

Database – Slide 5

“Core” Relational Algebra

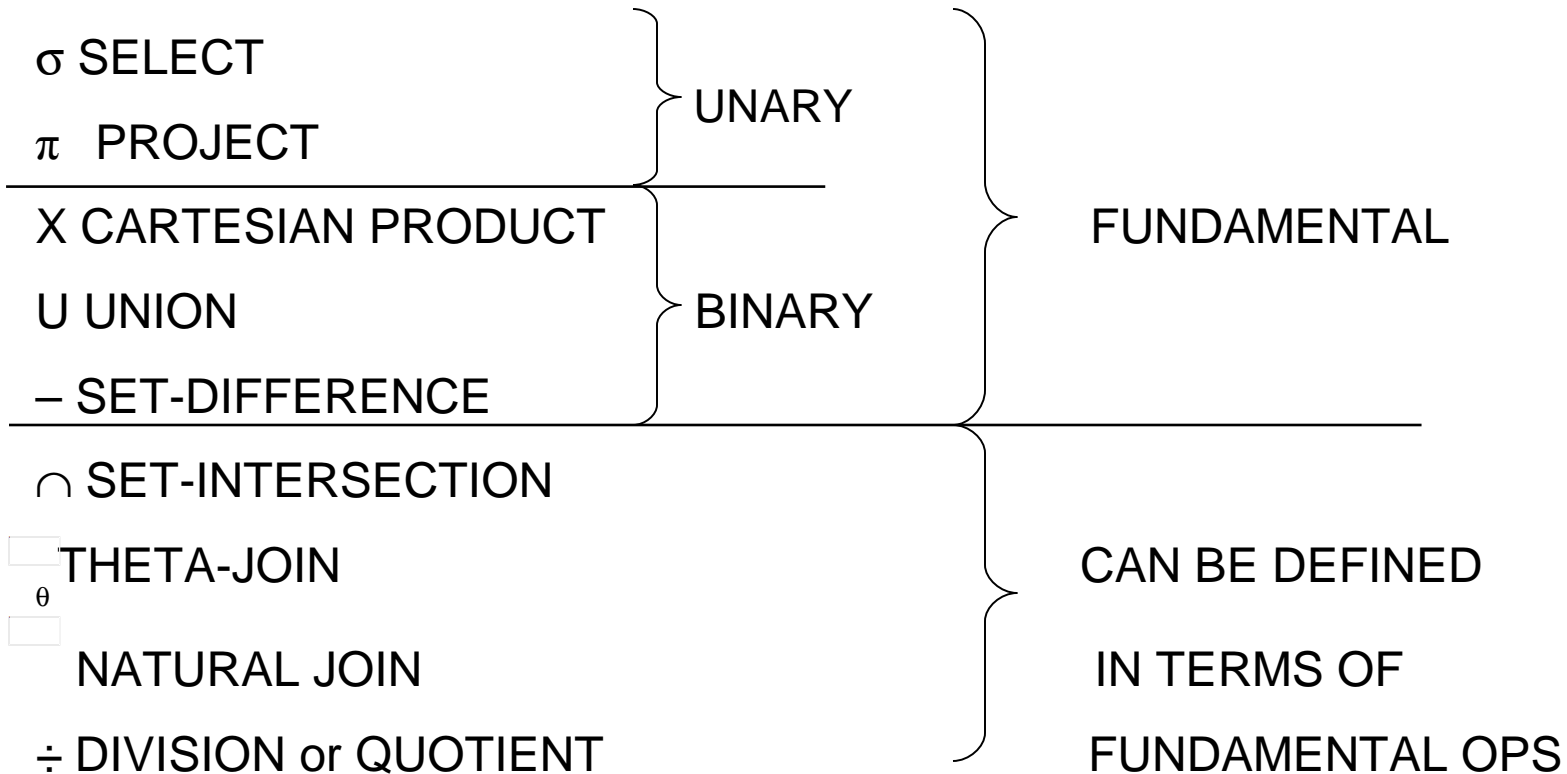
A small set of operators that allow us to manipulate relations in limited but useful ways. The operators are:

1. Union, intersection, and difference: the usual set operators.
 - ◆ But the relation schemas must be the same.
2. *Selection*: Picking certain rows from a relation.
3. *Projection*: Picking certain columns.
4. *Products and joins*: Composing relations in useful ways.
5. *Renaming* of relations and their attributes.

