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.

A	B	C	
a1	b1	c1	provenance1
a2	b1	c1	provenance2
a2	b2	c2	provenance3
a2	b2	c3	provenance4

Provenance Annotations

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

A	B	
a1	b1	X1
a2	b1	X2
a2	b2	X3

- Next, compute the provenance expression inductively, based on the query plan

Join Operator

$\bowtie$

A	B	
a1	b1	X1
a2	b1	X2
a2	b2	X3

B	C	
b1	c1	Y1
b2	c2	Y2
b2	c3	Y3

=

A	B	C	
a1	b1	c1	$X1 \cdot Y1$
a2	b1	c1	$X2 \cdot Y1$
a2	b2	c2	$X3 \cdot Y2$
a2	b2	c3	$X3 \cdot Y3$

Projection Operator

Π

A	B	
a1	b1	X1
a1	b2	X2
a2	b1	X3
a2	b2	X4
a2	b3	X5

=

A	
a1	X1+X2
a2	X3+X4+X5

Union Operator

$\cup$

A	B	
a1	b1	X1
a2	b2	X2

A	B	
a2	b2	Y1
a3	b3	Y2

=

A	B	
a1	b1	X1
a2	b2	X2+Y1
a3	b3	X3

Selection Operator

$\sigma_{A=a1}$

A	B	
a1	b1	X1
a1	b2	X2
a2	b1	X3
a2	b2	X4
a2	b3	X5

=

A	B	
a1	b1	X1
a1	b2	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

$\sigma_{A=a1}$

A	B	
a1	b1	X1
a1	b2	X2
a2	b1	X3
a2	b2	X4
a2	b3	X5

=

A	B	
a1	b1	$X1 \cdot 1$
a1	b2	$X2 \cdot 1$
a2	b1	$X3 \cdot 0$
a2	b2	$X4 \cdot 0$
a2	b3	$X5 \cdot 0$

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

$$\Pi_{AC}(R) \bowtie \Pi_{BC}(R) =$$

R =

A	B	C	
a	b	c	X
d	b	e	Y
f	g	e	Z

A	B	C	
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	$Z \cdot Z$

Discuss in class what these annotations mean

Simple Example 2

$\sigma_{C=e}(R) =$

R =

A	B	C	
a	b	c	X
d	b	e	Y
f	g	e	Z

A	B	C	
a	b	c	$0 = X \cdot 0$
d	b	e	$Y = Y \cdot 1$
f	g	e	$Z = Z \cdot 1$

Discuss in class what these annotations mean

Complex Example

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

R =

A	B	C	
a	b	c	X
d	b	e	Y
f	g	e	Z

A	C	
a	c	$(X \cdot X + X \cdot X) \cdot 0 = 0 \cdot 2 \cdot X^2 = 0$
a	e	$X \cdot Y \cdot 1 = X \cdot Y$
d	c	$Y \cdot X \cdot 0 = 0$
d	e	$(Y \cdot Y + Y \cdot Z + Y \cdot Y) \cdot 1 = 2 \cdot Y^2 + Y \cdot Z$
f	e	$(Z \cdot Z + Z \cdot Y + Z \cdot Z) \cdot 1 = 2 \cdot Z^2 + Y \cdot Z$

Discuss in class what these annotations mean

K-Relations

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

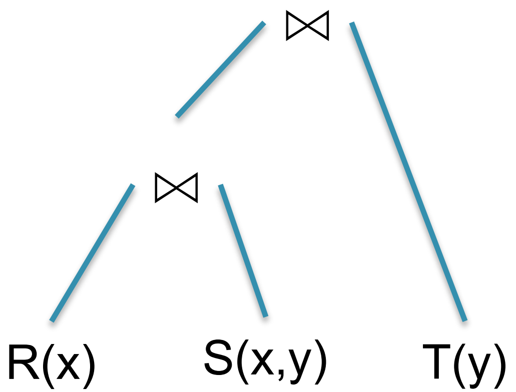
The problem:

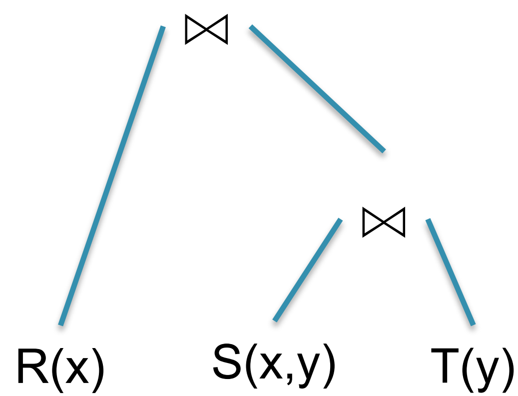
- We have defined provenance for a query plan P
- Given a query Q, we want the provenance to be independent of the plan
- Needed: if P1=P2,
  then provenance(P1) = Provenance(P2)

