Evaluation of Recursive Queries
Part 2: Pushing Selections

Aggregate Operators

The < ... > notation in the head indicates grouping; the remaining arguments (Part, in this example) are the GROUP BY fields.

In order to apply such a rule, must have all of Assembly relation available.

Stratification with respect to use of < ... > is the usual restriction to deal with this problem; similar to negation.

\[
\text{NumParts}(\text{Part}, \text{SUM}(\text{Qty})) \leftarrow \text{Assembly}(\text{Part}, \text{Subpt}, \text{Qty}).
\]

Evalution of Datalog Programs

\[\text{WITH RECURSIVE Comp(Part, Subpt) AS}
\begin{align*}
\text{(SELECT A1.Part, A1.Subpt FROM Assembly A1)} \\
\text{(SELECT A2.Part, C1.Subpt FROM Assembly A2, Comp C1 WHERE A2.Subpt=C1.Part)} \\
\text{SELECT * FROM Comp C2}
\end{align*}\]

Avoiding Repeated Inferences

\[\text{Seminaive Fixpoint Evaluation: Avoid repeated inferences by ensuring that when a rule is applied, at least one of the body facts was generated in the most recent iteration. (Which means this inference could not have been carried out in earlier iterations.)}
\]

- For each recursive table \( P \), use a table \( \text{delta}_P \) to store the \( P \) tuples generated in the previous iteration.
- Rewrite the program to use the delta tables, and update the delta tables between iterations.

\[\text{Comp(Part, Subpt) :- Assembly(Part, Part2, Qty),}
\text{delta}_{\text{Comp}}(\text{Part2, Subpt}).\]

Avoiding Unnecessary Inferences

\[\text{There is a tuple (S1,S2) in SameLev if there is a path up from S1 to some node and down to S2 with the same number of up and down edges.}
\]

\[\begin{align*}
\text{SameLev}(S1, S2) & \leftarrow \text{Assembly}(P1, S1, Q1), \text{Assembly}(P1, S2, Q2), \\
\text{SameLev}(S1, S2) & \leftarrow \text{Assembly}(P1, S1, Q1), \\
\text{SameLev}(P1, P2) & \leftarrow \text{Assembly}(P2, S2, Q2). \\
\end{align*}\]
Avoiding Unnecessary Inferences

- Suppose that we want to find all SameLev tuples with spoke in the first column. We should “push” this selection into the fixpoint computation to avoid unnecessary inferences.
- But we can’t just compute SameLev tuples with spoke in the first column, because some other SameLev tuples are needed to compute all such tuples:

\[
\text{SameLev}(\text{spoke}, \text{seat}) \leftarrow \text{Assembly}(\text{wheel}, \text{spoke}, 2),
\]

\[
\text{SameLev}(\text{wheel}, \text{frame}) \leftarrow \text{Assembly}(\text{frame}, \text{seat}, 1).
\]

“Magic Sets” Idea

- Idea: Define a “filter” table that computes all relevant values, and restrict the computation of SameLev to infer only tuples with a relevant value in the first column.

\[
\text{Magic SL}(P1) \leftarrow \text{Magic SL}(S1), \text{Assembly}(P1, S1, Q1).
\]

\[
\text{Magic SL}(\text{spoke}).
\]

\[
\text{SameLev}(S1, S2) \leftarrow \text{Magic SL}(S1), \text{Assembly}(P1, S1, Q1), \text{Assembly}(P1, S2, Q2).
\]

The Magic Sets Algorithm

- Generate an “adorned” program
  - Program is rewritten to make the pattern of bound and free arguments in the query explicit; evaluating SameLevel with the first argument bound to a constant is quite different from evaluating it with the second argument bound
  - This step was omitted for simplicity in previous slide
- Add filters of the form “Magic P” to each rule in the adorned program that defines a predicate P to restrict these rules
- Define new rules to define the filter tables of the form Magic_P

Generating Adorned Rules

- The adorned program for the query pattern SameLevel, assuming a left-to-right order of rule evaluation:

\[
\text{SameLev}^b (S1, S2) \leftarrow \text{Assembly}(P1, S1, Q1), \text{Assembly}(P1, S2, Q2).
\]

\[
\text{SameLev}^b (S1, S2) \leftarrow \text{Assembly}(P1, S1, Q1), \text{Assembly}(P1, S2, Q2).
\]

Defining Magic Tables

- After modifying each rule in the adorned program by adding filter “Magic” predicates, a rule for Magic_P is generated from each occurrence O of P in the body of such a rule:
  - Delete everything to the right of O
  - Add the prefix “Magic” and delete the free columns of O
  - Move O, with these changes, into the head of the rule

\[
\text{SameLev}^b (S1, S2) \leftarrow \text{Magic SL}^b(S1), \text{Assembly}(P1, S1, Q1), \text{Assembly}(P1, S2, Q2).
\]

\[
\text{Magic SL}^b(P1) \leftarrow \text{Magic SL}^b(S1), \text{Assembly}(P1, S1, Q1).
\]