Relational Algebra

- limited expressive power (subset of possible queries)
- good optimizer possible
- rich enough language to express enough useful things

Finiteness



Extra Example Relations

DEPOSIT(branchName, acctNo, custName, balance)

CUSTOMER(custName, street, custCity)

BORROW(branchName, loan-no, custName, amount)

BRANCH(branchName, assets, branchCity)

CLIENT(custName, emplName)

Borrow	BN	L#	CN	AMT
T1	Midtown	123	Fred	600
T2	Midtown	234	Sally	1200
T3	Midtown	235	Sally	1500
T4	Downtown	612	Tom	2000

Selection

$$R_1 = \sigma_C(R_2)$$

where C is a condition involving the attributes of relation R_2 .

Example

Relation Sells:

$$\text{JoeMenu} = \sigma_{\text{bar}=\text{Joe's}}(\text{Sells})$$

SELECT (σ)

$\text{arity}(\sigma(R)) = \text{arity}(R)$

$0 \leq \text{card}(\sigma(R)) \leq \text{card}(R)$

$\sigma_c(R)$ $\sigma_c(R) \subseteq (R)$

c is selection condition: terms of form: attr op value attr op attr

op is one of < = > \leq \neq \geq

example of term: branch-name = 'Midtown'

terms are connected by $\wedge \vee \neg$

$\sigma_{\text{branchName} = \text{'Midtown'} \wedge \text{amount} > 1000}$ **(Borrow)**

$\sigma_{\text{custName} = \text{empName}}$ **(client)**

Projection

$$R_1 = \pi_L(R_2)$$

where L is a list of attributes from the schema of R_2 .

Example

$$\pi_{\text{beer, price}}(\text{Sells})$$

- Notice elimination of duplicate tuples.

Projection (π)

$$0 \leq \text{card}(\pi_A(R)) \leq \text{card}(R)$$

$$\text{arity}(\pi_A(R)) = m \leq \text{arity}(R) = k$$

$$\pi_{i_1, \dots, i_m}(R) \quad 1 \leq i_j \leq k \text{ distinct}$$

produces set of m-tuples $\langle a_1, \dots, a_m \rangle$

such that \exists k-tuple $\langle b_1, \dots, b_k \rangle$ in R where $a_j = b_{i_j}$ for $j = 1, \dots, m$

$$\pi_{\text{branchName, custName}}(\text{Borrow})$$

Midtown Fred

Midtown Sally

Winter 2003
Downtown Tom

Product

$$R = R_1 \times R_2$$

pairs each tuple t_1 of R_1 with each tuple t_2 of R_2 and puts in R a tuple $t_1 t_2$.

Cartesian Product (\times)

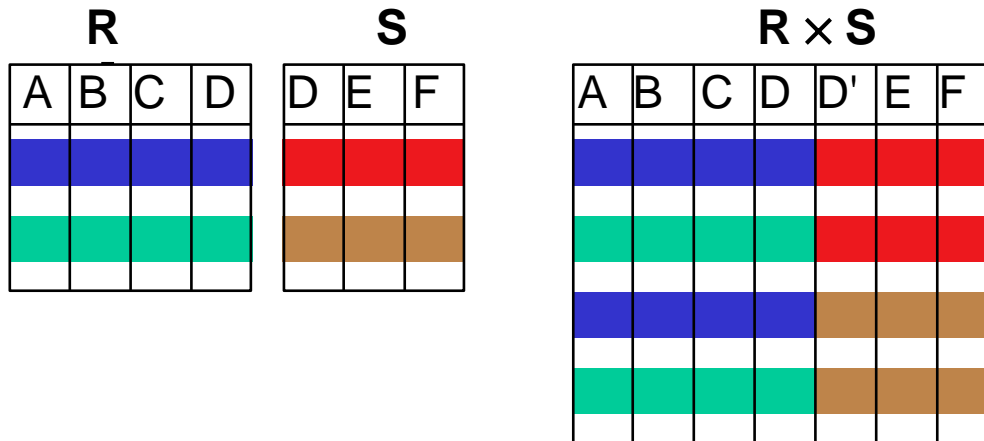
$$\text{arity}(\mathbf{R}) = k_1 \quad \text{arity}(\mathbf{R} \times \mathbf{S}) = k_1 + k_2$$