Example

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

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





R=

x	
a	X

S=

x	y	
a	b	Y

T=

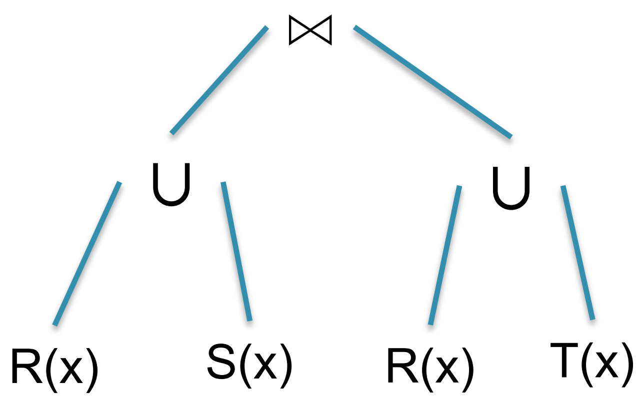
y	
b	Z

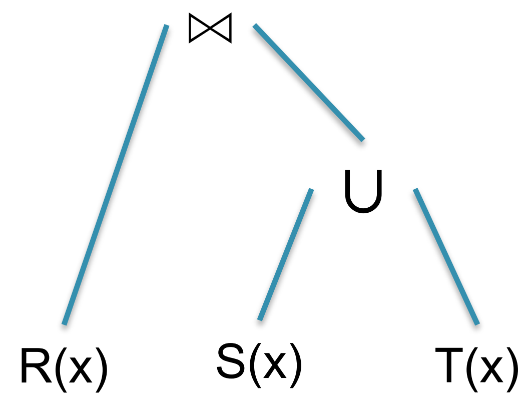
Example

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

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

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





R=

x	
a	X

S=

x	
a	Y

T=

x	
a	Z

Identities on Provenance Expressions

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: $x \cdot 0 = 0 \cdot x = 0$

Identities on Provenance Expressions

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.

Example

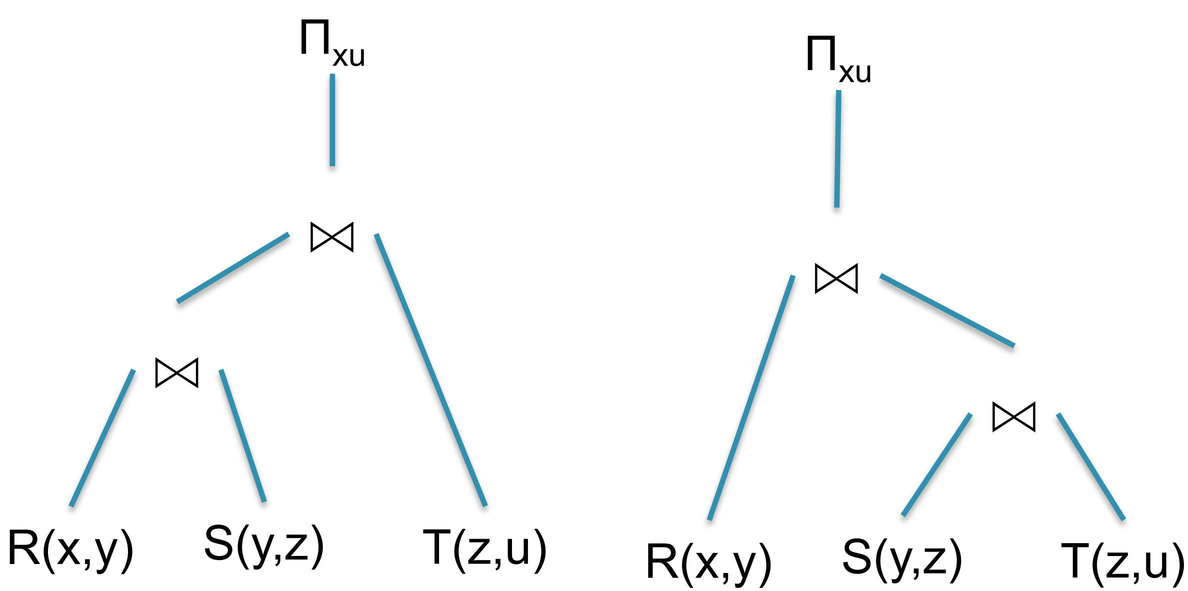
q(x,u) :- R(x,y), S(y,z), T(z,u)

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

x	y	
a	b1	X1
a	b2	X2

y	z	
b1	c1	Y1
b1	c2	Y2
b2	c2	Y3

z	u	
c1	d	Z1
c2	d	Z2



Applications

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

R =

A	B	C	
a	b	c	X
d	b	e	Y
f	g	e	Z

A	C	
a	c	0
a	e	$X \cdot Y$
d	e	$2 \cdot Y^2 + Y \cdot Z$
f	e	$2 \cdot Z^2 + Y \cdot Z$

Q: Suppose we delete the tuple (d,b,e) from R. Which tuple(s) disappear from the result?

