Relational Algebra

COMP 3380 - Databases: Concepts and Usage

Department of Computer Science
The University of Manitoba
Fall 2006

Relational Query Languages

- **Query languages (QLs):** Allow manipulation and retrieval of data from a DB
- Relational model supports simple & powerful QLs:
  - Strong formal foundation based on logic
  - Allows for much optimization
- Query languages ≠ programming languages
  - QLs not intended to be used for complex calculations
  - QLs support easy & efficient access to large data sets

Formal Relational Query Languages

- Two mathematical query languages form the basis for "real" languages (e.g., SQL), and for implementation:
  - **Relational Algebra:** User specifies what data is required & *how to get those data*
    - Procedural (*or* operational)
    - Very useful for representing execution plans
  - **Relational Calculus:** User specifies what data is required *without specifying how to get those data*
    - Declarative (i.e., non-procedural)

Preliminaries

- A query is applied to *relation instances*, and **the result of a query is also a *relation instance***
  - Schemas of input relations for a query are fixed
  - The schema for the result of a given query is fixed
    - Determined by definition of query language constructs
- Positional *vs.* named-attribute notation:
  - **Positional notation** easier for formal definitions, **named-attribute notation** more readable.
  - Both used in SQL

Relational Algebra

- Basic operations:
  - **Selection (σ):** Selects a subset of rows from a relation
  - **Projection (π):** Extracts wanted columns from a relation
  - **Cartesian-product (x):** Allows us to combine two relations
  - **Set-difference (–):** Returns the tuples that are in R1 but not in R2
  - **Union (∪):** Returns the tuples that are in R1 or R2

COMP 3380 (Fall 2006), Leung 5

Relational Algebra

- Additional operations:
  - **Intersection (∩)**
  - **Join (⋈)**
  - **Division (÷)**
  - **Rename (*ρ*)**
  - **Assignment (←)**

  Not essential, but (very) useful
- Since each operation returns a relation, operations can be *composed* (Algebra is "closed")

COMP 3380 (Fall 2006), Leung 6

Selection

- **Selects rows** that satisfy the *selection condition*
- $\sigma_p(R) = \{t \mid t \in R \text{ and } p(t)\}$ where p is a formula in propositional calculus consisting of terms connected by $\wedge$, $\vee$, or $\neg$. Each term is either
  - attr_1 *op* attr_2, or
  - attr *op* constant,

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

Selection

- **No duplicates in result**
- *Schema* of result identical to schema of the (only) input relation
- *Result* relation can be the input for another relational algebra operation (*Operator composition*)
- E.g., consider *Acct* (*acctNum, branchName, bal*). Find all those tuples of *Acct* about the Vancouver branch:

$$\sigma_{branchName='Vancouver'}(Acct)$$

Example: Selection

- Relation *R*:

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

- $\sigma_{A=B \wedge D > 5}(R)$:

A	B	C	D
α	α	1	7
β	β	23	10

Projection

- **Projects columns**, i.e., extracts only attributes that are in the *projection list*
- $\pi_{A1,\, A2,\, \ldots,\, Ak}$ **(*R*)** where A1, A2, …, A*k* are attributes in the relation *R*
- Useful when we do not want to retrieve unnecessary data, thereby reducing the query cost and/or hiding irrelevant data from the user
- Projection operator has to **eliminate duplicates**, since relations are *sets*

Projection

- *Schema* of result contains exactly the attributes in the projection list, with the same names that they had in the (only) input relation.
- E.g., consider *Acct* (*acctNum*, *branchName*, *bal*). List the account number and balance for each account (i.e., do not list the *branchName* attribute of *Acct*):

$$\pi_{acctNum,\ bal}\ (Acct)$$

Example: Projection

- Relation R:

A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

- $\pi_{A,C}\ (R)$:

