CS748T
Distributed Database Management
Lecturer: Prof. M. Tamer Özsu

Semantic Data Control
in
Distributed Database Environment

by

Lubomir Stanchev

February 2001
University of Waterloo
INTRODUCTION

• Semantic Data Control is:
  • Checking the validity of a database

• The rules that determine whether a database is valid are called:
  • Constraints

• This presentation covers how to impose constraints

• Enforcing constraints is important because:
  • databases in which constraints are not imposed could have multiple interpretations:
    • i.e. become unusable
OUTLINE

• Type of Constraints
  • Data Constraints
    • Duplication Constraints
    • Integrity Constraints
  • Update Constraints

• Applications of Semantic Data Control

• Algorithms for Enforcing Constraints
  • Materialized View Maintenance Algorithms
  • Materialized View Update Algorithms
  • Integrity Constraint Compilation
  • Integrity Constraint Restoring Algorithms

• Conclusion and Future Research
DATA CONSTRAINTS

Note: Replica constraints are a special kind of view constraints

Figure 1. Distributed database data constraint classification

Figure 2. A distributed database example
UPDATE CONSTRAINTS

- Specify what kind of updates are allowed on the data.
- An update constraint consists of a triple \((P^+, P^-, P^+)\)
- Example:

\[
\begin{align*}
R_1(A, B) \\
R_2(B, C)
\end{align*}
\]

constraint: \(\forall t_1 \in R_1 \exists t_2 \in R_2 \ (t_1.B = t_2.A)\)
compiled constraint:

\[
\begin{align*}
&\text{(\(\exists t_2(t_1.B = t_2.B)\), TRUE,} \\
&t_1(\text{new}).B = t_1(\text{old}).B \lor \exists t_2(t_1(\text{new}).B = t_2.B)) \text{ on } R_1 \\
&\text{(TRUE, } \forall \exists t_1(t_1.B = t_2.B),} \\
&t_2(\text{new}).B = t_2(\text{old}).B \lor \forall \exists t_1(t_1.B = t_2(\text{new}).B)\text{ on table } R_2.
\end{align*}
\]

- Dynamic Constraint
  \((T, P^+, P^-, P^+)\)

Note: Dynamic update constraints can are specified usually as multiple update constraints.
# IMPOSING CONSTRAINTS

<table>
<thead>
<tr>
<th>constraint operation</th>
<th>static update constraints to be imposed</th>
<th>imposing dynamic duplicate constraints</th>
<th>imposing static duplicate constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>update to a materialized view</td>
<td>integrity constraint preserving + guaranteeing feasible translation to the underlying data</td>
<td>through deferred materialized view update algorithm</td>
<td>through immediate materialized view update algorithm</td>
</tr>
<tr>
<td>update to underlying data sources</td>
<td>integrity constraint preserving</td>
<td>through deferred materialized view maintenance algorithm</td>
<td>through immediate materialized view maintenance algorithm</td>
</tr>
</tbody>
</table>

Table 1. Shows how duplication constraints are imposed.

<table>
<thead>
<tr>
<th>imposing static integrity constraints</th>
<th>imposing dynamic integrity constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>through update constraints</td>
<td>by applying a deferred integrity constraint restoring algorithm</td>
</tr>
<tr>
<td>by applying an immediate integrity constraint restoring algorithm</td>
<td></td>
</tr>
</tbody>
</table>

Table 2. Shows how integrity constraints are imposed
EXAMPLE OF CONSTRAINT ENFORCEMENT

Table X

A: integer;
B: integer
key constraint: A is primary key,

Table Y

A: integer;
B: integer
key constraint: A is primary key,

View \( V = X \text{ union } Y \)

A: integer;
B: integer
key constraint: A is primary key,

Figure 4. An example database instance

(1,2) \quad X

(1,2) \quad Y

insert (1,3) \quad V
DATA WAREHOUSE APPLICATION

Figure 5. The data warehouse model
DATA VISUALIZATION APPLICATION

Materialized View Maintenance

MOBILE SYSTEMS

Figure 7. Mobile System Example

2. Materialized View Maintenance
Classification of Materialized View Maintenance Algorithms

Figure 8. Classification of materialized view maintenance algorithms
Incremental View Maintenance without Auxiliary Views