Applications

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

R =

A	B	C	
a	b	c	X
d	b	e	Y
f	g	e	Z

A	C	
a	c	0
a	e	$X \cdot Y$
d	e	$2 \cdot Y^2 + Y \cdot Z$
f	e	$2 \cdot Z^2 + Y \cdot Z$

=

A	C	
a	c	0
a	e	0
d	e	0
f	e	$2 \cdot Z^2$

Q: Suppose we delete the tuple (d,b,e) from R. Which tuple(s) disappear from the result?

A: 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 =

A	B	C	
a	b	c	X
d	b	e	Y
f	g	e	Z

A	C	
a	c	0
a	e	$X \cdot Y$
d	e	$2 \cdot Y^2 + Y \cdot Z$
f	e	$2 \cdot Z^2 + Y \cdot Z$

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

Applications

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

R =

A	B	C	
a	b	c	X
d	b	e	Y
f	g	e	Z

A	C	
a	c	0
a	e	$X \cdot Y$
d	e	$2 \cdot Y^2 + Y \cdot Z$
f	e	$2 \cdot Z^2 + Y \cdot Z$

A	C	
a	c	0
a	e	9
d	e	27
f	e	27

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

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

R =

A	B	C	
a	b	c	X
d	b	e	Y
f	g	e	Z

A	C	
a	c	0
a	e	$X \cdot Y$
d	e	$2 \cdot Y^2 + Y \cdot Z$
f	e	$2 \cdot Z^2 + Y \cdot Z$

➔

A	C	
a	c	-
a	e	X, Y
d	e	Y, Z
f	e	Y, Z

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)

R =

A	B	C	
a	b	c	X
d	b	e	Y
f	g	e	Z

Q =

A	C	
d	e	{Y,Z}

Lineage [CuiWidomWiener'00]

Set of contributing tuples

Semiring: $(\mathrm{Lin}(X), +, \cup, \perp, \varnothing)$

Source: Green ICDT 2009

Examples, define the semi-ring (in class)

Semirings for various models of provenance (2)

R =

A	B	C	
a	b	c	X
d	b	e	Y
f	g	e	Z

Q =

A	C	
d	e	{{Y},{Y,Z}}

Why-provenance [Buneman'08]

Set of sets of witnesses

Semiring: $(\mathrm{Why}(X), \cup, \Cup, \emptyset, \{\emptyset\})$

Examples, define the semi-ring (in class)

Source: Tannen, EDBT 2010

Semirings for various models of provenance (3)

R =

A	B	C	
a	b	c	X
d	b	e	Y
f	g	e	Z

Q =

A	C	
d	e	{Y}

Why-provenance [Buneman'08]

Set of sets of *minimal* witnesses

Semiring: (PosBool(X), $\wedge$, $\vee$, $\top$, $\perp$)

Source: Tannen, EDBT 2010

Examples, define the semi-ring (in class)

Semirings for various models of provenance (4)

R =

A	B	C	
a	b	c	X
d	b	e	Y
f	g	e	Z

Q =

A	C	
d	e	[{Y}, {Y}, {Y,Z}]

Notation:

{ } set

[ ] bag

Trio lineage [Das Sarma'08]

Bags of sets of witnesses

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

Source: Tannen, EDBT 2010

Examples, define the semi-ring (in class)

Semirings for various models of provenance (5)

R =

A	B	C	
a	b	c	X
d	b	e	Y
f	g	e	Z

Q =

A	C	
d	e	{[Y,Y], [Y,Z]}

Notation:

{ } set

[ ] bag

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 =

A	B	C	
a	b	c	X
d	b	e	Y
f	g	e	Z

Q =

A	C	
d	e	[[Y,Y], [Y,Y], [Y,Z]]

Notation:

{ } set

[ ] bag

Provenance polynomials [Green'07]

Bags of bags of contributing tuples

Semiring: $(N[X], +, \cdot, 0, 1)$

Source: Tannen, EDBT 2010

Application

<u>Discretionary Access Control</u> [LaPadula]

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

R =

A	B	C
a	b	c
d	b	e
f	g	e

X=C

Y=P

Z=T

A	C	
a	c	$2 \cdot X^2$ = ?
a	e	$X \cdot Y$ = ?
d	e	$2 \cdot Y^2 + Y \cdot Z$ = ?
f	e	$2 \cdot Z^2 + Y \cdot Z$ = ?