A	C
α	1
α	1
β	1
β	2

➔

A	C
α	1
β	1
β	2

Union

- ***R*** $\cup$ ***S*** takes two input relations *R* and *S*, which must be **union-compatible**
  - (i.e., same number of attributes &
  - "corresponding" attributes have the same domain), &

  returns a relation instance containing all tuples that occur in *R* or *S*
- ***R*** $\cup$ ***S*** = $\{t \mid t \in R \text{ or } t \in S\}$
- *Schema* of result identical to schema of input relations
- E.g., consider *Depositor* (*custName, acctNum*) & *Borrower* (*custName, acctNum*). Find the names of all customers with an account or a loan (i.e., either an account or a loan, or both):

$$\pi_{custName}\ (Depositor) \cup \pi_{custName}\ (Borrower)$$

Example: Union

- Relations *R, S*:

R

A	B
α	1
α	2
β	1

S

A	B
α	2
β	3

- *R* $\cup$ *S*:

A	B
α	1
α	2
β	1
β	3

Intersection

- ***R* ∩ *S*** takes two input relations *R* and *S*, which must be **union-compatible**, & returns a relation instance containing all tuples that occur in both *R* and *S*
- ***R* ∩ *S*** = $\{t \mid t \in R \text{ and } t \in S\}$
- *Schema* of result identical to schema of input relations
- E.g., consider *Depositor* (*custName, acctNum*) & *Borrower* (*custName, acctNum*). Find the names of all customers with both an account and a loan:

Example: Intersection

- Relations *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

Set Difference

- ***R – S*** takes two input relations *R* and *S*, which must be **union-compatible**, & returns a relation instance containing all tuples that occur in *R* but not in *S*
- ***R – S*** = $\{t \mid t \in R \text{ and } t \notin S\}$
- *Schema* of result identical to schema of input relations
- E.g., consider *Depositor* (*custName, acctNum*) & *Borrower* (*custName, acctNum*). Find the names of all customers with an account but not a loan:

Example: Set Difference

- Relations *R, S*:

R

A	B
α	1
α	2
β	1

S

A	B
α	2
β	3

- *R – S*:

A	B
α	1
β	1

Cartesian Product (or Cross Product)

- Pairs each row of R with each row of S
- $\boldsymbol{R} \times \boldsymbol{S} = \{t\,q \mid t \in R \text{ and } q \in S\}$
- *Result schema* has 1 attr per attr of R and S, with attribute names "inherited" if possible.
- If schema(R) and schema(S) are not disjoint (➔ ambiguous attribute names):
  - **prefix attributes with the relation names**, *or*
  - rename the attributes.

Example: Cartesian Product

- Relations R, S:

A	B
α	1
β	2

R

A	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

S

- $T = R \times S$:

R.A	B	S.A	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

Rename

- Allows us to name (& refer to) the results of RA expressions; allows us to refer to a relation by more than one name.
- Avoids ambiguity when the same relation appears in a query more than once
- $\boldsymbol{\rho_Y (E)}$ returns the expression E under the name Y
- E.g., $R \times \rho_{R'} (R)$
- E.g., $\rho_{T(\mathbf{A}, B, \mathbf{C}, D, E)} (R \times S)$
- <u>NOTE</u>: Formal RA does not have names for relations

Join (or Natural Join)

- $\boldsymbol{R \bowtie S}$ joins tuples in relations R and S
- An equi-join in which equalities are specified on *all* attrs having the same name in R & S
- *Result schema* similar to Cartesian-product, but only *one* copy of attributes for which equality is specified
- Fewer tuples than Cartesian-product, might be able to compute more efficiently

Join (or Natural Join)

- E.g., Consider R (A, $\boldsymbol{B}$, C, $\boldsymbol{D}$) & S ($\boldsymbol{B}$, $\boldsymbol{D}$, E).
  - Schema of $T = R \bowtie S$ is (A, $\boldsymbol{B}$, C, $\boldsymbol{D}$, E)
  - $T = R \bowtie S$

    $= \pi_{A,\ R.B,\ C,\ R.D,\ E}\ (\sigma_{R.B=S.B\ \wedge\ R.D=S.D}\ (R \times S))$
- <u>NOTE</u>: If there are *no common attributes* in relations P and Q, then P ⋈ Q = P x Q.

