Relational Algebra

BASIC OPERATIONS

What is an “Algebra”

Mathematical system consisting of:

- *Operands* -- **values** from which new values can be constructed.
- *Operators* -- **symbols** denoting procedures that **construct** new values from given values.

What is Relational Algebra?

An **algebra** whose **operands** are **relations**.

Operators are designed to do the most common things that we need to do with relations in a database.

- The result is an algebra that can be used as a *query language* for relations.

Core Relational Algebra

Union, intersection, and difference.

- Usual set operations, but *both operands must have the same relation schema*.

Selection: picking certain rows.

Projection: picking certain columns.

Products and joins: compositions of relations.

Renaming of relations (schema), i.e., the name of attributes and/or the name of relation itself (not affect the tuples of a relation).

Selection

R1 := σ_C(R2)

- *C* is a **condition** (as in "if" statements) that refers to attributes of R2.
- R1 is all those tuples of R2 that **satisfy *C***.

Example: Selection

Relation Sells:

bar	beer	price
Joe's	Bud	2.50
Joe's	Miller	2.75
Sue's	Bud	2.50
Sue's	Miller	3.00

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

bar	beer	price
Joe's	Bud	2.50
Joe's	Miller	2.75

Projection

R1 := π_{L} (R2)

- *L* is a **list of attributes** from the schema of R2.
- R1 is **constructed** by looking at each tuple of R2, extracting the attributes on list *L*, in the order specified, and creating from those components a tuple for R1.
- **Eliminate duplicate** tuples, if any.

Example: Projection

Relation Sells:

bar	beer	price
Joe's	Bud	2.50
Joe's	Miller	2.75
Sue's	Bud	2.50
Sue's	Miller	3.00

Prices := $\Pi_{beer,price}$(Sells):

beer	price
Bud	2.50
Miller	2.75
Miller	3.00

Extended Projection

Using the same π_L operator, we allow the list *L* to contain arbitrary expressions involving attributes:

1. Arithmetic on attributes, e.g., *A*+*B* -> *C*.
2. Duplicate occurrences of the same attribute.

Example: Extended Projection

R =

A	B
1	2
3	4

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

C	A1	A2
3	1	1
7	3	3

Product

R3 := R1 X R2

- Pair **each tuple t1** of R1 with **each tuple t2** of R2.
- **Concatenation t1t2** is a tuple of R3.
- **Schema** of R3 is the **attributes** of R1 and then R2, in order.
- But beware attribute *A* of the same name in R1 and R2: use R1.*A* and R2.*A*.

Example: R3 := R1 X R2

R1(

A,	B
1	2
3	4

)

R2(

B,	C
5	6
7	8
9	10

)

R3(

A,	R1.B,	R2.B,	C
1	2	5	6
1	2	7	8
1	2	9	10
3	4	5	6
3	4	7	8
3	4	9	10

)

Theta-Join

R3 := R1 $\bowtie_C$ R2

- Take the product R1 X R2.
- Then apply σ_C to the result.

As for σ, C can be any boolean-valued condition.

- e.g., A θ B, where θ is =, <, etc.; hence the name "theta-join."

Example: Theta Join

Sells(

bar,	beer,	price
Joe's	Bud	2.50
Joe's	Miller	2.75
Sue's	Bud	2.50
Sue's	Coors	3.00

)

Bars(

name,	addr
Joe's	Maple St.
Sue's	River Rd.

)

BarInfo := Sells $\bowtie_{\text{Sells.bar = Bars.name}}$ Bars

BarInfo(

bar,	beer,	price,	name,	addr
Joe's	Bud	2.50	Joe's	Maple St.
Joe's	Miller	2.75	Joe's	Maple St.
Sue's	Bud	2.50	Sue's	River Rd.
Sue's	Coors	3.00	Sue's	River Rd.

)

Natural Join

A useful join variant (*natural* join) connects two relations by:

- **Equating attributes** of the **same name**, and
- **Projecting** out one copy of each pair of equated attributes.

Denoted R3 := R1 ⋈ R2.

