



## Lecture 4 of 42

### Relational Joins

Wednesday, 30 January 2008

William H. Hsu  
Department of Computing and Information Sciences, KSU

KSOL course page: <http://snipurl.com/va60>  
Course web site: <http://www.kddresearch.org/Courses/Spring-2008/CIS560>  
Instructor home page: <http://www.cis.ksu.edu/~bhsu>

Reading for Next Class:  
Chapter 2, Silberschatz *et al.*, 5<sup>th</sup> edition



## Chapter 2: Relational Model

- Structure of Relational Databases
- Fundamental Relational-Algebra-Operations
- Additional Relational-Algebra-Operations
- Extended Relational-Algebra-Operations
- Null Values
- Modification of the Database





## Relational Algebra: Review

- Procedural language
- Six basic operators
  - \* select:  $\sigma$
  - \* project:  $\pi$
  - \* union:  $\cup$
  - \* set difference:  $-$
  - \* Cartesian product:  $\times$
  - \* rename:  $\rho$
- The operators take one or two relations as inputs and produce a new relation as a result.



## Review: Finding Max using Self-Join

- Find the largest account balance
  - \* Strategy:
    - ⇒ Find those balances that are *not* the largest
      - ◆ Rename *account* relation as *d* so that we can compare each account balance with all others
    - ⇒ Use set difference to find those account balances that were *not* found in the earlier step.
  - \* The query is:

$$\pi_{balance}(account) - \pi_{account.balance}(\sigma_{account.balance < d.balance}(account \times \rho_d(account)))$$

$$\pi_b(a) - \left[ \pi_{a,b}(account \times \rho_d(account)) \right]$$





## Formal Definition

- A basic expression in the relational algebra consists of either one of the following:
  - \* A relation in the database  $r(R)$   $s(S)$
  - \* A constant relation
- Let  $E_1$  and  $E_2$  be relational-algebra expressions; the following are all relational-algebra expressions:
  - \*  $E_1 \cup E_2$
  - \*  $E_1 - E_2$
  - \*  $E_1 \times E_2$
  - \*  $\sigma_P(E_1)$ ,  $P$  is a predicate on attributes in  $E_1$
  - \*  $\Pi_S(E_1)$ ,  $S$  is a list consisting of some of the attributes in  $E_1$
  - \*  $\rho_x(E_1)$ ,  $x$  is the new name for the result of  $E_1$



## Additional Operations

We define additional operations that do not add any power to the relational algebra, but that simplify common queries.

- Set intersection  $\cap$  *defn in terms of  $\sigma$  &  $\pi$*
- Natural join  $\bowtie$
- Division  $\div$
- Assignment

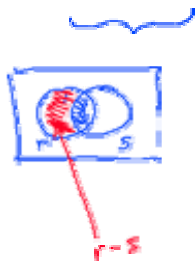
$$r \bowtie s \equiv \sigma_{\substack{R.A=S.A \\ R.B=S.B}}(r \times s)$$





## Set-Intersection Operation

- Notation:  $r \cap s$
- Defined as:
- $r \cap s = \{ t \mid t \in r \text{ and } t \in s \}$
- Assume:
  - \*  $r, s$  have the same arity
  - \* attributes of  $r$  and  $s$  are compatible
- Note:  $r \cap s = r - (r - s)$



## Set-Intersection Operation – Example

- Relation  $r, s$ :

A	B
$\alpha$	1
$\alpha$	2
$\beta$	1

$r$

A	B
$\alpha$	2
$\beta$	3

$s$

- $r \cap s$

A	B
$\alpha$	2

$$|r \cap s| \leq |r|$$

$$|r \cap s| \leq |s|$$

$$|r \cap s| \leq \min(|r|, |s|)$$



## Natural-Join Operation

- Notation:  $r \bowtie s$
- Let  $r$  and  $s$  be relations on schemas  $R$  and  $S$  respectively. Then,  $\bowtie s$  is a relation on schema  $R \cup S$  obtained as follows:
  - \* Consider each pair of tuples  $t_r$  from  $r$  and  $t_s$  from  $s$ .
  - \* If  $t_r$  and  $t_s$  have the same value on each of the attributes in  $R \cap S$ , add a tuple  $t$  to the result, where
    - $\Rightarrow t$  has the same value as  $t_r$  on  $r$
    - $\Rightarrow t$  has the same value as  $t_s$  on  $s$

- Example:

$R = (A, B, C, D)$

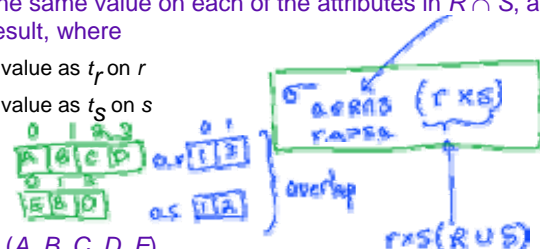
$S = (E, B, D)$

\* Result schema =  $(A, B, C, D, E)$

\*  $r \bowtie s$  is defined as:

$$\pi_{r.A, r.B, r.C, r.D, s.E} (\sigma_{r.B=s.B \wedge r.D=s.D} (r \times s))$$

a.e. RUS



## Natural Join Operation – Example

- Relations  $r, s$ :