Application

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??

Application

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??

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

Application

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??

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

Application

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??

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?

Application

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??

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

What are the labels of these records?

R =

A	B	C
a	b	c
d	b	e
f	g	e

X=C
Y=P
Z=T

A	C	
a	c	$2 \cdot X^2$
a	e	$X \cdot Y$
d	e	$2 \cdot Y^2 + Y \cdot Z$
f	e	$2 \cdot Z^2 + Y \cdot Z$

(A, min, max, 0, P), where A = P < C < S < T < 0

Application

<u>Discretionary Access Control</u> [LaPadula]

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

R =

A	B	C	
a	b	c	X=C
d	b	e	Y=P
f	g	e	Z=T

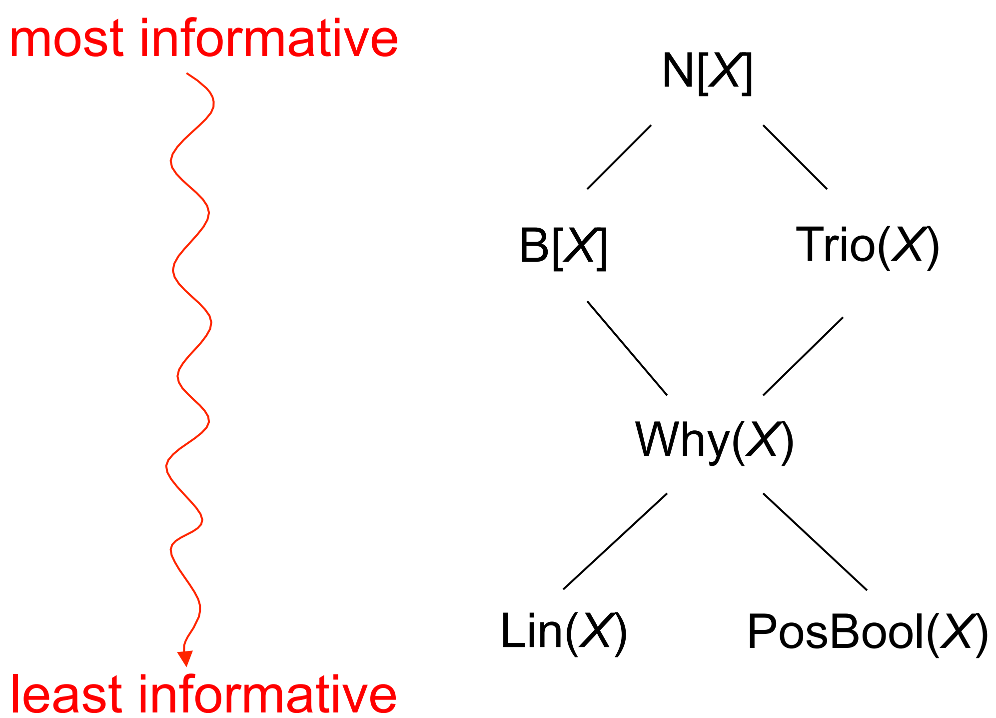
A	C	
a	c	$2 \cdot X^2 = C$
a	e	$X \cdot Y = C$
d	e	$2 \cdot Y^2 + Y \cdot Z = C$
f	e	$2 \cdot Z^2 + Y \cdot Z = T$

(A, min, max, 0, P), where A = P < C < S < T < 0

Semirings

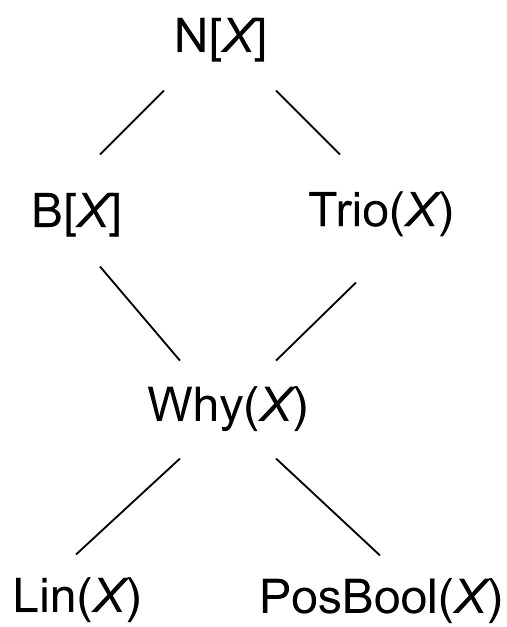
$(B, \wedge, \vee, \top, \bot)$	Set semantics
$(\mathbb{N}, +, \cdot, 0, 1)$	Bag semantics
$(P(\Omega), \cup, \cap, \varnothing, \Omega)$	Probabilistic events [FuhrRölleke 97]
$(\mathrm{BoolExp}(X), \wedge, \vee, \top, \bot)$	Conditional tables (c-tables) [ImielinskiLipski 84]
$(R_+^\infty, \min, +, 1, 0)$	Tropical semiring (cost/distrust score/confidence need)
$(A, \min, \max, 0, P)$ where $A = P < C < S < T < 0$	Access control levels [PODS8]