Example: Join (1)

- Relations R, S:

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

R

B	D	E
1	**a**	**α**
1	a	γ
2	**b**	**δ**

S

- $T = R \bowtie S$:

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**

Example: Join (2)

- Relations *R*, *S*:

R

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

S

B	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	$\in$

- $T = R \bowtie S$:

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

Relaxations of (Natural) Join: Equi-Join & Condition Join

- **Join** (or **natural join**):
  - An equi-join in which equalities are specified on *all* attributes having the same name in *R* & *S*
- **Equi-join**:
  - A special case of condition join where *cond* contains only **equalities**
  - E.g., $U = \boldsymbol{R} \bowtie_B \boldsymbol{S} = \pi_{A, \mathbf{R.B}, C, \mathbf{R.D}, \mathbf{S.D}, E} (\sigma_{\mathbf{R.B=S.B}} (R \times S))$
- **Condition join** (aka **theta-join**):
  - $R \bowtie_{cond} S = \sigma_{cond} (R \times S)$
  - E.g., $V = \boldsymbol{R} \bowtie_{\mathbf{R.B < S.B}} \boldsymbol{S} = \pi_{A, R.B, C, R.D, S.B, S.D, E} (\sigma_{\mathbf{R.B<S.B}} (R \times S))$

Extensions of (Inner) Join: Outer Joins

- Extension of (inner) joins that avoids loss of information.
- Compute the **inner join** & then add tuples from one relation that do not match tuples in the other relation to the result of the join.
- Variations:
  - **Left outer join** (⟕)
  - **Right outer join** (⟖)
  - **(Full) outer join** (⟗)

Example: Outer Joins

- Relations *R, S*:

R

A	B
α	1
β	2
γ	3

S

A	C
β	b
γ	c
δ	d

- $R \bowtie S$:

A	B	C
β	2	b
γ	3	c

R ⟕ *S*:

A	B	C
α	1	NULL
β	2	b
γ	3	c

R ⟖ *S*:

A	B	C
β	2	b
γ	3	c
δ	NULL	d

R ⟗ *S*:

A	B	C
α	1	NULL
β	2	b
γ	3	c
δ	NULL	d

Division

- Suited to queries that include the phrase "for all" (e.g., "find boats that are reserved by **all** sailors")
- $\boldsymbol{R \div S} = \{t \mid t \in \pi_{schema(R-S)}(R) \wedge \forall u \in S\ (tu \in R)\}$
  - $R \div S = \{ <x> \mid \forall y \in S\ (\exists <x,y> \in R) \}$
  - i.e., $R \div S$ contains all x tuples (boats in the answer) such that for ***every*** y tuple (every sailor) in S, there is an xy tuple (a reservation made by sailor y on boat x) in R
  - i.e., If the set of y values (sailors) associated with an x value (boat) in R contains all y values in S (i.e., boat x reserved by all sailors), then the x value is in $R \div S$
- In general, x and y can be any lists of attrs; y is the list of attrs in S, and $x \cup y$ is the list of attrs of R.

COMP 3380 (Fall 2006), Leung 29

Division

- Division is not essential op; just a useful shorthand.
  - (Also true of joins, but joins are so common that systems implement joins specially)
- <u>Idea</u>: For $R \div S$, compute all x values that are **not** "**dis**qualified" by some y value in S.
  - x value is *disqualified* if by attaching y value from S, we obtain an xy tuple that is not in R.
  - $\boldsymbol{R \div S = \pi_x(R) - \pi_x((\pi_x(R) \times S) - R)}$

COMP 3380 (Fall 2006), Leung 30

Example: Division (1)

- E.g., consider *Sailors*(<u>*sID*</u>, *sName, rating, age*) & *Reserves*(<u>*sID, bID, rDate*</u>). **Find IDs of the boats who have reserved *all* sailors**:

  $$\pi_{sID,\, bID}\,(\boldsymbol{Reserves}) \div \pi_{sID}\,(\boldsymbol{Sailors})$$

  $= \{t \mid t \in \pi_{bID}\,(Reserves) \wedge \forall u \in \pi_{sID}\,(Sailors)\ (tu \in \pi_{sID,\, bID}\,(Reserves))\}$