$$\text{arity}(\mathbf{S}) = k_2 \quad \text{card}(\mathbf{R} \times \mathbf{S}) = \text{card}(\mathbf{R}) \times \text{card}(\mathbf{S})$$

$\mathbf{R} \times \mathbf{S}$ is the set all possible $(k_1 + k_2)$ -tuples

whose first k_1 attributes are a tuple in \mathbf{R}

last k_2 attributes are a tuple in \mathbf{S}



Theta-Join

$$R = R_1 \underset{C}{\boxtimes} R_2$$

is equivalent to $R = \sigma_C(R_1 \times R_2)$.

Theta-Join $R \bowtie_{i \theta j} S$

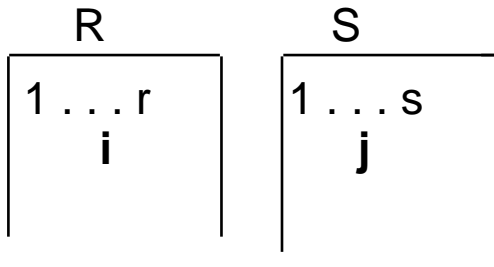
$$\text{arity}(R) = r$$

$$\text{arity}(S) = s$$

$$\text{arity}(R \bowtie_{\theta} S) = r + s$$

$$\sigma_{i \theta j} (R \times S)$$

$$0 \leq \text{card}(R \bowtie_{\theta} S) \leq \text{card}(R) \times \text{card}(S)$$



θ can be $< > = \neq \leq \geq$

If equal (=), then it is an
EQUIJOIN

$$R \bowtie_c S = \sigma_c (R \times S)$$

$$R(A B C) \bowtie_{R.A < S.D} S(C D E)$$

result has schema $T(A B C C' D E)$

R(ABC)	S(CDE)	T(ABCC'DE)
1 3 5	2 1 1	1 3 5 1 2 2
2 4 6	1 2 2	1 3 5 3 3 4
3 5 7	3 3 4	1 3 5 4 4 3
4 6 8	4 4 3	2 4 6 3 3 4
		2 4 6 4 4 3
		3 5 7 4 4 3

Natural Join

$$R = R_1 \bowtie R_2$$

calls for the theta-join of R_1 and R_2 with the condition that all attributes of the same name be equated. Then, one column for each pair of equated attributes is projected out.

Example

Suppose the attribute name in relation `Bars` was changed to `bar`, to match the `bar` name in `Sells`.

$$\text{BarInfo} = \text{Sells} \bowtie \text{Bars}$$

Renaming

$\rho_{S(A_1, \dots, A_n)}(R)$ produces a relation identical to R but named S and with attributes, in order, named A_1, \dots, A_n .

Example

`Bars =`

$\rho_{R(\text{bar}, \text{addr})}(\text{Bars}) =$

- The name of the second relation is R .

Union ($R \cup S$) $\text{arity}(R) = \text{arity}(S) = \text{arity}(R \cup S)$

$\max(\text{card}(R), \text{card}(S)) \leq \text{card}(R \cup S) \leq \text{card}(R) + \text{card}(S)$

set of tuples in R or S or both

$R \subseteq R \cup S$

$S \subseteq R \cup S$

Find customers of Perryridge Branch

$\pi_{\text{Cust-Name}} (\sigma_{\text{Branch-Name} = \text{"Perryridge"}} (\text{BORROW} \cup \text{DEPOSIT}))$

