CSIE30600/CSIEB0290

Database Systems

Lecture 4: Relational Algebra and Calculus

Outline

- Relational Algebra
  - Unary Relational Operations
  - Relational Algebra Operations From Set Theory
  - Binary Relational Operations
  - Additional Relational Operations
  - Examples of Queries in Relational Algebra
- Relational Calculus*
  - Tuple Relational Calculus
  - Domain Relational Calculus
- Example Database Application (COMPANY)
- Overview of the QBE language(based on relational calculus)*

Relational Query Languages

- *Query languages: Allow manipulation and retrieval* of data from a database.
- Relational model supports simple, powerful QLs:
  - Simple data structure – sets!
  - Easy to understand, easy to manipulate
  - 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 Langs

- 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. (Nonoperational, *declarative.)*

Basics of Relational Algebra

- An algebra consists of operators and atomic operands
- Expressions can be constructed by applying operators to atomic operands and/or other expressions
  - Operations can be composed -- **algebra is closed**
  - Parentheses are needed to group operators
- Algebra of arithmetic: operands are variables and constants, and operators are the usual arithmetic operators
  - E.g., (x+y)*2 or ((x+7)/(y-3)) + x
- Relational algebra: operands are variables that stand for relations (sets of tuples), and operations include *union, intersection, selection, projection, Cartesian product, etc*
  - E.g., ($\pi_{c\text{-}owner}$CheckingAccount) ∩ ($\pi_{s\text{-}owner}$SavingsAccount)

Relational Algebra Overview

- Relational algebra is the basic set of operations for the relational model
- These operations enable a user to specify **basic retrieval requests** (or **queries**)
- The result of an operation is a new relation, which may have been formed from one or more input relations
  - This property makes the algebra "closed" (all objects in relational algebra are relations)

Relational Algebra Overview (cont.)

- The algebra operations thus produce new relations
  - These can be further manipulated using operations of the same algebra
- A sequence of relational algebra operations forms a relational algebra **expression**
  - The result of a relational algebra expression is also a relation that represents the result of a database query (or retrieval request)

Relational Algebra Overview

- Relational Algebra consists of several groups of operations
  - Unary operations
    - SELECT (symbol: σ (sigma))
    - PROJECT (symbol: π (pi))
    - RENAME (symbol: ρ (rho))
  - Operations from Set Theory
    - UNION ($\cup$), INTERSECTION ($\cap$), DIFFERENCE (or MINUS, –)
    - CARTESIAN PRODUCT ($\times$)

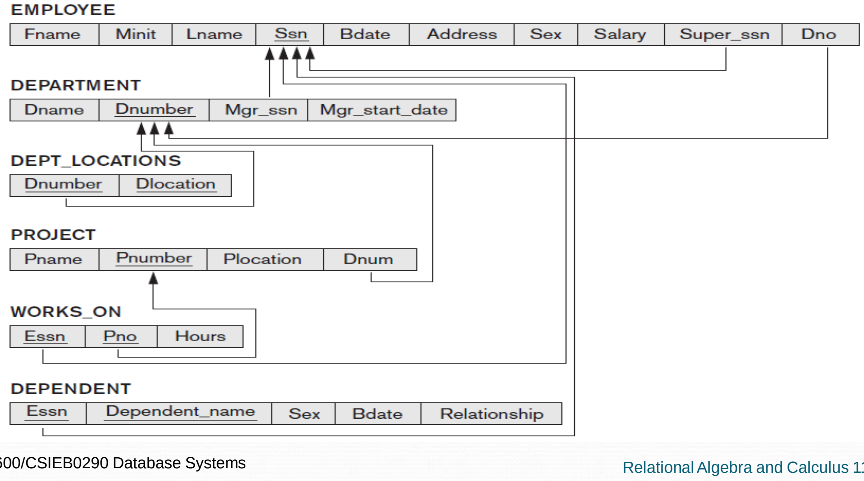
Relational Algebra Overview

- Binary operations
  - JOIN (several variations of JOIN exist)
  - DIVISION
- Additional operations
  - OUTER JOINS, OUTER UNION
  - AGGREGATE FUNCTIONS (These compute summary of information: for example, SUM, COUNT, AVG, MIN, MAX)

Database Schema for COMPANY

- Many examples discussed below refer to the COMPANY database shown here.

Figure 5.7
Referential integrity constraints displayed on the COMPANY relational database schema.



Example: database state

Figure 5.6
One possible database state for the COMPANY relational database schema.

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

CSIE30600/CSIEB0290 Database Systems

Relational Algebra and Calculus 12

Example: database state

Figure 5.6
One possible database state for the COMPANY relational database schema.

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

CSIE30600/CSIEB0290 Database Systems

Relational Algebra and Calculus 13

Select Operation(σ) – Example

Relations *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

Unary Relational Operations: SELECT

- The SELECT operation (denoted by σ (sigma)) is used to select the tuples from a relation that satisfies a **selection condition**.
- Examples:
  - Select the EMPLOYEE tuples whose department number is 4:

    $$\sigma_{\text{DNO} = 4}(\text{EMPLOYEE})$$

  - Select the employee tuples whose salary is greater than $30,000:

    $$\sigma_{\text{SALARY} > 30{,}000}(\text{EMPLOYEE})$$

SELECT

- In general, the select operation is denoted by

$$\sigma_{\text{<selection condition>}}(R)$$

where
- the symbol σ (sigma) is the *select* operator
- the selection condition is a Boolean (conditional) expression specified on the attributes of relation R
- Selection condition contains clauses of the form

  <attribute name> <comparison op> <constant value>

  or

  <attribute name> <comparison op> <attribute name>
- Clauses can be combined with AND, OR, and NOT
- tuples that make the condition **true** are selected
- tuples that make the condition **false** are filtered out

SELECT: Formal Definition

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

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

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

Each term is one of:

<attribute> *op* <attribute> or <constant>

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

- Example of selection:

$$\sigma_{Dname=\text{"Research"}}(DEPARTMENT)$$

SELECT: Properties

- The SELECT operation $\sigma_{<\text{selection condition}>}(R)$ produces a relation S that has the same schema (same attributes) as R
- SELECT σ is commutative:
  - $\sigma_{<c1>}(\sigma_{<c2>}(R)) = \sigma_{<c2>}(\sigma_{<c1>}(R))$
- Because of commutativity property, a cascade (sequence) of SELECT operations may be applied in any order:
  - $\sigma_{<c1>}(\sigma_{<c2>}(\sigma_{<c3>}(R))) = \sigma_{<c2>}(\sigma_{<c3>}(\sigma_{<c1>}(R)))$
- A cascade of SELECT operations may be replaced by a single selection with a conjunction of all the conditions:
  - $\sigma_{<c1>}(\sigma_{<c2>}(\sigma_{<c3>}(R))) = \sigma_{<c1>\ \text{AND}\ <c2>\ \text{AND}\ <c3>}(R)$
- #tuples in the result S ≤ #tuples in R
- Fraction of tuples selected by a selection condition is called the selectivity.

Project Operation(π) – Example

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

Unary Operation: PROJECT

- PROJECT Operation is denoted by π (pi)
- This operation keeps certain columns (attributes) from a relation and discards the other columns.
  - PROJECT creates a vertical partitioning
    - The list of specified attributes is kept in each tuple
    - The other attributes in each tuple are discarded
- Example: To list each employee's first and last name and salary, the following is used:

$$\pi_{\text{LNAME, FNAME, SALARY}}(\text{EMPLOYEE})$$

PROJECT Operations: (cont.)

- The general form of the *project* operation is:

$$\pi_{\text{<attribute list>}}(\text{R})$$

  - π (pi) is used to represent the *project* operation
  - <attribute list> is the desired list of attributes from relation R.
- The project operation removes any duplicate tuples
  - This is because the result of the project operation must be a set of tuples
  - Mathematical sets *do not allow* duplicate elements.

