

CH4 RELATIONAL ALGEBRA AND CALCULUS

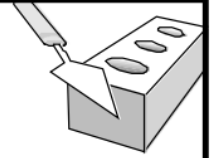
- CH4 RELATIONAL ALGEBRA AND CALCULUS
 - What is the foundation for relational query languages like SQL? What is the difference between procedural and declarative languages?
 - What is relational algebra, and why is it important?
 - What are the basic algebra operators, and how are they combined to write complex queries?
 - Selection and Projection
 - Set Operations
 - Join
 - Division
 - More Example
 - What is relational calculus, and why is it important?
 - Tuple Relational Calculus
 - Domain Relational Calculus

What is the foundation for relational query languages like SQL? What is the difference between procedural and declarative languages?

Query languages

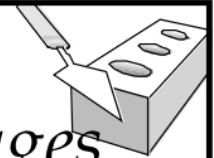
specialized languages for asking questions, or queries, that involve the data in a database.

Relational Query Languages



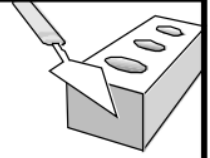
- ❖ Query languages: Allow manipulation and retrieval of data from a database.
- ❖ Relational model supports simple, powerful QLs:
 - Strong formal foundation based on logic.
 - Allows for much optimization.
- ❖ Query Languages != programming languages!
 - QLs not expected to be “Turing complete”.
 - 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: More operational, very useful for representing execution plans.
 - Relational Calculus: Lets users describe what they want, rather than how to compute it. (Non-operational, declarative.)

The inputs and outputs of a query are relations. A query is evaluated using *instances* of each input relation and it produces an instance of the output relation.



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 (but query will run regardless of instance!)
 - The schema for the *result* of a given query is also fixed! Determined by definition of query language constructs.
- ❖ Positional vs. named-field notation:
 - Positional notation easier for formal definitions, named-field notation more readable.
 - Both used in SQL

Example Instances

R1

<u>sid</u>	<u>bid</u>	<u>day</u>
22	101	10/10/96
58	103	11/12/96

❖ “Sailors” and “Reserves” relations for our examples.

❖ We’ll use positional or named field notation, assume that names of fields in query results are ‘inherited’ from names of fields in query input relations.

S1

<u>sid</u>	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S2

<u>sid</u>	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

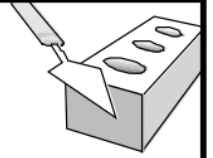
5

What is relational algebra, and why is it important?

Relational algebra is one of the two formal query languages associated with the relational model. Each relational query describes a step-by-step procedure for computing the desired answer, based on the order in which operators are applied in the query.

What are the basic algebra operators, and how are they combined to write complex queries?

Selection and Projection



Relational Algebra

❖ Basic operations:

- Selection (σ) Selects a subset of rows from relation.
- Projection (π) Deletes unwanted columns from relation.
- Cross-product (\times) Allows us to combine two relations.
- Set-difference ($-$) Tuples in reln. 1, but not in reln. 2.
- Union (\cup) Tuples in reln. 1 and in reln. 2.

❖ Additional operations:

- Intersection, join, division, renaming: Not essential, but (very!) useful.

❖ Since each operation returns a relation, operations can be *composed*! (Algebra is “closed”.)

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

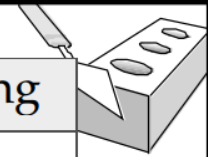
6

The projection operator π allows us to extract columns from a relation; for example, we can find out all

sailor names and ratings by using π .

