



Lecture 3 of 42

Relational Databases Discussion: Problem Set 1

Monday, 28 January 2008

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

KSOL course page: <http://snipurl.com/1pq4c>
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.*, 5th edition – next week
Problem Set 1



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: σ
 - * project: Π
 - * union: \cup
 - * set difference: $-$
 - * Cartesian product: \times
 - * rename: ρ
- The operators take one or two relations as inputs and produce a new relation as a result.



Select Operation

- Notation: $\sigma_p(r)$
- p is called the **selection predicate**
- Defined as:

$$\sigma_p(r) = \{t \mid t \in r \text{ and } p(t)\}$$

Where p is a formula in propositional calculus consisting of **terms** connected by : \wedge (**and**), \vee (**or**), \neg (**not**)

Each **term** is one of:

$\langle \text{attribute} \rangle \text{ op } \langle \text{attribute} \rangle$ or $\langle \text{constant} \rangle$

where op is one of: $=, \neq, >, \geq, <, \leq$

- Example of selection:

$$\sigma_{\text{branch_name}=\text{"Perryridge"}}(\text{account})$$





Project Operation

- Notation:

$$\Pi_{A_1, A_2, \dots, A_k}(r)$$

where A_1, A_2 are attribute names and r is a relation name.

- The result is defined as the relation of k columns obtained by erasing the columns that are not listed
- Duplicate rows removed from result, since relations are sets
- Example: To eliminate the *branch_name* attribute of *account*

$$\Pi_{\text{account_number, balance}}(\text{account})$$



Union Operation

- Notation: $r \cup s$
- Defined as:

$$r \cup s = \{t \mid t \in r \text{ or } t \in s\}$$

- For $r \cup s$ to be valid.
 1. r, s must have the **same arity** (same number of attributes)
 2. The attribute domains must be **compatible** (example: 2nd column of r deals with the same type of values as does the 2nd column of s)
- Example: to find all customers with either an account or a loan

$$\Pi_{\text{customer_name}}(\text{depositor}) \cup \Pi_{\text{customer_name}}(\text{borrower})$$





Set Difference Operation

- Notation $r - s$
- Defined as:
$$r - s = \{t \mid t \in r \text{ and } t \notin s\}$$
- Set differences must be taken between **compatible** relations.
 - * r and s must have the same arity
 - * attribute domains of r and s must be compatible



Cartesian-Product Operation

- Notation $r \times s$
- Defined as:
$$r \times s = \{tq \mid t \in r \text{ and } q \in s\}$$
- Assume that attributes of $r(R)$ and $s(S)$ are disjoint. (That is, $R \cap S = \emptyset$).
- If attributes of $r(R)$ and $s(S)$ are not disjoint, then renaming must be used.





Composition of Operations

- Can build expressions using multiple operations
- Example: $\sigma_{A=C}(r \times s)$
- $r \times s$

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

- $\sigma_{A=C}(r \times s)$

A	B	C	D	E
α	1	α	10	a
β	2	β	10	a
β	2	β	20	b



Rename Operation

- Allows us to name, and therefore to refer to, the results of relational-algebra expressions.
- Allows us to refer to a relation by more than one name.
- Example:

$$\rho_X(E)$$

returns the expression E under the name X

- If a relational-algebra expression E has arity n , then

$$\rho_{X(A_1, A_2, \dots, A_n)}(E)$$

returns the result of expression E under the name X , and with the attributes renamed to A_1, A_2, \dots, A_n .





Banking Example

- *branch* (*branch_name*, *branch_city*, *assets*)
- *customer* (*customer_name*, *customer_street*, *customer_city*)
- *account* (*account_number*, *branch_name*, *balance*)
- *loan* (*loan_number*, *branch_name*, *amount*)
- *depositor* (*customer_name*, *account_number*)
- *borrower* (*customer_name*, *loan_number*)



Example Queries

- Find all loans of over \$1200

$$\sigma_{amount > 1200} (loan)$$

- Find the loan number for each loan of an amount greater than \$1200

$$\Pi_{loan_number} (\sigma_{amount > 1200} (loan))$$





Example Queries

- Find the names of all customers who have a loan, an account, or both, from the bank

$$\Pi_{customer_name}(borrower) \cup \Pi_{customer_name}(depositor)$$

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

$$\Pi_{customer_name}(borrower) \cap \Pi_{customer_name}(depositor)$$



Example Queries

- Find the names of all customers who have a loan at the Perryridge branch.

$$\Pi_{customer_name}(\sigma_{branch_name="Perryridge"}(\sigma_{borrower.loan_number = loan.loan_number}(borrower \times loan)))$$

- Find the names of all customers who have a loan at the Perryridge branch but do not have an account at any branch of the bank.

$$\Pi_{customer_name}(\sigma_{branch_name = "Perryridge"}$$

$$(\sigma_{borrower.loan_number = loan.loan_number}(borrower \times loan))) - \Pi_{customer_name}(depositor)$$





Example Queries

- Find the names of all customers who have a loan at the Perryridge branch.

- Query 1

$$\Pi_{\text{customer_name}} (\sigma_{\text{branch_name} = \text{"Perryridge"}} (\sigma_{\text{borrower.loan_number} = \text{loan.loan_number}} (\text{borrower} \times \text{loan})))$$

- Query 2

$$\Pi_{\text{customer_name}} (\sigma_{\text{loan.loan_number} = \text{borrower.loan_number}} (\sigma_{\text{branch_name} = \text{"Perryridge"}} (\text{loan}) \times \text{borrower}))$$


Example Queries

- 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:

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




Formal Definition

- A basic expression in the relational algebra consists of either one of the following:
 - * A relation in the database
 - * 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
- Natural join
- Division
- Assignment





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
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- $r \cap s$

A	B
α	2





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:
 - $\bowtie \Pi_{r.A, r.B, r.C, r.D, s.E} (\sigma_{r.B=s.B \wedge r.D=s.D} (r \times s))$



Natural Join Operation – Example

- Relations r, s :

A	B	C	D
α	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

r

B	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	ϵ

s

- $r \bowtie s$

A	B	C	D	E
α	1	α	a	α
α	1	α	a	γ
α	1	γ	a	α
α	1	γ	a	γ
δ	2	β	b	δ





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
α	1
α	2
α	3
β	1
γ	1
δ	1
δ	3
δ	4
ϵ	6
ϵ	1
β	2

B
1
2

s

- $r \div s$:

A
α
β

r





Another Division Example

- Relations r, s :

A	B	C	D	E
α	a	α	a	1
α	a	γ	a	1
α	a	γ	b	1
β	a	γ	a	1
β	a	γ	b	3
γ	a	γ	a	1
γ	a	γ	b	1
γ	a	β	b	1

r

D	E
a	1
b	1

s

- $r \div s$:

A	B	C
α	a	γ
γ	a	γ



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
- * $\Pi_{R-S}(\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r)$ gives those tuples t in $\Pi_{R-S}(r)$ such that for some tuple $u \in s$, $tu \notin r$.



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

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

- * 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 at 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

$$\Pi_{customer_name, loan_number, amount}(borrower \bowtie loan)$$




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.

$$\Pi_{customer_name, branch_name} (depositor \bowtie account) \div \Pi_{branch_name} (\sigma_{branch_city = \text{"Brooklyn"}} (branch))$$





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
---	---	---

α	α	7
α	β	7
β	β	3
β	β	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

branch_name \mathcal{G} **sum**(*balance*) (*account*)

<i>branch_name</i>	sum (<i>balance</i>)
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