PROJECT: Formal Definition

- Notation: $\pi_{A_1, A_2, \ldots, A_k}(r)$

  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 keep only the *Pname* and *Pnumber* attributes of PROJECT

  $\pi_{Pname,\ Pnumber}(PROJECT)$

PROJECT: Properties

- Degree: #attributes in <attribute list>
- #tuples in the result ≤ #tuples in R
  - If the list of attributes includes a *key* of R, then the #tuples in the result of PROJECT = #tuples in R
- PROJECT is **not** commutative (Why?)
  - $\pi_{<list1>}(\pi_{<list2>}(R)) = \pi_{<list1>}(R)$ as long as <list1> ⊆ <list2>

Examples of SELECT and PROJECT

Figure 8.1
Results of SELECT and PROJECT operations. (a) $\sigma_{(\text{Dno}=4 \textbf{ AND } \text{Salary}>25000) \text{ OR } (\text{Dno}=5 \textbf{ AND } \text{Salary}>30000)}$(EMPLOYEE). (b) $\pi_{\text{Lname, Fname, Salary}}$(EMPLOYEE). (c) $\pi_{\text{Sex, Salary}}$(EMPLOYEE).

(a)

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5

(b)

Lname	Fname	Salary
Smith	John	30000
Wong	Franklin	40000
Zelaya	Alicia	25000
Wallace	Jennifer	43000
Narayan	Ramesh	38000
English	Joyce	25000
Jabbar	Ahmad	25000
Borg	James	55000

(c)

Sex	Salary
M	30000
M	40000
F	25000
F	43000
M	38000
M	25000
M	55000

Relational Algebra Expressions

- We may want to apply several relational algebra operations one after the other
  - Either we can write the operations as a single in-line expression by nesting the operations, or
  - We can write a sequence of operations through the creation of intermediate result relations by assignment($\leftarrow$).
- In the latter case, we must create intermediate results and give names to these relations.

Assignment (←)

- The assignment operation (←) provides a convenient way to express complex queries.
  - Write query as a series of assignments 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:

$$temp1 \leftarrow \pi_{R-S}(r)$$

$$temp2 \leftarrow \pi_{R-S}((temp1 \times s) - \pi_{R-S,S}(r))$$

$$result \leftarrow temp1 - temp2$$

- The result to the right of the ← is assigned to the relation variable on the left of the ←.
- May use the variable in subsequent expressions

In-line Expression vs Sequence of Operations (Examples)

- To retrieve the first name, last name, and salary of all employees who work in department number 5, we must apply a select and a project operation
- We can write a single in-line expression as follows:
  - $\pi_{\text{FNAME, LNAME, SALARY}}(\sigma_{\text{DNO}=5}(\text{EMPLOYEE}))$
- OR we can explicitly show the sequence of operations, giving a name to each intermediate relation:
  - $\boxed{\text{DEP5\_EMPS}} \leftarrow \sigma_{\text{DNO}=5}(\text{EMPLOYEE})$
  - $\text{RESULT} \leftarrow \pi_{\text{FNAME, LNAME, SALARY}}(\boxed{\text{DEP5\_EMPS}})$

Unary Operations: RENAME(ρ)

- The RENAME operator is denoted by ρ (rho)
- In some cases, we may want to *rename* the attributes of a relation or the relation name or both
  - Useful when a query requires multiple operations
  - Necessary in some cases (see JOIN operation later)

RENAME Operation

- The general RENAME operation ρ can be expressed by any of the following forms:
  - $\rho_{S(B1, B2, \ldots, Bn)}(R)$ changes both:
    - the relation name to S, *and*
    - the column (attribute) names to B1, B1, ...Bn
  - $\rho_S(R)$ changes:
    - the *relation name* only to S
  - $\rho_{(B1, B2, \ldots, Bn)}(R)$ changes:
    - the *column (attribute) names* only to B1, B1, ...Bn

RENAME

- For convenience, we also use a *shorthand* for renaming attributes :
  - If we write:
    - RESULT $\leftarrow \pi_{\text{FNAME, LNAME, SALARY}}$(DEP5_EMPS)
    - RESULT will have the *same attribute names* as DEP5_EMPS (same attributes as EMPLOYEE)
  - If we write:
    - RESULT(F, M, L, S, B, A, SX, SAL, SU, DNO)$\leftarrow$ $\rho_{\text{RESULT(F.M.L.S.B,A,SX,SAL,SU, DNO)}}$(DEP5_EMPS)
    - The 10 attributes of DEP5_EMPS are *renamed* to F, M, L, S, B, A, SX, SAL, SU, DNO, respectively

Example: multiple operations and RENAME

(a)

Fname	Lname	Salary
John	Smith	30000
Franklin	Wong	40000
Ramesh	Narayan	38000
Joyce	English	25000

(b)
TEMP

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston,TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston,TX	M	40000	888665555	5
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble,TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5

R

First_name	Last_name	Salary
John	Smith	30000
Franklin	Wong	40000
Ramesh	Narayan	38000
Joyce	English	25000

Figure 8.2
Results of a sequence of operations. (a) $\pi_{\text{Fname, Lname, Salary}}(\sigma_{\text{Dno=5}}(\text{EMPLOYEE}))$.
(b) Using intermediate relations and renaming of attributes.

Union Operation(∪) – Example

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
α	2
β	1
β	3

Operations from Set Theory: UNION

- UNION Operation
  - Binary operation, denoted by ∪
  - The result of **R ∪ S**, is a relation that includes all tuples that are either in R or in S or in both
  - Duplicate tuples are eliminated
  - The two operand relations R and S must be "type compatible" (or UNION compatible)
    - R and S must have same #attributes (degree)
    - Each pair of corresponding attributes must be type compatible (have same or compatible domains)

UNION Operation

- Example:
  - To retrieve the social security numbers of all employees who either *work in department 5* (RESULT1 below) or *directly supervise an employee who works in department 5* (RESULT2 below)
  - We can use the UNION operation as follows:

$$DEP5\_EMPS \leftarrow \sigma_{DNO=5}(EMPLOYEE)$$
$$RESULT1 \leftarrow \pi_{SSN}(DEP5\_EMPS)$$
$$RESULT2(SSN) \leftarrow \pi_{SUPERSSN}(DEP5\_EMPS)$$
$$RESULT \leftarrow RESULT1 \cup RESULT2$$

  - The union operation produces the tuples that are in either RESULT1 or RESULT2 or both

Example of the result of a UNION operation

RESULT1

Ssn
123456789
333445555
666884444
453453453

RESULT2

Ssn
333445555
888665555

RESULT

Ssn
123456789
333445555
666884444
453453453
888665555

Figure 8.3
Result of the UNION operation RESULT ← RESULT1 ∪ RESULT2.