Nested Queries in SQL (No Recursion)

```
SELECT E, Sal, Avg, Ecnt
FROM emp(E, Sal, D, J),
dinfo(D, Avg, Ecnt)
WHERE J = "Sr pgmer"
```

```
SELECT D, AVG(Sal), count(*)
FROM emp
GROUPBY D
```

“Find senior programmers and their salary, and also average salary and headcount in their depts.”
### Example – Datalog and Magic

**Datalog**

\[ \text{Einfo}(E, \text{Sal}, \text{Avg}, \text{Ecnt}) : J = \text{Sr pgmer}, \text{emp}(E, \text{Sal}, D, J), \text{dinfo}(D, \text{Avg}, \text{Ecnt}). \]

\[ \text{dinfo}(D, A, C) : \text{\ldots} \]

**MAGIC**

\[ m_{\text{emp}}(J) : J = \text{Sr pgmer}. \]

\[ m_{\text{dinfo}}(D) : \{ J = \text{Sr pgmer} \}, m_{\text{emp}}(J), \text{emp}(E, \text{Sal}, D, J). \]

\[ \text{dinfo}(D, A, C) : m_{\text{dinfo}}(D), \text{\ldots} \]

### Magic

1. **Identifies subqueries**
   - Idea: Use rules of the form:
     - If \(<\ldots>\) is a (sub)query and also conditions \(<\ldots>\) hold,
     - Then \(<\ldots>\) are also subqueries.

2. **Restricts computation**
   - Idea: Modify the view definition by joining with the table of queries.
   - This join acts as a “Filter.”

3. **Classify queries**
   - Idea: Using “Adornments”, or “Query forms”. All queries of the form \(p(v_1, v_2)\) are “MAGIC” tuples \(m_p(v_1, v_2)\).

4. **Magic on subqueries**
   - Can use magic for \(q_1\) subqueries:
     - \(m_{q_1}(v) : \ldots p(x, y, z). \)
   - \(q_1\) subqueries handled some other way.
   - So, suitable for rule-based optimizer.

### Dealing with Subqueries – Other ways

**CORRELATION**

- When a subquery is generated, compute all answers, then continue.
  - Not (gasp!) set-oriented.
  - Current DB solution. (DB2 etc.)

**PROLOG**

- When a subquery is generated, compute one answer, then continue.
  - May have to “BACKTRACK.”

### Example – Recursion, Duplicates

**SELECT**

\[ Ename \]

**FROM**

\[ \text{emp e1} \]

**WHERE**

\[ \text{Job} = \text{Sr pgmer} \AND \text{Sal} > ( \text{SELECT} \text{AVG(Sal)} \FROM \text{emp e2} \WHERE e2.D = e1.D ) \]

“Find all senior programmers, the average salary of her/his department is computed.

- Not set-oriented.
- Possible redundancy.

### Correlation

**SELECT**

\[ \text{Ename} \]

**FROM**

\[ \text{emp e1} \]

**WHERE**

\[ \text{Job} = \text{Sr pgmer} \AND \text{Sal} > ( \text{SELECT} \text{AVG(Sal)} \FROM \text{emp e2} \WHERE e2.D = e1.D ) \]

**INNER**

\[ \{ \text{SELECT} \text{AVG(e2.Sal)} \FROM \text{emp e2} \WHERE e2.D = e1.D \} \]

For each senior programmer, the average salary of her/his department is computed.

- Not set-oriented.
- Possible redundancy.

### Decorrelation

**SELECT**

\[ \text{Ename, dep_avgsal} \]

**FROM**

\[ \text{emp, dep_avgsal} \]

**WHERE**

\[ \text{Job} = \text{Sr pgmer} \AND \text{Sal} > \text{Avsal} \AND \text{emp.D} = \text{dep_avgsal.D} \]

\[ \text{dep_avgsal AS dep_avgsal} \]

**SELECT**

\[ D, \text{AVG(Sal)} \]

**FROM**

\[ \text{emp} \]

**GROUP BY**

\[ D \]

- Set-oriented, no redundancy.
- But… irrelevant computation.
Voila! Magic!