A provenance hierarchy



A provenance hierarchy

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

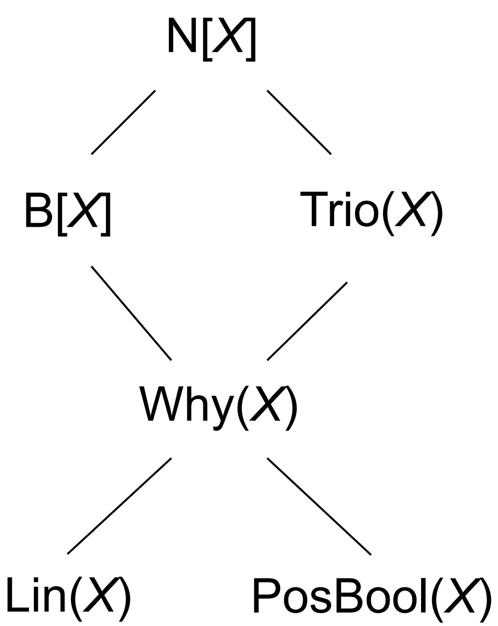


A path downward from K_1 to K_2 indicates that there exists an **onto (surjective) semiring homomorphism** $h : K_1 \rightarrow K_2$

A provenance hierarchy

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

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



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

N[X]

drop coefficients

$x^2y + xy + y^2 + z$ B[X]

Trio(X) $3xy + 5y + z$

drop exponents

Why(X)

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

N[X]

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

B[X]

Trio(X)

drop exponents
$3xy + 5y + z$

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

Why(X)

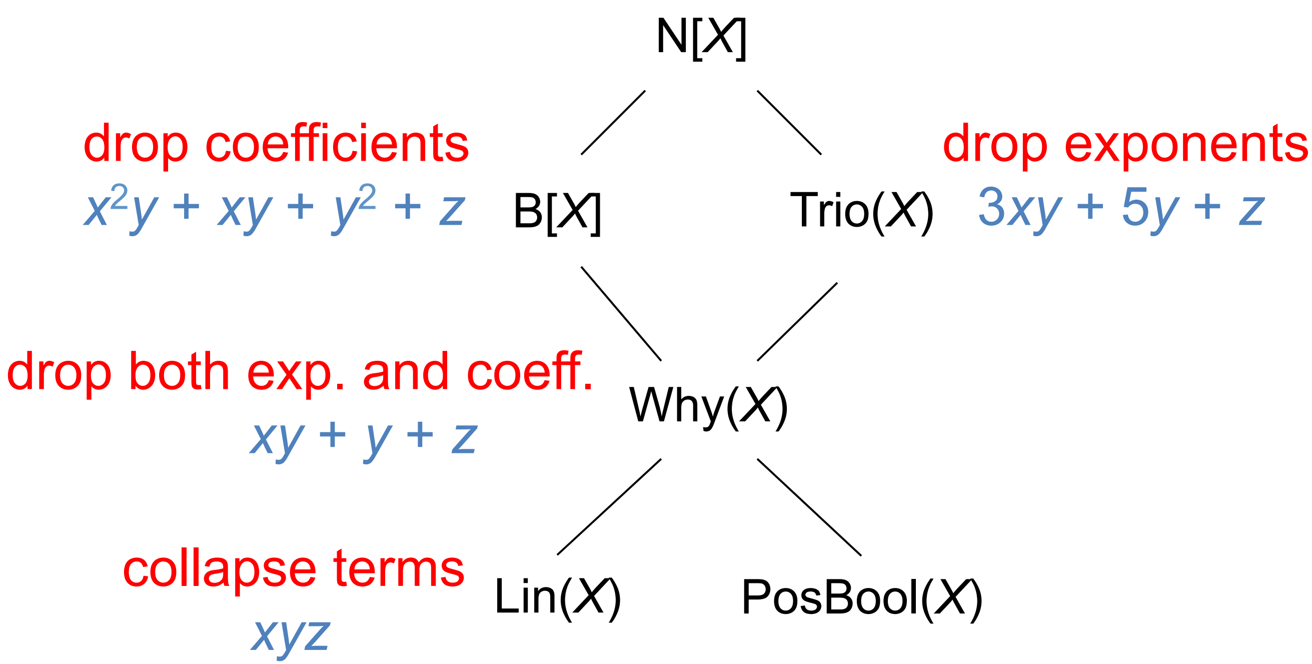
Lin(X)

