CSEP 544

Lecture 9:
Provenance, Views
Announcements

• Homework 5:
  – See schedule examples in today’s email
  – Minor mistakes fixed yesterday (see email)
  – Homework due next Monday

• Reading assignment next week:
  – Long paper + short paper = 1 review

• Final Exam
  – Take home exam Saturday-Sunday 3/15-16
Data Provenance
Data Provenance

• Provenance inside the DBMS
  – Will discuss today

• Provenance outside of the DBMS
  – Much more messy; there is a standard, OPM (Open Provenance Model)
Provenance Annotations

- Some query produces an output table T(A,B,C)
- We store it over some period of time
- Later we ask: “where did this tuple come from?”
- The “provenance annotation” answers this.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>b1</td>
<td>c1</td>
<td></td>
</tr>
<tr>
<td>a2</td>
<td>b1</td>
<td>c1</td>
<td>provenance1</td>
</tr>
<tr>
<td>a2</td>
<td>b2</td>
<td>c2</td>
<td>provenance2</td>
</tr>
<tr>
<td>a2</td>
<td>b2</td>
<td>c3</td>
<td>provenance3</td>
</tr>
<tr>
<td>a2</td>
<td>b2</td>
<td>c3</td>
<td>provenance4</td>
</tr>
</tbody>
</table>
Provenance Annotations

• Start by annotating each tuple in the original database with a unique identifier; can be the Tuple Id (TID)

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>a1</td>
<td>b1</td>
<td>X1</td>
</tr>
<tr>
<td>a2</td>
<td>b1</td>
<td>X2</td>
</tr>
<tr>
<td>a2</td>
<td>b2</td>
<td>X3</td>
</tr>
</tbody>
</table>

• Next, compute the provenance expression inductively, based on the query plan
Join Operator

\[ \begin{array}{c|c|c} 
A & B & \text{Join} \\
\hline 
a1 & b1 & \text{X1} \\
a2 & b1 & \text{X2} \\
a2 & b2 & \text{X3} \\
\end{array} \]

\[ \begin{array}{c|c|c} 
B & C \\
\hline 
b1 & c1 & \text{Y1} \\
b2 & c2 & \text{Y2} \\
b2 & c3 & \text{Y3} \\
\end{array} \]

\[ \begin{array}{c|c|c} 
A & B & C \\
\hline 
a1 & b1 & c1 & \text{X1} \cdot \text{Y1} \\
a2 & b1 & c1 & \text{X2} \cdot \text{Y1} \\
a2 & b2 & c2 & \text{X3} \cdot \text{Y2} \\
a2 & b2 & c3 & \text{X3} \cdot \text{Y3} \\
\end{array} \]
Projection Operator

\[ \Pi = A_{a1}X_1 + X_2 + a_{a2}X_3 + X_4 + X_5 \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>b1</td>
<td>X1</td>
</tr>
<tr>
<td>a1</td>
<td>b2</td>
<td>X2</td>
</tr>
<tr>
<td>a2</td>
<td>b1</td>
<td>X3</td>
</tr>
<tr>
<td>a2</td>
<td>b2</td>
<td>X4</td>
</tr>
<tr>
<td>a2</td>
<td>b3</td>
<td>X5</td>
</tr>
</tbody>
</table>
Union Operator

\[
\begin{array}{c|c}
A & B \\
\hline
a1 & b1 \\
a2 & b2 \\
\end{array}
\quad \begin{array}{c|c}
A & B \\
\hline
a2 & b2 \\
a3 & b3 \\
\end{array}
\]

U

\[
\begin{array}{c|c}
A & B \\
\hline
a1 & b1 \\
a2 & b2 \\
a3 & b3 \\
\end{array}
\quad \begin{array}{c|c}
A & B \\
\hline
a2 & b2 \\
a3 & b3 \\
\end{array}
\]

X1 \quad X2+Y1

\begin{array}{c|c}
X1 & Y1 \\
\hline
a1 & b1 \\
a2 & b2 \\
a3 & b3 \\
\end{array}
\quad \begin{array}{c|c}
X1 & X2+Y1 \\
\hline
a1 & b1 \\
a2 & b2 \\
a3 & b3 \\
\end{array}
\quad \begin{array}{c|c}
X1 & X3 \\
\hline
a1 & b1 \\
a2 & b2 \\
a3 & b3 \\
\end{array}
\]
Selection Operator

$$\sigma_{A=a_1}$$

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>b1</td>
<td>X1</td>
</tr>
<tr>
<td>a1</td>
<td>b2</td>
<td>X2</td>
</tr>
<tr>
<td>a2</td>
<td>b1</td>
<td>X3</td>
</tr>
<tr>
<td>a2</td>
<td>b2</td>
<td>X4</td>
</tr>
<tr>
<td>a2</td>
<td>b3</td>
<td>X5</td>
</tr>
</tbody>
</table>

$$= \quad \begin{array}{cc}
A & B \\
a1 & b1 \\
a1 & b2 \\
\end{array} \quad X1 \\
\begin{array}{cc}
A & B \\
a1 & b2 \\
\end{array} \quad X2$$

We could simply remove the tuples filtered out. But it’s better to keep them around (we’ll see why). What is their annotation?
Selection Operator

\[ \sigma_{A=a1} \]

\begin{array}{|c|c|}
\hline
A & B \\
\hline
a1 & b1 \\
a1 & b2 \\
a2 & b1 \\
a2 & b2 \\
a2 & b3 \\
\hline
\end{array}

\begin{array}{|c|c|}
\hline
A & B \\
\hline
a1 & b1 \\
a1 & b2 \\
a2 & b1 \\
a2 & b2 \\
a2 & b3 \\
\hline
\end{array}

\begin{array}{|c|}
\hline
X1 \\
X2 \\
X3 \\
X4 \\
X5 \\
\hline
\end{array}

= 

We could simply remove the tuples filtered out. But it’s better to keep them around (we’ll see why). What is their annotation?
Simple Example 1

\[ \Pi_{AC}(R) \Join \Pi_{BC}(R) = \]

\[
\begin{array}{c|c|c|c}
\hline
 & A & B & C \\
\hline
 a & b & c & X \cdot X \\
 d & b & e & Y \cdot Y \\
 d & g & e & Y \cdot Z \\
 f & b & e & Z \cdot Y \\
 f & g & e & Z \cdot Z \\
\hline
\end{array}
\]

