Relational Algebra and SQL

Chapter 6

Relational Query Languages

- Languages for describing queries on a relational database
- *Structured Query Language* (SQL)
  - Predominant application-level query language
  - Declarative
- *Relational Algebra*
  - Intermediate language used within DBMS
  - Procedural

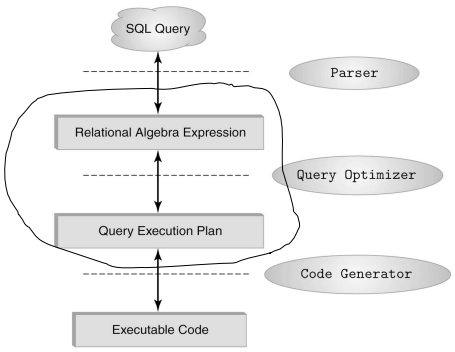
What is an Algebra?

- A language based on operators and a domain of values
- Operators map values taken from the domain into other domain values
- Hence, an expression involving operators and arguments produces a value in the domain
- When the domain is a set of all relations (and the operators are as described later), we get the *relational algebra*
- We refer to the expression as a *query* and the value produced as the *query result*

Relational Algebra

- *Domain*: set of relations
- *Basic operators*: select, project, union, set difference, Cartesian product
- *Derived operators*: set intersection, division, join
- *Procedural*: Relational expression specifies query by describing an algorithm (the sequence in which operators are applied) for determining the result of an expression

The Role of Relational Algebra in a DBMS



Select Operator

- Produce table containing subset of rows of argument table satisfying condition

  $\sigma_{condition}(relation)$
- Example:

Person

Id	*Name*	*Address*	*Hobby*
1123	John	123 Main	stamps
1123	John	123 Main	coins
5556	Mary	7 Lake Dr	hiking
9876	Bart	5 Pine St	stamps

$\sigma_{Hobby='stamps'}(\text{Person})$

Id	*Name*	*Address*	*Hobby*
1123	John	123 Main	stamps
9876	Bart	5 Pine St	stamps

Selection Condition

- Operators: $<, \leq, \geq, >, =, \neq$
- Simple selection condition:
  - <*attribute*> *operator* <*constant*>
  - <*attribute*> *operator* <*attribute*>
- <*condition*> AND <*condition*>
- <*condition*> OR <*condition*>
- NOT <*condition*>

7

Selection Condition - Examples

- $\sigma_{Id>3000 \text{ OR } Hobby=\text{'hiking'}}$ (Person)
- $\sigma_{Id>3000 \text{ AND } Id<3999}$ (Person)
- $\sigma_{\text{NOT}(Hobby=\text{'hiking'})}$ (Person)
- $\sigma_{Hobby\neq\text{'hiking'}}$ (Person)

8

Project Operator

- Produces table containing subset of columns of argument table

$\pi_{attribute\ list}(relation)$

- Example:

Person

Id	*Name*	*Address*	*Hobby*
1123	John	123 Main	stamps
1123	John	123 Main	coins
5556	Mary	7 Lake Dr	hiking
9876	Bart	5 Pine St	stamps

$\pi_{Name,Hobby}$(Person)

Name	*Hobby*
John	stamps
John	coins
Mary	hiking
Bart	stamps

9

Project Operator

- Example:

Person

Id	*Name*	*Address*	*Hobby*
1123	John	123 Main	stamps
1123	John	123 Main	coins
5556	Mary	7 Lake Dr	hiking
9876	Bart	5 Pine St	stamps

$\pi_{Name,Address}$(Person)

Name	*Address*
John	123 Main
Mary	7 Lake Dr
Bart	5 Pine St

Result is a table (no duplicates); can have fewer tuples than the original

10

Expressions

$\pi_{Id,\ Name}\ (\sigma_{Hobby=\text{'stamps' OR } Hobby=\text{'coins'}}\ (\text{Person})\ )$

Id	*Name*	*Address*	*Hobby*
1123	John	123 Main	stamps
1123	John	123 Main	coins
5556	Mary	7 Lake Dr	hiking
9876	Bart	5 Pine St	stamps

Person

Id	*Name*
1123	John
9876	Bart

Result

11

Set Operators

- Relation is a set of tuples, so set operations should apply: $\cap$, $\cup$, – (set difference)
- Result of combining two relations with a set operator is a relation => all its elements must be tuples having same structure
- Hence, scope of set operations limited to *union compatible relations*

12

Union Compatible Relations

- Two relations are *union compatible* if
  - Both have same number of columns
  - Names of attributes are the same in both
  - Attributes with the same name in both relations have the same domain
- Union compatible relations can be combined using *union*, *intersection*, and *set difference*

13

Example

Tables:

Person (*SSN, Name, Address, Hobby*)
Professor (*Id, Name, Office, Phone*)

are not union compatible. However

π_{Name} (Person) and π_{Name} (Professor)

are union compatible so

π_{Name} (Person) - π_{Name} (Professor)

makes sense.

