Algebra

- Based on operators and a domain of values
- Operators map arguments from domain into another domain value
- Hence, an expression involving operators and arguments produces a value in the domain.
- We refer to the expression as a query and the value produced as the result of that query

Relational Algebra

- Domain: set of relations
- Basic operators:
  - Select (σ)
  - project (Π)
  - union (U)
  - set difference (-)
  - Cartesian product (X)
- 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

Select Operator

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

$$\sigma_{<condition>}(<relation>)$$

- Example: $\sigma_{Hobby='stamps'}(Person)$

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

Selection Condition

- Operators: <, ≤, ≥, >, =, ≠
- Simple selection condition:
  - <attribute> operator <constant>
  - <attribute> operator <attribute>
- <condition> AND <condition>
- <condition> OR <condition>
- NOT <condition>

Selection Condition - Examples

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

Project Operator

- Produce table containing subset of columns of argument table

  $\Pi_{\text{<attribute-list>}}(\text{<relation>})$

- Example: $\Pi_{Name,\ Hobby}(Person)$

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

Name	Hobby
John	stamps
John	coins
Mary	hiking
Bart	stamps

Project Operator

Example: $\Pi_{Name,\ Address}\ (Person)$

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

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

Result is a table (no duplicates)

Expressions

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

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

Set Operators

- Relation is a set of tuples => set operations should apply
- 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*

Union Compatible Relations

- Two relations are union compatible if
  - Both have same number of columns
  - Domains of the corresponding attributes in both relations must be compatible
- Union compatible relations can be combined using union, intersection, and set difference

Example

Tables:

Person (SSN, Name, Address, Hobby)

Professor (Id, Name, Office, Phone)

are not union compatible. However

$\Pi_{Name}(Person)$ and $\Pi_{Name}(Professor)$

are union compatible and

$\Pi_{Name}(Person) - \Pi_{Name}(Professor)$

makes sense.

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

$R \times S$

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 $a = c$
- 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* use *expression* $[A_1, A_2, \ldots A_n]$

Example

Transcript (StudId, CrsCode, Semester, Grade)

Teaching (ProfId, CrsCode, Semester)

$$\Pi_{StudId,\ CrsCode}\ (Transcript)[StudId,\ SCrsCode] \times \Pi_{ProfId,\ CrsCode}(Teaching)\ [ProfId,\ PCrscode]$$

This is a relation with 4 attributes:
StudId, SCrsCode, ProfId, PCrsCode

Derived Operation: Join

The expression :

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

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 =, <, >, ≥ ≠, ≤, is referred to as the (theta) join of *R* and *S* and denoted:

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

Where *join-condition* and *join-condition ´* are (roughly) the same …

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

Theta Join – Example

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

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

The join yields a table with attributes:

Employee.Name, Employee.Id, Employee.Salary, MngrId
Manager.Name, Manager.Id, Manager.Salary

Equijoin Join - Example

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

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 commonly used since it combines related data in different relations.

Equijoin Join - Example

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

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

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 commonly used since it combines related data in different relations.

Climbers (C2):

CId	CName	Skill	Age
123	Edmund	EXP	80
214	Arnold	BEG	25
313	Bridget	EXP	33
212	James	MED	27

Climbs (C1):

CId	RId	Date	Duration
123	1	10/10/88	5
123	3	11/08/87	1
313	1	12/08/89	5
214	2	08/07/92	2
313	1	06/07/94	3

$$\sigma_{CId:1=CId:2}(\text{Climbs} \times \text{Climbers}):$$

C1.CId	**RId**	**Date**	**Duration**	**C2.CId**	**CName**	**Skill**	**Age**
123	1	10/10/88	5	123	Edmund	EXP	80
123	3	11/08/87	1	123	Edmund	EXP	80
313	1	12/08/89	5	313	Bridget	EXP	33
214	2	08/07/92	2	214	Arnold	BEG	25
313	1	06/07/94	3	313	Bridget	EXP	33

Natural JOIN (*)

- In an EQUIJOIN $R \leftarrow R_1 \infty_c R_2$, the join attribute of R_2 appear *redundantly* in the result relation R.
- In a NATURAL JOIN, the *redundant join attributes* of R_2 are *eliminated* from R. The equality condition is *implied* and need not be specified.