Discuss in class what these annotations mean.
Discuss in class what these annotations mean
Complex Example

\[ \sigma_{C=e} \prod_{AC}( \prod_{AC}(R) \bowtie \prod_{BC}(R) \cup \prod_{AB}(R) \bowtie \prod_{BC}(R)) = \]

\[ R = \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>d</td>
<td>b</td>
<td>e</td>
</tr>
<tr>
<td>f</td>
<td>g</td>
<td>e</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>c</td>
<td>(X \cdot X + X \cdot X) \cdot 0 = 0 \cdot 2 \cdot X^2 = 0</td>
</tr>
<tr>
<td>a</td>
<td>e</td>
<td>X \cdot Y \cdot 1 = X \cdot Y</td>
</tr>
<tr>
<td>d</td>
<td>c</td>
<td>Y \cdot X \cdot 0 = 0</td>
</tr>
<tr>
<td>d</td>
<td>e</td>
<td>(Y \cdot Y + Y \cdot Z + Y \cdot Y) \cdot 1 = 2 \cdot Y^2 + Y \cdot Z</td>
</tr>
<tr>
<td>f</td>
<td>e</td>
<td>(Z \cdot Z + Z \cdot Y + Z \cdot Z) \cdot 1 = 2 \cdot Z^2 + Y \cdot Z</td>
</tr>
</tbody>
</table>

Discuss in class what these annotations mean
**Definition.** A K-relation is a relation where each tuple is annotated with an element from the set K.

What we have described so far is an extension of the positive operations of the relational algebra to K-relations.

We assumed that K has the operators +, \cdot.
Identities on Provenance Expressions

The problem:

• We have defined provenance for a query plan $P$

• Given a query $Q$, we want the provenance to be independent of the plan

• Needed: if $P_1 = P_2$, then $\text{provenance}(P_1) = \text{Provenance}(P_2)$
Example

$q(x, y) := R(x), S(x, y), T(y)$

Do these plans compute the same provenance for the output $(a, b)$?

\[
\begin{array}{c}
R(x) \swarrow \\
S(x, y) \\
T(y)
\end{array}
\]

\[
\begin{array}{c}
R(x) \swarrow \\
S(x, y) \searrow \\
T(y)
\end{array}
\]

\[
R = \begin{array}{c}
\begin{array}{c}
\hline
x \\
a \\
\end{array} \\
X
\end{array}
\]

\[
S = \begin{array}{c}
\begin{array}{cc}
\hline
x & y \\
a & b \\
\end{array} \\
Y
\end{array}
\]

\[
T = \begin{array}{c}
\begin{array}{c}
\hline
y \\
b \\
\end{array} \\
Z
\end{array}
\]
Example

Do these two plans compute the same provenance expression for the output (a)?

q(x) := R(x), S(x)
q(x) := R(x), T(x)

V(x) := S(x)
V(x) := T(x)
q(x) := R(x), V(x)

R =
\[
\begin{array}{c}
x \\ a \\ X
\end{array}
\]

S =
\[
\begin{array}{c}
x \\ a \\ Y
\end{array}
\]

T =
\[
\begin{array}{c}
x \\ a \\ Z
\end{array}
\]
**Definition.** A structure \((K, +, \cdot, 0, 1)\) is called a **commutative semiring** if:

1. \((K, +, 0)\) is a **commutative monoid**:
   a. + is associative: \((x+y)+z=x+(y+z)\)
   b. + is commutative: \(x+y=y+x\)
   c. 0 is the identity for +: \(x+0=0+x=x\)

2. \((K, \cdot, 1)\) is a **commutative monoid**:
   a. … (similar identities)

3. \(\cdot\) distributes over +: \(x \cdot (y+z) = x \cdot y + x \cdot z\)

4. For all \(x\): \(x \cdot 0 = 0 \cdot x = 0\)
Theorem. The standard identities of the Bag algebra hold for K-relations iff \((K, +, \cdot, 0, 1)\) is a commutative semiring.

Definition. A structure \((K, +, \cdot, 0, 1)\) is called a commutative semiring if:

1. \((K,+ ,0)\) is a commutative monoid:
   a. + is associative: \((x+y)+z=x+(y+z)\)
   b. + is commutative: \(x+y=y+x\)
   c. 0 is the identity for +: \(x+0=0+x=x\)

2. \((K, \cdot, 1)\) is a commutative monoid:
   a. … (similar identities)

3. \(\cdot\) distributes over +: \(x \cdot (y+z) = x \cdot y + x \cdot z\)

4. For all \(x\): \(x \cdot 0 = 0 \cdot x = 0\)
Example

\[ q(x,u) := R(x,y), S(y,z), T(z,u) \]