PosBool(X)

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

A provenance hierarchy

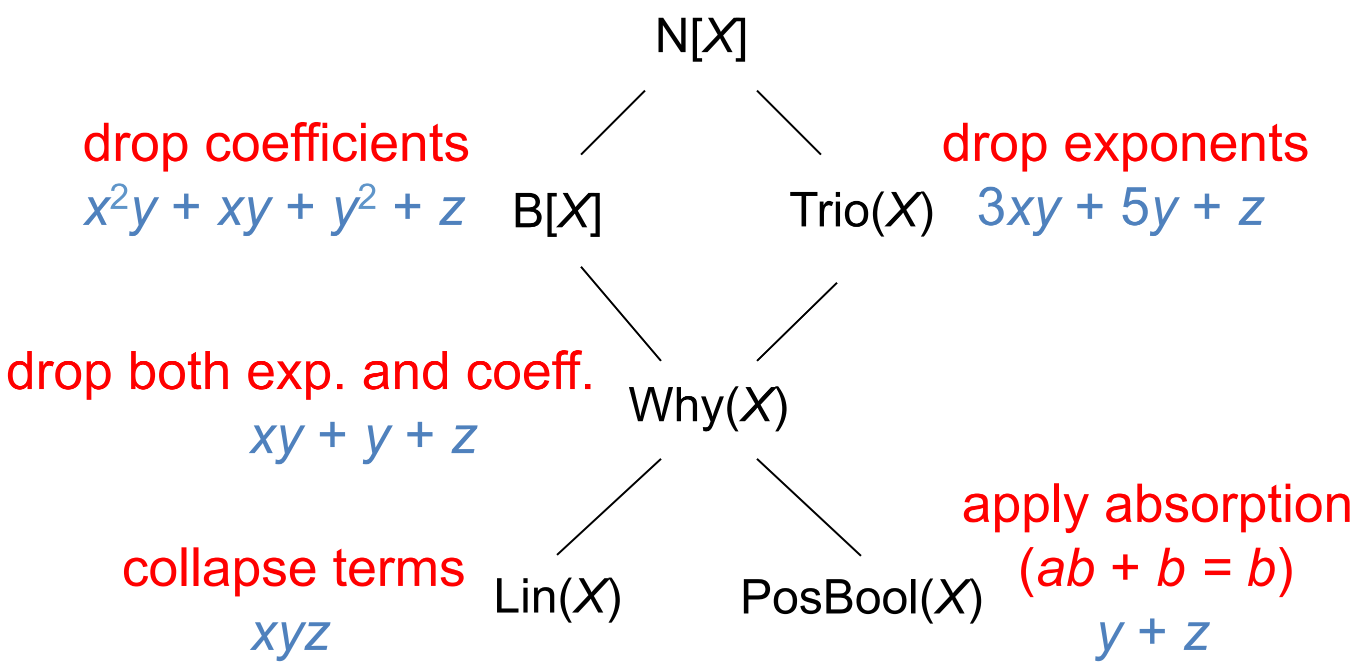
Example: $2x^2y + xy + 5y^2 + z$



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

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

$\mathbb{N}[X]$

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

$\mathbb{B}[X]$

$\mathrm{Trio}(X)$

drop exponents
$3xy + 5y + z$

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

$\mathrm{Why}(X)$

collapse terms
xyz

$\mathrm{Lin}(X)$

$\mathrm{PosBool}(X)$

apply absorption
$(ab + b = b)$
$y + z$

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

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

StorePrice(store, price)

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)

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

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

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

StorePrice(store, price)

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
```

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

StorePrice(store, price)

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
```

Modified query:

```
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
```

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

StorePrice(store, price)

Query Modification

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

```
SELECT DISTINCT u.customer, u.store
FROM Purchase u, Purchase x, Product y
WHERE u.store = x.store
    AND y.price > 1000
    AND x.product = y.pname
```

Notice that Purchase occurs twice. Why?

Modified query:

```
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
```

Modified and unnested query:

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

StorePrice(store, price)

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%'
```

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

StorePrice(store, price)

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%'
```

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%'
```

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

StorePrice(store, price)

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%’
```

We can further optimize! How?