Union Operation – Formal Definition

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

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

- The result of $r \cup s$, is a relation that includes all tuples in r or in s or in both r and s
- Duplicate tuples are eliminated
- For $r \cup s$ to be valid.
  1. *r, s* must have the *same* **arity** (#attributes)
  2. The attribute domains must be **compatible** (corresponding columns must have same type of values)

Operations from Set Theory

- Type compatibility of operands is required for the binary set operation UNION $\cup$, (also for INTERSECTION $\cap$, and SET DIFFERENCE –, to be discussed later)
- R1(A1, A2, ..., An) and R2(B1, B2, ..., Bn) are type compatible if:
  - they have the same number of attributes, and
  - the domains of corresponding attributes are type compatible (i.e. dom(Ai)=dom(Bi) for i=1, 2, ..., n).
- The resulting relation for R1$\cup$R2 (also for R1$\cap$R2, or R1–R2) has the same attribute names as the **first** operand relation R1 (by convention)

Operations from Set Theory: INTERSECTION(∩)

- INTERSECTION is denoted by ∩
- The result of the operation **R** ∩ **S**, is a relation that includes all tuples that are in both R and S
  - The attribute names in the result will be the same as the attribute names in R
- The two operand relations R and S must be "type compatible"

Set Difference Operation – Example

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

Operations from Set Theory: SET DIFFERENCE(–)

- SET DIFFERENCE (also called MINUS or EXCEPT) is denoted by –
- The result of R – S, is a relation that includes all tuples that are in R but not in S
  - The attribute names in the result will be the same as the attribute names in R
- The two operand relations R and S must be “type compatible”

Set Difference – Formal Definition

- 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

Examples: UNION, INTERSECT, DIFFERENCE

Figure 8.4
The set operations UNION, INTERSECTION, and MINUS. (a) Two union-compatible relations. (b) STUDENT ∪ INSTRUCTOR. (c) STUDENT ∩ INSTRUCTOR. (d) STUDENT − INSTRUCTOR. (e) INSTRUCTOR − STUDENT.

(a) STUDENT

Fn	Ln
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

INSTRUCTOR

Fname	Lname
John	Smith
Ricardo	Browne
Susan	Yao
Francis	Johnson
Ramesh	Shah

(b)

Fn	Ln
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert
John	Smith
Ricardo	Browne
Francis	Johnson

(c)

Fn	Ln
Susan	Yao
Ramesh	Shah

(d)

Fn	Ln
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

(e)

Fname	Lname
John	Smith
Ricardo	Browne
Francis	Johnson

Properties of UNION, INTERSECT, DIFFERENCE

- Notice that both union and intersection are commutative operations; that is
  - $R \cup S = S \cup R$, and $R \cap S = S \cap R$
- Both union and intersection can be treated as n-ary operations applicable to any number of relations as both are associative operations; that is
  - $R \cup (S \cup T) = (R \cup S) \cup T$
  - $(R \cap S) \cap T = R \cap (S \cap T)$
- The difference operation is not commutative; that is, in general
  - $R - S \neq S - R$

Challenge Question

- How could you express the intersection operation if you didn't have an intersection operator in relational algebra? [Hint: Can you express Intersection using only the Difference operator?]
- A ∩ B = ???

Cartesian-Product Operation(×) – Example

Relations *r, s*

r

A	B
α	1
β	2

s

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

r x 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

CARTESIAN PRODUCT

- CARTESIAN(or CROSS) PRODUCT (x):
  - used to combine tuples from two relations
  - Denoted by R(A1, A2, . . ., An) x S(B1, B2, . . ., Bm)
  - Result is a relation Q with degree n + m attributes:
    - Q(A1, A2, . . ., An, B1, B2, . . ., Bm)
  - Q has one tuple for each combination of tuples—one from R and one from S.
  - Hence, if R has n_R tuples ($|R| = n_R$), and S has n_S tuples, then R x S will have $n_R * n_S$ tuples.
  - R and S do NOT have to be "type compatible"

CARTESIAN PRODUCT (cont.)

- Generally, CROSS PRODUCT is not a meaningful operation
  - Can become meaningful when followed by other operations
- Example (not meaningful):
  - FEMALE_EMPS ← $\sigma_{SEX='F'}$(EMPLOYEE)
  - EMPNAMES ← $\pi_{FNAME, LNAME, SSN}$(FEMALE_EMPS)
  - EMP_DEPENDENTS ← EMPNAMES × DEPENDENT
- EMP_DEPENDENTS will contain every combination of EMPNAMES and DEPENDENT
  - whether or not they are actually related

CARTESIAN PRODUCT (cont.)

- To keep only combinations where the DEPENDENT is related to the EMPLOYEE, we add a SELECT operation as follows
- Example (meaningful):
  - FEMALE_EMPS $\leftarrow \sigma_{SEX='F'}$(EMPLOYEE)
  - EMPNAMES $\leftarrow \pi_{FNAME,\ LNAME,\ SSN}$(FEMALE_EMPS)
  - EMP_DEPENDENTS $\leftarrow$ EMPNAMES $\times$ DEPENDENT
  - ACTUAL_DEPS $\leftarrow \sigma_{SSN=ESSN}$(EMP_DEPENDENTS)
  - RESULT $\leftarrow \pi_{FNAME,\ LNAME,\ DEPENDENT\_NAME}$(ACTUAL_DEPS)
- RESULT will now contain the name of female employees and their dependents.

Example: CARTESIAN PRODUCT

Figure 8.5
The CARTESIAN PRODUCT (CROSS PRODUCT) operation.

FEMALE_EMPS

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ss
Alicia	J	Zelaya	999887777	1968-07-19	3321Castle, Spring, TX	F	25000	98765432
Jennifer	S	Wallace	987654321	1941-06-20	291Berry, Bellaire, TX	F	43000	8886655
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	3334455

EMPNAMES

Fname	Lname	Ssn
Alicia	Zelaya	999887777
Jennifer	Wallace	987654321
Joyce	English	453453453

EMP_DEPENDENTS

Fname	Lname	Ssn	Essn	Dependent_name	Sex	Bdate	. . .
Alicia	Zelaya	999887777	333445555	Alice	F	1986-04-05	. . .
Alicia	Zelaya	999887777	333445555	Theodore	M	1983-10-25	. . .
Alicia	Zelaya	999887777	333445555	Joy	F	1958-05-03	. . .
Alicia	Zelaya	999887777	987654321	Abner	M	1942-02-28	. . .
Alicia	Zelaya	999887777	123456789	Michael	M	1988-01-04	. . .
Alicia	Zelaya	999887777	123456789	Alice	F	1988-12-30	. . .
Alicia	Zelaya	999887777	123456789	Elizabeth	F	1967-05-05	. . .
Jennifer	Wallace	987654321	333445555	Alice	F	1986-04-05	. . .
Jennifer	Wallace	987654321	333445555	Theodore	M	1983-10-25	. . .
Jennifer	Wallace	987654321	333445555	Joy	F	1958-05-03	. . .
Jennifer	Wallace	987654321	987654321	Abner	M	1942-02-28	. . .
Jennifer	Wallace	987654321	123456789	Michael	M	1988-01-04	. . .
Jennifer	Wallace	987654321	123456789	Alice	F	1988-12-30	. . .
Jennifer	Wallace	987654321	123456789	Elizabeth	F	1967-05-05	. . .
Joyce	English	453453453	333445555	Alice	F	1986-04-05	. . .
Joyce	English	453453453	333445555	Theodore	M	1983-10-25	. . .
Joyce	English	453453453	333445555	Joy	F	1958-05-03	. . .
Joyce	English	453453453	987654321	Abner	M	1942-02-28	. . .
Joyce	English	453453453	123456789	Michael	M	1988-01-04	. . .
Joyce	English	453453453	123456789	Alice	F	1988-12-30	. . .
Joyce	English	453453453	123456789	Elizabeth	F	1967-05-05	. . .

ACTUAL_DEPENDENTS

Fname	Lname	Ssn	Essn	Dependent_name	Sex	Bdate	. . .
Jennifer	Wallace	987654321	987654321	Abner	M	1942-02-28	. . .

RESULT

Fname	Lname	Dependent_name
Jennifer	Wallace	Abner

Cartesian-Product – Formal Definition

- 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.