  $= \{<bID> \mid \forall\ sID \in \pi_{sID}\,(Sailors)\ (\exists <sID, bID> \in \pi_{sID,\, bID}\,(Reserves))\}$
- The result contains the ID of each boat x such that for ***every*** sailor y in *Sailors*, there is a reservation on boat x made by sailor y in *Reserves.*
- If the set of sIDs associated with any bID in *Reserves* contains **all** the sIDs in *Sailors* (i.e., a boat reserved by all sailors), then such a bID is in the result of the division $\pi_{sID,\, bID}\,(Reserves) \div \pi_{sID}\,(Sailors)$.
- In other words, bID is *disqualified* if by attaching sID from *Sailors*, we obtain a <sID, bID> tuple that is not in *Reserves*:

  $$\pi_{sID,\, bID}\,(Reserves) \div \pi_{sID}\,(Sailors)$$

  $$= \pi_{bID}(\boldsymbol{Reserves}) - \pi_{bID}\,((\pi_{sID}(\boldsymbol{Sailors}) \times \pi_{bID}(\boldsymbol{Reserves})) - \pi_{sID,bID}(\boldsymbol{Reserves}))$$

Example: Division (2)

- E.g., consider *Boats* (<u>*bID*</u>, *bName, color*) & *Reserves* (<u>*sID, bID, rDate*</u>). **Find IDs of the sailors who have reserved *all* boats**:

  $$\pi_{sID,\, bID}\,(\boldsymbol{Reserves}) \div \pi_{bID}\,(\boldsymbol{Boats})$$

  $= \{t \mid t \in \pi_{sID}\,(Reserves) \wedge \forall u \in \pi_{bID}\,(Boats)\ (tu \in \pi_{sID,\, bID}\,(Reserves))\}$

  $= \{<sID> \mid \forall\ bID \in \pi_{bID}\,(Boats)\ (\exists <sID, bID> \in \pi_{sID,\, bID}\,(Reserves))\}$
- The result contains the ID of each sailor x such that for ***every*** boat y in *Boats*, there is a reservation made by sailor x on boat y in *Reserves.*
- If the set of bIDs associated with any sID in *Reserves* contains **all** the bIDs in *Boats* (i.e., a sailor reserved all boats), then such an sID is in the result of the division $\pi_{sID,\, bID}\,(Reserves) \div \pi_{bID}\,(Boats)$.
- In other words, sID is *disqualified* if by attaching bID from *Boats*, we obtain a <sID, bID> tuple that is not in *Reserves*:

  $$\pi_{sID,\, bID}\,(Reserves) \div \pi_{bID}\,(Boats)$$

  $$= \pi_{sID}(\boldsymbol{Reserves}) - \pi_{sID}\,((\pi_{sID}(\boldsymbol{Reserves}) \times \pi_{bID}(\boldsymbol{Boats})) - \pi_{sID,bID}(\boldsymbol{Reserves}))$$

Example: Division (3)

sno	pno
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

R

pno
p2

S1

pno
p2
p4

S2

pno
p1
p2
p4

S3

sno
s1
s2
s3
s4

R÷S1

sno
s1
s4

R÷S2

sno
s1

R÷S3

Example: Division (4)

- Relations *R, S*:

A	B
α	1
α	2
α	3
β	1
β	2
γ	1
δ	1
δ	3
δ	4
$\in$	6
$\in$	1

R

B
1
2

S

Group by **A**,
find the groups having all ***B***

- *R* ÷ *S*:

A
α
β

Example: Division (5)

- 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

Group by **ABC**,
find the grps having all ***DE***

D	***E***
a	***1***
b	***1***

S

- *R* ÷ *S*:

A	**B**	**C**
α	a	γ
γ	a	γ

Examples: Division (3-5)

- E.g., Consider *R* (*sno*, ***pno***) & *S* (***pno***).
  - Schema of *T* = *R* ÷ *S* is (**sno**)
  - $T = R \div S = \pi_{sno}(R) - \pi_{sno}((\pi_{sno}(R) \times S) - R)$
- E.g., Consider *R* (*A*, ***B***) & *S* (***B***).
  - Schema of *T* = *R* ÷ *S* is (**A**)
  - $T = R \div S = \pi_{A}(R) - \pi_{A}((\pi_{A}(R) \times S) - R)$
- E.g., Consider *R* (*A*, *B*, *C*, ***D*, *E***) & *S* (***D*, *E***).
  - Schema of *T* = *R* ÷ *S* is (**A, B, C**)
  - $T = R \div S = \pi_{A,B,C}(R) - \pi_{A,B,C}((\pi_{A,B,C}(R) \times S) - R)$