In class: compute the provenance of the output \((a,b)\) for both plans.
Applications

\[ \sigma_{C=e} \Pi_{AC} ( \Pi_{AC}(R) \Join \Pi_{BC}(R) \cup \Pi_{AB}(R) \Join \Pi_{BC}(R) ) = \]

\[ R = \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>d</td>
<td>b</td>
<td>e</td>
</tr>
<tr>
<td>f</td>
<td>g</td>
<td>e</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>c</td>
</tr>
<tr>
<td>a</td>
<td>e</td>
</tr>
<tr>
<td>d</td>
<td>e</td>
</tr>
<tr>
<td>f</td>
<td>e</td>
</tr>
</tbody>
</table>

\[ Q: \text{Suppose we delete the tuple } (d,b,e) \text{ from } R. \text{ Which tuple(s) disappear from the result?} \]
Applications

\[ \sigma_{C=e} \Pi_{AC}(R) \bowtie \Pi_{BC}(R) \cup \Pi_{AB}(R) \bowtie \Pi_{BC}(R) = \]

\[ R = \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>d</td>
<td>b</td>
<td>e</td>
</tr>
<tr>
<td>f</td>
<td>g</td>
<td>e</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>X</th>
<th>A</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>c</td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>e</td>
<td>(X \cdot Y)</td>
</tr>
<tr>
<td>d</td>
<td>e</td>
<td>(2 \cdot Y^2 + Y \cdot Z)</td>
</tr>
<tr>
<td>f</td>
<td>e</td>
<td>(2 \cdot Z^2 + Y \cdot Z)</td>
</tr>
</tbody>
</table>

\[ Q: \text{Suppose we delete the tuple (d,b,e) from R. Which tuple(s) disappear from the result?} \]

\[ A: \text{Set } Y=0 \]
Applications

\[ \sigma_{C=e} \Pi_{AC}(\Pi_{AC}(R) \bowtie \Pi_{BC}(R) \cup \Pi_{AB}(R) \bowtie \Pi_{BC}(R)) = \]

\[ R = \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>d</td>
<td>b</td>
<td>e</td>
</tr>
<tr>
<td>f</td>
<td>g</td>
<td>e</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>c</td>
</tr>
<tr>
<td>a</td>
<td>e</td>
</tr>
<tr>
<td>d</td>
<td>e</td>
</tr>
<tr>
<td>f</td>
<td>e</td>
</tr>
</tbody>
</table>

Q: Suppose each tuple in R occurs 3 times (bag semantics). How many times occurs each tuple in the answer?
Applications

\[ \sigma_{C=e} \Pi_{\text{AC}}( \Pi_{\text{AC}}(R) \Join \Pi_{\text{BC}}(R) \cup \Pi_{\text{AB}}(R) \Join \Pi_{\text{BC}}(R)) = \]

\[ R = \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>d</td>
<td>b</td>
<td>e</td>
</tr>
<tr>
<td>f</td>
<td>g</td>
<td>e</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>c</td>
</tr>
<tr>
<td>a</td>
<td>e</td>
</tr>
<tr>
<td>d</td>
<td>e</td>
</tr>
<tr>
<td>f</td>
<td>e</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>c</td>
</tr>
<tr>
<td>a</td>
<td>e</td>
</tr>
<tr>
<td>d</td>
<td>e</td>
</tr>
<tr>
<td>f</td>
<td>e</td>
</tr>
</tbody>
</table>

Q: Suppose each tuple in R occurs 3 times (bag semantics). How many times occurs each tuple in the answer?

A. Set X=Y=Z=3
Sets of Contributing Tuples

$$\sigma_{C=e} \Pi_{AC}(\Pi_{AC}(R) \bowtie \Pi_{BC}(R)) = \Pi_{AB}(R) \bowtie \Pi_{BC}(R))$$

\[ R = \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>d</td>
<td>b</td>
<td>e</td>
</tr>
<tr>
<td>f</td>
<td>g</td>
<td>e</td>
</tr>
</tbody>
</table>

\[ X \quad Y \quad Z \]

<table>
<thead>
<tr>
<th>A</th>
<th>C</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>c</td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>e</td>
<td>X \cdot Y</td>
</tr>
<tr>
<td>d</td>
<td>e</td>
<td>2 \cdot Y^2 + Y \cdot Z</td>
</tr>
<tr>
<td>f</td>
<td>e</td>
<td>2 \cdot Z^2 + Y \cdot Z</td>
</tr>
</tbody>
</table>

Trace only the set of input tuples that contributed to an output tuple

This is also a semi-ring! Which one?
Variants of Provenance

• Depending on the application we may want to tune the degree of detail that we keep in the provenance

• Historically, researchers have first proposed ad-hoc definitions of provenance (often called lineage)

• Later, all these were proven to be special cases of semi-rings
Semirings for various models of provenance (1)

**Lineage** [Cui\textit{et al.}’00]
Set of contributing tuples

**Semiring:** \((\text{Lin}(X), +, \cup, \bot, \varnothing)\)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>d</td>
<td>b</td>
<td>e</td>
</tr>
<tr>
<td>f</td>
<td>g</td>
<td>e</td>
</tr>
</tbody>
</table>

\( \quad R = \quad \)

\( \quad Q = \quad \)

<table>
<thead>
<tr>
<th>A</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\( \quad \{Y,Z\} \quad \)

Examples, define the semi-ring (in class)
Semirings for various models of provenance (2)

\[ R = \]
\[
\begin{array}{ccc}
A & B & C \\
\hline
a & b & c \\
d & b & e \\
f & g & e \\
\end{array}
\]

\[ X \]

\[ Q = \]
\[
\begin{array}{cc}
A & C \\
\hline
d & e \\
\end{array}
\]

\[ \{Y\}, \{Y,Z\} \]

**Why-provenance** [Buneman’08]

Set of sets of witnesses

**Semiring:** \((\text{Why}(X), \cup, \sqcup, \emptyset, \{\emptyset\})\)

Examples, define the semi-ring (in class)
Semirings for various models of provenance (3)

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>X</td>
</tr>
<tr>
<td>d</td>
<td>b</td>
<td>e</td>
<td>Y</td>
</tr>
<tr>
<td>f</td>
<td>g</td>
<td>e</td>
<td>Z</td>
</tr>
</tbody>
</table>

\[ R = \]

\[ Q = \]

\[ A \]

\[ C \]

\[ \{Y\} \]

**Why-provenance** [Buneman’08]

Set of sets of *minimal* witnesses

**Semiring:** \((\text{PosBool}(X), \wedge, \vee, \top, \bot)\)

Examples, define the semi-ring (in class)
Semirings for various models of provenance (4)