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 the 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.

$$\pi_{customer\_name} (\sigma_{branch\_name=\text{"Perryridge"}} (\sigma_{borrower.loan\_number = loan.loan\_number} (borrower \times loan)))$$

- 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

$$\pi_{customer\_name} (\sigma_{branch\_name = \text{"Perryridge"}} (\sigma_{borrower.loan\_number = loan.loan\_number}(borrower \times loan))) - \pi_{customer\_name}(depositor)$$

Example Queries

- Find the names of all customers who have a loan at the Perryridge branch.
  - Query 1

$$\pi_{\text{customer\_name}} (\sigma_{\text{branch\_name = "Perryridge"}} (\sigma_{\text{borrower.loan\_number = loan.loan\_number}} (\text{borrower x loan})))$$

  - Query 2

$$\pi_{\text{customer\_name}}(\sigma_{\text{loan.loan\_number = borrower.loan\_number}} ((\sigma_{\text{branch\_name = "Perryridge"}} (\text{loan})) \times \text{borrower}))$$

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:

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

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

Completeness

- Set of relational algebra operations $\{\sigma, \pi, \cup, \rho, -, \times\}$ is a **complete set**
  - Any relational algebra operation can be expressed as a sequence of operations from this set

Additional Operations

- We define additional operations that do not add any power to the relational algebra, but that simplify common queries.
- Set intersection
- Join operation
- Division
- Assignment

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

Set-Intersection Operation – Formal Definition

- 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: $r \cap s = r - (r - s)$

Binary Relational Operations: JOIN

- JOIN Operation (denoted by $\bowtie$)
  - The sequence of CARTESIAN PRODECT followed by SELECT is used quite commonly to identify and select related tuples from two relations
  - A special operation, called JOIN combines this sequence into a single operation
  - This operation is very important for any relational database with more than a single relation, because it allows us *combine related tuples* from various relations

JOIN (cont.)

- The general form of a join operation on two relations R(A1, A2, . . ., An) and S(B1, B2, . . ., Bm) is:

$$R \bowtie_{\text{<join condition>}} S$$

where R and S can be any relations that result from general *relational algebra expressions*.

JOIN (cont.)

- Example: Suppose that we want to retrieve the name of the manager of each department.
  - To get the manager's name, we need to combine each DEPARTMENT tuple with the EMPLOYEE tuple whose SSN value matches the MGRSSN value in the department tuple.
  - We do this by using the join $\bowtie$ operation.
  - DEPT_MGR ← DEPARTMENT $\bowtie_{\text{MGRSSN=SSN}}$ EMPLOYEE
- MGRSSN=SSN is the join condition
  - Combines each department record with the employee who manages the department
  - The join condition can also be specified as DEPARTMENT.MGRSSN = EMPLOYEE.SSN

Example of applying the JOIN operation

Figure 8.6
Result of the JOIN operation DEPT_MGR ← DEPARTMENT $\bowtie_{\text{Mgr\_ssn=Ssn}}$ EMPLOYEE.

DEPT_MGR

Dname	Dnumber	Mgr_ssn	...	Fname	Minit	Lname	Ssn	...
Research	5	333445555	...	Franklin	T	Wong	333445555	...
Administration	4	987654321	...	Jennifer	S	Wallace	987654321	...
Headquarters	1	888665555	...	James	E	Borg	888665555	...

DEPT_MGR ← DEPARTMENT $\bowtie_{\text{MGRSSN=SSN}}$ EMPLOYEE

JOIN: More Example

$\bowtie$ = join

Account $\bowtie_{\text{Number=Account}}$ Deposit

Account	Number	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	103	J. Smith	5000.00	savings
	104	M. Jones	1000.00	checking
	105	H. Martin	10,000.00	checking

Deposit	Account	Transaction-id	Date	Amount
	102	1	10/22/00	500.00
	102	2	10/29/00	200.00
	104	3	10/29/00	1000.00
	105	4	11/2/00	10,000.00

	Number	Owner	Balance	Type	Account	Transaction-id	Date	Amount
	102	W. Wei	2000.00	checking	102	1	10/22/00	500.00
	102	W. Wei	2000.00	checking	102	2	10/29/00	200.00
	104	M. Jones	1000.00	checking	104	3	10/29/00	1000.00
	105	H. Martin	10,000.00	checking	105	4	11/2/00	10000.00

JOIN: More Example

Account $\bowtie_{\text{(Number=Account and Amount>700)}}$ Deposit

Account	Number	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	103	J. Smith	5000.00	savings
	104	M. Jones	1000.00	checking
	105	H. Martin	10,000.00	checking

Deposit	Account	T-id	Date	Amount
	102	1	10/22/00	500.00
	102	2	10/29/00	200.00
	104	3	10/29/00	1000.00
	105	4	11/2/00	10,000.00

	Number	Owner	Balance	Type	Account	T-id	Date	Amount
	104	M. Jones	1000.00	checking	104	3	10/29/00	1000.00
	105	H. Martin	10,000.00	checking	105	4	11/2/00	10000.00

Challenge Question

- How could you express the "join" operation if you didn't have a join operator in relational algebra? [Hint: are there other operators that you could use, in combination?]

JOIN using × and σ

- *Condition Join:* $R \bowtie_c S = \sigma_c (R \times S)$
- Sometimes called a *theta-join*.
- *Result schema same as that of cross-product*
- Fewer tuples than cross-product, might be able to compute more efficiently

Some properties of JOIN

- Consider the following JOIN operation:
  - R(A1, A2, . . ., An) $\bowtie_{R.Ai=S.Bj}$ S(B1, B2, . . ., Bm)
  - Result is a relation Q with degree n + m attributes:
    - Q(A1, A2, . . ., An, B1, B2, . . ., Bm), in that order.
  - The resulting relation state has one tuple for each combination of tuples—r from R and s from S, but only if they satisfy the join condition r[Ai]=s[Bj]
  - Hence, if R has n_R, and S has n_S tuples, then the result will generally have *less than* $n_R * n_S$ tuples.
  - Only related tuples (based on the join condition) will appear in the result

Some properties of JOIN

- The general case of JOIN operation is called a Theta-join: R $\bowtie_{\theta}$ S
- The join condition is called *theta*
- *Theta* can be any general boolean expression on the attributes of R and S; for example:
  - R.Ai<S.Bj AND (R.Ak=S.Bl OR R.Ap<S.Bq)
- Most join conditions involve one or more equality conditions “AND”ed together; for example:
  - R.Ai=S.Bj AND R.Ak=S.Bl AND R.Ap=S.Bq

Binary Relational Operations: EQUIJOIN

- EQUIJOIN Operation
- The most common use of join involves join conditions with *equality comparisons* only
- Such a join, where the only comparison operator used is =, is called an EQUIJOIN.
  - In the result of an EQUIJOIN we always have one or more pairs of attributes (whose names need not be identical) that have identical values in every tuple.
  - The JOIN seen in the previous example was an EQUIJOIN.

NATURAL JOIN Operation

- NATURAL JOIN Operation
  - Another variation of JOIN called NATURAL JOIN — denoted by * — was created to get rid of the second (superfluous) attribute in an EQUIJOIN condition.
  - The standard definition of natural join requires that the two join attributes, or each pair of corresponding join attributes, *have the same name* in both relations
  - If this is not the case, a renaming operation is applied first.

NATURAL JOIN (cont.)

- Example: Apply a natural join on the DNUMBER attributes of DEPARTMENT and DEPT_LOCATIONS, it is sufficient to write:
  - DEPT_LOCS ← DEPARTMENT * DEPT_LOCATIONS
- Only attribute with the same name is DNUMBER
- An implicit join condition is created based on this attribute:

  DEPARTMENT.DNUMBER=DEPT_LOCATIONS.DNUMBER

NATURAL JOIN (cont.)