- Over SPJ queries: \( \pi_A(\sigma_Y(R_1 \bowtie R_2 \bowtie \ldots R_n) \)
  - \( V=\sigma_Y(R) \)
  - \( \sigma_Y(R') = \sigma_Y(R+\Delta R) = \sigma_Y(R)+\sigma_Y(\Delta R) = V + \sigma_Y(\Delta R) \)
  - \( V= \pi_A (R) \)
  - \( \pi_1(R+\Delta R) \neq \pi_1(R)+\pi_1(\Delta R) \)
  - e.g.
    - \( R=\{(2,1),(2,2),(3,1)\} \)
    - \( \Delta R=\{-(2,1)\} \)
    - \( \pi_1(R+\Delta R)= \pi_1(\{(2,2),(3,1)\})=\{(2),(3)\} \)
    - \( \pi_1(R)+\pi_1(\Delta R)=\{(2),(3)\}-\{2\}=\{3\} \)
  - The problem is that duplicate eliminating projection is not distributive relative to union
  - Solution - use duplicate preserving projections
    - Add tuple IDs to each table and require for them to participate in the attribute list of every projection. In our example:
      - \( R = \{(1,2,1),(2,2,2),(3,3,1)\} \)
      - \( \Delta R = \{-(1,2,1)\} \)
      - \( \pi_{1,2}(R+\Delta R) = \pi_{1,2}(\{(2,2,2),(3,3,1)\}) = \{(2,2),(3,3)\} \)
      - \( \pi_{1,2}(R+\Delta R) = \pi_{1,2}(\{(2,2,2),(3,3,1)\}) = \{(2,2),(3,3)\} \)
    - Implement duplicate preserving projections by tuple counting. In our example:
      - \( \pi_1(R+\Delta R) = \pi_1(\{(2,2,1), (3,1,1)\}) = \{(2,2),(3,1)\} \)
      - \( \pi_1(R) + \pi_1(\Delta R) = \{(2,2),(3,1)\}-\{(2,1)\} = \{(2,2),(3,1)\} \)
Incremental View Maintenance without Auxiliary Views (Cont'd)

- Over Join Queries: \( V = R_1 \bowtie R_2 \bowtie R_3 \)

\[ V' = (R_1 + \Delta R_1) \bowtie (R_2 + \Delta R_2) \bowtie (R_3 + \Delta R_3) = \]
\[ R_1 \bowtie R_2 \bowtie R_3 + R_1 \bowtie R_2 \bowtie \Delta R_3 + \ldots + R_1 \bowtie \Delta R_2 \bowtie \Delta R_3. \]

- We have used that join is distributive relative to union
- We will have \( 2^n - 1 \) expressions for \( \Delta V \) for a \( n \)-way join

Optimizations
- Tuple Marking:

<table>
<thead>
<tr>
<th>( R_1 )</th>
<th>( R_2 )</th>
<th>( R_1 \bowtie R_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>insert</td>
<td>insert</td>
<td>insert</td>
</tr>
<tr>
<td>insert</td>
<td>delete</td>
<td>ignore</td>
</tr>
<tr>
<td>insert</td>
<td>old</td>
<td>insert</td>
</tr>
<tr>
<td>delete</td>
<td>insert</td>
<td>ignore</td>
</tr>
<tr>
<td>delete</td>
<td>delete</td>
<td>delete</td>
</tr>
<tr>
<td>delete</td>
<td>old</td>
<td>delete</td>
</tr>
<tr>
<td>old</td>
<td>insert</td>
<td>insert</td>
</tr>
<tr>
<td>old</td>
<td>delete</td>
<td>delete</td>
</tr>
<tr>
<td>old</td>
<td>old</td>
<td>old</td>
</tr>
</tbody>
</table>

- Exploiting Integrity Constraints:

- \( R_1 = (\Delta, B), R_2 = (B, C) \), f.k. constraint \( R_1.B = R_2.B \)
- \( V = R_1 \bowtie R_2, \)
- \( \Delta V = \Delta R_1 \bowtie \Delta R_2 + \Delta R_1 \bowtie R_2 + R_1 \bowtie \Delta R_2 \)
- but \( R_1 \bowtie \Delta R_2 \) will be empty
- \( \Delta V = \Delta R_1 \bowtie \Delta R_2 + \Delta R_1 \bowtie R_2 \)
Incremental View Maintenance without Auxiliary Views (Cont'd)