More Examples

- Consider the following 3 relations:
  - *Sailors* (*sID, sName, rating, age*)
  - *Boats* (*bID, bName, color*)
  - *Reserves* (*sID, bID, rDate*)
- *S1* is an instance of *Sailor*, *R1* is an instance of *Reserves*
- Retrieve rows about sailors with rating > 8:

$$\sigma_{rating>8}\ (S1)$$

- Find all sailor names & ratings:

$$\pi_{sName,\ rating}\ (S1)$$

- Find only the ages of sailors:

$$\pi_{age}\ (S1)$$

COMP 3380 (Fall 2006), Leung 37

More Examples

- Compute the names & ratings of 8^{+}-rated sailors:

$$\pi_{sName,\ rating}\ (\sigma_{rating \geq 8}\ (S1))$$

- Suppose there is another instance of *Sailor* (say, *S2*)
- Retrieve rows about sailors in either *S1* or *S2*:

$$S1 \cup S2$$

- Retrieve rows about sailors in both *S1* and *S2*:

$$S1 \cap S2$$

- Retrieve rows about sailors in *S1* but not in *S2*:

$$S1 - S2$$

COMP 3380 (Fall 2006), Leung 38

More Examples

- Pair each tuple of sailors with each tuple of reserves:

$$S1 \times R1$$

- Find all sailors who have reserved boats:

$$S1 \bowtie R1$$

- Find names of sailors who have reserved boat #103:

$$\pi_{sName}\,(\sigma_{bID=103}\,(S1 \bowtie R1))$$

- Find names of sailors who have reserved a red boat:

$$\pi_{sName}\,(\sigma_{color=\text{'red'}}\,(B1) \bowtie R1 \bowtie S1)$$

COMP 3380 (Fall 2006), Leung 39

More Examples

- Find names of sailors who have reserved a red or a green boat:

$$\pi_{sName}\,(\sigma_{color=\text{'red'} \vee color=\text{'green'}}\,(B1) \bowtie R1 \bowtie S1)$$

or

$$\pi_{sName}\,(\pi_{sID}\,([\sigma_{color=\text{'red'}}\,(B1) \cup \sigma_{color=\text{'green'}}\,(B1)] \bowtie R1) \bowtie S1)$$

or

$$\pi_{sName}\,(S1 \bowtie [\pi_{sID}\,(\sigma_{color=\text{'red'}}\,(B1) \bowtie R1) \cup \pi_{sID}\,(\sigma_{color=\text{'green'}}\,(B1) \bowtie R1)])$$

COMP 3380 (Fall 2006), Leung 40

More Examples

- Find names of sailors who have reserved a red and a green boat:

$$\pi_{sName}(S1 \bowtie [\pi_{sID}(\sigma_{color='red'}(B1) \bowtie R1) \cap \pi_{sID}(\sigma_{color='green'}(B1) \bowtie R1)])$$

How about the following?

$$\pi_{sName}(\sigma_{color='red' \wedge color='green'}(B1) \bowtie R1 \bowtie S1)$$

How about the following?

$$\pi_{sName}(\pi_{sID}[\sigma_{color='red'}(B1) \cap \sigma_{color='green'}(B1)] \bowtie R1 \bowtie S1)$$

COMP 3380 (Fall 2006), Leung 41

More Examples

- Find names of sailors who have reserved a red but not a green boat:

$$\pi_{sName}(S1 \bowtie [\pi_{sID}(\sigma_{color='red'}(B1) \bowtie R1) - \pi_{sID}(\sigma_{color='green'}(B1) \bowtie R1)])$$

How about the following?

$$\pi_{sName}(\sigma_{color='red' \wedge color\neq'green'}(B1) \bowtie R1 \bowtie S1)$$

How about the following?

$$\pi_{sName}(\pi_{sID}[\sigma_{color='red'}(B1) - \sigma_{color='green'}(B1)] \bowtie R1 \bowtie S1)$$