R =

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>d</td>
<td>b</td>
<td>e</td>
</tr>
<tr>
<td>f</td>
<td>g</td>
<td>e</td>
</tr>
</tbody>
</table>

X

Y

Z

Q =

<table>
<thead>
<tr>
<th>A</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>e</td>
</tr>
</tbody>
</table>

Q = \{Y\}, \{Y\}, \{Y,Z\}

Trio lineage [Das Sarma’08]

Bags of sets of witnesses

Semiring: (Trio(X), +, \cdot, 0, 1)

Examples, define the semi-ring (in class)
Semirings for various models of provenance (5)

\[
R = \begin{array}{ccc} 
A & B & C \\
\hline
a & b & c \\
d & b & e \\
f & g & e \\
\end{array} \quad \text{X}
\]

\[
Q = \begin{array}{cc} 
A & C \\
\hline
d & e \\
\end{array} \quad \text{Y, Z}
\]

Polynomials with boolean coefficients [Green’09]
Sets of bags of contributing tuples
Semiring: \((B[X], +, \cdot, 0, 1)\)

Source: Tannen, EDBT 2010
Semirings for various models of provenance (6)

\[
R = \begin{array}{ccc}
A & B & C \\
\hline
a & b & c \\
d & b & e \\
f & g & e \\
\end{array}
\]

\[
Q = \begin{array}{cc}
A & C \\
\hline
\end{array}
\]

Notation:

\[
\{ \} \text{ set} \\
\[ \] \text{ bag}
\]

Provenance polynomials \cite{Green'07}

Bags of bags of contributing tuples

Semiring: \((\mathbb{N}[X], +, \cdot, 0, 1)\)
### Discretionary Access Control [LaPadula]
- Public = P
- Confidential = C
- Secret = S
- Top Secret = T
- No Such Thing… = 0

**R =**

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X=C</td>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>Y=P</td>
<td>d</td>
<td>b</td>
<td>e</td>
</tr>
<tr>
<td>Z=T</td>
<td>f</td>
<td>g</td>
<td>e</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 \cdot X^2 = ?</td>
<td>a</td>
<td>c</td>
</tr>
<tr>
<td>X \cdot Y = ?</td>
<td>a</td>
<td>e</td>
</tr>
<tr>
<td>2 \cdot Y^2 + Y \cdot Z = ?</td>
<td>d</td>
<td>e</td>
</tr>
<tr>
<td>2 \cdot Z^2 + Y \cdot Z = ?</td>
<td>f</td>
<td>e</td>
</tr>
</tbody>
</table>
**Discretionary Access Control** [LaPadula]
- Public = P
- Confidential = C
- Secret = S
- Top Secret = T
- No Such Thing… = 0

Alice has clearance S:
- Alice can read C data
- Alice cannot read T data
- Alice can write T data
- Alice cannot read C data

Why??
Discretionary Access Control [LaPadula]

- Public = P
- Confidential = C
- Secret = S
- Top Secret = T
- No Such Thing… = 0

Alice has clearance S:
- Alice can read C data
- Alice cannot read T data
- Alice can write T data
- Alice cannot read C data

Q: Join record A labeled C with record B labeled S. What is the label of (A,B)?

Why??
Discretionary Access Control [LaPadula]

- Public = P
- Confidential = C
- Secret = S
- Top Secret = T
- No Such Thing… = 0

Alice has clearance S:
- Alice can read C data
- Alice cannot read T data
- Alice can write T data
- Alice cannot read C data

Q: Join record A labeled C with record B labeled S. What is the label of (A,B)?
A: S
Discretionary Access Control [LaPadula]
- Public = P
- Confidential = C
- Secret = S
- Top Secret = T
- No Such Thing… = 0

Alice has clearance S:
- Alice can read C data
- Alice cannot read T data
- Alice can write T data
- Alice cannot read C data

Q: Join record A labeled C with record B labeled S. What is the label of (A,B)?
A: S

Q: Eliminate duplicates \{A, A, A, A\} labeled T, C, C, S. What is the label of A?
Discretionary Access Control [LaPadula]

- Public = P
- Confidential = C
- Secret = S
- Top Secret = T
- No Such Thing... = 0

Alice has clearance S:
- Alice can read C data
- Alice cannot read T data
- Alice can write T data
- Alice cannot read C data

Q: Join record A labeled C with record B labeled S. What is the label of (A,B)?
A: S

Q: Eliminate duplicates {A, A, A, A} labeled T, C, C, S. What is the label of A?
A: C
**Application**

**Discretionary Access Control** [LaPadula]
- Public = $P$
- Confidential = $C$
- Secret = $S$
- Top Secret = $T$
- No Such Thing… = 0

$$R = \begin{array}{ccc}
A & B & C \\
\hline
a & b & c \\
d & b & e \\
f & g & e \\
\end{array}$$

$$X = C \quad Y = P \quad Z = T$$

<table>
<thead>
<tr>
<th>A</th>
<th>C</th>
<th>\quad X = 2 \cdot X^2</th>
<th>\quad X \cdot Y</th>
<th>\quad 2 \cdot Y^2 + Y \cdot Z</th>
<th>\quad 2 \cdot Z^2 + Y \cdot Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>c</td>
<td>2 \cdot X^2</td>
<td>X \cdot Y</td>
<td>2 \cdot Y^2 + Y \cdot Z</td>
<td>2 \cdot Z^2 + Y \cdot Z</td>
</tr>
<tr>
<td>a</td>
<td>e</td>
<td>X \cdot Y</td>
<td>2 \cdot Y^2 + Y \cdot Z</td>
<td>2 \cdot Z^2 + Y \cdot Z</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>e</td>
<td>2 \cdot Y^2 + Y \cdot Z</td>
<td>2 \cdot Z^2 + Y \cdot Z</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$(A, \text{min}, \text{max}, 0, P), \text{ where } A = P < C < S < T < 0$
**Application**