Projection

- ❖ Deletes attributes that are not in *projection list*.
- ❖ *Schema* of result contains exactly the fields in the projection list, with the same names that they had in the (only) input relation.
- ❖ Projection operator has to eliminate *duplicates*! (Why??)
 - Note: real systems typically don't do duplicate elimination unless the user explicitly asks for it. (Why not?)



sname	rating
yuppy	9
lubber	8
guppy	5
rusty	10

$\pi_{sname, rating}(S2)$

age
35.0
55.5

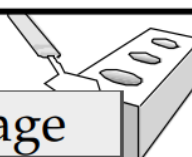
$\pi_{age}(S2)$

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

7

The important point to note is that, although three sailors are aged 35, a single tuple with age=35.0 appears in the result of the projection. This follows from the definition of a relation as a set of tuples. In practice, real systems often omit the expensive step of eliminating duplicate tuples, leading to relations that are multisets.

Selection



sid	sname	rating	age
28	yuppy	9	35.0
58	rusty	10	35.0

- ❖ Selects rows that satisfy *selection condition*.
- ❖ No duplicates in result! (Why?)
- ❖ *Schema* of result identical to schema of (only) input relation.
- ❖ *Result* relation can be the *input* for another relational algebra operation! (*Operator composition*.)

$$\sigma_{rating > 8}(S2)$$

sname	rating
yuppy	9
rusty	10

$$\pi_{sname, rating}(\sigma_{rating > 8}(S2))$$

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

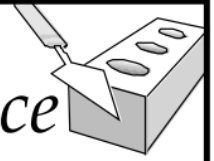
8

Set Operations

The following standard operations on sets are also available in relational algebra: \cup (union), \cap (intersection), $-$ (set-difference), and \times (cross-product).

- **Union:** $R \cup S$ returns a relation instance containing all tuples that occur in *either* relation instance R or relation instance S (or both). R and S must be *union-compatible*, and the schema of the result is defined to be identical to the schema of R .
Two relation instances are said to be union-compatible if the following conditions hold:
 1. they have the same number of the fields, and
 2. corresponding fields, taken in order from left to right, have the same *domains*.
 Note that field names are not used in defining union-compatibility. For convenience, we will assume that the fields of $R \cup S$ inherit names from R , if the fields of R have names. (This assumption is implicit in defining the schema of $R \cup S$ to be identical to the schema of R , as stated earlier.)
- **Intersection:** $R \cap S$ returns a relation instance containing all tuples that occur in both R and S . The relations R and S must be union-compatible, and the schema of the result is defined to be identical to the schema of R .
- **set-difference:** $R - S$ returns a relation instance containing all tuples that occur in R but not in S . The relations R and S must be union-compatible, and the schema of the result is defined to be identical to the schema of R .

Union, Intersection, Set-Difference



- ❖ All of these operations take two input relations, which must be union-compatible:

- Same number of fields.
- 'Corresponding' fields have the same type.

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0
44	guppy	5	35.0
28	yuppy	9	35.0

$S1 \cup S2$

- ❖ What is the *schema* of result?

sid	sname	rating	age
22	dustin	7	45.0

$S1 - S2$

sid	sname	rating	age
31	lubber	8	55.5
58	rusty	10	35.0

$S1 \cap S2$

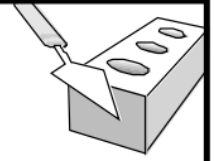
Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

9

- **Cross-product:** $R \times S$ returns a relation instance whose schema contains all the fields of R (in the same order as they appear in R) followed by all the fields of S (in the same order as they appear in S). The result of $R \times S$ contains one tuple r, s (the concatenation of tuples r and s) for each pair of tuples $r \in R, s \in S$. The cross-product operation is sometimes called **Cartesian product**.

We use the convention that the fields of $R \times S$ inherit names from the corresponding fields of R and S . It is possible for both R and S to contain one or more fields having the same name; this situation creates a naming conflict. The corresponding fields in $R \times S$ are unnamed and are referred to solely by position.

Cross-Product



- ❖ Each row of S1 is paired with each row of R1.
- ❖ *Result schema* has one field per field of S1 and R1, with field names 'inherited' if possible.
 - *Conflict*: Both S1 and R1 have a field called *sid*.