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%’
```

Modified and unnested query:

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

StorePrice(store, price)

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%'
```

Assuming Product.pname is a key
and Purchase.product is a foreign key

Modified and unnested query:

Final Query

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

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

Resumes

SSN	Name	Address	Resume	Picture
234234	Mary	Huston	Clob1…	Blob1…
345345	Sue	Seattle	Clob2…	Blob2…
345343	Joan	Seattle	Clob3…	Blob3…
234234	Ann	Portland	Clob4…	Blob4…

T1

SSN	Name	Address
234234	Mary	Huston
345345	Sue	Seattle
. . .		

T2

SSN	Resume
234234	Clob1…
345345	Clob2…

T3

SSN	Picture
234234	Blob1…
345345	Blob2…

Vertical Partitioning

```
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 → enables reconstructing the table
  - Compress → great compression, minimize I/O
  - Updates = VERY, VERY expensive
- Companies: *C-Store* and *Vertica*

Horizontal Partitioning

Customers

SSN	Name	City
234234	Mary	Huston
345345	Sue	Seattle
345343	Joan	Seattle
234234	Ann	Portland
--	Frank	Calgary
--	Jean	Montreal

CustomersInHuston

SSN	Name	City
234234	Mary	Huston

CustomersInSeattle

SSN	Name	City
345345	Sue	Seattle
345343	Joan	Seattle

. . . . .

Horizontal Partitioning

```
CREATE VIEW Customers AS
    CustomersInHuston
        UNION ALL
    CustomersInSeattle
        UNION ALL
     . . .
```

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:

```
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?)

Purchase(customer, product, store)
Product(<u>pname</u>, price)

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

Views and Security

Fred is not allowed to see Balance

John is not allowed to see >0 balances

Customers:

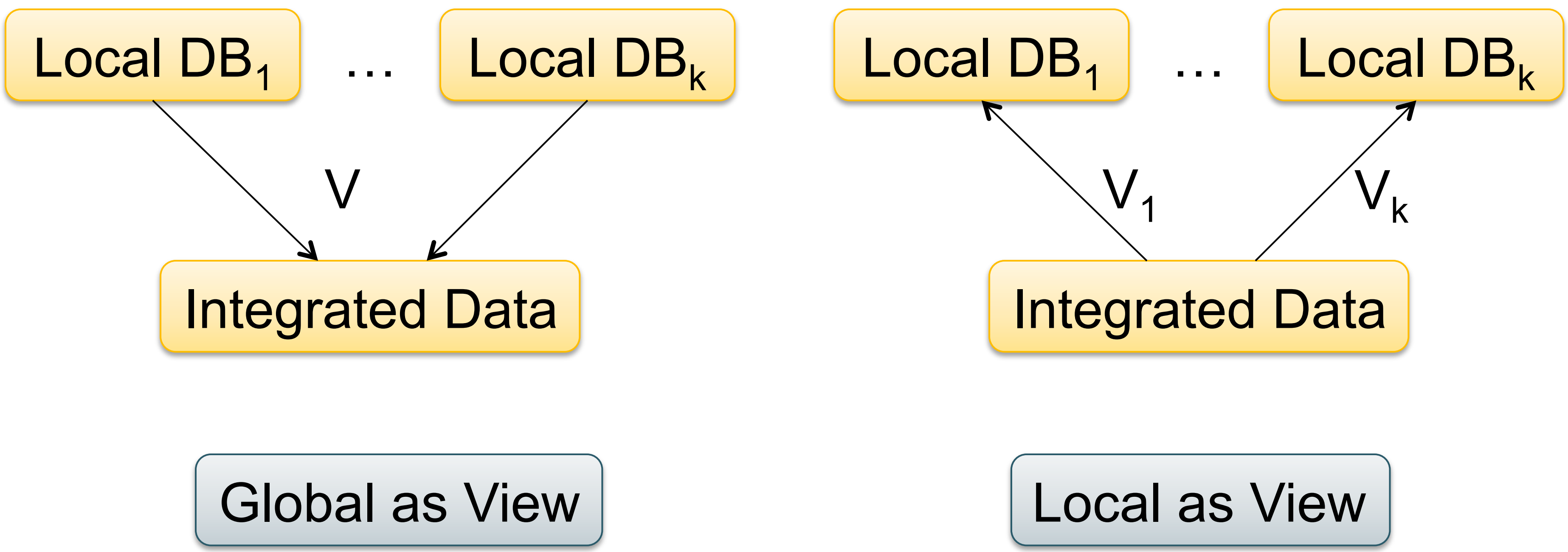
Name	Address	Balance
Mary	Huston	450.99
Sue	Seattle	-240
Joan	Seattle	333.25
Ann	Portland	-520

Name	Address	Balance
Mary	Huston	450.99
Sue	Seattle	-240
Joan	Seattle	333.25
Ann	Portland	-520