Difference($R - S$)

$$\text{arity}(R) = \text{arity}(S) = \text{arity}(R - S)$$

$$0 \leq \text{card}(R - S) \leq \text{card}(R) \qquad \emptyset \subseteq R - S \subseteq R$$

is the tuples in R not in S

Depositors of Perryridge who aren't borrowers of Perryridge

$\pi_{\text{custName}} (\sigma_{\text{branchName} = \text{'Perryridge'}} (\text{DEPOSIT} - \text{BORROW}))$

Deposit < Perryridge, 36, Pat, 500 >

Borrow < Perryridge, 72, Pat, 10000 >

$\pi_{\text{custName}} (\sigma_{\text{branchName} = \text{'Perryridge'}} (\text{DEPOSIT}))$ —

$\pi_{\text{custName}} (\sigma_{\text{branchName} = \text{'Perryridge'}} (\text{BORROW}))$

Does $\sigma (\pi (D) - \pi (B))$ work?

Combining Operations

Algebra =

2. Basis arguments +
3. Ways of constructing expressions.

For relational algebra:

5. Arguments = variables standing for relations + finite, constant relations.
 6. Expressions constructed by applying one of the operators + parentheses.
- Query = expression of relational algebra.

$\pi_{\text{custName,custCity}}$

$(\sigma_{\text{Client.Banker-Name} = \text{'Johnson'}}$
 $(\text{Client} \times \text{Customer})) =$

$\pi_{\text{cust-Name,custCity}}(\text{Customer})$

- Is this always true? Is this what we wanted?

$\pi_{\text{Client.custName, Customer.custCity}}$

$(\sigma_{\text{Client.bankerName} = \text{'Johnson'}}$
 $\wedge \text{Client.custName} = \text{Customer.custName}$
 $(\text{Client} \times \text{Customer}))$

$\pi_{\text{Client.custName, Customer.custCity}}$

$(\sigma_{\text{Client.custName} = \text{Customer.custName}}$

$(\text{Customer} \times \pi_{\text{custName}}$

$(\sigma_{\text{Client.bankerName} = \text{'Johnson'}}(\text{Client}))))$

SET INTERSECTION

$(R \cap S)$

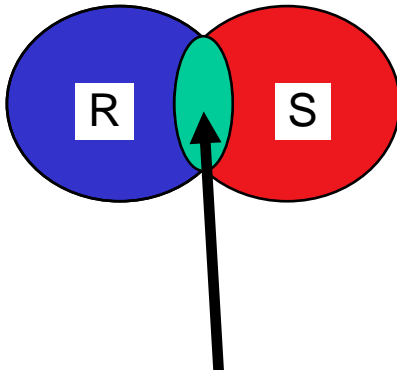
$$\text{arity}(R) = \text{arity}(S) = \text{arity}(R \cap S)$$

$$0 \leq \text{card}(R \cap S) \leq \min(\text{card}(R), \text{card}(S))$$

tuples both in R and in S

$$\emptyset \subseteq R \cap S \subseteq R$$

$$\emptyset \subseteq R \cap S \subseteq S$$



$$R - (R - S) = R \cap S$$

Operator Precedence

The normal way to group operators is:

2. Unary operators σ , π , and ρ have highest precedence.
 3. Next highest are the “multiplicative” operators, \square , \square_c , and \times .
 4. Lowest are the “additive” operators, \cup , \cap , and — .
- But there is no universal agreement, so we always put parentheses *around* the argument of a unary operator, and it is a good idea to group all binary operators with parentheses *enclosing* their arguments.

