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

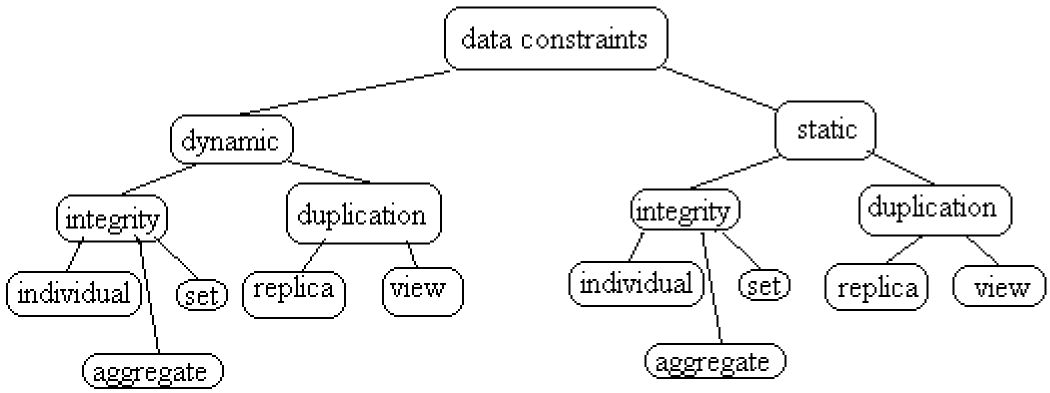


Figure 1. Distributed database data constraint classification

Note: Replica constraints are a special kind of view constraints

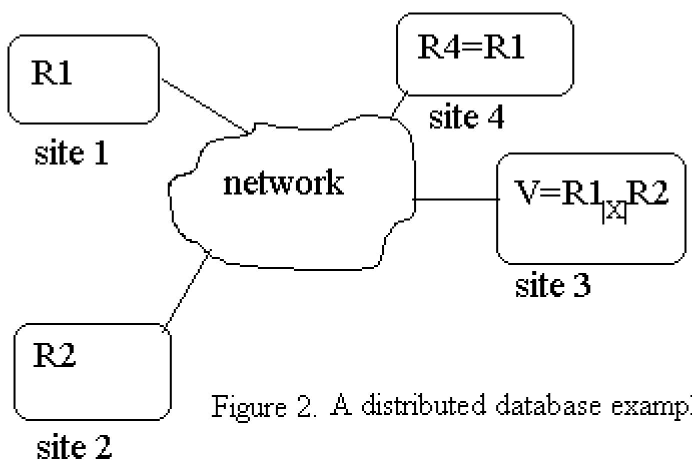


Figure 2. A distributed database example

UPDATE CONSTRAINTS

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

  $R_1(\underline{A}, B)$
  $R_2(\underline{B}, C)$

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

  - $(\exists t_2 (t_1.B = t_2.B)$, TRUE,
    $t_1(\text{new}).B = t_1(\text{old}).B \vee \exists t_2(t_1(\text{new}).B = t_2.B))$ on R_1
  - (TRUE, $\neg\exists t_1 (t_1.B = t_2.B)$,
    $t_2(\text{new}).B = t_2(\text{old}).B \vee \neg\exists t_1(t_1.B = t_2(\text{new}).B))$ on table R_2.