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

```
CREATE VIEW BadCreditCustomers
    SELECT *
    FROM Customers
    WHERE Balance < 0
```

Data Integration Terminology



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  CustomersInHuston  AS
    (SELECT * FROM Customers
     WHERE city = ‘Huston’)
….
```

```
SELECT name FROM Customers
WHERE   city = ‘Seattle’
```

→

```
SELECT name
FROM CustomersInSeattle
```

Indexes are Materialized Views

Product(pid, name, weight, price, …)

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

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

"Covering indexes":
query uses
only the indexes

Indexes as LAV:

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

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

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(<u>pname</u>, city)

Have this materialized view:

```
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
```

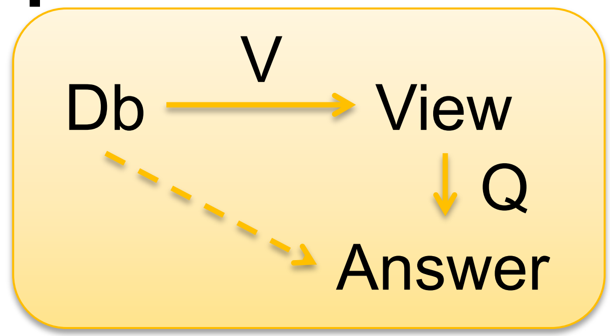
```
SELECT  y.buyer, y.seller
FROM    Person x, Purchase y
WHERE   x.city = 'Seattle'   AND
        x..pname = y.buyer AND
        y.product='gizmo'
```

"Maximally contained rewriting"

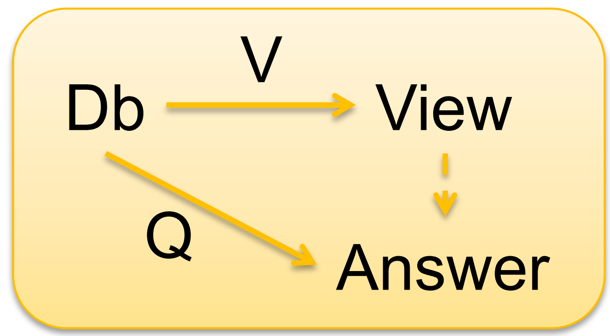
```
SELECT  buyer, seller
FROM    DifferentView
WHERE   product= 'gizmo'
```

Technical Aspects

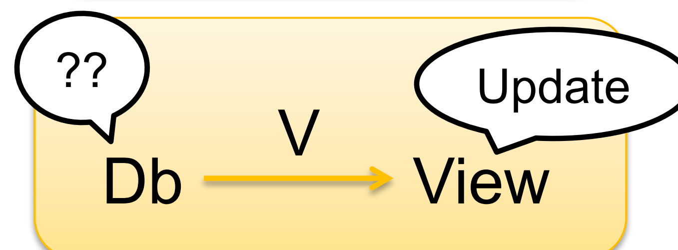
- View inlining, or query modification



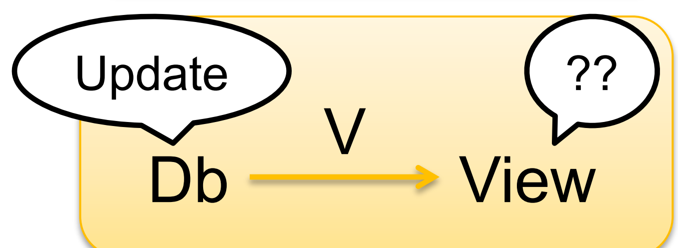
- Query answering using views



- 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

Updating Views

Purchase(customer, product, store)
Product(<u>pname</u>, price)

```
INSERT
INTO Expensive-Product
VALUES('Gizmo')
```

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

Updateable view

Updatable Views

- Have a virtual view V(A1, A2, …) over tables R1, R2, …
- User wants to *update* a tuple in V
  - Insert/modify/delete
- Can we translate this into updates to R1, R2, … ?
- If yes: V = “an updateable view”
- If not: V = “a non-updateable view”

Updating Views

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

```
INSERT
INTO Expensive-Product
VALUES('Gizmo')
```

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

```
INSERT
INTO Product
VALUES('Gizmo', NULL)
```

Updateable view

Updating Views

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

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

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

Updateable view

Updating Views

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

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

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

Updateable view

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

Note this

Updating Views

Purchase(customer, product, store)
Product(<u>pname</u>, price)

INSERT INTO CustomerPrice
VALUES('Joe', 200)

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

? ? ? ? ?

Non-updateable view

Most views are non-updateable

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(<u>pid</u>, name, category, price)

```
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
```