A	B	C	D
$\alpha$	1	$\alpha$	a
$\beta$	2	$\gamma$	a
$\gamma$	4	$\beta$	b
$\alpha$	1	$\gamma$	a
$\delta$	2	$\beta$	b

$r$

B	D	E
1	a	$\alpha$
3	a	$\beta$
1	a	$\gamma$
2	b	$\delta$
3	b	$\epsilon$

$s$

- $r \bowtie s$

A	B	C	D	E
$\alpha$	1	$\alpha$	a	$\alpha$
$\alpha$	1	$\alpha$	a	$\gamma$
$\alpha$	1	$\gamma$	a	$\alpha$
$\alpha$	1	$\gamma$	a	$\gamma$
$\delta$	2	$\beta$	b	$\delta$



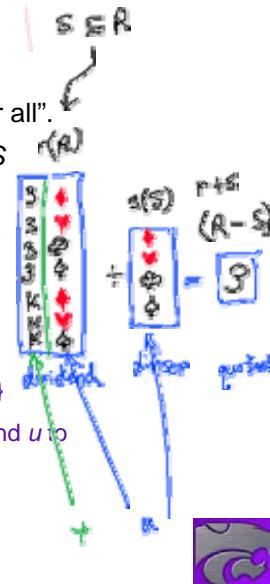
## Division Operation

- Notation:  $r \div s$
- Suited to queries that include the phrase “for all”.
- Let  $r$  and  $s$  be relations on schemas  $R$  and  $S$  respectively where
  - \*  $R = (A_1, \dots, A_m, B_1, \dots, B_n)$
  - \*  $S = (B_1, \dots, B_n)$

The result of  $r \div s$  is a relation on schema  $R - S = (A_1, \dots, A_m)$

$$r \div s = \{ t \mid t \in \Pi_{R-S}(r) \wedge \forall u \in s (tu \in r) \}$$

Where  $tu$  means the concatenation of tuples  $t$  and  $u$  to produce a single tuple



## Division Operation – Example

■ Relations  $r, s$ :

A	B
$\alpha$	1
$\alpha$	2
$\alpha$	3
$\beta$	1
$\gamma$	1
$\delta$	1
$\delta$	3
$\delta$	4
$\epsilon$	6
$\epsilon$	1
$\beta$	2

B
1
2

$s$

■  $r \div s$ :

A
$\alpha$
$\beta$

$r$



## Another Division Example

Relations  $r, s$ :

A	B	C	D	E
$\alpha$	a	$\alpha$	a	1
$\alpha$	a	$\gamma$	a	1
$\alpha$	a	$\gamma$	b	1
$\beta$	a	$\gamma$	a	1
$\beta$	a	$\gamma$	b	3
$\gamma$	a	$\gamma$	a	1
$\gamma$	a	$\gamma$	b	1
$\gamma$	a	$\beta$	h	1

$r$

D	E
a	1
b	1

$s$

$r \div s$ :

A	B	C
$\alpha$	a	$\gamma$
$\gamma$	a	$\gamma$



## Division Operation (Cont.)

Property

- \* Let  $q = r \div s$
- \* Then  $q$  is the largest relation satisfying  $q \times s \subseteq r$

Definition in terms of the basic algebra operation

Let  $r(R)$  and  $s(S)$  be relations, and let  $S \subseteq R$

$$r \div s = \Pi_{R-S}(r) - \Pi_{R-S}((\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r))$$

To see why:

- \*  $\Pi_{R-S,S}(r)$  simply reorders attributes of  $r$
- \*  $\Pi_{R-S}(\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r)$  is those tuples  $t$  in  $\Pi_{R-S}(r)$  such that for some  $u \in s, tu \notin r$ .

"less than a full book"

$$r \supseteq s \times q$$

$$r \div s = q$$

$$q = \text{def } \arg \max_{p \subseteq R} p \times s \subseteq r$$



## Assignment Operation

- The assignment operation ( $\leftarrow$ ) provides a convenient way to express complex queries.
  - \* Write query as a sequential program consisting of
    - $\Rightarrow$  a series of assignments
    - $\Rightarrow$  followed by an expression whose value is displayed as a result of the query.
  - \* Assignment must always be made to a temporary relation variable.
- Example: Write  $r \div s$  as

$temp1 \leftarrow \Pi_{R-S}(r)$   
 $temp2 \leftarrow \Pi_{R-S}((temp1 \times s) - \Pi_{R-S,S}(r))$   
 $result = temp1 - temp2$

- \* The result to the right of the  $\leftarrow$  is assigned to the relation variable on the left of the  $\leftarrow$ .
- \* May use variable in subsequent expressions.