Example

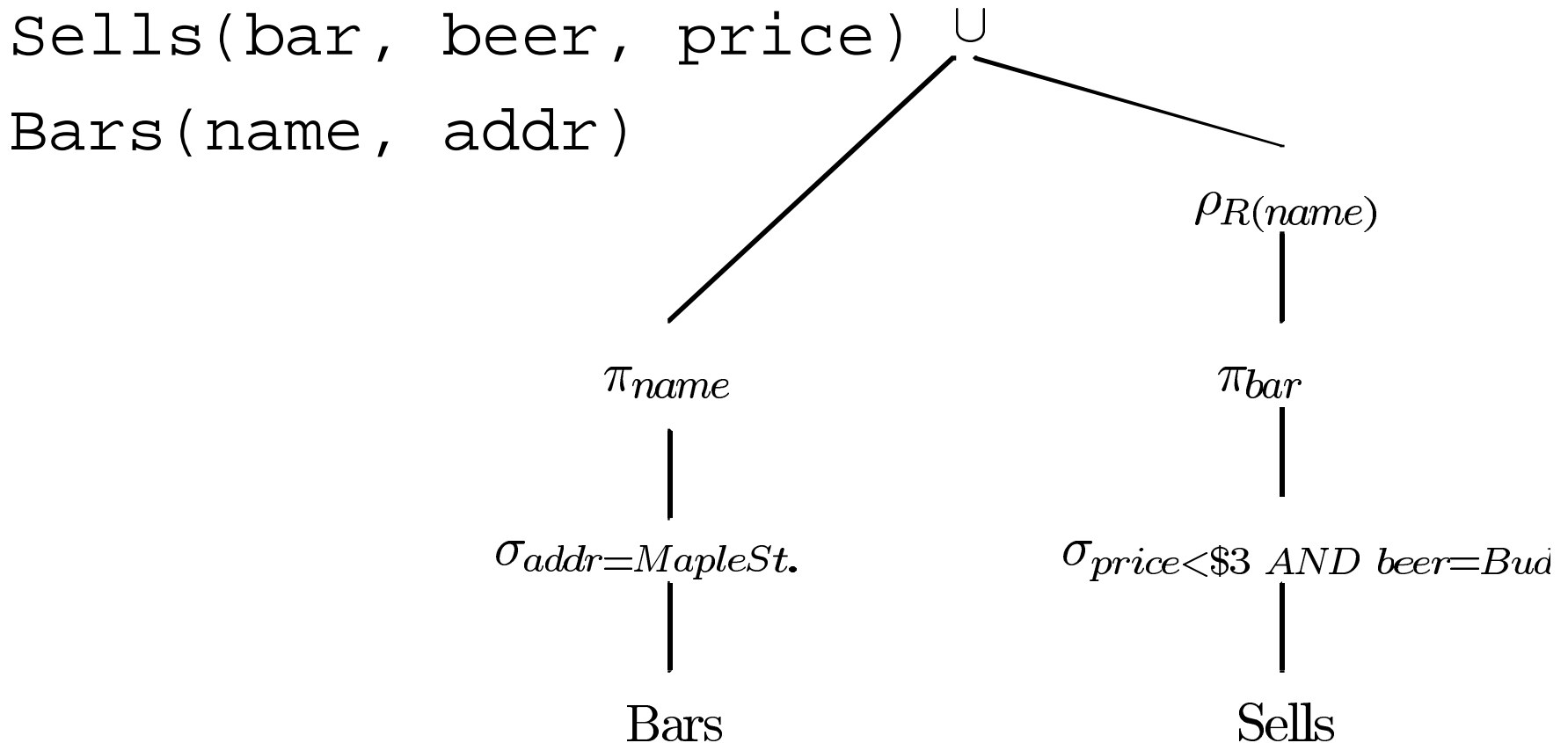
Group $R \cup \sigma S \square T$ as $R \cup (\sigma(S) \square T)$.

Each Expression Needs a Schema

- If \cup , \cap , $-$ applied, schemas are the same, so use this schema.
- Projection: use the attributes listed in the projection.
- Selection: no change in schema.
- Product $R \times S$: use attributes of R and S .
 - ◆ But if they share an attribute A , prefix it with the relation name, as $R.A$, $S.A$.
- Theta-join: same as product.
- Natural join: use attributes from each relation; common attributes are merged anyway.
- Renaming: whatever it says.

