corpus**
  - (125M web tables)
  - [Website](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 $\omega/\Omega$ + ReOpt.</td>
<td>1.13</td>
<td>0.28</td>
</tr>
</tbody>
</table>
Performance of query 9, by selectivity

- **k = 1**
  - Invariant Caching
  - Static $\omega/\Omega$
  - Dynamic $\omega/\Omega + \text{ReOpt.}$

- **k = 3**
  - Invariant Caching
  - Static $\omega/\Omega$
  - Dynamic $\omega/\Omega + \text{ReOpt.}$

- **k = 5**
  - Invariant Caching
  - Static $\omega/\Omega$
  - Dynamic $\omega/\Omega + \text{ReOpt.}$

- **k = 10**
  - Invariant Caching
  - Static $\omega/\Omega$
  - Dynamic $\omega/\Omega + \text{ReOpt.}$
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)