- Dynamic Constraint
  $(T, P^{+}, P^{-}, P^{+-})$

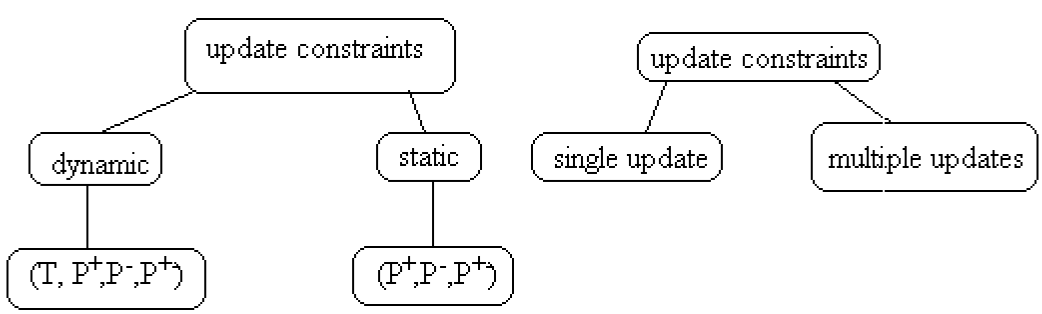


Figure3. Update Constraints Classification

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

IMPOSING CONSTRAINTS

constraint / operation	static update constraints to be imposed	imposing dynamic duplicate constraints	imposing static duplicate constraints
update to a materialized view	integrity constraint preserving + guaranteeing feasible translation to the underlying data	through deferred materialized view update algorithm	through immediate materialized view update algorithm
update to underlying data sources	integrity constraint preserving	through deferred materialized view maintenance algorithm	through immediate materialized view maintenance algorithm

Table 1. Shows how duplication constraints are imposed.

imposing static integrity constraints	**imposing dynamic integrity constraints**
through update constraints	by applying a deferred integrity constraint restoring algorithm
by applying an immediate integrity constraint restoring algorithm	

Table 2. Shows how integrity constraints are imposed

EXAMPLE OF CONSTRAINT ENROFORCEMENT

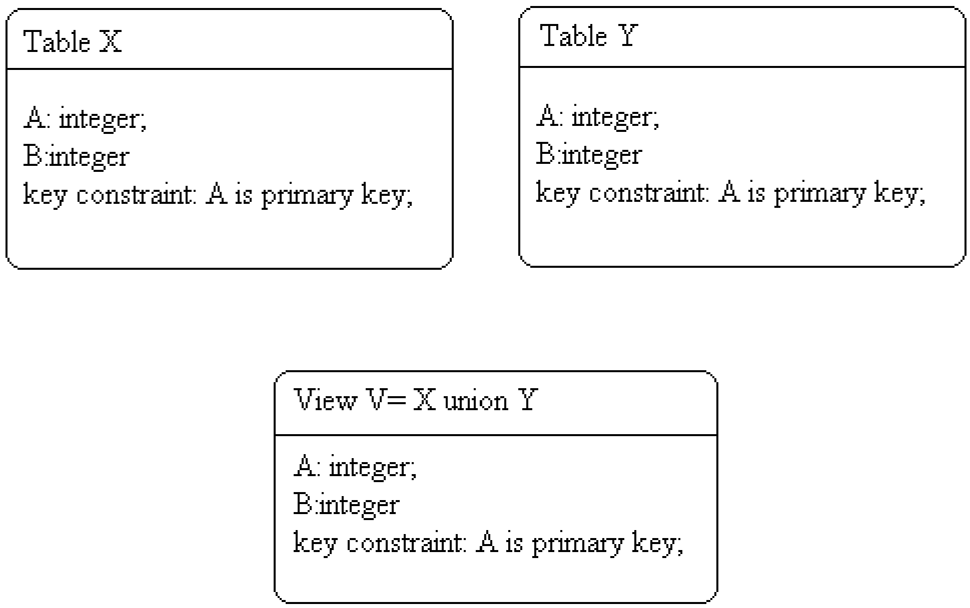
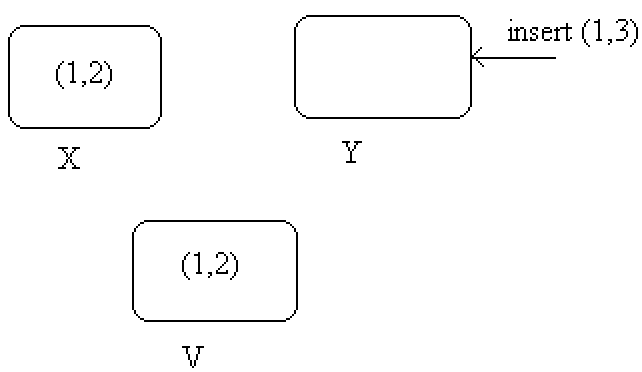