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

- Renaming operator: $\rho (C(1 \rightarrow sid1, 5 \rightarrow sid2), S1 \times R1)$

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

10

renaming operator ρ .

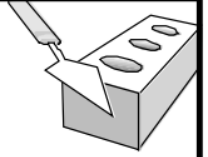
Join

The most general version of the join operation accepts a join condition c and a pair of relation instances as arguments and returns a relation instance. The *join condition* is identical to a *selection condition* in form. The operation is defined as follows:

$$\bowtie_c S = \sigma_c (R \times S)$$

Thus \bowtie is defined to be a cross-product followed by a selection. Note that the condition c can (and typically does) refer to attributes of both R and S . The reference to an attribute of a relation, say, R , can be by position (of the form $R.i$) or by name (of the form $R.name$).

Joins



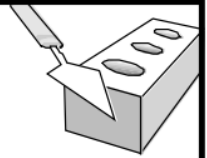
❖ Condition Join: $R \bowtie_c S = \sigma_c(R \times S)$

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	58	103	11/12/96

$$S1 \bowtie_{S1.sid < R1.sid} R1$$

- ❖ *Result schema* same as that of cross-product.
- ❖ Fewer tuples than cross-product, might be able to compute more efficiently
- ❖ Sometimes called a *theta-join*.

Joins



❖ Equi-Join: A special case of condition join where the condition c contains only *equalities*.

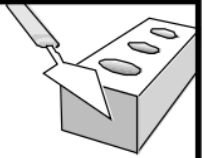
sid	sname	rating	age	bid	day
22	dustin	7	45.0	101	10/10/96
58	rusty	10	35.0	103	11/12/96

$$S1 \bowtie_{sid} R1$$

- ❖ *Result schema* similar to cross-product, but only one copy of fields for which equality is specified.
- ❖ Natural Join: Equijoin on *all* common fields.

Division

Division



- ❖ Not supported as a primitive operator, but useful for expressing queries like:

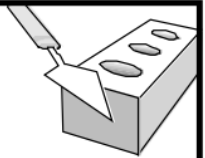
Find sailors who have reserved all boats.

- ❖ Let A have 2 fields, x and y ; B have only field y :
 - $A/B = \{ \langle x \rangle \mid \exists \langle x, y \rangle \in A \ \forall \langle y \rangle \in B \}$
 - i.e., A/B contains all x tuples (sailors) such that for every y tuple (boat) in B , there is an xy tuple in A .
 - Or: If the set of y values (boats) associated with an x value (sailor) in A contains all y values in B , the x value is in A/B .
- ❖ In general, x and y can be any lists of fields; y is the list of fields in B , and $x \cup y$ is the list of fields of A .

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

13

Examples of Division A/B



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

A

pno
p2

$B1$

sno
s1
s2
s3
s4

$A/B1$

pno
p2
p4

$B2$

sno
s1
s4

$A/B2$

pno
p1
p2
p4

$B3$

sno
s1

$A/B3$

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

14



Expressing A/B Using Basic Operators

- ❖ Division is not essential op; just a useful shorthand.
 - (Also true of joins, but joins are so common that systems implement joins specially.)
- ❖ *Idea:* For A/B , compute all x values that are not 'disqualified' by some y value in B .
 - x value is *disqualified* if by attaching y value from B , we obtain an xy tuple that is not in A .

Disqualified x values: $\pi_x ((\pi_x(A) \times B) - A)$

A/B : $\pi_x(A) -$ all disqualified tuples

More Example

<i>sid</i>	<i>sname</i>	<i>rating</i>	<i>age</i>
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Figure 4.15 An Instance *S3* of Sailors

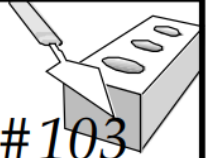
<i>sid</i>	<i>bid</i>	<i>day</i>
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Figure 4.16 An Instance *R2* of Reserves

<i>bid</i>	<i>bname</i>	<i>color</i>
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Figure 4.17 An Instance *B1* of Boats

