Incomplete Databases on the Cheap

Todd J. Green
University of Pennsylvania
May 6, 2005
Example

Consider a database given by

\[
\begin{array}{|c|c|}
\hline
\text{SUPPLIER} & \text{LOCATION} \\
\hline
\text{Smith} & \text{London} \\
\hline
\end{array}
\quad \begin{array}{|c|c|}
\hline
\text{SUPPLIER} & \text{PRODUCT} \\
\hline
\text{Smith} & \text{Nails} \\
\hline
\end{array}
\]

and suppose that the relational view

\[
\pi_{\text{LOCATION, PRODUCT}}(SL \bowtie SP)
\]

is updated by adding two tuples, \langle \text{New York, Bolts}\rangle and \langle \text{Los Angeles, Nuts}\rangle.
Example

It is then natural to define the effect of this update to be:

\[
\begin{array}{|c|c|}
\hline
\text{SUPPLIER} & \text{LOCATION} \\
\hline
\text{Smith} & \text{London} \\
\text{x} & \text{New York} \\
\text{y} & \text{Los Angeles} \\
\hline
\end{array}
\quad \begin{array}{|c|c|}
\hline
\text{SUPPLIER} & \text{PRODUCT} \\
\hline
\text{Smith} & \text{Nails} \\
\text{x} & \text{Bolts} \\
\text{y} & \text{Nuts} \\
\hline
\end{array}
\]

The formalism we are using here is called a V-table. The V-table allows one to join on the unknown values. More precisely, one can prove that V-tables form a weak representation system with respect to the positive relational algebra.
Example

This is a nice formalism, but how does it relate to what DBMSs actually support? They support something not quite as powerful, at least on the face of it, called Codd tables.

<table>
<thead>
<tr>
<th>SUPPLIER</th>
<th>LOCATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>London</td>
</tr>
<tr>
<td></td>
<td>New York</td>
</tr>
<tr>
<td></td>
<td>Los Angeles</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SUPPLIER</th>
<th>PRODUCT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>Nails</td>
</tr>
<tr>
<td></td>
<td>Bolts</td>
</tr>
<tr>
<td></td>
<td>Nuts</td>
</tr>
</tbody>
</table>

The nulls are unlabeled, so we can’t join on them. More precisely, one can prove that Codd tables do not form a weak representation system even for conjunctive queries.
Encoding V-tables with Codd tables

However, it turns out that we can use Codd tables to *encode* V-tables, such that with a little query translation, we are able to join on the labeled null values.

<table>
<thead>
<tr>
<th>SUPPLIER</th>
<th>LOCATION</th>
<th>SUP-NULL</th>
<th>LOC-NULL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>London</td>
<td>@</td>
<td>@</td>
</tr>
<tr>
<td>@</td>
<td>New York</td>
<td>x</td>
<td>@</td>
</tr>
<tr>
<td>@</td>
<td>Los Angeles</td>
<td>y</td>
<td>@</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SUPPLIER</th>
<th>PRODUCT</th>
<th>SUP-NULL</th>
<th>PROD-NULL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>Nails</td>
<td>@</td>
<td>@</td>
</tr>
<tr>
<td>@</td>
<td>Bolts</td>
<td>x</td>
<td>@</td>
</tr>
<tr>
<td>@</td>
<td>Nuts</td>
<td>y</td>
<td>@</td>
</tr>
</tbody>
</table>
Encoding Theorem

We can state this result more precisely:

**Theorem 1.** *For every V-table \( T \), one can construct a Codd table \( T' \) such that, for every positive relational query \( q \) over \( T \), \( q \) can be rewritten into a positive relational query \( q' \) over \( T' \) such that*

\[
\bigcap \text{Mod}(q(T)) = \bigcap \text{Mod}(q'(T')).
\]
Encoding Theorem

Proof. (Sketch) $T'$ is constructed as in the preceding example. $q'$ is constructed by first computing $\overline{q}$ from $q$:

\[
\begin{align*}
q_1 \times q_2 & \rightarrow \overline{q_1} \times \overline{q_2}, \\
\pi_A(q_1) & \rightarrow \pi_A,\mathcal{N}_A(\overline{q_1}), \\
q_1 \cup q_2 & \rightarrow \overline{q_1} \cup \overline{q_2}, \\
\sigma_{A=c}(q_1) & \rightarrow \sigma_{A=c}(\overline{q_1}), \\
\sigma_{A=A'}(q_1) & \rightarrow \sigma_{A=A' \vee \mathcal{N}_A \mathcal{N}_{A'}}(\overline{q_1}), \\
\overline{T} & \rightarrow T'.
\end{align*}
\]

Now let $q' = \pi_A(\overline{q})$ where $A$ is the set of attributes in the result of $q$ (i.e., discard the null columns). \qed
Implementation

This theorem forms a foundation for a practical implementation.

We implemented in Java a middleware layer that gives the illusion of V-table support in Oracle. The layer is a JDBC driver which wraps the real Oracle JDBC driver, translating queries on the fly.

We also extended the SQL syntax to allow one to talk about the labeled nulls.
Example (Updates)

For example, the update

```sql
insert into s1 values (null('x'), 'New York');
insert into sp values (null('x'), 'Bolts');
```

translates to

```sql
insert into s1 values (null, 'New York', 'x', null);
insert into sp values (null, 'Bolts', 'x', null);
```
and the query

```sql
select s.*, p.product
from sl s, sp p
where s.supplier = p.supplier;
```

translates to

```sql
select s.supplier, s.location, p.product
from sl s, sp p
where s.supplier = p.supplier
  or s.supplier_null = p.supplier_null;
```
Experimental Evaluation

To see how much overhead the encoding scheme introduces, we compared performance of the TPC-H benchmark with and without the encoding.

The values in the null label columns running the benchmark were all null, as the evaluation was designed to measure overhead in the case where the scheme is giving no new answers. We used a 100MB data set.

Note that we did not use any indexes, so the numbers on the following slides represent a worst-case performance scenario (the next experiment we plan to run will use indexes).
# TPC-H Benchmark Results

<table>
<thead>
<tr>
<th>Query</th>
<th>Base time (sec)</th>
<th>Rewrite time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>67</td>
<td>65</td>
</tr>
<tr>
<td>Q2</td>
<td>154</td>
<td>340</td>
</tr>
<tr>
<td>Q3</td>
<td>160</td>
<td>703</td>
</tr>
<tr>
<td>Q4</td>
<td>34,144</td>
<td>35,595</td>
</tr>
<tr>
<td>Q5</td>
<td>917</td>
<td>5260</td>
</tr>
<tr>
<td>Q6</td>
<td>37</td>
<td>34</td>
</tr>
</tbody>
</table>
TPC-H Benchmark Results

![Graph showing TPC-H Benchmark Results](image-url)