**Discretionary Access Control** [LaPadula]
- Public = \( P \)
- Confidential = \( C \)
- Secret = \( S \)
- Top Secret = \( T \)
- No Such Thing… = 0

\[
R = \begin{array}{ccc}
A & B & C \\
a & b & c \\
d & b & e \\
f & g & e \\
\end{array}
\]

\[
\begin{array}{c|c|c}
A & C & \\
\hline
\text{X=C} & 2 \cdot X^2 = C \\
\text{Y=P} & X \cdot Y = C \\
\text{Z=T} & 2 \cdot Y^2 + Y \cdot Z = C \\
\end{array}
\]

\[
\begin{array}{c|c|c}
A & C & \\
\hline
\text{X=C} & 2 \cdot X^2 = C \\
\text{Y=P} & X \cdot Y = C \\
\text{Z=T} & 2 \cdot Y^2 + Y \cdot Z = C \\
\end{array}
\]

\[(A, \min, \max, 0, P), \text{ where } A = P < C < S < T < 0\]
Semirings

<table>
<thead>
<tr>
<th>Semiring</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>((B, \land, \lor, \top, \bot))</td>
<td>Set semantics</td>
</tr>
<tr>
<td>((\mathbb{N}, +, \cdot, 0, 1))</td>
<td>Bag semantics</td>
</tr>
<tr>
<td>((P(\Omega), \cup, \cap, \emptyset, \Omega))</td>
<td>Probabilistic events [FuhrRölleke 97]</td>
</tr>
<tr>
<td>((\text{BoolExp}(X), \land, \lor, \top, \bot))</td>
<td>Conditional tables (c-tables) [ImielinskiLipski 84]</td>
</tr>
<tr>
<td>((\mathbb{R}_+^\infty, \min, +, 1, 0))</td>
<td>Tropical semiring (cost/distrust score/confidence need)</td>
</tr>
<tr>
<td>((A, \min, \max, 0, P)) where (A = P &lt; C &lt; S &lt; T &lt; 0)</td>
<td>Access control levels [PODS8]</td>
</tr>
</tbody>
</table>
A provenance hierarchy

most informative

least informative

N[\mathcal{X}]

B[\mathcal{X}]

Trio(\mathcal{X})

Why(\mathcal{X})

Lin(\mathcal{X})

PosBool(\mathcal{X})
A provenance hierarchy

Example: $2x^2y + xy + 5y^2 + z$

A path downward from $K_1$ to $K_2$ indicates that there exists an onto (surjective) semiring homomorphism $h : K_1 \rightarrow K_2$
A provenance hierarchy

Example: $2x^2y + xy + 5y^2 + z$

A path downward from $K_1$ to $K_2$ indicates that there exists an onto (surjective) semiring homomorphism $h : K_1 \to K_2$
A provenance hierarchy

Example: $2x^2y + xy + 5y^2 + z$

\[
\frac{N[X]}{\text{drop coefficients}} \quad \frac{B[X]}{x^2y + xy + y^2 + z} \quad \text{drop exponents} \quad \frac{\text{Trio}(X)}{3xy + 5y + z}
\]

\[
\frac{\text{Why}(X)}{\text{why}} \quad \frac{\text{Lin}(X)}{\text{linear}} \quad \frac{\text{PosBool}(X)}{\text{positive boolean}}
\]

A path downward from $K_1$ to $K_2$ indicates that there exists an **onto** (surjective) semiring homomorphism $h : K_1 \rightarrow K_2$
A provenance hierarchy

Example: $2x^2y + xy + 5y^2 + z$

- $N[X]$
  - drop coefficients: $x^2y + xy + y^2 + z$
  - drop exponents: $B[X]$
- $Trio(X)$: $3xy + 5y + z$
- $Why(X)$
  - drop both exp. and coeff.: $xy + y + z$
- $Lin(X)$
- $PosBool(X)$

A path downward from $K_1$ to $K_2$ indicates that there exists an onto (surjective) semiring homomorphism $h : K_1 \rightarrow K_2$
A provenance hierarchy

Example: $2x^2y + xy + 5y^2 + z$

$\text{N}[X]$

- drop coefficients
  - $x^2y + xy + y^2 + z$

$\text{B}[X]$

- drop both exp. and coeff.
  - $xy + y + z$

- collapse terms
  - $xyz$

$\text{Trio}(X) \quad 3xy + 5y + z$

$\text{Why}(X)$

- drop exponents
  - $3xy + 5y + z$

$\text{Lin}(X)$

- collapse terms
  - $xyz$

$\text{PosBool}(X)$

A path downward from $K_1$ to $K_2$ indicates that there exists an onto (surjective) semiring homomorphism $h: K_1 \rightarrow K_2$
A provenance hierarchy

Example: $2x^2y + xy + 5y^2 + z$

A path downward from $K_1$ to $K_2$ indicates that there exists an onto (surjective) semiring homomorphism $h : K_1 \rightarrow K_2$
Using homomorphisms to relate models

Example: $2x^2y + xy + 5y^2 + z$

- **N[X]**
  - drop coefficients
  - $x^2y + xy + y^2 + z$
- **B[X]**
  - drop both exp. and coeff.
  - $xy + y + z$
- **Trio(X)**
  - drop exponents
  - $3xy + 5y + z$
- **Why(X)**
  - apply absorption ($ab + b = b$)
  - $y + z$
- **Lin(X)**
  - collapse terms
  - $xyz$
- **PosBool(X)**
  -

**Homomorphism?**

$h(x+y) = h(x)+h(y)$

$h(xy)=h(x)h(y)$

$h(0)$=$0$

$h(1)$=$1$

Moreover, for these homomorphisms

$h(x)$=$x$
Views
Overview

Views are ubiquitous in data management:

• Used in SQL as names for predefined queries

• More generally, any derived data is a view
Views

• A view in SQL =
  – A table computed from other tables, s.t., whenever the base tables are updated, the view is updated too

• More generally:
  – A view is derived data that keeps track of changes in the original data

• Compare:
  – A function computes a value from other values, but does not keep track of changes to the inputs
A Simple View

Create a view that returns for each store
the prices of products purchased at that store

CREATE VIEW StorePrice AS
SELECT DISTINCT x.store, y.price
FROM Purchase x, Product y
WHERE x.product = y.pname

This is like a new table
StorePrice(store, price)
We Use a View Like Any Table

• A "high end" store is a store that sell some products over 1000.
• For each customer, return all the high end stores that they visit.

```
SELECT DISTINCT u.name, u.store
FROM Purchase u, StorePrice v
WHERE u.store = v.store
    AND v.price > 1000
```
Types of Views

- **Virtual views**
  - Used in databases
  - Computed only on-demand – slow at runtime
  - Always up to date

- **Materialized views**
  - Used in data warehouses
  - Pre-computed offline – fast at runtime
  - May have stale data (must recompute or update)
  - Indexes *are* materialized views
For each customer, find all the high end stores that they visit.

**CREATE VIEW** StorePrice AS

```
SELECT DISTINCT x.store, y.price
FROM Purchase x, Product y
WHERE x.product = y.pname
```

```
SELECT DISTINCT u.name, u.store
FROM Purchase u, StorePrice v
WHERE u.store = v.store
    AND v.price > 1000
```
Query Modification

For each customer, find all the high end stores that they visit.

CREATE VIEW StorePrice AS
SELECT DISTINCT x.store, y.price
FROM Purchase x, Product y
WHERE x.product = y.pname

SELECT DISTINCT u.name, u.store
FROM Purchase u, StorePrice v
WHERE u.store = v.store
AND v.price > 1000

SELECT DISTINCT u.customer, u.store
FROM Purchase u,
(SELECT DISTINCT x.store, y.price
FROM Purchase x, Product y
WHERE x.product = y.pname) v
WHERE u.store = v.store
AND v.price > 1000
For each customer, find all the high end stores that they visit.

Original query:

\[
\begin{align*}
\text{SELECT} & \quad \text{DISTINCT } u.\text{customer}, u.\text{store} \\
\text{FROM} & \quad \text{Purchase } u, \text{Purchase } x, \text{Product } y \\
\text{WHERE} & \quad u.\text{store} = x.\text{store} \\
\text{AND} & \quad y.\text{price} > 1000 \\
\text{AND} & \quad x.\text{product} = y.\text{pname}
\end{align*}
\]

Modified query:

\[
\begin{align*}
\text{SELECT} & \quad \text{DISTINCT } u.\text{customer}, u.\text{store} \\
\text{FROM} & \quad \text{Purchase } u, \text{Purchase } x, \text{Product } y \\
\text{WHERE} & \quad u.\text{store} = x.\text{store} \\
\text{AND} & \quad y.\text{price} > 1000 \\
\text{AND} & \quad x.\text{product} = y.\text{pname}
\end{align*}
\]

Modified and unnested query:

\[
\begin{align*}
\text{SELECT} & \quad \text{DISTINCT } u.\text{customer}, u.\text{store} \\
\text{FROM} & \quad \text{Purchase } u, \\
\text{\quad (SELECT} & \quad \text{DISTINCT } x.\text{store}, y.\text{price} \\
\text{\quad \text{FROM} & \quad \text{Purchase } x, \text{Product } y \\
\text{\quad \quad \text{WHERE} & \quad x.\text{product} = y.\text{pname})} v \\
\text{\quad \text{WHERE} & \quad u.\text{store} = v.\text{store} \\
\text{\quad \text{AND} & \quad v.\text{price} > 1000}
\end{align*}
\]

Notice that Purchase occurs twice. Why?
Further Virtual View Optimization

Retrieve all stores whose name contains ACME

```
CREATE VIEW StorePrice AS
    SELECT DISTINCT x.store, y.price
    FROM Purchase x, Product y
    WHERE x.product = y.pname

SELECT DISTINCT v.store
FROM StorePrice v
WHERE v.store like '%ACME%'
```
Retrieve all stores whose name contains ACME

CREATE VIEW StorePrice AS
SELECT DISTINCT x.store, y.price
FROM Purchase x, Product y
WHERE x.product = y.pname

SELECT DISTINCT v.store
FROM StorePrice v
WHERE v.store like ‘%ACME%’

Modified query:

(SELECT DISTINCT x.store, y.price
FROM Purchase x, Product y
WHERE x.product = y.pname) v
WHERE v.store like ‘%ACME%’
Further Virtual View Optimization

Retrieve all stores whose name contains ACME

**SELECT DISTINCT x.store FROM Purchase x, Product y WHERE x.product = y.pname AND x.store like '%ACME%'

**Modified query:**

**SELECT DISTINCT v.store FROM**

(SELECT DISTINCT x.store, y.price FROM Purchase x, Product y WHERE x.product = y.pname) v

WHERE v.store like '%ACME%'

We can further optimize! How?
Retrieve all stores whose name contains ACME

```
SELECT DISTINCT x.store 
FROM Purchase x, Product y 
WHERE x.product = y.pname 
    AND x.store like '%ACME%'
```

Modified and unnested query:

```
SELECT DISTINCT x.store 
FROM Purchase x 
WHERE x.store like '%ACME%'
```

Final Query

Assuming Product.pname is a key and Purchase.product is a foreign key
Applications of Virtual Views

• **Increased physical data independence.** E.g.
  – Vertical data partitioning
  – Horizontal data partitioning

• **Logical data independence.** E.g.
  – Change schemas of base relations (i.e., stored tables)

• **Security**
  – View reveals only what the users are allowed to know
### Physical Data Independence: Vertical Partitioning