14

Cartesian Product

- If R and S are two relations, $R \times S$ is the set of all concatenated tuples $<x,y>$, where x is a tuple in R and y is a tuple in S
  - (R and S need not be union compatible)
- $R \times S$ is expensive to compute:
  - Factor of two in the size of each row
  - Quadratic in the number of rows

a	*b*
x1	x2
x3	x4

R

c	*d*
y1	y2
y3	y4

S

a	*b*	*c*	*d*
x1	x2	y1	y2
x1	x2	y3	y4
x3	x4	y1	y2
x3	x4	y3	y4

$R \times S$

15

Renaming

- Result of expression evaluation is a relation
- Attributes of relation must have distinct names. This is not guaranteed with Cartesian product
  - e.g., suppose in previous example *a* and *c* have the same name
- Renaming operator tidies this up. To assign the names $A_1, A_2, \ldots A_n$ to the attributes of the *n* column relation produced by expression *expr* use

  expr $[A_1, A_2, \ldots A_n]$

16

Example

Transcript (*StudId, CrsCode, Semester, Grade*)
Teaching (*ProfId, CrsCode, Semester*)

$\pi_{StudId, CrsCode}$ (Transcript)[*StudId, CrsCode1*]
$\times \pi_{ProfId, CrsCode}$(Teaching) [*ProfId, Crscode2*]

This is a relation with 4 attributes:
StudId, CrsCode1, ProfId, CrsCode2

17

Derived Operation: Join

A (*general* or *theta*) *join* of R and S is the expression

$R \bowtie_{join\text{-}condition} S$

where *join-condition* is a *conjunction* of terms:

A_i *oper* B_i

in which A_i is an attribute of R; B_i is an attribute of S; and *oper* is one of $=, <, >, \geq \neq, \leq$.

The meaning is:

$\sigma_{join\text{-}condition'}(R \times S)$

where *join-condition* and *join-condition´* are the same, except for possible renamings of attributes (next)

18

Join and Renaming

- **Problem**: *R* and *S* might have attributes with the same name – in which case the Cartesian product is not defined
- **Solution**:
  - Rename attributes prior to forming the product and use new names in *join-condition´*.
  - Common attribute names are qualified with relation names in the result of the join

19

Theta Join – Example

Output the names of all employees that earn more than their managers.

$\pi_{\text{Employee.}Name}$ (Employee $\bowtie_{MngrId=Id \text{ AND } Salary>Salary}$ Manager)

The join yields a table with attributes:
Employee.*Name*, Employee.*Id*, Employee.*Salary*, *MngrId*
Manager.*Name*, Manager.*Id*, Manager.*Salary*

20

Equijoin Join - Example

Equijoin: Join condition is a conjunction of *equalities*.

$\pi_{Name,CrsCode}$(Student $\bowtie_{Id=StudId}$ $\sigma_{Grade='A'}$ (Transcript))

Student

Id	*Name*	*Addr*	*Status*
111	John	…..	…..
222	Mary	…..	…..
333	Bill	…..	…..
444	Joe	…..	…..

Transcript

StudId	*CrsCode*	*Sem*	*Grade*
111	CSE305	S00	B
222	CSE306	S99	A
333	CSE304	F99	A

Mary	CSE306
Bill	CSE304

The equijoin is used very frequently since it combines related data in different relations.

21

Natural Join

- Special case of equijoin:
  - join condition equates *all* and *only* those attributes with the same name (condition doesn't have to be explicitly stated)
  - duplicate columns eliminated from the result

Transcript (*StudId, CrsCode, Sem, Grade*)
Teaching (*ProfId, CrsCode, Sem*)

Transcript $\bowtie$ Teaching =

$\pi_{StudId,\ Transcript.CrsCode,\ Transcript.Sem,\ Grade,\ ProfId}$
(Transcript $\bowtie_{CrsCode=CrsCode \text{ AND } Sem=Sem}$ Teaching)
[*StudId, CrsCode, Sem, Grade, ProfId*]

22

Natural Join (con't)

- More generally:

$$R \bowtie S = \pi_{attr\text{-}list} (\sigma_{join\text{-}cond} (R \times S))$$

where
attr-list = *attributes* (*R*) ∪ *attributes* (*S*)
(duplicates are eliminated) and *join-cond* has the form:
$A_1 = A_1$ AND … AND $A_n = A_n$
where
$\{A_1 \ldots A_n\}$ = *attributes*(*R*) ∩ *attributes*(*S*)

23

Natural Join Example

- List all Id's of students who took at least two different courses:

π_{StudId} ($\sigma_{CrsCode \neq CrsCode2}$ (
Transcript $\bowtie$
Transcript [*StudId, CrsCode2, Sem2, Grade2*]))

(don't join on *CrsCode*, *Sem*, and *Grade* attributes)

24