- Another example: Q ← R(A,B,C,D) * S(C,D,E)
  - The implicit join condition includes ***each pair*** of attributes with the same name, "AND"ed together:
    - R.C=S.C AND R.D.=S.D
  - Result keeps only one attribute of each such pair:
    - Q(A,B,C,D,E)

Example of NATURAL JOIN

(a)

PROJ_DEPT

Pname	Pnumber	Plocation	Dnum	Dname	Mgr_ssn	Mgr_start_date
ProductX	1	Bellaire	5	Research	333445555	1988-05-22
ProductY	2	Sugarland	5	Research	333445555	1988-05-22
ProductZ	3	Houston	5	Research	333445555	1988-05-22
Computerization	10	Stafford	4	Administration	987654321	1995-01-01
Reorganization	20	Houston	1	Headquarters	888665555	1981-06-19
Newbenefits	30	Stafford	4	Administration	987654321	1995-01-01

(b)

DEPT_LOCS

Dname	Dnumber	Mgr_ssn	Mgr_start_date	Location
Headquarters	1	888665555	1981-06-19	Houston
Administration	4	987654321	1995-01-01	Stafford
Research	5	333445555	1988-05-22	Bellaire
Research	5	333445555	1988-05-22	Sugarland
Research	5	333445555	1988-05-22	Houston

Figure 8.7
Results of two natural join operations. (a) proj_dept ← project * dept.
(b) dept_locs ← department * dept_locations.

Challenge Question

- How could you express the natural join operation if you didn’t have a natural join operator in relational algebra?
- Consider you have two relations R(A,B,C) and S(B,C,D).

Division Operation

- Notation: $r \div s$
- Suited to queries that include "for all".
- Let r and s be relations on schemas R and S respectively where
  - $R = (A_1, \ldots, A_m, B_1, \ldots, B_n)$
  - $S = (B_1, \ldots, B_n)$

  The result of $r \div s$ is a relation on schema

  $R - S = (A_1, \ldots, A_m)$

$$r \div s = \{ t \mid t \in \pi_{R-S}(r) \wedge \forall u \in s \, (tu \in r) \}$$

where tu means the concatenation of tuples t and u to produce a single tuple

Division Operation – Example

Relations *r, s*

r

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

s

B
1
2

$r \div s$

A
α
β

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

Examples of Division: Suppliers and Parts

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

A

pno
p2

B1

pno
p2
p4

B2

pno
p1
p2
p4

B3

sno
s1
s2
s3
s4

A/B1

sno
s1
s4

A/B2

sno
s1

A/B3

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 $S \subseteq R$

$$r \div s = \pi_{R-S}(r) - \pi_{R-S}\left((\pi_{R-S}(r) \times s) - \pi_{R-S,S}(r) \right)$$

- To see why
  - $\pi_{R-S,S}(r)$ simply reorders attributes of r
  - $\pi_{R-S}\left((\pi_{R-S}(r) \times s) - \pi_{R-S,S}(r) \right)$ gives those tuples t in

    $\pi_{R-S}(r)$ such that for some tuple $u \in s$, $tu \notin r$.

Table 8.1 Operations of Relational Algebra

OPERATION	PURPOSE	NOTATION
SELECT	Selects all tuples that satisfy the selection condition from a relation R.	$\sigma_{\langle\text{selection condition}\rangle}(R)$
PROJECT	Produces a new relation with only some of the attributes of R, and removes duplicate tuples.	$\pi_{\langle\text{attribute list}\rangle}(R)$
THETA JOIN	Produces all combinations of tuples from R_1 and R_2 that satisfy the join condition.	$R_1 \bowtie_{\langle\text{join condition}\rangle} R_2$
EQUIJOIN	Produces all the combinations of tuples from R_1 and R_2 that satisfy a join condition with only equality comparisons.	$R_1 \bowtie_{\langle\text{join condition}\rangle} R_2$, OR $R_1 \bowtie_{(\langle\text{join attributes 1}\rangle),(\langle\text{join attributes 2}\rangle)} R_2$
NATURAL JOIN	Same as EQUIJOIN except that the join attributes of R_2 are not included in the resulting relation; if the join attributes have the same names, they do not have to be specified at all.	$R_1 *_{\langle\text{join condition}\rangle} R_2$, OR $R_1 *_{(\langle\text{join attributes 1}\rangle),(\langle\text{join attributes 2}\rangle)} R_2$ OR $R_1 * R_2$

UNION	Produces a relation that includes all the tuples in R_1 or R_2 or both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cup R_2$
INTERSECTION	Produces a relation that includes all the tuples in both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cap R_2$
DIFFERENCE	Produces a relation that includes all the tuples in R_1 that are not in R_2; R_1 and R_2 must be union compatible.	$R_1 - R_2$
CARTESIAN PRODUCT	Produces a relation that has the attributes of R_1 and R_2 and includes as tuples all possible combinations of tuples from R_1 and R_2.	$R_1 \times R_2$
DIVISION	Produces a relation $R(X)$ that includes all tuples $t[X]$ in $R_1(Z)$ that appear in R_1 in combination with every tuple from $R_2(Y)$, where $Z = X \cup Y$.	$R_1(Z) \div R_2(Y)$

Notation for Query Trees

- **Query tree**
  - Represents the input relations of query as leaf nodes of the tree
  - Represents the relational algebra operations as internal nodes

Example of a Query Tree

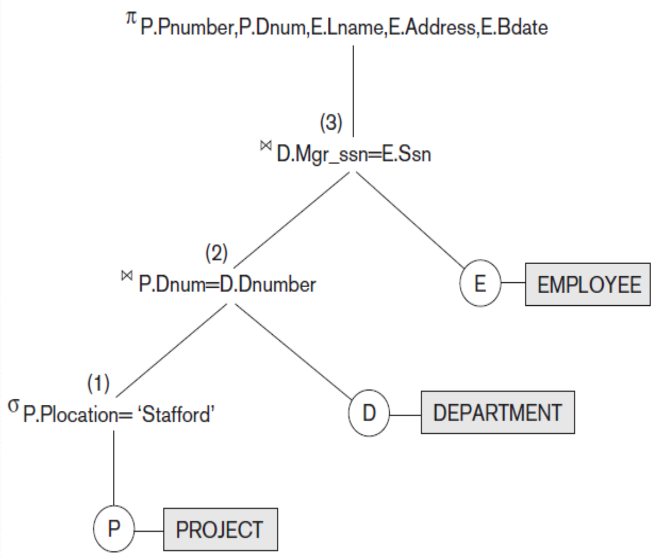


Figure 8.9
Query tree corresponding to the relational algebra expression for Q2.

Bank Example Queries

- Find the name of all customers who have a loan at the bank and the loan amount

$$\pi_{customer\_name,\ loan\_number,\ amount}(borrower \bowtie loan)$$

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

$$\pi_{customer\_name}(borrower) \cap \pi_{customer\_name}(depositor)$$

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.

$\pi_{customer\_name,\ branch\_name}\ (depositor \bowtie account)$
$\div\ \pi_{branch\_name}\ (\sigma_{branch\_city\ =\ \text{"Brooklyn"}}\ (branch))$

Additional Relational Operations

- **Generalized projection**
  - Allows functions of attributes to be included in the projection list

$$\pi_{F1,\ F2,\ \ldots,\ Fn}\ (R)$$

- **Aggregate functions** and **grouping**
  - Common functions applied to collections of numeric values
  - Include SUM, AVERAGE, MAXIMUM, and MINIMUM

Aggregate Functions and Grouping

- To specify mathematical aggregate functions on collections of values from the database.
- Examples: retrieving the average or total salary of all employees or the total number of employee tuples.
  - These functions are used in simple statistical queries that summarize information from the database tuples.
- Common functions on numeric values include
  - SUM, AVERAGE, MAXIMUM, and MINIMUM.
- The COUNT function is used for counting tuples or values.