Example

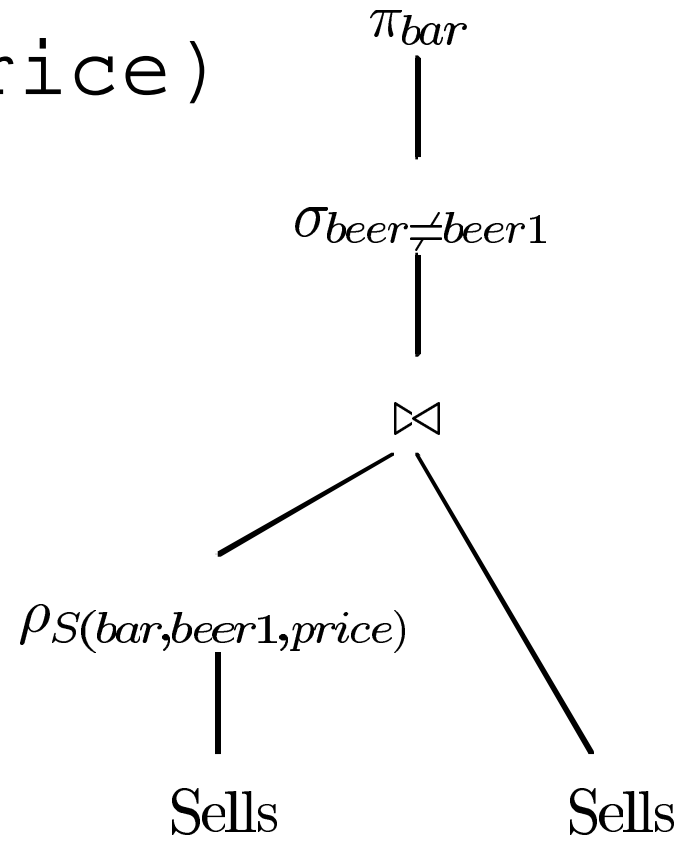
- Find the bars that are either on Maple Street or sell Bud for less than \$3.



Example

Find the bars that sell two different beers at the same price.

`Sells(bar, beer, price)`



Linear Notation for Expressions

- Invent new names for intermediate relations, and assign them values that are algebraic expressions.
- Renaming of attributes implicit in schema of new relation.

Example

Find the bars that are either on Maple Street or sell Bud for less than \$3.

`Sells(bar, beer, price)`

`Bars(name, addr)`

`R1(name) := $\pi_{\text{name}}(\sigma_{\text{addr} = \text{Maple St.}}(\text{Bars}))$`

`R2(name) := $\pi_{\text{bar}}(\sigma_{\text{beer}=\text{Bud AND price}<\$3}(\text{Sells}))$`

`R3(name) := R1 \cup R2`

Why Decomposition “Works”?

What does it mean to “work”? Why can’t we just tear sets of attributes apart as we like?

- Answer: the decomposed relations need to represent the same information as the original.
 - ◆ We must be able to reconstruct the original from the decomposed relations.

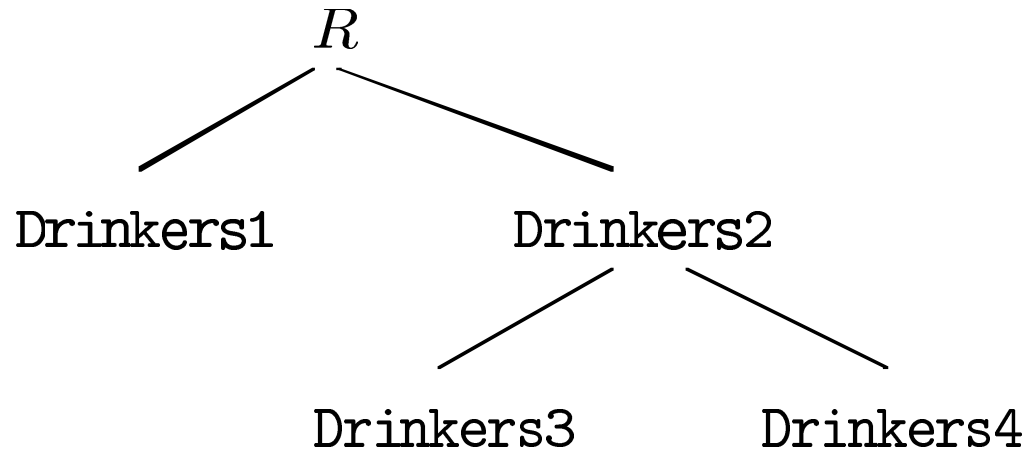
Projection and Join Connect the Original and Decomposed Relations

- Suppose R is decomposed into S and T .
We project R onto S and onto T .