- Over aggregate queries with min/max:
  - \( R_I = \{(1,2),(2,1),(2,4)\} \)
  - \( V = F_{\text{min}(2)}(R_I) = \{(1,2),(2,1)\} \)
  - If (1,2) is deleted from \( R_I \):
    - time to do incremental refresh = time to do direct update
    - reason: min, max are not self-maintainable aggregates

- Over aggregate queries with count/sum/sum:
  - \( V = F_{\text{sum}(2)}(R_I) = \{(1,2),(2,5)\} \)
  - If (1,2) is deleted from \( R_I \), updating \( V \) to \( \{(1,0),(2,5)\} \) is clearly wrong
  - **Solution:** use tuple accounting again
    - \( V \) will be stored as \( \{(1,2,#1),(2,5,#2)\} \)
    - when (1,2) is deleted from \( R_I \), \( V \) will be correctly updated to \( \{(2,5)\} \).
Incremental View Maintenance with Auxiliary Views

- Each materialized view may have a number of auxiliary views associated with it.
- The purpose of the auxiliary views is to allow the corresponding materialized view to be maintained without the base tables to be queried.
- Formally: \((\text{auxiliary_views} \{R_i \}_{i=1}^n, \Sigma, D, Q, \text{changes})\) holds if
  \[
  (\exists \text{ query } Q' ) \text{ s.t. } (\forall \mu \in \text{changes}(\Sigma)) [\text{if } (D+\mu) \in \text{Models}(\Sigma) \text{ then } Q(D+\mu) = Q'(\mu, Q(D), \{R_i \}^n_{i=1})]
  \]
- Basic idea: Reduce the size of the auxiliary views as much as possible by exploiting integrity constraints
- Example:
  \[
  X(A,B,C), Z(B,D,M), Y(C,F,G)
  \]
  \[
  V = \pi_{A,F,D}(\sigma_{D > 10 \land F < 3}(X \bowtie_{X.C = Y.C} Y \bowtie_{X.B = Z.B} Z))
  \]
  \[
  V = \pi_{A,B,C}(X) \bowtie_{X.C = Y.C} \pi_{C,F}(\sigma_{F < 3}(Y)) \bowtie_{X.B = Z.B} \pi_{B,D}(\sigma_{D > 10}(Z))
  \]
  The above rewriting is possible because the projection is duplicate preserving
  \[
  V_{new} = \pi_{A,B,C}(X + \Delta X) \bowtie_{X.C = Y.C} \pi_{C,F}(\sigma_{F < 3}(Y + \Delta Y)) \bowtie_{X.B = Z.B} \pi_{B,D}(\sigma_{D > 10}(Z + \Delta Z))
  \]
  But \(\Delta Y \bowtie X\) and \(\Delta Y \bowtie Z\) will be empty, so
  \[
  V_{new} = V + \pi_{A,B,C}(\Delta X) \bowtie_{X,C = Y,C} \pi_{C,F}(\sigma_{F < 3}(\Delta Y)) \bowtie_{X.B = Z.B} \pi_{B,D}(\sigma_{D > 10}(\Delta Z)) + \pi_{A,B,C}(\Delta X) \bowtie_{X,C = Y,C} \pi_{C,F}(\sigma_{F < 3}(\Delta Y)) \bowtie_{X.B = Z.B} \pi_{B,D}(\sigma_{D > 10}(\Delta Z))
  \]
  Therefore we need only the auxiliary views \(\pi_{B,D}(\sigma_{D > 10}(Z))\) and \(\pi_{C,F}(\sigma_{F < 3}(Y))\)

Note: The two type of algorithms use the same formula simplification based on integrity constraints
Materialized View Update Algorithms