Aggregate Function

- Use of the aggregate functional operation $\mathcal{F}$
  - $\mathcal{F}_{\text{MAX Salary}}$ (EMPLOYEE) retrieves the maximum salary value from the EMPLOYEE relation
  - $\mathcal{F}_{\text{MIN Salary}}$ (EMPLOYEE) retrieves the minimum Salary value from the EMPLOYEE relation
  - $\mathcal{F}_{\text{SUM Salary}}$ (EMPLOYEE) retrieves the sum of the Salary from the EMPLOYEE relation
  - $\mathcal{F}_{\text{COUNT SSN, AVERAGE Salary}}$ (EMPLOYEE) computes the count (number) of employees and their average salary
    - Note: count just counts the number of rows, without removing duplicates

Aggregate Operation – Example

Relation *r*:

A	B	C
α	α	7
α	β	7
β	β	3
β	β	10

$\mathcal{F}_{\textbf{sum(c)}}$ (r)

sum(*c*)
27

Using Grouping with Aggregation

- The previous examples all summarized one or more attributes for a set of tuples
  - Maximum Salary or Count (number of) Ssn
- Grouping can be combined with Aggregate Functions
- Example: For each department, retrieve the DNO, COUNT SSN, and AVERAGE SALARY
- A variation of aggregate operation $\mathcal{F}$ allows this:
  - Grouping attribute placed to left of symbol
  - Aggregate functions to right of symbol
  - $_{\text{DNO}}\,\mathcal{F}_{\text{COUNT SSN, AVERAGE Salary}}$ (EMPLOYEE)
- Above operation groups employees by DNO (department number) and computes the count of employees and average salary per department

Group Aggregation - Example

(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

Figure 8.10
The aggregate function operation.

a. $\rho_{R(\text{Dno, No\_of\_employees, Average\_sal})}(_{\text{Dno}}\,\Im_{\text{COUNT Ssn, AVERAGE Salary}}\,(\text{EMPLOYEE}))$.

b. $_{\text{Dno}}\,\Im_{\text{COUNT Ssn, AVERAGE Salary}}(\text{EMPLOYEE})$.

c. $\Im_{\text{COUNT Ssn, AVERAGE Salary}}(\text{EMPLOYEE})$.

Aggregate Operation – More Example

- Relation *account* grouped by *branch-name*:

branch_name	*account_number*	*balance*
Perryridge	A-102	400
Perryridge	A-201	900
Brighton	A-217	750
Brighton	A-215	750
Redwood	A-222	700

$$_{branch\_name}\mathcal{F}_{\mathbf{sum}(balance)}\ (account)$$

branch_name	**sum**(*balance*)
Perryridge	1300
Brighton	1500
Redwood	700

Recursive Closure Operations

- Operation applied to a recursive relationship between tuples of same type
- What is the result of the following sequence of queries ?

$$BORG\_SSN \leftarrow \pi_{Ssn}(\sigma_{Fname='James' \text{ AND } Lname='Borg'}(EMPLOYEE))$$
$$SUPERVISION(Ssn1, Ssn2) \leftarrow \pi_{Ssn,Super\_ssn}(EMPLOYEE)$$
$$RESULT1(Ssn) \leftarrow \pi_{Ssn1}(SUPERVISION \bowtie_{Ssn2=Ssn} BORG\_SSN)$$

OUTER JOIN Operations

- **Outer joins**
  - Keep all tuples in R, or all those in S, or all those in both relations regardless of whether or not they have matching tuples in the other relation
  - **Types**
    - **LEFT OUTER JOIN**, **RIGHT OUTER JOIN**, **FULL OUTER JOIN** ⟗
  - Example:

$$\text{TEMP} \leftarrow (\text{EMPLOYEE} ⟕_{\text{Ssn=Mgr\_ssn}} \text{DEPARTMENT})$$

$$\text{RESULT} \leftarrow \pi_{\text{Fname, Minit, Lname, Dname}}(\text{TEMP})$$

The OUTER UNION Operation

- Take union of tuples from two relations that have some common attributes
  - Not union (type) compatible
- **Partially compatible**
  - All tuples from both relations included in the result
  - Tuples with the same value combination will appear only once

Queries in Relational Algebra

Query 1. Retrieve the name and address of all employees who work for the 'Research' department.

RESEARCH_DEPT ← $\sigma_{\text{Dname='Research'}}$(DEPARTMENT)
RESEARCH_EMPS ← (RESEARCH_DEPT $\bowtie_{\text{Dnumber=Dno}}$ EMPLOYEE)
RESULT ← $\pi_{\text{Fname, Lname, Address}}$(RESEARCH_EMPS)

As a single in-line expression, this query becomes:

$\pi_{\text{Fname, Lname, Address}}$ ($\sigma_{\text{Dname='Research'}}$(DEPARTMENT $\bowtie_{\text{Dnumber=Dno}}$(EMPLOYEE))

Queries in Relational Algebra

Query 2. For every project located in 'Stafford', list the project number, the controlling department number, and the department manager's last name, address, and birth date.

STAFFORD_PROJS ← $\sigma_{\text{Plocation='Stafford'}}$(PROJECT)
CONTR_DEPTS ← (STAFFORD_PROJS $\bowtie_{\text{Dnum=Dnumber}}$ DEPARTMENT)
PROJ_DEPT_MGRS ← (CONTR_DEPTS $\bowtie_{\text{Mgr\_ssn=Ssn}}$ EMPLOYEE)
RESULT ← $\pi_{\text{Pnumber, Dnum, Lname, Address, Bdate}}$(PROJ_DEPT_MGRS)

Query 3. Find the names of employees who work on *all* the projects controlled by department number 5.

DEPT5_PROJS ← $\rho_{(\text{Pno})}$(π_{Pnumber}($\sigma_{\text{Dnum=5}}$(PROJECT)))
EMP_PROJ ← $\rho_{(\text{Ssn, Pno})}$($\pi_{\text{Essn, Pno}}$(WORKS_ON))
RESULT_EMP_SSNS ← EMP_PROJ ÷ DEPT5_PROJS
RESULT ← $\pi_{\text{Lname, Fname}}$(RESULT_EMP_SSNS * EMPLOYEE)

Queries in Relational Algebra

Query 6. Retrieve the names of employees who have no dependents.

This is an example of the type of query that uses the MINUS (SET DIFFERENCE) operation.

$\text{ALL\_EMPS} \leftarrow \pi_{\text{Ssn}}(\text{EMPLOYEE})$
$\text{EMPS\_WITH\_DEPS(Ssn)} \leftarrow \pi_{\text{Essn}}(\text{DEPENDENT})$
$\text{EMPS\_WITHOUT\_DEPS} \leftarrow (\text{ALL\_EMPS} - \text{EMPS\_WITH\_DEPS})$
$\text{RESULT} \leftarrow \pi_{\text{Lname, Fname}}(\text{EMPS\_WITHOUT\_DEPS} * \text{EMPLOYEE})$

Query 7. List the names of managers who have at least one dependent.

$\text{MGRS(Ssn)} \leftarrow \pi_{\text{Mgr\_ssn}}(\text{DEPARTMENT})$
$\text{EMPS\_WITH\_DEPS(Ssn)} \leftarrow \pi_{\text{Essn}}(\text{DEPENDENT})$
$\text{MGRS\_WITH\_DEPS} \leftarrow (\text{MGRS} \cap \text{EMPS\_WITH\_DEPS})$
$\text{RESULT} \leftarrow \pi_{\text{Lname, Fname}}(\text{MGRS\_WITH\_DEPS} * \text{EMPLOYEE})$

Question

- Relational Algebra is not Turing complete. There are operations that cannot be expressed in relational algebra.
- What is the advantage of using this language to query a database?
  - *By limiting the scope of the operations, it is possible to automatically optimize queries.*

Relational Calculus

- A relational calculus expression creates a new relation, which is specified in terms of variables that range over rows of the stored database relations (in tuple calculus) or over columns of the stored relations (in domain calculus).
- In a calculus expression, there is *no order of operations* to specify how to retrieve the query result—a calculus expression specifies only what information the result should contain.
  - This is the main distinguishing feature between relational algebra and relational calculus.