Example: Natural Join

Sells(

bar,	beer,	price
Joe's	Bud	2.50
Joe's	Miller	2.75
Sue's	Bud	2.50
Sue's	Coors	3.00

)

Bars(

bar,	addr
Joe's	Maple St.
Sue's	River Rd.

)

BarInfo := Sells ⋈ Bars

Note: Bars.name has become Bars.bar to make the natural join "work."

BarInfo(

bar,	beer,	price,	addr
Joe's	Bud	2.50	Maple St.
Joe's	Milller	2.75	Maple St.
Sue's	Bud	2.50	River Rd.
Sue's	Coors	3.00	River Rd.

)

Renaming

The ρ operator gives a **new schema** to a relation.

R1 := $\rho_{R1(A1,\ldots,An)}$(R2) makes R1 be a relation with attributes A1,…,A*n* and the same tuples as R2.

Simplified notation: R1(A1,…,A*n*) := R2.

Example: Renaming

Bars(

name,	addr
Joe's	Maple St.
Sue's	River Rd.

)

R(bar, addr) := Bars

R(

bar,	addr
Joe's	Maple St.
Sue's	River Rd.

)

Building Complex Expressions

Combine operators with parentheses and precedence rules.

Three notations, just as in arithmetic:

1. Sequences of assignment statements.
2. Expressions with several operators.
3. Expression trees.

Sequences of Assignments

Create temporary relation names.

Renaming can be applied by giving relations a list of attributes.

Example: R3 := R1 $\bowtie_C$ R2 can be written:

R4 := R1 X R2

R3 := σ_C(R4)

Expressions in a Single Assignment

Example: the theta-join R3 := R1 $\bowtie_C$ R2 can be written: R3 := σ_C (R1 X R2)

Precedence of relational operators:

1. [σ, π, ρ] (highest).
2. [x, ⋈].
3. ∩.
4. [∪, —]

Expression Trees

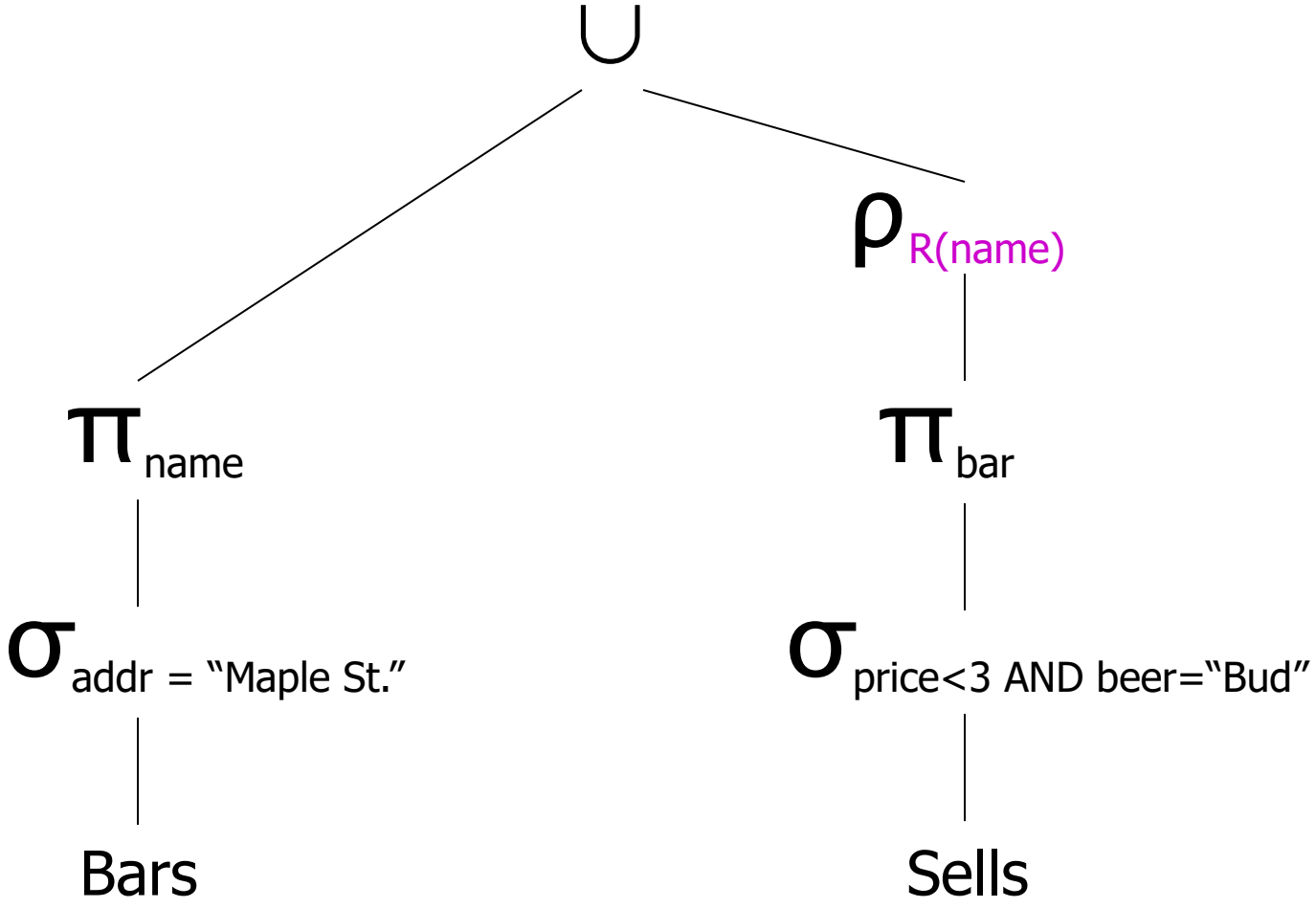
Leaves are **operands** --- standing for relations