Climbs∞Climbers :

CId	RId	Date	Duration	CName	Skill	Age
123	1	10/10/88	5	Edmund	EXP	80
123	3	11/08/87	1	Edmund	EXP	80
313	1	12/08/89	5	Bridget	EXP	33
214	2	08/07/92	2	Arnold	BEG	25
313	1	06/07/94	3	Bridget	EXP	33

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)

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

Natural Join (con't)

- More generally:

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

where

attr-list = attributes (R) $\cup$ *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)* $\cap$ *attributes(S)*

Natural Join Example

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

Natural Join Example

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

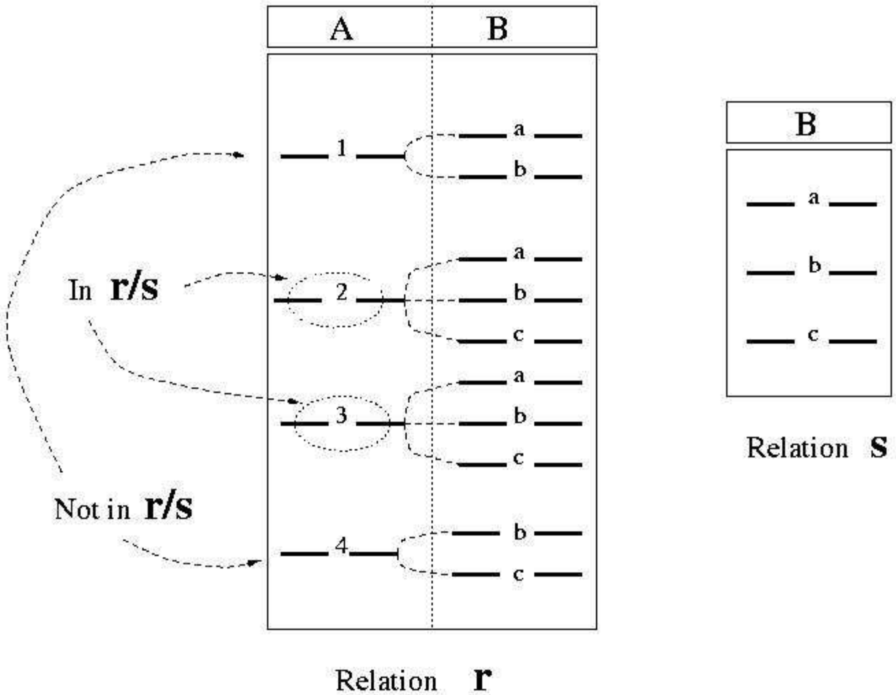
$$\Pi_{StudId} \,(\, \sigma_{CrsCode \neq CrsCode2} \,($$
$$Transcript \quad *$$
$$Transcript\ [StudId,\ CrsCode2,\ Sem2,\ Grade2]))$$

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

Division

- Goal: Produce the tuples in one relation, r, that match *all* tuples in another relation, s
  - $r\ (A_1, \ldots A_n, B_1, \ldots B_m)$
  - $s\ (B_1 \ldots B_m)$
  - r/s, with attributes $A_1, \ldots A_n$, is the set of all tuples $<a>$ such that for every tuple $<b>$ in s, $<a,b>$ is in r
- Can be expressed in terms of projection, set difference, and cross-product

Division (con't)



Division - Example

- List the Ids of students who have passed *all* courses that were taught in spring 2000

Division - Example

- List the Ids of students who have passed *all* courses that were taught in spring 2000
- *Numerator*: StudId and CrsCode for every course passed by every student
  - $\pi_{StudId,\ CrsCode}\ (\sigma_{Grade \neq \text{'F'}}\ (Transcript)\ )$
- *Denominator*: CrsCode of all courses taught in spring 2000
  - $\Pi_{CrsCode}\ (\sigma_{Semester=\text{'S2000'}}\ (Teaching)\ )$
- Result is *numerator/denominator*