Relational Calculus (Cont.)

- Relational calculus is considered to be a nonprocedural or declarative language.
- This differs from relational algebra, where we must write a *sequence of operations* to specify a retrieval request; hence relational algebra can be considered as a procedural way of stating a query.
- Any retrieval that can be specified in basic relational algebra can also be specified in relational calculus (and vice versa)

Tuple Relational Calculus

- The tuple relational calculus is based on specifying a number of tuple variables.
- Each tuple variable usually ranges over a particular database relation, meaning that the variable may take as its value any tuple from that relation.
- A simple tuple relational calculus query is of the form

  {t | COND(t)}

  - where t is a tuple variable and COND (t) is a conditional expression involving t.
  - The result of such a query is the set of all tuples t that satisfy COND (t).

Tuple Relational Calculus

- **Tuple variables**
  - Ranges over a particular database relation
- **Satisfy** COND(*t*):
- Specify:
  - **Range relation** *R* of *t*
  - Select particular combinations of tuples
  - Set of attributes to be retrieved (**requested attributes**)

Tuple Relational Calculus

- General expression of tuple relational calculus is of the form:

$$\{t_1.A_j, t_2.A_k, \ldots, t_n.A_m \mid \mathrm{COND}(t_1, t_2, \ldots, t_n, t_{n+1}, t_{n+2}, \ldots, t_{n+m})\}$$

- **Truth value** of an atom
  - Evaluates to either TRUE or FALSE for a specific combination of tuples
- **Formula** (Boolean condition)
  - Made up of one or more atoms connected via logical operators **AND**, **OR**, and **NOT**

Tuple Relational Calculus

- Example: Find the first and last names of all employees whose salary is above $50,000.

  {t.FNAME, t.LNAME | EMPLOYEE(t) AND t.SALARY>50000}

- The condition EMPLOYEE(t) specifies that the range relation of tuple variable t is EMPLOYEE.
- The first and last name (PROJECTION $\pi_{\text{FNAME, LNAME}}$) of each EMPLOYEE tuple t that satisfies the condition t.SALARY>50000 (SELECTION $\sigma_{\text{SALARY >50000}}$) will be retrieved.

Conditional Expression

1. Set of attributes and constants
2. Set of comparison operators: (e.g., $<, \leq, =, \neq, >, \geq$)
3. Set of connectives: and ($\wedge$), or ($\vee$), not ($\neg$)
4. Implication ($\Rightarrow$): $x \Rightarrow y$, if x is true, then y is true

$$x \Rightarrow y \equiv \neg x \vee y$$

5. Set of quantifiers:
   - $\exists\, t \in r\, (Q\,(t\,)) \equiv$ "there exists" a tuple in t in relation r such that predicate $Q\,(t\,)$ is true
   - $\forall t \in r\, (Q\,(t\,)) \equiv Q$ is true "for all" tuples t in relation r

Relational Algebra and Calculus 110

The Existential and Universal Quantifiers

- Two special symbols called quantifiers can appear in formulas; these are the universal quantifier ($\forall$) and the existential quantifier ($\exists$).
- Informally, a tuple variable t is bound if it is quantified, meaning that it appears in an ($\forall$t) or ($\exists$t) clause; otherwise, it is free.

The Existential and Universal Quantifiers

- If F is a formula, then so are (∃t)(F) and (∀t)(F), where t is a tuple variable.
  - The formula (∃t)(F) is true if the formula F evaluates to true for **some** (at least one) tuple assigned to free occurrences of t in F; otherwise (∃t)(F) is false.
  - The formula (∀t)(F) is true if the formula F evaluates to true for **every** tuple (in the universe) assigned to free occurrences of t in F; otherwise (∀t)(F) is false.

The Existential and Universal Quantifiers

- ∀ is called the universal or "for all" quantifier because every tuple in "the universe of" tuples must make F true to make the quantified formula true.
- ∃ is called the existential or "there exists" quantifier because any tuple that exists in "the universe of" tuples may make F true to make the quantified formula true.

Example Query Using Existential Quantifier

- The only *free tuple variables* in a relational calculus expression should be those that appear to the left of the bar (|).
  - In above query, t is the only free variable; it is then *bound successively* to each tuple.

Query 1. List the name and address of all employees who work for the 'Research' department.

Q1: {*t*.Fname, *t*.Lname, *t*.Address | EMPLOYEE(*t*) **AND** (∃*d*)(DEPARTMENT(*d*) **AND** *d*.Dname='Research' **AND** *d*.Dnumber=*t*.Dno)}

Example Query Using Existential Quantifier

- If a tuple *satisfies the conditions* specified in the query, the attributes FNAME, LNAME, and ADDRESS are retrieved for each such tuple.
  - The conditions EMPLOYEE (t) and DEPARTMENT(d) specify the range relations for t and d.
  - The condition d.DNAME = 'Research' is a selection condition and corresponds to a SELECT operation in the relational algebra, whereas the condition d.DNUMBER = t.DNO is a JOIN condition.

Example Query Using Universal Quantifier

- Find the names of employees who work on *all* the projects controlled by department number 5.

  {e.LNAME, e.FNAME | EMPLOYEE(e) and (
  (∀x)(not(PROJECT(x)) or not(x.DNUM=5) OR
  ((∃w)(WORKS_ON(w) and w.ESSN=e.SSN and
  x.PNUMBER=w.PNO))))}
- Exclude from the universal quantification all tuples that we are not interested in by making the condition true *for all such tuples*.
  - The first tuples to exclude (by making them evaluate automatically to true) are those that are not in the relation R of interest.

Example Query Using Universal Quantifier

- In query above, using the expression **not(PROJECT(x))** inside the universally quantified formula evaluates to true all tuples x that are not in the PROJECT relation.
  - Then we exclude the tuples we are not interested in from R itself. The expression not(x.DNUM=5) evaluates to true all tuples x that are in the project relation but are not controlled by department 5.
- Finally, we specify a condition that must hold on all the remaining tuples in R.

  ((∃w)(WORKS_ON(w) and w.ESSN=e.SSN and
  x.PNUMBER=w.PNO)

Tuple Calculus – More Example

Query 4. Make a list of project numbers for projects that involve an employee whose last name is 'Smith', either as a worker or as manager of the controlling department for the project.

Q4: { p.Pnumber | PROJECT(p) **AND** ((($\exists e$)($\exists w$)(EMPLOYEE(e)
AND WORKS_ON(w) **AND** w.Pno=p.Pnumber
AND e.Lname='Smith' **AND** e.Ssn=w.Essn))
OR
(($\exists m$)($\exists d$)(EMPLOYEE(m) **AND** DEPARTMENT(d)
AND p.Dnum=d.Dnumber **AND** d.Mgr_ssn=m.Ssn
AND m.Lname='Smith')))}

Using the Universal Quantifier in Queries

Query 3. List the names of employees who work on *all* the projects controlled by department number 5. One way to specify this query is to use the universal quantifier as shown:

Q3: {e.Lname, e.Fname | EMPLOYEE(e) **AND** (($\forall x$)(**NOT**(PROJECT(x)) **OR NOT**
(x.Dnum=5) **OR** (($\exists w$)(WORKS_ON(w) **AND** w.Essn=e.Ssn **AND**
x.Pnumber=w.Pno))))}

Q3A: {e.Lname, e.Fname | EMPLOYEE(e) **AND** (**NOT** ($\exists x$) (PROJECT(x) **AND**
(x.Dnum=5) and (**NOT** ($\exists w$)(WORKS_ON(w) **AND** w.Essn=e.Ssn
AND x.Pnumber=w.Pno))))}