Interior nodes are **operators**, applied to their child or children.

Example: Tree for a Query

Using the relations Bars(name, addr) and Sells(bar, beer, price), find the names of all the bars that are either on Maple St. or sell Bud for less than $3.

As a Tree:

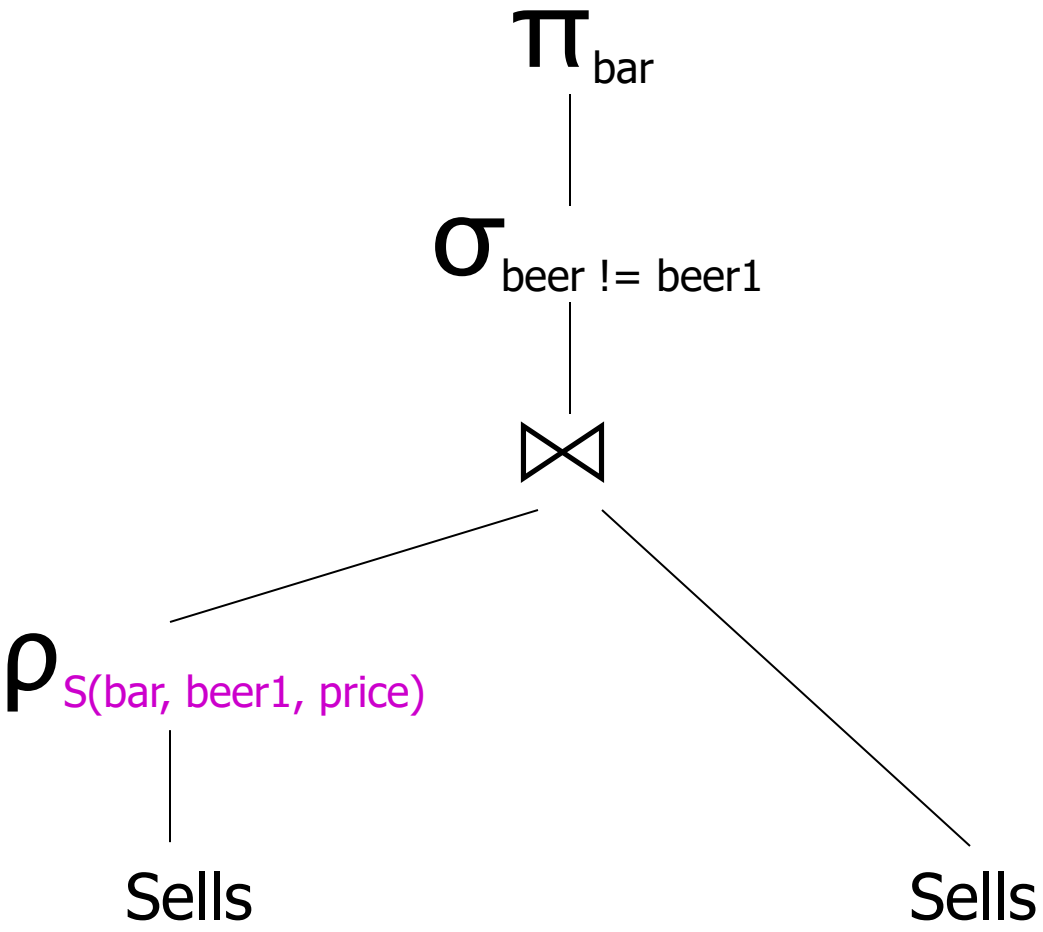

∪
ρ R(name)
π name
π bar
σ addr = "Maple St."
σ price<3 AND beer="Bud"
Bars
Sells

Example: Self-Join

Using Sells(bar, beer, price), find the bars that sell two different beers at the same price.

Strategy: by **renaming**, define a copy of Sells, called S(bar, beer1, price). The natural join of Sells and S consists of quadruples (bar, beer, beer1, price) such that the bar sells both beers at this price.

The Tree



Schemas for Results

Union, intersection, and difference: the **schemas** of the two operands must be the same, so use that schema for the result.

Selection: **schema** of the result is the same as the schema of the operand.

Projection: list of attributes tells us the schema.

Schemas for Results --- (2)

Product: schema is the attributes of both relations.

- Use R.*A*, etc., to distinguish two attributes named *A*.

Theta-join: same as product.

Natural join: union of the attributes of the two relations.

Renaming: the operator tells the schema.

Relational Algebra on Bags

A *bag* (or *multiset*) is like a set, but an element may appear more than once.

Example: {1,2,1,3} is a bag.

Example: {1,2,3} is also a bag that happens to be a set.

Why Bags?

SQL, the most important query language for relational databases, is actually a bag language.

Some operations are more efficient on bags than sets.

Operations on Bags