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Address</th>
<th>Resume</th>
<th>Picture</th>
</tr>
</thead>
<tbody>
<tr>
<td>234234</td>
<td>Mary</td>
<td>Huston</td>
<td>Clob1…</td>
<td>Blob1…</td>
</tr>
<tr>
<td>345345</td>
<td>Sue</td>
<td>Seattle</td>
<td>Clob2…</td>
<td>Blob2…</td>
</tr>
<tr>
<td>345343</td>
<td>Joan</td>
<td>Seattle</td>
<td>Clob3…</td>
<td>Blob3…</td>
</tr>
<tr>
<td>234234</td>
<td>Ann</td>
<td>Portland</td>
<td>Clob4…</td>
<td>Blob4…</td>
</tr>
</tbody>
</table>

**T1**

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>234234</td>
<td>Mary</td>
<td>Huston</td>
</tr>
<tr>
<td>345345</td>
<td>Sue</td>
<td>Seattle</td>
</tr>
<tr>
<td>…</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**T2**

<table>
<thead>
<tr>
<th>SSN</th>
<th>Resume</th>
</tr>
</thead>
<tbody>
<tr>
<td>234234</td>
<td>Clob1…</td>
</tr>
<tr>
<td>345345</td>
<td>Clob2…</td>
</tr>
</tbody>
</table>

**T3**

<table>
<thead>
<tr>
<th>SSN</th>
<th>Picture</th>
</tr>
</thead>
<tbody>
<tr>
<td>234234</td>
<td>Blob1…</td>
</tr>
<tr>
<td>345345</td>
<td>Blob2…</td>
</tr>
</tbody>
</table>
CREATE VIEW Resumes AS
SELECT T1.ssn, T1.name, T1.address, T2.resume, T3.picture
FROM T1, T2, T3
WHERE T1.ssn = T2.ssn and T2.ssn = T3.ssn
Vertical Partitioning

```
SELECT address
FROM    Resumes
WHERE   name = 'Sue'
```

We want the system to query only table T1.

Will that happen?
Vertical Partitioning

• Hot trend in databases today for analytics
• Main idea:
  – Storage = Column(TID, value) pairs
  – Sort by TID $\Rightarrow$ enables reconstructing the table
  – Compress $\Rightarrow$ great compression, minimize I/O
  – Updates = VERY, VERY expensive
• Companies: **C-Store** and **Vertica**
### Horizontal Partitioning

**Customers**

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>234234</td>
<td>Mary</td>
<td>Huston</td>
</tr>
<tr>
<td>345345</td>
<td>Sue</td>
<td>Seattle</td>
</tr>
<tr>
<td>345343</td>
<td>Joan</td>
<td>Seattle</td>
</tr>
<tr>
<td>234234</td>
<td>Ann</td>
<td>Portland</td>
</tr>
<tr>
<td>--</td>
<td>Frank</td>
<td>Calgary</td>
</tr>
<tr>
<td>--</td>
<td>Jean</td>
<td>Montreal</td>
</tr>
</tbody>
</table>

**CustomersInHuston**

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>234234</td>
<td>Mary</td>
<td>Huston</td>
</tr>
</tbody>
</table>

**CustomersInSeattle**

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>345345</td>
<td>Sue</td>
<td>Seattle</td>
</tr>
<tr>
<td>345343</td>
<td>Joan</td>
<td>Seattle</td>
</tr>
</tbody>
</table>

...
CREATE VIEW Customers AS
  CustomersInHouston
  UNION ALL
  CustomersInSeattle
  UNION ALL
  ...

Horizontal Partitioning
Horizontal Partitioning

```
SELECT name
FROM Customers
WHERE city = 'Seattle'
```

Which tables are queried by the system?

WHY ???
Horizontal Partitioning

SELECT name
FROM Customers
WHERE city = 'Seattle'

Now even humans can’t tell which table contains customers in Seattle

CREATE VIEW Customers AS
CustomersInXXX
UNION ALL
CustomersInYYY
UNION ALL

...
Horizontal Partitioning

A hack around the problem:

```sql
CREATE VIEW Customers AS
(SELECT SSN, name, 'Huston' as city
FROM CustomersInHuston)
UNION ALL
(SELECT SSN, name, 'Seattle' as city
FROM CustomersInSeattle)
UNION ALL
...
```
Horizontal Partitioning

```
SELECT name
FROM Customers
WHERE city = 'Seattle'
```

```
SELECT name
FROM CustomersInSeattle
```
Denormalization

• Pre-compute a view that is the join of several tables
• The view is now a relation that is not in BCNF (why not?)

```
CREATE VIEW CustomerPurchase AS
SELECT x.customer, x.store, y.pname, y.price
FROM Purchase x, Product y
WHERE x.product = y.pname
```

Purchase/customer, product, store
Product/pname, price
### Views and Security

**Customers:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>Huston</td>
<td>450.99</td>
</tr>
<tr>
<td>Sue</td>
<td>Seattle</td>
<td>-240</td>
</tr>
<tr>
<td>Joan</td>
<td>Seattle</td>
<td>333.25</td>
</tr>
<tr>
<td>Ann</td>
<td>Portland</td>
<td>-520</td>
</tr>
</tbody>
</table>

Fred is not allowed to see Balance

John is not allowed to see >0 balances

**CREATE VIEW PublicCustomers**

```sql
CREATE VIEW PublicCustomers
SELECT Name, Address
FROM Customers
```

**CREATE VIEW BadCreditCustomers**

```sql
CREATE VIEW BadCreditCustomers
SELECT *
FROM Customers
WHERE Balance < 0
```
Which one needs query expansion, which one needs query answering using views?
Horizontal Partitioning as LAV

CREATE VIEW CustomersInSeattle AS
(SELECT * FROM Customers
WHERE city = 'Seattle')

CREATE VIEW CustomersInHouston AS
(SELECT * FROM Customers
WHERE city = 'Houston')

....