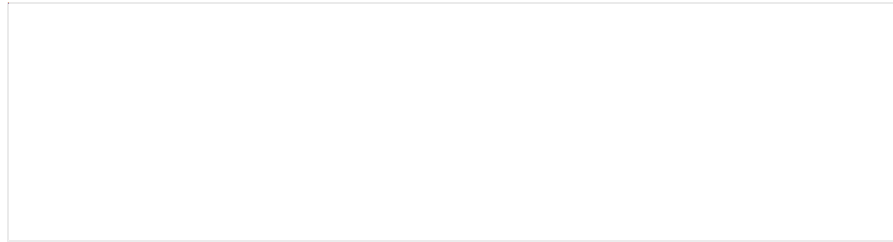
Example

$R =$

- Recall we decomposed this relation as:



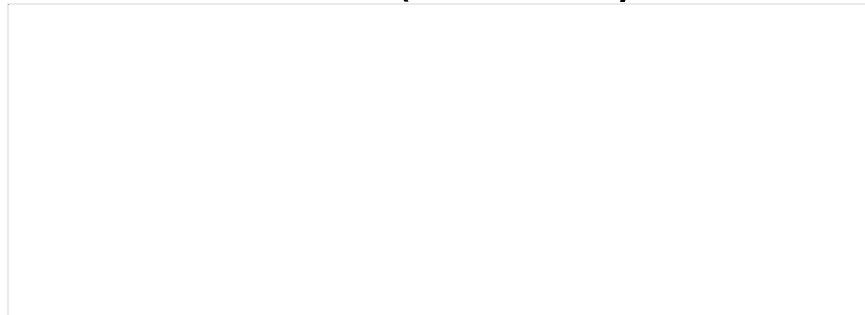
Project onto Drinkers1 (name, addr, favoriteBeer):



Project onto Drinkers3 (beersLiked, manf):



Project onto Drinkers4 (name, beersLiked):



Reconstruction of Original

Can we figure out the original relation from the decomposed relations?

- Sometimes, if we natural join the relations.

Example

`Drinkers3` \bowtie `Drinkers4` =

- Join of above with `Drinkers1` = original R .

Theorem

Suppose we decompose a relation with schema XYZ into XY and XZ and project the relation for XYZ onto XY and XZ .

Then $XY \sqcap XZ$ is *guaranteed* to reconstruct XYZ if and only if $X \twoheadrightarrow Y$ (or equivalently, $X \twoheadrightarrow Z$).

- Usually, the MVD is really a FD, $X \rightarrow Y$ or $X \rightarrow Z$.
- BCNF: When we decompose XYZ into XY and XZ , it is because there is a FD $X \rightarrow Y$ or $X \rightarrow Z$ that violates BCNF.
 - ◆ Thus, we can always reconstruct XYZ from its projections onto XY and XZ .
- 4NF: when we decompose XYZ into XY and XZ , it is because there is an MVD $X \twoheadrightarrow Y$ or $X \twoheadrightarrow Z$ that violates 4NF.
 - ◆ Again, we can reconstruct XYZ from its projections onto XY and XZ .

Bag Semantics

A relation (in SQL, at least) is really a *bag* or *multiset*.

- It may contain the same tuple more than once, although there is no specified order (unlike a list).
- Example: $\{1,2,1,3\}$ is a bag and not a set.
- Select, project, and join work for bags as well as sets.
 - ◆ Just work on a tuple-by-tuple basis, and don't eliminate duplicates.

Bag Union

Sum the times an element appears in the two bags.

- Example: $\{1,2,1\} \cup \{1,2,3,3\} = \{1,1,1,2,2,3,3\}$.

Bag Intersection

Take the minimum of the number of occurrences in each bag.