Find names of sailors who've reserved boat #103



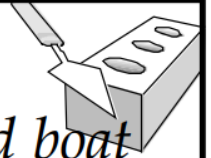
❖ Solution 1: $\pi_{sname}((\sigma_{bid=103} Reserves) \bowtie Sailors)$

❖ Solution 2: $\rho (Temp1, \sigma_{bid=103} Reserves)$

$\rho (Temp2, Temp1 \bowtie Sailors)$

$\pi_{sname}(Temp2)$

❖ Solution 3: $\pi_{sname}(\sigma_{bid=103}(Reserves \bowtie Sailors))$



Find names of sailors who've reserved a red boat

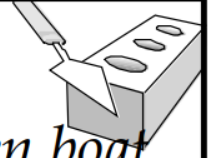
- ❖ Information about boat color only available in Boats; so need an extra join:

$$\pi_{sname}((\sigma_{color='red'}Boats) \bowtie Reserves \bowtie Sailors)$$

- ❖ A more efficient solution:

$$\pi_{sname}(\pi_{sid}((\pi_{bid} \sigma_{color='red'}Boats) \bowtie Res) \bowtie Sailors)$$

A query optimizer can find this, given the first solution!

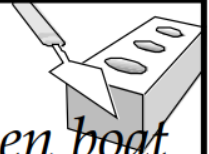


Find sailors who've reserved a red or a green boat

- ❖ Can identify all red or green boats, then find sailors who've reserved one of these boats:

$$\rho \text{ (Tempboats, } (\sigma_{color='red' \vee color='green'} \text{Boats}))$$
$$\pi_{sname}(\text{Tempboats} \bowtie \text{Reserves} \bowtie \text{Sailors})$$

- ❖ Can also define Tempboats using union! (How?)
- ❖ What happens if \vee is replaced by \wedge in this query?



Find sailors who've reserved a red and a green boat

- ❖ Previous approach won't work! Must identify sailors who've reserved red boats, sailors who've reserved green boats, then find the intersection (note that *sid* is a key for Sailors):

$$\rho (Tempred, \pi_{sid}((\sigma_{color='red'} Boats) \bowtie Reserves))$$

$$\rho (Tempgreen, \pi_{sid}((\sigma_{color='green'} Boats) \bowtie Reserves))$$

$$\pi_{sname}((Tempred \cap Tempgreen) \bowtie Sailors)$$



Find the names of sailors who've reserved all boats

- ❖ Uses division; schemas of the input relations to / must be carefully chosen:

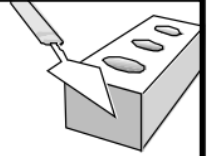
$$\rho (Tempsids, (\pi_{sid,bid} Reserves) / (\pi_{bid} Boats))$$

$$\pi_{sname} (Tempsids \sqcap Sailors)$$

- ❖ To find sailors who've reserved all 'Interlake' boats:

$$..... / \pi_{bid} (\sigma_{bname='Interlake'} Boats)$$

Summary

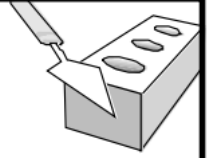


- ❖ The relational model has rigorously defined query languages that are simple and powerful.
- ❖ Relational algebra is more operational; useful as internal representation for query evaluation plans.
- ❖ Several ways of expressing a given query; a query optimizer should choose the most efficient version.

What is relational calculus, and why is it important?