SELECT name FROM Customers
WHERE city = 'Seattle'

SELECT name FROM CustomersInSeattle
Indexes are Materialized Views

CREATE INDEX W ON Product(weight)
CREATE INDEX P ON Product(price)

Indexes as LAV:

CREATE VIEW W AS
  SELECT weight, pid
  FROM Product y
CREATE VIEW P AS
  SELECT price, pid
  FROM Product y

SELECT weight, price
FROM Product
WHERE weight > 10
  and price < 100

“Covering indexes”: query uses only the indexes

SELECT x.weight, y.price
FROM W x, P y
WHERE x.weight > 10
  and y.price < 100
  and x.pid = y.pid

Product(pid, name, weight, price, …)
Answering Queries Using Views

• We have several materialized views:
  – V1, V2, ..., Vn

• Given a query Q
  – Answer it by using views instead of base tables

• Variation: Query rewriting using views
  – Answer it by rewriting it to another query first

• Example: if the views are indexes, then we rewrite the query to use indexes
Rewriting Queries Using Views

Purchase(buyer, seller, product, store)
Person(pname, city)

CREATE VIEW SeattleView AS
SELECT y.buyer, y.seller, y.product, y.store
FROM Person x, Purchase y
WHERE x.city = 'Seattle' AND x.pname = y.buyer

Goal: rewrite this query in terms of the view

SELECT y.buyer, y.seller
FROM Person x, Purchase y
WHERE x.city = 'Seattle' AND x.pname = y.buyer AND y.product = 'gizmo'
Rewriting Queries Using Views

```
SELECT y.buyer, y.seller
FROM Person x, Purchase y
WHERE x.city = 'Seattle' AND x..pname = y.buyer AND y.product='gizmo'
```

```
SELECT buyer, seller
FROM SeattleView
WHERE product='gizmo'
```
Rewriting is not always possible

CREATE VIEW DifferentView AS
SELECT y.buyer, y.seller, y.product, y.store
FROM Person x, Purchase y, Product z
WHERE x.city = 'Seattle' AND
      x.pname = y.buyer AND
      y.product = z.name AND
      z.price < 100

SELECT y.buyer, y.seller
FROM Person x, Purchase y
WHERE x.city = 'Seattle' AND
      x.pname = y.buyer AND
      y.product = 'gizmo'

SELECT buyer, seller
FROM DifferentView
WHERE product = 'gizmo'

"Maximally contained rewriting"
Technical Aspects

- View inlining, or query modification
- Query answering using views
- Updating views
- Incremental view update
Technical Aspects of Views

• Simplifying queries after the views have been in-lined
  – Query un-nesting
  – Query minimization

• Handling updates
  – Updating virtual views
  – Incremental update of materialized views
Creating a View

CREATE VIEW Expensive-Product AS
  SELECT pname
  FROM Product
  WHERE price > 100

Updateable view

Purchase(customer, product, store)
Product(pname, price)

INSERT INTO Expensive-Product
VALUES(‘Gizmo’)
Updatable Views

• Have a virtual view $V(A_1, A_2, \ldots)$ over tables $R_1, R_2, \ldots$
• User wants to update a tuple in $V$
  – Insert/modify/delete
• Can we translate this into updates to $R_1, R_2, \ldots$?
• If yes: $V = \text{“an updateable view”}$
• If not: $V = \text{“a non-updateable view”}$
Updating Views

Purchase(customer, product, store)
Product(pname, price)

CREATE VIEW Expensive-Product AS
SELECT pname
FROM Product
WHERE price > 100

INSERT INTO Expensive-Product VALUES('Gizmo')

INSERT INTO Product VALUES('Gizmo', NULL)
Updating Views

Purchase(customer, product, store)
Product(pname, price)

CREATE VIEW AcmePurchase AS
SELECT customer, product
FROM Purchase
WHERE store = 'AcmeStore'

INSERT INTO AcmePurchase
VALUES('Joe', 'Gizmo')
Updating Views

Purchase(customer, product, store)
Product(pname, price)

CREATE VIEW AcmePurchase AS
  SELECT customer, product
  FROM Purchase
  WHERE store = 'AcmeStore'

INSERT INTO AcmePurchase VALUES('Joe', 'Gizmo')

INSERT INTO Purchase VALUES('Joe', 'Gizmo', NULL)

Updateable view

Note this
Updating Views

Most views are non-updateable.

CREATE VIEW CustomerPrice AS
    SELECT x.customer, y.price
    FROM Purchase x, Product y
    WHERE x.product = y.pname

INSERT INTO CustomerPrice
VALUES('Joe', 200)
Incremental View Update

Also known as view synchronization

• Immediate synchronization = after each update

• Deferred synchronization
  – Lazy = at query time
  – Periodic
  – Forced = manual
Incremental View Update

Order(cid, pid, date)
Product(pid, name, price)

CREATE VIEW FullOrder AS
SELECT x.cid, x.pid, x.date, y.name, y.price
FROM Order x, Product y
WHERE x.pid = y.pid

UPDATE Product
SET price = price / 2
WHERE pid = '12345'

UPDATE FullOrder
SET price = price / 2
WHERE pid = '12345'

No need to recompute the entire view!
Incremental View Update

Product(pid, name, category, price)

CREATE VIEW Categories AS
SELECT DISTINCT category
FROM Product

DELETE Product
WHERE pid = '12345'

DELETE Categories
WHERE category in
(SELECT category
FROM Product
WHERE pid = '12345')

It doesn’t work! Why? How can we fix it?
Incremental View Update

Product\((\text{pid}, \text{name}, \text{category}, \text{price})\)

```sql
CREATE VIEW Categories AS
SELECT category, count(*) as c
FROM Product
GROUP BY category
```

DELETE Product
WHERE pid = '12345'

UPDATE Categories
SET c = c-1
WHERE category in
(SELECT category
FROM Product
WHERE pid = '12345');
DELETE Categories
WHERE c = 0