- Example: $\{1,2,1\} \cap \{1,2,3,3\} = \{1,2\}$.

Bag Difference

Proper-subtract the number of occurrences in the two bags.

- Example: $\{1,2,1\} - \{1,2,3,3\} = \{1\}$.

Laws for Bags Differ From Laws for Sets

- Some familiar laws continue to hold for bags.
 - ◆ Examples: union and intersection are still commutative and associative.
- But other laws that hold for sets do *not* hold for bags.

Example

$R \cap (S \cup T) \equiv (R \cap S) \cup (R \cap T)$ holds for sets.

- Let R , S , and T each be the bag $\{1\}$.
- Left side: $S \cup T = \{1,1\}$; $R \cap (S \cup T) = \{1\}$.
- Right side: $R \cap S = R \cap T = \{1\}$;
 $(R \cap S) \cup (R \cap T) = \{1\} \cup \{1\} = \{1,1\} \neq \{1\}$.

Extended (“Nonclassical”) Relational Algebra

Adds features needed for SQL, bags.

2. Duplicate-elimination operator δ .
3. Extended projection.
4. Sorting operator τ .
5. Grouping-and-aggregation operator γ .
6. Outerjoin operator \boxplus .

Duplicate Elimination

$\delta(R)$ = relation with one copy of each tuple that appears one or more times in R .

Example

$R =$

A	B
1	2
3	4
1	2

$\delta(R) =$

A	B
1	2
3	4

Sorting

$\forall \tau_L(R) =$ list of tuples of R , ordered according to attributes on list L .

- Note that result type is outside the normal types (set or bag) for relational algebra.
 - ◆ Consequence: τ cannot be followed by other relational operators.

Example

$R =$	<u>A</u>	<u>B</u>
	1	3
	3	4
	5	2

$\tau_B(R) = [(5,2), (1,3), (3,4)]$.

Extended Projection

Allow the columns in the projection to be functions of one or more columns in the argument relation.

Example

$$R =$$

<u>A</u>	<u>B</u>
1	2
3	4

$$\pi_{A+B, A, A}(R) =$$

<u>A+B</u>	<u>A1</u>	<u>A2</u>
3	1	1
7	3	3

Aggregation Operators

- These are not relational operators; rather they summarize a column in some way.
- Five standard operators: Sum, Average, Count, Min, and Max.

Grouping Operator

$\gamma_L(R)$, where L is a list of elements that are either

- b) Individual (*grouping*) attributes or
- c) Of the form $\theta(A)$, where θ is an aggregation operator and A the attribute to which it is applied,

is computed by:

5. Group R according to all the grouping attributes on list L .
6. Within each group, compute $\theta(A)$, for each element $\theta(A)$ on list L .
7. Result is the relation whose columns consist of one tuple for each group. The components of that tuple are the values associated with each element of L for that group.

Example

Let $R =$

bar	beer	price
Joe's	Bud	2.00
Joe's	Miller	2.75
Sue's	Bud	2.50
Sue's	Coors	3.00
Mel's	Miller	3.25

Compute $\gamma_{beer, AVG(price)}(R)$.

1. Group by the grouping attribute(\$), beer in this case:

bar	beer	price
Joe's	Bud	2.00
Sue's	Bud	2.50
Joe's	Miller	2.75
Mel's	Miller	3.25
Sue's	Coors	3.00

2. Compute average of price within groups:

beer	AVG(price)
Bud	2.25
Miller	3.00
Coors	3.00

Outerjoin

The normal join can “lose” information, because a tuple that doesn’t join with any from the other relation (*dangles*) has no vestage in the join result.

- The *null value* \perp can be used to “pad” dangling tuples so they appear in the join.
- Gives us the *outerjoin* operator \bowtie° .
- Variations: theta-outerjoin, left- and right-outerjoin (pad only dangling tuples from the left (respectively, right)).

Example

 $R =$

A	B
1	2
3	4

 $S =$

B	C
4	5
6	7

 $R \bowtie S =$

A	B	C
3	4	5
1	2	\perp
\perp	6	7

part of natural join

part of right-outerjoin

part of left-outerjoin