Figure 4. An example database instance



DATA WAREHOUSE APPLICATION

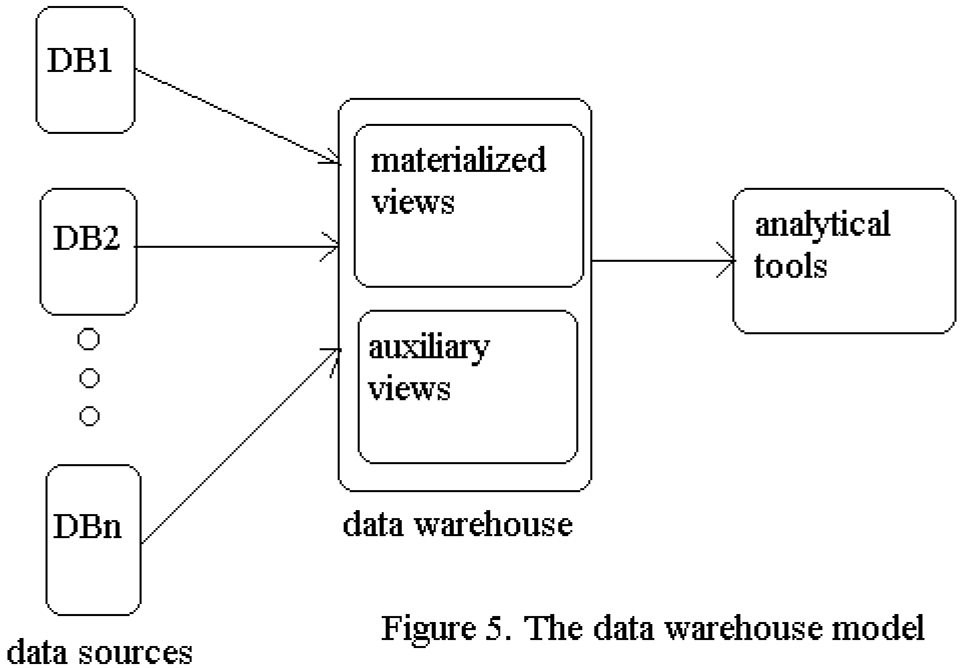


Figure 5. The data warehouse model

DATA VISUALIZATION APPLICATION

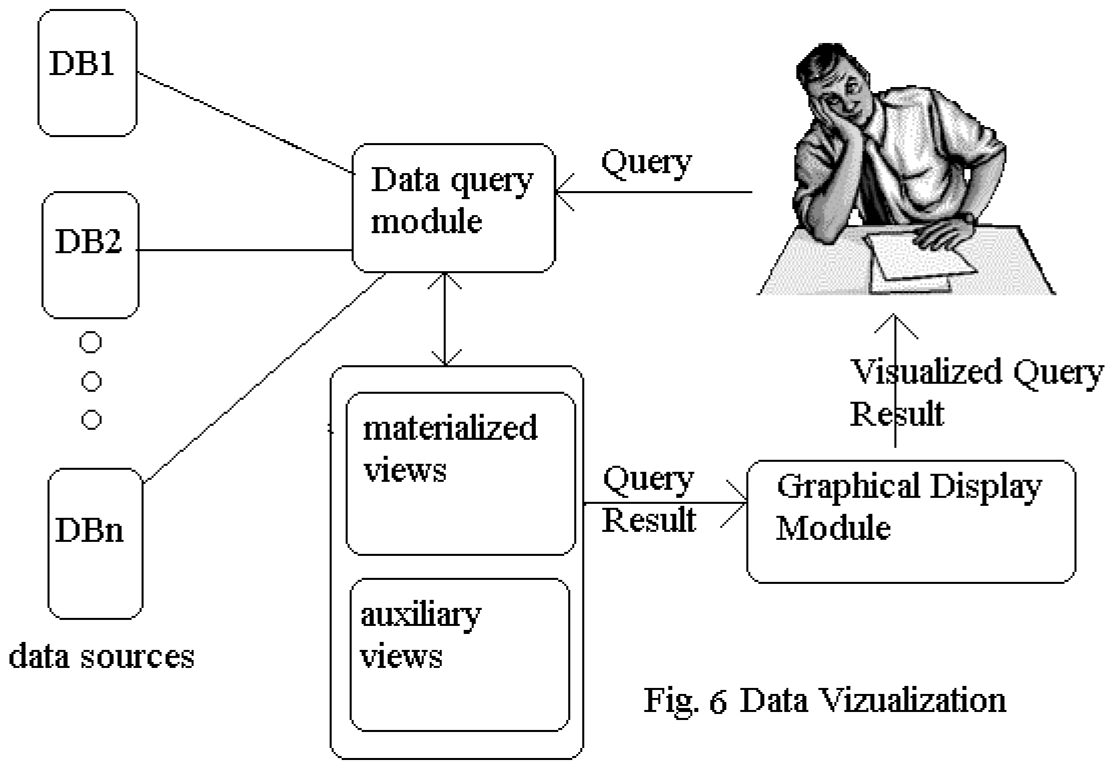


Fig. 6 Data Vizualization

MOBLIE SYSTEMS

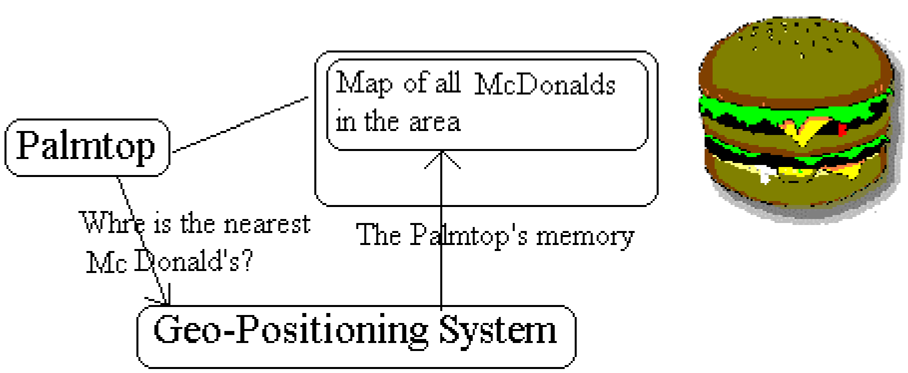


Figure 7. Mobile System Example

2. Materialized View Maintenance

Classification of Materialized View Maintenance Algorithms

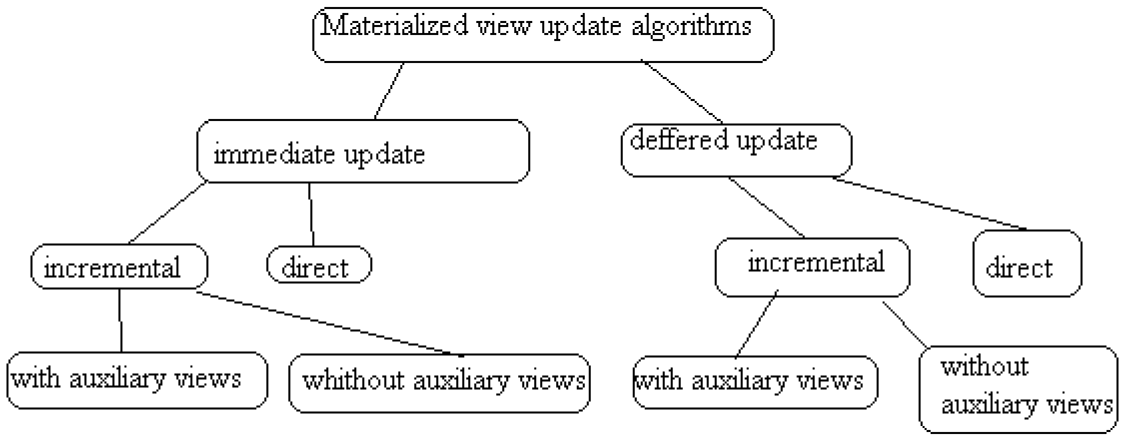


Figure 8. Classification of materialized view maintenance algorithms

Incremental View Maintenance without Auxiliary Views

- Over SPJ queries: $\pi_A(\sigma_\gamma(R_1 \bowtie R_2 \bowtie \ldots R_n)$
  - $V = \sigma_\gamma(R)$
    - $\sigma_\gamma(R') = \sigma_\gamma(R + {}_\Delta R) = \sigma_\gamma(R) + \sigma_\gamma({}_\Delta R) = V + \sigma_\gamma({}_\Delta R)$
  - $V = \pi_A(R)$
    - $\pi_1(R + {}_\Delta R) \neq \pi_1(\mathrm{R}) + \pi_1({}_\Delta R)$
    - e.g.