COMP 3380 (Fall 2006), Leung 42

More Examples

- Find IDs of sailors who have reserved all boats:

$$\pi_{sID,\ bID}\,(R1) \div \pi_{bID}\,(B1)$$

- Find names of sailors who have reserved all boats:

$$\pi_{sName}\ (S1 \bowtie (\pi_{sID,\ bID}\,(R1) \div \pi_{bID}\,(B1)))$$

Modification of the DB

- The content of the DB may be modified using the following operations:
  - Insertion
  - Deletion
  - Update
- All these operations can be expressed using the **assignment operator** ($\leftarrow$)

Insertion

- To insert data into a relation, we either:
  - specify a tuple to be inserted, or
  - write a query whose result is a set of tuples to be inserted
- In RA, an insertion is expressed by:

$$\boldsymbol{R \leftarrow R \cup E}$$

  where R is a relation and E is a RA expression.
- The insertion of a single tuple is expressed by letting E be a constant relation containing one tuple.

Example: Insertion

- Consider *Acct* (*acctNum, branchName, bal*) & *Depositor* (*custName, acctNum*).
- Insert information into the DB specifying that Smith has $1200 in account A973 at the Vancouver branch.

$$Acct \leftarrow Acct \cup \{(\text{'A973'}, \text{'Vancouver'}, 1200)\}$$

$$Depositor \leftarrow Depositor \cup \{(\text{'Smith'}, \text{'A973'})\}$$

Deletion

- A delete request is expressed similarly to a query, except instead of displaying tuples to the user, the selected tuples are removed from the database.
- Can delete only *whole tuples*; cannot delete values on only particular attributes
- A deletion is expressed in RA by:

$$\boldsymbol{R \leftarrow R - E}$$

where R is a relation and E is a RA query.

Example: Deletion

- Consider *Acct* (*acctNum, branchName, bal*) & *Loan* (*loanNum, branchName, amt*).
- Delete all account records in the Toronto branch

$$Acct \leftarrow Acct - \sigma_{branchName\,=\,\text{'Toronto'}}\,(Acct)$$

- Delete all loan records with amount in the range of \$0 to \$50

$$Loan \leftarrow Loan - \sigma_{amt \geq 0\ \wedge\ amt \leq 50}\,(Loan)$$

Update

- A mechanism to change a value in a tuple without charging *all* values in the tuple
- $\boldsymbol{R \leftarrow \pi_{F1,\, F2,\, \ldots,\, Fk} (R)}$
- Each F_i is either
  - the i-th attribute of R (if the i-th attribute is not updated), or
  - an expression, involving only constants and the attributes of R, which gives the new value for the attribute F_i (if the attribute F_i is to be updated)

Example: Update

- Consider *Acct* (*acctNum, branchName, bal*).
- Make interest payments by increasing all balances by 5%

$$Acct \leftarrow \pi_{acctNum,\ branchName,\ bal*1.05} (Acct)$$

- <u>Other Notation:</u> $\boldsymbol{\delta}_{bal \leftarrow bal*1.05} (Acct)$

Example: Update

- Consider *Acct* (*acctNum*, *branchName*, *bal*).
- Pay all accounts with balances over $10,000 a 6% interest and pay all others 5%

$$Acct \leftarrow \pi_{acctNum,\ branchName,\ bal*1.06}\ (\sigma_{bal>10000}\ (Acct))$$
$$\cup\ \pi_{acctNum,\ branchName,\ bal*1.05}\ (\sigma_{bal\leq 10000}\ (Acct))$$

- Other Notation:

$$\boldsymbol{\delta}_{bal\ \leftarrow\ bal*1.06}\ (\sigma_{bal>10000}\ (Acct))$$
$$\boldsymbol{\delta}_{bal\ \leftarrow\ bal*1.05}\ (\sigma_{bal\leq 10000}\ (Acct))$$

Summary

- The relational model has rigorously defined query languages that are simple and powerful.
- Relational algebra (RA) is more operational; useful as internal representation for query evaluation plans.
- *Several ways* of expressing a given query; a query optimizer should choose the most efficient version.