\[
\begin{align*}
\text{msg}(D) & \quad \text{AS} \\
\text{SELECT} & \quad \text{DISTINCT} \\
\text{emp, dep_avgsal} & \quad \text{D} \\
\text{FROM} & \quad \text{emp} \\
\text{WHERE} & \quad \text{Job = "Sr pgmr" AND Sal > dep_avgsal.D}
\end{align*}
\]

\[
\begin{align*}
\text{dep_avgsal}(D, ASal) & \quad \text{AS} \\
\text{SELECT} & \quad \text{D} \\
\text{AVG(Sal)} & \quad \text{ASal} \\
\text{FROM} & \quad \text{msg, emp} \\
\text{WHERE} & \quad \text{msg.D = emp.D} \\
\text{GROUP} & \quad \text{D}
\end{align*}
\]

From Datalog to SQL

- Conditions
  - \( X + Y > 10 \)
- Grouping and aggregation
  - \( \text{avg} = 25 \)

Multisets
- If you don’t remove duplicates. That is a feature!

Example - Conditions

\[
\begin{align*}
\text{SELECT} & \quad \text{Ename, Mgr} \\
\text{FROM} & \quad \text{emp, dept} \\
\text{WHERE} & \quad \text{Job = "Sr Pgmr" AND Sal > 50000 AND emp.D = dept.D}
\end{align*}
\]

Datalog to SQL: A summary

- Conditions
  - Magic transformation is followed by some “GROUNDING” steps.
- Multisets (Duplicates)
  - # copies of a tuple = # of derivations
- Operationally
  - Just skip duplicate checks
- Magic
  - All “magic” tables are DISTINCT
- Groupby, Aggregates
  - Must check if restrictions (selections, conditions) can be “pushed down”
  - With recursion, may need stratification.

Comparing Magic and Correlation

We must consider three factors:
1. Binding propagation
2. Repeated work (duplicates)
3. Set-Orientiation

<table>
<thead>
<tr>
<th>Correlation</th>
<th>Magic</th>
</tr>
</thead>
<tbody>
<tr>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>× (✓)</td>
<td>√</td>
</tr>
<tr>
<td>×</td>
<td>✓</td>
</tr>
</tbody>
</table>

Experiments

Experiments run on DB V2R2 DBMS.

Benchmark DB

<table>
<thead>
<tr>
<th>Table</th>
<th>Tuple Size</th>
<th># Tuples</th>
<th># Column</th>
<th># 4k Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>item</td>
<td>34</td>
<td>170,000</td>
<td>4</td>
<td>1,850</td>
</tr>
<tr>
<td>wkc</td>
<td>28</td>
<td>500</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>ltl</td>
<td>78</td>
<td>2,580,000</td>
<td>13</td>
<td>57,980</td>
</tr>
<tr>
<td>ltp</td>
<td>43</td>
<td>339,440</td>
<td>14</td>
<td>4,250</td>
</tr>
</tbody>
</table>
Results

Experiment 1
Binding propagation, no duplicates, set-orientation not significant.

<table>
<thead>
<tr>
<th>Query</th>
<th>Time</th>
<th>I/O</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>0.25</td>
<td>0.46</td>
</tr>
<tr>
<td>Correlated</td>
<td>0.40</td>
<td>0.06</td>
</tr>
<tr>
<td>Magic</td>
<td>0.46</td>
<td>0.25</td>
</tr>
</tbody>
</table>

Experiment 2
Binding set contains duplicates (~100), set-orientation not significant.

<table>
<thead>
<tr>
<th>Query</th>
<th>Time</th>
<th>I/O</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>0.06</td>
<td>0.40</td>
</tr>
<tr>
<td>Correlated</td>
<td>0.10</td>
<td>0.05</td>
</tr>
<tr>
<td>Magic</td>
<td>0.25</td>
<td>0.06</td>
</tr>
</tbody>
</table>

Results – Cont.

Experiment 3
Binding set has some duplicates, set-orientation is significant. (Bindings on non-index column)

<table>
<thead>
<tr>
<th>Query</th>
<th>Time</th>
<th>I/O</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>465</td>
<td>55</td>
</tr>
<tr>
<td>Correlated</td>
<td>430</td>
<td>46</td>
</tr>
<tr>
<td>Magic</td>
<td>136</td>
<td>62</td>
</tr>
</tbody>
</table>

Experiment 4
Variant of experiments 3 with more expensive subquery (10 bindings).

<table>
<thead>
<tr>
<th>Query</th>
<th>Time</th>
<th>I/O</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>8.6</td>
<td>5.2</td>
</tr>
<tr>
<td>Correlated</td>
<td>24.7</td>
<td>2.5</td>
</tr>
<tr>
<td>Magic</td>
<td>55</td>
<td>46</td>
</tr>
</tbody>
</table>

Conclusions

- Magic is:
  - Applicable to full SQL.
  - Suitable for rule-based optimization.
  - Efficient.
  - Stable.
  - Parallelizable.