Complete Set of Relational Algebra Operations

- the set {б, Π, U, - , X } is called a *complete set* of relational algebra operations. Any query language *equivalent to* these operations is called **relationally complete**.
- All the basic operations discussed so far can be described as a sequence of *only* the above set
- Additional operations were not part of the *original* relational algebra
  - Aggregate functions (SUM, AVG), grouping
  - Outer Join, outer union

Additional Relational Operations

- Aggregate Functions and Grouping
- Outer JOIN and Outer UNION

Aggregate functions

- Functions are often applied to sets of values or sets of tuples in DB applications
  - SUM, COUNT, AVERAGE, MIN, MAX
  - $_{\text{<grouping attributes>}}\,\boldsymbol{F}\,_{\text{<function list>}}$ (R)
  - Grouping attributes are optional
- Example:

  List the average salary of **all** employees (no grouping needed):

  For each department, retrieve the department number, the number of employees, and the average salary:

Aggregate functions

- Functions are often applied to sets of values or sets of tuples in DB applications
  - SUM, COUNT, AVERAGE, MIN, MAX
  - $_{\text{<grouping attributes>}}\,\boldsymbol{F}\,_{\text{<function list>}}$ (R)
  - Grouping attributes are optional
- Example:

  List the average salary of **all** employees (no grouping needed):

  $R(AVGSAL) \leftarrow \boldsymbol{F}_{\text{AVG SALARY}}$ (EMPLOYEE)

  For each department, retrieve the department number, the number of employees, and the average salary:

  $R(DNO,NUMEMPS,AVGSAL) \leftarrow$

  $_{\text{DNO}}\,\boldsymbol{F}\,_{\text{COUNT SSN, AVERAGE SALARY}}$ (EMPLOYEE)

  - DNO is called the ***grouping attribute***

Figure 7.16 An illustration of the AGGREGATE FUNCTION operation. (a) $R(\text{DNO, NO\_OF\_EMPLOYEES, AVERAGE\_SAL}) \leftarrow {}_{\text{DNO}}\mathfrak{F}_{\text{COUNT SSN,AVERAGE SALARY}}(\text{EMPLOYEE})$. (b) ${}_{\text{DNO}}\mathfrak{F}_{\text{COUNT SSN,AVERAGE SALARY}}(\text{EMPLOYEE})$. (c) $\mathfrak{F}_{\text{COUNT SSN,AVERAGE SALARY}}(\text{EMPLOYEE})$.

(a)

R	DNO	NO_OF_EMPLOYEES	AVERAGE_SAL
	5	4	33250
	4	3	31000
	1	1	55000

(b)

DNO	COUNT_SSN	AVERAGE_SALARY
5	4	33250
4	3	31000
1	1	55000

(c)

COUNT_SSN	AVERAGE_SALARY
8	35125

Outer join

- In a regular EQUIJOIN or NATURAL JOIN operation, tuples in R1 or R2 that do not have matching tuples in the other relation *do not appear in the result*
- Tuples with null in the join attributes are also eliminated.
- Some queries require all tuples in R1 (or R2 or both) to appear in the result
- When no matching tuples are found, **null**s are placed for the missing attributes

Outer join (Cont.)

- **LEFT OUTER JOIN**:
  - R1 left∞ R2 lets every tuple in R1 appear in the result
- **RIGHT OUTER JOIN**:
  - R1 ∞right R2 lets every tuple in R2 appear in the result
- **FULL OUTER JOIN**:
  - R1 left∞right R2 lets every tuple in R1 or R2 appear in the result

Figure 7.18 The LEFT OUTER JOIN operation.

RESULT	FNAME	MINIT	LNAME	DNAME
	John	B	Smith	null
	Franklin	T	Wong	Research
	Alicia	J	Zelaya	null
	Jennifer	S	Wallace	Administration
	Ramesh	K	Narayan	null
	Joyce	A	English	null
	Ahmad	V	Jabbar	null
	James	E	Borg	Headquarters

Outer UNION

- If two relations are **not union compatible**
- Partial compatible
- example