# Relational Algebra

Operators
Expression Trees



## What is an "Algebra"

- Mathematical system consisting of:
  - *Operands* --- variables or 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 or variables that represent 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.



## Roadmap

- There is a core relational algebra that has traditionally been thought of as *the* relational algebra.
- But there are several other operators we shall add to the core in order to model better the language SQL --- the principal language used in relational database systems.



## Core Relational Algebra

- Union, intersection, and difference.
  - Usual set operations, but require both operands have the same relation schema.
- Selection: picking certain rows.
- Projection: picking certain columns.
- Products and joins: compositions of relations.
- Renaming of relations and attributes.



## Selection

- R1 := SELECT$_C$(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

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 := $SELECT_{bar="Joe's"}$(Sells):

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

7

## Projection

- R1 := $PROJ_L$ (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.

8

## Example

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 := $PROJ_{beer,price}$(Sells):

| beer | price |
|---|---|
| Bud | 2.50 |
| Miller | 2.75 |
| Miller | 3.00 |

9

## Product

- R3 := R1 * 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 R2, in order.
  - But beware attribute *A* of the same name in R1 and R2: use R1.*A* and R2.*A*.

10

## Example: R3 := R1 * 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 |

)

11

## Theta-Join

- R3 := R1 $JOIN_C$ R2
  - Take the product R1 * R2.
  - Then apply $SELECT_C$ to the result.
- As for SELECT, *C* can be any boolean-valued condition.
  - Historic versions of this operator allowed only A theta B, where theta was =, <, etc.; hence the name "theta-join."

12

## Example

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 JOIN $_{\text{Sells.bar = Bars.name}}$ 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. |

)

13

## Natural Join

- A frequent type of join connects two relations by:
  - Equating attributes of the same name, and
  - Projecting out one copy of each pair of equated attributes.
- Called *natural* join.
- Denoted R3 := R1 JOIN R2.

14

## Example

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 JOIN 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. |

)

15

## Renaming

- The RENAME operator gives a new schema to a relation.
- R1 := RENAME$_{R1(A1,\ldots,An)}$(R2) makes R1 be a relation with attributes A1,…,A*n* and the same tuples as R2.
- Simplified notation: R1(A1,…,A*n*) := R2.

16

## Example

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. |

)

17

## Building Complex Expressions

- Algebras allow us to express sequences of operations in a natural way.
  - Example: in arithmetic --- $(x + 4)*(y - 3)$.
- Relational algebra allows the same.
- Three notations, just as in arithmetic:
  1. Sequences of assignment statements.
  2. Expressions with several operators.
  3. Expression trees.

18

## Sequences of Assignments

- Create temporary relation names.
- Renaming can be implied by giving relations a list of attributes.
- Example: R3 := R1 $JOIN_C$ R2 can be written:

  R4 := R1 * R2

  R3 := $SELECT_C$(R4)

19

## Expressions in a Single Assignment

- Example: the theta-join R3 := R1 $JOIN_C$ R2 can be written: R3 := $SELECT_C$ (R1 * R2)
- Precedence of relational operators:
  1. Unary operators --- select, project, rename --- have highest precedence, bind first.
  2. Then come products and joins.
  3. Then intersection.
  4. Finally, union and set difference bind last.
- But you can always insert parentheses to force the order you desire.

20

## Expression Trees

- Leaves are operands --- either variables standing for relations or particular, constant relations.
- Interior nodes are operators, applied to their child or children.

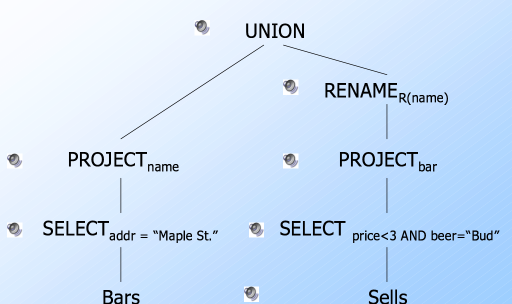
21

## Example

- 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.

22

## As a Tree:




23

## Example

- 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.

24

## The Tree


25

## Schemas for Interior Nodes

- An expression tree defines a schema for the relation associated with each interior node.
- Similarly, a sequence of assignments defines a schema for each relation on the left of the := sign.

26

## Schema-Defining Rules 1

- For 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.

27

## Schema-Defining Rules 2

- Product: the schema is the attributes of both relations.
  - Use R.*A*, etc., to distinguish two attributes named *A*.
- Theta-join: same as product.
- Natural join: use attributes of both relations.
  - Shared attribute names are merged.
- Renaming: the operator tells the schema.

28

## Relational Algebra on Bags

- A *bag* is like a set, but an element may appear more than once.
  - *Multiset* is another name for "bag."
- Example: {1,2,1,3} is a bag. {1,2,3} is also a bag that happens to be a set.
- Bags also resemble lists, but order in a bag is unimportant.
  - Example: {1,2,1} = {1,1,2} as bags, but [1,2,1] != [1,1,2] as lists.

29

## Why Bags?

- SQL, the most important query language for relational databases is actually a bag language.
  - SQL will eliminate duplicates, but usually only if you ask it to do so explicitly.
- Some operations, like projection, are much more efficient on bags than sets.

30

## Operations on Bags

- Selection applies to each tuple, so its effect on bags is like its effect on sets.
- 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.