      - $R = \{(2,1),(2,2),(3,1)\}$
      - ${}_\Delta R = \{-(2,1)\}$
      - $\pi_1(R + {}_\Delta R) = \pi_1(\{(2,2),(3,1)\}) = \{(2),(3)\}$
      - $\pi_1(\mathrm{R}) + \pi_1({}_\Delta R) = \{(2),(3)\} - \{(2)\} = \{(3)\}$
    - 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:
        - $\pi_1(R + {}_\Delta R) = \pi_1(\{(2,2,\#1), (3,1,\#1)\})$
          $= \{(2,\#1),(3,\#1)\}$
        - $\pi_1(R) + \pi_1({}_\Delta R) = \{(2,\#2),(3,\#1)\} - \{(2,\#1)\}$
          $= \{(2,\#1),(3,\#1)\}$

Incremental View Maintenance without Auxiliary Views (Cont'd)

- Over Join Queries: $V = R_1 \bowtie R_2 \bowtie R_3$
  - $V' = (R_1 +_\Delta R_1) \bowtie (R_2 +_\Delta R_2) \bowtie (R_3 +_\Delta R_3) =$
    $R_1 \bowtie R_2 \bowtie R_3 + R_1 \bowtie R_2 \bowtie {}_\Delta R_3 + ... + {}_\Delta R_1 \bowtie {}_\Delta R_2 \bowtie {}_\Delta R_3$.
  - We have used that join is distributive relative to union
  - We will have 2^n-1 expressions for ΔV for a n-way join
- Optimizations
  - Tuple Marking:

R_1	R_2	$R_1 \bowtie R_2$
insert	insert	insert
insert	delete	ignore
insert	old	insert
delete	insert	ignore
delete	delete	delete
delete	old	delete
old	insert	insert
old	delete	delete
old	old	old

  - Exploiting Integrity Constraints:
    - $R_1 = (\underline{A}, B)$, $R_2 = (\underline{B}, C)$, 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_1={(1,2),(2,1),(2,4)}
  - $V = {}_1\mathcal{F}_{\mathrm{min(2)}}(R_1)$ = {(1,2),(2,1)}
  - If (1,2) is deleted from R_1
    - 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:
  - $V = {}_1\mathcal{F}_{\mathrm{sum(2)}}(R_1)$ = {(1,2),(2,5)}
  - If (1,2) is deleted from R_1, 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_1, 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: (auxiliary_views $\{R_i\}_{i=1}^{n}$,Σ,D,Q,changes) holds if

($\exists$ query Q') s.t. ($\forall\mu\in$changes(Σ)) [if (D+μ)∈Models(Σ) then Q(D+μ)=Q'(μ,Q(D), $\{R_i\}_{i=1}^{n}$)]