Tuple Relational Calculus

Relational Calculus



- ❖ Comes in two flavors: *Tuple relational calculus* (TRC) and *Domain relational calculus* (DRC).
- ❖ Calculus has *variables, constants, comparison ops, logical connectives* and *quantifiers*.
 - TRC: Variables range over (i.e., get bound to) *tuples*.
 - DRC: Variables range over *domain elements* (= field values).
 - Both TRC and DRC are simple subsets of first-order logic.
- ❖ Expressions in the calculus are called *formulas*. An answer tuple is essentially an assignment of constants to variables that make the formula evaluate to *true*.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

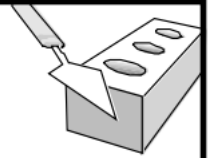
2

A tuple relational calculus query has the form $\{ T \mid p(T) \}$, where T is a tuple variable and $p(T)$ denotes a formula that describes T . (Q11) Find all sailors with a rating above 7. $\{ S \mid S \in \text{Sailors} \wedge S.\text{rating} > 7 \}$

Domain Relational Calculus

domain variable

a variable that ranges over the values in the domain of some attribute



Domain Relational Calculus

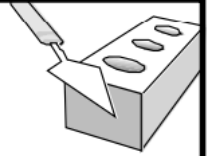
❖ *Query* has the form:

$$\{\langle x_1, x_2, \dots, x_n \rangle \mid p(\langle x_1, x_2, \dots, x_n \rangle)\}$$

❖ *Answer* includes all tuples $\langle x_1, x_2, \dots, x_n \rangle$ that make the formula $p(\langle x_1, x_2, \dots, x_n \rangle)$ be true.

❖ Formula is recursively defined, starting with simple *atomic formulas* (getting tuples from relations or making comparisons of values), and building bigger and better formulas using the *logical connectives*.

DRC Formulas



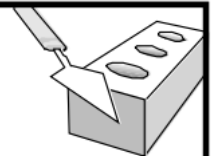
❖ Atomic formula:

- $\langle x_1, x_2, \dots, x_n \rangle \in Rname$, or $X \text{ op } Y$, or $X \text{ op } \text{constant}$
- op is one of $<, >, =, \leq, \geq, \neq$

❖ Formula:

- an atomic formula, or
- $\neg p, p \wedge q, p \vee q$, where p and q are formulas, or
- $\exists X(p(X))$, where variable X is *free* in $p(X)$, or
- $\forall X(p(X))$, where variable X is *free* in $p(X)$
- ❖ The use of quantifiers $\exists X$ and $\forall X$ is said to bind X .
 - A variable that is not bound is free.

Free and Bound Variables



- ❖ The use of quantifiers $\exists X$ and $\forall X$ in a formula is said to bind X .
 - A variable that is not bound is free.
- ❖ Let us revisit the definition of a query:

$$\left\{ \langle x_1, x_2, \dots, x_n \rangle \mid p(\langle x_1, x_2, \dots, x_n \rangle) \right\}$$

- ❖ There is an important restriction: the variables x_1, \dots, x_n that appear to the left of \mid must be the *only* free variables in the formula $p(\dots)$.

Find all sailors with a rating above 7



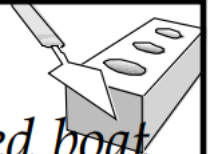
$$\{\langle I, N, T, A \rangle \mid \langle I, N, T, A \rangle \in \text{Sailors} \wedge T > 7\}$$

- ❖ The condition $\langle I, N, T, A \rangle \in \text{Sailors}$ ensures that the domain variables I , N , T and A are bound to fields of the same *Sailors* tuple.
- ❖ The term $\langle I, N, T, A \rangle$ to the left of ' \mid ' (which should be read as *such that*) says that every tuple $\langle I, N, T, A \rangle$ that satisfies $T > 7$ is in the answer.
- ❖ Modify this query to answer:
 - Find sailors who are older than 18 or have a rating under 9, and are called 'Joe'.

Find sailors rated > 7 who've reserved boat #103

$$\{ \langle I, N, T, A \rangle \mid \langle I, N, T, A \rangle \in \text{Sailors} \wedge T > 7 \wedge \\ \exists Ir, Br, D \left(\langle Ir, Br, D \rangle \in \text{Reserves} \wedge Ir = I \wedge Br = 103 \right) \}$$

- ❖ We have used $\exists Ir, Br, D (\dots)$ as a shorthand for $\exists Ir (\exists Br (\exists D (\dots)))$
- ❖ Note the use of \exists to find a tuple in Reserves that 'joins with' the Sailors tuple under consideration.