## Bank Example Queries

- Find the names of all customers who have a loan and an account in bank.

$\Pi_{customer\_name}(borrower) \cap \Pi_{customer\_name}(depositor)$

- Find the name of all customers who have a loan at the bank and the loan amount

$\Pi_{customer\_name, loan\_number, amount}(borrower \bowtie loan)$

SQL  
 AS e<sub>1</sub> → IN e<sub>2</sub>  
 IN e<sub>2</sub>  
 AS e<sub>1</sub> → IN e<sub>2</sub>  
 IN e<sub>2</sub>





## Bank Example Queries

- Find all customers who have an account from at least the "Downtown" and the Uptown" branches.

- Query 1

$$\Pi_{customer\_name} (\sigma_{branch\_name = \text{"Downtown"}} (depositor \bowtie account)) \cap$$

$$\Pi_{customer\_name} (\sigma_{branch\_name = \text{"Uptown"}} (depositor \bowtie account))$$

- Query 2

$$\Pi_{customer\_name, branch\_name} (depositor \bowtie account) \div \rho_{temp(branch\_name)} (\{(\text{"Downtown"}), (\text{"Uptown"})\})$$

Note that Query 2 uses a constant relation.



## Example Queries

- Find all customers who have an account at all branches located in Brooklyn city.

$$\Pi_{customer\_name, branch\_name} (depositor \bowtie account) \div \Pi_{branch\_name} (\sigma_{branch\_city = \text{"Brooklyn"}} (branch))$$

$$R \supseteq S$$

$$S \subseteq R$$

$$\neq$$

$$\neq$$



## Extended Relational-Algebra-Operations

- Generalized Projection
- Aggregate Functions
- Outer Join



## Generalized Projection

- Extends the projection operation by allowing arithmetic functions to be used in the projection list.

$$\Pi_{F_1, F_2, \dots, F_n}(E)$$

- $E$  is any relational-algebra expression
- Each of  $F_1, F_2, \dots, F_n$  are arithmetic expressions involving constants and attributes in the schema of  $E$ .
- Given relation  $credit\_info(customer\_name, limit, credit\_balance)$ , find how much more each person can spend:

$$\Pi_{customer\_name, limit - credit\_balance}(credit\_info)$$



## Aggregate Functions and Operations

- **Aggregation function** takes a collection of values and returns a single value as a result.

**avg**: average value  
**min**: minimum value  
**max**: maximum value  
**sum**: sum of values  
**count**: number of values

- **Aggregate operation** in relational algebra

$$g_{G_1, G_2, \dots, G_n, F_1(A_1), F_2(A_2), \dots, F_n(A_n)}(E)$$

$E$  is any relational-algebra expression

- \*  $G_1, G_2, \dots, G_n$  is a list of attributes on which to group (can be empty)
- \* Each  $F_i$  is an aggregate function
- \* Each  $A_i$  is an attribute name



## Aggregate Operation – Example

- Relation  $r$ :

A	B	C
$\alpha$	$\alpha$	7
$\alpha$	$\beta$	7
$\beta$	$\beta$	3
$\beta$	$\beta$	10

- $g_{\text{sum}(c)}(r)$

sum(c)
27





## Aggregate Operation – Example

- Relation *account* grouped by *branch-name*:

<i>branch_name</i>	<i>account_number</i>	<i>balance</i>
Perryridge	A-102	400
Perryridge	A-201	900
Brighton	A-217	750
Brighton	A-215	750
Redwood	A-222	700

*Take F(A) for every G;*  
*branch\_name*  $\mathcal{G}$  **sum(balance)** (*account*)

<i>branch_name</i>	<b>sum(balance)</b>
Perryridge	1300
Brighton	1500
Redwood	700



## Aggregate Functions (Cont.)

- Result of aggregation does not have a name
  - \* Can use rename operation to give it a name
  - \* For convenience, we permit renaming as part of aggregate operation

*branch\_name*  $\mathcal{G}$  **sum(balance)** **as sum\_balance** (*account*)



## Outer Join

- An extension of the join operation that avoids loss of information.
- Computes the join and then adds tuples from one relation that does not match tuples in the other relation to the result of the join.
- Uses *null* values:
  - \* *null* signifies that the value is unknown or does not exist
  - \* All comparisons involving *null* are (roughly speaking) **false** by definition.
    - ⇒ We shall study precise meaning of comparisons with nulls later



## Outer Join – Example

- Relation *loan*

<i>loan_number</i>	<i>branch_name</i>	<i>amount</i>
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

- Relation *borrower*

<i>customer_name</i>	<i>loan_number</i>
Jones	L-170
Smith	L-230
Hayes	L-155