- Basic idea: Reduce the size of the auxiliary views as much as possible by exploiting integrity constraints
- Example:
  - $X(\underline{A},B,C)$, $Z(\underline{B},D,M)$, $Y(\underline{C},F,G)$
  - $V=\pi_{A,F,D}(\sigma_{D>10\ \wedge\ F<3}(X\bowtie_{X.C=Y.C}Y\bowtie_{X.B=Z.B}Z)$
  - $V=\pi_{A,B,C}(X)\bowtie_{X.C=Y.C}\pi_{C,F}(\sigma_{F<3}(Y))\bowtie_{X.B=Z.B}\pi_{B,D}(\sigma_{D>10}(Z))$
  - 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
  - $V_{new}=\text{V}+\pi_{A,B,C}(\Delta X)\bowtie_{X.C=Y.C}\pi_{C,F}(\sigma_{F<3}(\Delta Y))\bowtie_{X.B=Z.B}$ $\pi_{B,D}(\sigma_{D>10}(\Delta Z))+\pi_{A,B,C}(\Delta X)\bowtie_{X.C=Y.C}\pi_{C,F}(\sigma_{F<3}(\Delta Y))\bowtie_{X.B=Z.B}$ $\pi_{B,D}(\sigma_{D>10}(Z+\Delta Z))+\pi_{A,B,C}(\Delta X)\bowtie_{X.C=Y.C}\pi_{C,F}(\sigma_{F<3}(Y+\Delta Y))\bowtie_{X.B=Z.B}$ $\pi_{B,D}(\sigma_{D>10}(\Delta Z))$
  - 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 $\forall D_1, D_2 \in \text{Models}(\Sigma)$, $Q_1(D_1)=Q_1(D_2) \Rightarrow Q_2(D_1)=Q_2(D_2)$.
  - $V_1 \geq V_2$ 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
  - $\boldsymbol{S}_1 \leq \boldsymbol{S}_2$ if

  $\forall D_1, D_2 \in \text{Models}(\Sigma)$,

  $[\forall Q_1 \in \mathbf{S_1}\ Q_1(D_1)=Q_1(D_2)] \Rightarrow [\forall Q_2 \in \mathbf{S_2}\ Q_2(D_1)=Q_2(D_2)]$
  - V_1 and V_2 over $\boldsymbol{R}$ are complement if $\{V_1, V_2\}$ is equivalent to $\boldsymbol{R}$.
  - V_1 is complement to V_2 relative to $\boldsymbol{R}$ means that V_2 contains the missing information from V_1 relative to $\boldsymbol{R}$.
  - If $V \neq \boldsymbol{R}$ and $V \wedge R \neq \emptyset$ then V has more than one complement
  - A translation of a view V with underlying tables $\boldsymbol{R}$ can be identified uniquely by specifying which complement of V is to remain unchanged during the update.

Materialized View Update Algorithms (Cont'd)

- Example

EMP	DEP
A	1
B	1
C	m

Employee Table X

DEP	MGR
1	X
m	Y

Department table Y

V1=Y
V2=select EMP,MGR
from X inner join Y
=

EMP	MGR
A	X
B	X
C	Y

Figure 9. Example for view update

- 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)),

Constraint Compilation

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

$(Q_1x_1)\ ...(Q_nx_n)\vee A_i$

- Where A_i is of the form $P(x_1,...x_k,Q(x_1,...x_k))$ where Q is a query and P is a build in predicate like <,=,∈.
- 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:

∀*j* {[PROJ(*j*)]=>[*j*∈(select * from PROJ as P where P.NAMD="CAD") => (select sum(G.DUR) from ASG as G where G.PNO=j.PNO)<100)]}

  - To impose update constraint, define a materialized view
    - V = (select * from PROJ as P where P.NAMD="CAD")
  - Impose the update constraint
    - (t=∅,FALSE,t(new).A<100) on V

Integrity Constraint Restoring Algorithms

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

$$Repair(D,\Sigma,IC) = \{D' \mid D' \in Models(\Sigma,IC) \wedge [\forall (D'' \in Models(\Sigma,IC))\ |D-D'| <= |D-D''|]\}$$

- 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