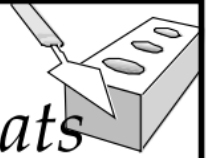


Find sailors rated > 7 who've reserved a red boat

$$\left\{ \langle I, N, T, A \rangle \mid \langle I, N, T, A \rangle \in \text{Sailors} \wedge T > 7 \wedge \right. \\ \left. \exists Ir, Br, D \left(\langle Ir, Br, D \rangle \in \text{Reserves} \wedge Ir = I \wedge \right. \right. \\ \left. \left. \exists B, BN, C \left(\langle B, BN, C \rangle \in \text{Boats} \wedge B = Br \wedge C = 'red' \right) \right) \right\}$$

- ❖ Observe how the parentheses control the scope of each quantifier's binding.
- ❖ This may look cumbersome, but with a good user interface, it is very intuitive. (MS Access, QBE)

Find sailors who've reserved all boats



$$\begin{aligned} & \{ \langle I, N, T, A \rangle \mid \langle I, N, T, A \rangle \in \text{Sailors} \wedge \\ & \quad \forall B, BN, C \left(\neg \langle B, BN, C \rangle \in \text{Boats} \right) \vee \\ & \quad \left(\exists Ir, Br, D \left(\langle Ir, Br, D \rangle \in \text{Reserves} \wedge I = Ir \wedge Br = B \right) \right) \} \end{aligned}$$

- ❖ Find all sailors I such that for each 3-tuple $\langle B, BN, C \rangle$ either it is not a tuple in Boats or there is a tuple in Reserves showing that sailor I has reserved it.

This query can be read as follows: "Find all values of N such that some tuple $\langle I, N, T, A \rangle$ in Sailors satisfies the following condition: For every B, BN, C , either this is not a tuple in Boats or there is some tuple Ir, Br, D in Reserves that proves that Sailor I has reserved boat B ." The \forall quantifier allows the domain variables B, BN , and C to range over all values in their respective attribute domains, and the pattern $\neg \langle B, BN, C \rangle \in$

Boats) \forall is necessary to restrict attention to those values that appear in tuples of Boats.

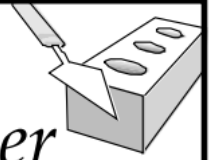
Find sailors who've reserved all boats (again!)

$$\{ \langle I, N, T, A \rangle \mid \langle I, N, T, A \rangle \in \text{Sailors} \wedge \\ \forall \langle B, BN, C \rangle \in \text{Boats} \\ \left(\exists \langle Ir, Br, D \rangle \in \text{Reserves} (I = Ir \wedge Br = B) \right) \}$$

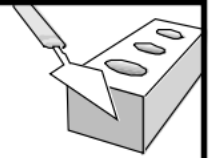
- ❖ Simpler notation, same query. (Much clearer!)
- ❖ To find sailors who've reserved all red boats:

$$\dots \left\{ C \neq \text{'red'} \vee \exists \langle Ir, Br, D \rangle \in \text{Reserves} (I = Ir \wedge Br = B) \right\}$$

Unsafe Queries, Expressive Power



- ❖ It is possible to write syntactically correct calculus queries that have an infinite number of answers! Such queries are called unsafe.
 - e.g., $\{S \mid \neg (S \in \text{Sailors})\}$
- ❖ It is known that every query that can be expressed in relational algebra can be expressed as a safe query in DRC / TRC; the converse is also true.
- ❖ Relational Completeness: Query language (e.g., SQL) can express every query that is expressible in relational algebra/calculus.



Summary

- ❖ Relational calculus is non-operational, and users define queries in terms of what they want, not in terms of how to compute it. (Declarativeness.)
- ❖ Algebra and safe calculus have same expressive power, leading to the notion of relational completeness.