- A materialized view update algorithm tries to translate updates made to materialized views to the underlying tables.
- Definitions:
  - $V_1 \geq V_2$ iff $\forall D_1, D_2 \in $Models$(\Sigma)$, $Q_1(D_1) = Q_1(D_2) \Rightarrow Q_2(D_1) = Q_2(D_2)$.
  - $V_1 \geq V_2$ means $V_1$ is more informative than $V_2$
  - If $V_1 \leq V_2$ and $V_2 \leq V_1$ then $V_1 = V_2$, i.e. the two views are equivalent
  - $S_1 \leq S_2$ if $\forall D_1, D_2 \in $Models$(\Sigma)$,
    
    \[ \forall Q_1 \in S_1 Q_1(D_1) = Q_1(D_2) \Rightarrow \forall Q_2 \in S_2 Q_2(D_1) = Q_2(D_2) \]\n  - $V_1$ and $V_2$ over $R$ are complement if $\{V_1, V_2\}$ is equivalent to $R$.
  - $V_1$ is complement to $V_2$ relative to $R$ means that $V_2$ contains the missing information from $V_1$ relative to $R$.
  - If $V \neq R$ and $V \cap R \neq \emptyset$ then $V$ has more than one complement
  - A translation of a view $V$ with underlying tables $R$ can be identified uniquely by specifying which complement of $V$ is to remain unchanged during the update.
Materialized View Update Algorithms (Cont'd)

- Example

<table>
<thead>
<tr>
<th>EMP</th>
<th>DEP</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>m</td>
</tr>
</tbody>
</table>

Employee Table X

\[
V_1 = Y
\]
\[
V_2 = \text{select EMP, MGR from } X \text{ inner join } Y
\]
\[
= \begin{array}{|c|c|}
  \hline
  \text{EMP} & \text{MGR} \\
  \hline
  A & X \\
  B & X \\
  C & Y \\
  \hline
\end{array}
\]

Department table Y

- We have the constraints that EMP is a primary key for the Employee table and Employee.DEP is a foreign key to the Department table.
- We have defined a translation of $V_2$ such that view $V_1$ is to remain invariant under the translation.
- If employee $A$ is replaced by employee $F$ in $V_2$, employee $A$ will be replaced by employee $F$ in table $X$.
- If $X$ is to be the invariant compliment of $V_2$, the above update is unfeasible.
- Solution: update constraint on $V_2$:
  - (FALSE, FALSE, t.new(EMP) = t.old(EMP)),

Figure 9. Example for view update
Constraint Compilation

• This is the process of imposing integrity constraints by defining appropriate update constraints (see slide 5)
• In general an integrity constraint is:

\[(Q_1x_1) \ldots (Q_nx_n) \lor A_i\]

• Where \(A_i\) is of the form \(P(x_1, \ldots x_k, Q(x_1, \ldots x_k))\) where \(Q\) is a query and \(P\) is a build in predicate like \(<, =, \in\).
• Example:
  • EMP(ENO, ENAME, TITLE), PROJ(PNO, PNAME, BUDGET).
  • Constraint: The total duration for all employees in the CAD project is less than 100
  • In the above form:

\[\forall j \{ [\text{PROJ}(j)] \Rightarrow [j \in (\text{select } * \text{ from PROJ as P where P.NAMD="CAD"}) \Rightarrow (\text{select } \text{sum(G.DUR) from ASG as G where G.PNO=j.PNO}<100)]}\]

• To impose update constraint, define a materialized view
  • \(V = (\text{select } * \text{ from PROJ as P where P.NAMD="CAD"})\)
  • Impose the update constraint
    • \((t=\emptyset, \text{FALSE}, t(\text{new}).A<100) \text{ on } V\)
Integrity Constraint Restoring Algorithms

- In a distributed database constraints may often be violated
- If this happens we define a repair as:

\[
\text{Repair}(D, \Sigma, IC) = \{ D' \mid D' \in \text{Models}(\Sigma, IC) \land \\
[\forall (D'' \in \text{Models}(\Sigma, IC)) |D-D'| \leq |D-D''|] \}
\]

- Example
  - \( R = \{(1,2),(1,3),(2,1)\} \), where the first attribute is a key.
  - The set of minimal repairs is \{ \{(1,2),(2,1)\}, \{(1,3),(2,1)\} \}.
  - \( R_1 = \{(1,2),(2,1)\} \) and \( R_2 = \{(1)\} \), constraint: \( R_1.2 \subseteq R_2.1 \)
    - delete the tuple (1,2) from \( R_1 \) or
    - add the tuple (2) to \( R_2 \).
Conclusion

• Formal framework for specifying and imposing constraints on the data in a distributed database environment.

• Open research problem:
  • Synchronizing the three type of algorithms that guarantee the validity of integrity, update and duplication constraints