31

## Example: Bag Selection

R(

| A, | B |
|---|---|
| 1 | 2 |
| 5 | 6 |
| 1 | 2 |

)

S(

| B, | C |
|---|---|
| 3 | 4 |
| 7 | 8 |

)

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

| A | B |
|---|---|
| 1 | 2 |
| 1 | 2 |

32

## Example: Bag Projection

R(

| A, | B |
|---|---|
| 1 | 2 |
| 5 | 6 |
| 1 | 2 |

)

S(

| B, | C |
|---|---|
| 3 | 4 |
| 7 | 8 |

)

$PROJECT_A$ (R) =

| A |
|---|
| 1 |
| 5 |
| 1 |

33

## Example: Bag Product

R(

| A, | B |
|---|---|
| 1 | 2 |
| 5 | 6 |
| 1 | 2 |

)

S(

| B, | C |
|---|---|
| 3 | 4 |
| 7 | 8 |

)

R * 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 |

34

## Example: Bag Theta-Join

R(

| A, | B |
|---|---|
| 1 | 2 |
| 5 | 6 |
| 1 | 2 |

)

S(

| B, | C |
|---|---|
| 3 | 4 |
| 7 | 8 |

)

R $JOIN_{R.B<S.B}$ 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 |

35

## Bag Union

- Union, intersection, and difference need new definitions for bags.
- 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} UNION {1,1,2,3,1} = {1,1,1,1,1,2,2,3}

36

## 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} INTER {1,2,3} = {1,2}.

37

## 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} – {1,2,3} = {1}.

38

## Beware: Bag Laws != Set Laws

- Not all algebraic laws that hold for sets also hold for bags.
- For one example, the commutative law for union (*R* UNION *S* = *S* UNION *R* ) *does* hold for bags.
  - Since addition is commutative, adding the number of times *x* appears in *R* and *S* doesn't depend on the order of *R* and *S*.

39

## An Example of Inequivalence

- Set union is *idempotent*, meaning that *S* UNION *S* = *S*.
- However, for bags, if *x* appears *n* times in *S*, then it appears 2*n* times in *S* UNION *S*.
- Thus *S* UNION *S* != *S* in general.

40

## The Extended Algebra

1. DELTA = eliminate duplicates from bags.
2. TAU = sort tuples.
3. *Extended projection* : arithmetic, duplication of columns.
4. GAMMA = grouping and aggregation.
5. OUTERJOIN: avoids "dangling tuples" = tuples that do not join with anything.

41

## Duplicate Elimination

- R1 := DELTA(R2).
- R1 consists of one copy of each tuple that appears in R2 one or more times.

42

## Example: Duplicate Elimination

R =

| A | B |
|---|---|
| 1 | 2 |
| 3 | 4 |
| 1 | 2 |

DELTA(R) =

| A | B |
|---|---|
| 1 | 2 |
| 3 | 4 |



## Sorting

- R1 := $TAU_L$ (R2).
  - $L$ is a list of some of the attributes of R2.
- R1 is the list of tuples of R2 sorted first on the value of the first attribute on $L$, then on the second attribute of $L$, and so on.
  - Break ties arbitrarily.
- TAU is the only operator whose result is neither a set nor a bag.



## Example: Sorting

R =

| A | B |
|---|---|
| 1 | 2 |
| 3 | 4 |
| 5 | 2 |

$TAU_B$ (R) = [(5,2), (1,2), (3,4)]



## Extended Projection

- Using the same $PROJ_L$ operator, we allow the list L to contain arbitrary expressions involving attributes, for example:
  1. Arithmetic on attributes, e.g., *A+B*.
  2. Duplicate occurrences of the same attribute.



## Example: Extended Projection

R =

| A | B |
|---|---|
| 1 | 2 |
| 3 | 4 |

$PROJ_{A+B,A,A}$ (R) =

| A+B | A1 | A2 |
|---|---|---|
| 3 | 1 | 1 |
| 7 | 3 | 3 |



## Aggregation Operators

- Aggregation operators are not operators of relational algebra.
- Rather, they apply to entire columns of a table and produce a single result.
- The most important examples: SUM, AVG, COUNT, MIN, and MAX.

## Example: Aggregation

R =

| A | B |
|---|---|
| 1 | 3 |
| 3 | 4 |
| 3 | 2 |

SUM(A) = 7
COUNT(A) = 3
MAX(B) = 4
AVG(B) = 3

49

## Grouping Operator

- R1 := GAMMA$_L$ (R2). *L* is a list of elements that are either:
  1. Individual (*grouping*) attributes.
  2. AGG(*A*), where AGG is one of the aggregation operators and *A* is an attribute.

50

## Applying GAMMA$_L$(R)

- Group *R* according to all the grouping attributes on list *L*.
  - That is, form one group for each distinct list of values for those attributes in *R*.
- Within each group, compute AGG(*A*) for each aggregation on list *L*.
- Result has grouping attributes and aggregations as attributes. One tuple for each list of values for the grouping attributes and their group's aggregations.

51

## Example: Grouping/Aggregation

R =

| A | B | C |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 1 | 2 | 5 |

GAMMA$_{A,B,AVG(C)}$ (R) = ??

First, group *R* :

| A | B | C |
|---|---|---|
| 1 | 2 | 3 |
| 1 | 2 | 5 |
| 4 | 5 | 6 |

Then, average *C* within groups:

| A | B | AVG(C) |
|---|---|---|
| 1 | 2 | 4 |
| 4 | 5 | 6 |

52

## Outerjoin

- Suppose we join *R* JOIN$_C$ *S*.
- A tuple of *R* that has no tuple of *S* with which it joins is said to be *dangling*.
  - Similarly for a tuple of *S*.
- Outerjoin preserves dangling tuples by padding them with a special NULL symbol in the result.

53

## Example: Outerjoin

R =

| A | B |
|---|---|
| 1 | 2 |
| 4 | 5 |

S =

| B | C |
|---|---|
| 2 | 3 |
| 6 | 7 |

(1,2) joins with (2,3), but the other two tuples are dangling.

R OUTERJOIN S =

| A | B | C |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | NULL |
| NULL | 6 | 7 |

54