Projection also applies to each tuple, but as a bag operator, we do not eliminate duplicates.

Products and joins are done on each pair of tuples, so duplicates in bags have no effect on how we operate.

Example: Bag Selection

R(

A,	B
1	2
5	6
1	2

)

$\sigma_{A+B<5}(R) =$

A	B
1	2
1	2

Example: Bag Projection

R(

A,	B
1	2
5	6
1	2

)

$\pi_A(R)$ =

A
1
5
1

Example: Bag Product

R(

A,	B
1	2
5	6
1	2

)

S(

B,	C
3	4
7	8

)

R X S =

A	R.B	S.B	C
1	2	3	4
1	2	7	8
5	6	3	4
5	6	7	8
1	2	3	4
1	2	7	8

Example: Bag Theta-Join

R(

A,	B
1	2
5	6
1	2

)

S(

B,	C
3	4
7	8

)

$R \bowtie_{R.B<S.B} S =$

A	R.B	S.B	C
1	2	3	4
1	2	7	8
5	6	7	8
1	2	3	4
1	2	7	8

Bag Union

An element appears in the union of two bags the sum of the number of times it appears in each bag.

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

Bag Intersection

An element appears in the intersection of two bags the minimum of the number of times it appears in either.

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

Bag Difference

An element appears in the difference $A - B$ of bags as many times as it appears in *A*, minus the number of times it appears in *B*.

- But never less than 0 times.

Example: {1,2,1,1} – {1,2,3} = {1,1}.

Beware: Bag Laws != Set Laws

Some, but *not all* algebraic laws that hold for sets also hold for bags.

Example: the commutative law for union ($R \cup S = S \cup R$) *does* hold for bags.

- Since addition is commutative, adding the number of times x appears in R and S does not depend on the order of R and S.

Actions

Review slides.

Read chapter about Relational Algebra (Chapter 5 in 2nd Edition of course book).

Next class: SQL!