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 the *loan_number, branch_name,* and *amount* for loans of over $1200

  $\{ t \mid t \in loan \wedge t[amount] > 1200 \}$

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

  $\{ t.loan\_number \mid t \in loan \wedge t.amount > 1200 \}$ or

  $\{ t \mid \exists\, s \in loan\ (t[loan\_number] = s[loan\_number]$
  $\wedge\ s[amount] > 1200) \}$

  Notice that a relation on schema [*loan_number*] is implicitly defined by the query

Example Queries

- Find the names of all customers having a loan, an account, or both at the bank

$\{ t \mid \exists s \in borrower\ (t[customer\_name] = s[customer\_name])$
$\quad \lor \exists u \in depositor\ (t[customer\_name] = u[customer\_name]) \}$

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

$\{ t \mid \exists s \in borrower\ (t[customer\_name] = s[customer\_name])$
$\quad \land \exists u \in depositor\ (t[customer\_name] = u[customer\_name]) \}$

Example Queries

- Find the names of all customers having a loan at the Perryridge branch

$\{ t \mid \exists s \in borrower\ (t[customer\_name] = s[customer\_name]$
$\quad \land \exists u \in loan\ (u[branch\_name] = \text{"Perryridge"}$
$\qquad \land\ u[loan\_number] = s[loan\_number])) \}$

- Find the names of all customers who have a loan at the Perryridge branch, but no account at any branch of the bank

$\{ t \mid \exists s \in borrower\ (t[customer\_name] = s[\text{customer\_name}]$
$\quad \land \exists u \in loan\ (u[branch\_name] = \text{"Perryridge"}$
$\qquad \land\ u[loan\_number] = s[\text{loan\_}number]))$
$\quad \land \neg \exists v \in depositor\ (v[customer\_name] =$
$\qquad t[customer\_name]) \}$

Example Queries

- Find the names of all customers having a loan from the Perryridge branch, and the cities in which they live

$\{ t \mid \exists s \in loan\ (s[branch\_name] = \text{"Perryridge"}$
$\wedge \exists u \in borrower\ (u[loan\_number] = s[loan\_numbe]$
$\wedge\ t[customer\_name] = u[customer\_name])$
$\wedge \exists v \in customer\ (u[customer\_name] = v[customer\_name]$
$\wedge\ t[customer\_city] = v[customer\_city]))) \}$

Notice that a relation on schema [*customer_name, customer_city*] is implicitly defined by the query.

Example Queries

- Find the names of all customers who have an account at all branches located in Brooklyn.

$\{ t \mid \exists\ r \in customer\ (t[customer\_name] = r[customer\_name])$
$\wedge (\forall\ u \in branch\ (u[branch\_city] = \text{"Brooklyn"} \Rightarrow$
$\exists\ s \in depositor\ (t[customer\_name] = s[customer\_name]$
$\wedge \exists\ w \in account\ (w[account\_number] = s[account\_number]$
$\wedge (w[branch\_name] = u[branch\_name])))) \}$

Safe Expressions

- Guaranteed to yield a finite number of tuples as its result
  - Otherwise expression is called unsafe
- Expression is safe
  - If all values in its result are from the domain of the expression

Domain Relational Calculus

- Another variation of relational calculus called the domain relational calculus, or simply, domain calculus is equivalent to tuple calculus and to relational algebra.
- The language QBE (Query-By-Example) related to domain calculus was developed almost concurrently to SQL at IBM Research, Yorktown Heights, New York.
  - Domain calculus was thought of as a way to explain what QBE does.
- Domain calculus differs from tuple calculus in the type of variables used in formulas:
  - Rather than having variables range over tuples, the variables range over single values from domains of attributes.
- To form a relation of degree n for a query result, we must have n of these domain variables— one for each attribute.

Domain Relational Calculus (cont.)

- An expression of the domain calculus is of the form

  $\{ x_1, x_2, \ldots, x_n \mid$

  $\quad COND(x_1, \ldots, x_n, x_{n+1}, x_{n+2}, \ldots, x_{n+m})\}$

  - where $x_1, x_2, \ldots, x_n, x_{n+1}, x_{n+2}, \ldots, x_{n+m}$ are domain variables that range over domains (of attributes)
  - and COND is a condition or formula of the domain relational calculus.

Example Query Using Domain Calculus

- Retrieve the birthdate and address of the employee whose name is 'John B. Smith'.
- Query :

  {uv | (∃ q) (∃ r) (∃ s) (∃ t) (∃ w) (∃ x) (∃ y) (∃ z)

  (EMPLOYEE(qrstuvwxyz) and q='John' and r='B' and s='Smith')}
- Abbreviated notation EMPLOYEE(qrstuvwxyz) uses the variables without the separating commas: EMPLOYEE(q,r,s,t,u,v,w,x,y,z)
- Ten variables for the employee relation are needed, one to range over the domain of each attribute in order.
  - Of the ten variables q, r, s, . . ., z, only u and v are free.

Relational Algebra and Calculus 129

Example Query Using Domain Calculus

- Specify the *requested attributes*, BDATE and ADDRESS, by the free domain variables u for BDATE and v for ADDRESS.
- Specify the condition for selecting a tuple following the bar (|)
  - namely, that the sequence of values assigned to the variables qrstuvwxyz be a tuple of the employee relation and that the values for q (FNAME), r (MINIT), and s (LNAME) be 'John', 'B', and 'Smith', respectively.

{uv | (∃ q) (∃ r) (∃ s) (EMPLOYEE(qrstuvwxyz) and q='John' and r='B' and s='Smith')}

Example Query Using Domain Calculus

Query 1. Retrieve the name and address of all employees who work for the 'Research' department.

Q1: $\{q, s, v \mid (\exists z)\ (\exists l)\ (\exists m)$ (EMPLOYEE($qrstuvwxyz$) **AND** DEPARTMENT($lmno$) **AND** l='Research' **AND** $m=z$)}

Query 2. For every project located in 'Stafford', list the project number, the controlling department number, and the department manager's last name, birth date, and address.

Q2: $\{i, k, s, u, v \mid (\exists j)(\exists m)(\exists n)(\exists t)$(PROJECT($hijk$) **AND** EMPLOYEE($qrstuvwxyz$) **AND** DEPARTMENT($lmno$) **AND** $k=m$ **AND** $n=t$ **AND** j='Stafford')}

More Examples on Bank

- Find the *loan_number, branch_name,* and *amount* for loans of over $1200

  $\{ \langle l, b, a \rangle \mid \langle l, b, a \rangle \in loan \land a > 1200 \}$

- Find the names of all customers who have a loan of over $1200

  $\{ \langle c \rangle \mid \exists\, l, b, a\ (\langle c, l \rangle \in borrower \land \langle l, b, a \rangle \in loan \land a > 1200) \}$

More Examples on Bank

- Find the names of all customers who have a loan from the Perryridge branch and the loan amount:

  $\{ \langle c, a \rangle \mid \exists\, l\ (\langle c, l \rangle \in borrower \land \exists b\ (\langle l, b, a \rangle \in loan \land b = \text{“Perryridge”})) \}$

  $\{ \langle c, a \rangle \mid \exists\, l\ (\langle c, l \rangle \in borrower \land \langle l, \text{“Perryridge”}, a \rangle \in loan) \}$

More Examples on Bank

- Find the names of all customers having a loan, an account, or both at the Perryridge branch:

$$\{ c \mid \exists\, l\, (< c, l > \in borrower \\ \wedge \exists\, b,a\, (< l, b, a > \in loan \wedge b = \text{“Perryridge”})) \\ \vee \exists\, a\, (< c, a > \in depositor \\ \wedge \exists\, b,n\, (< a, b, n > \in account \wedge b = \text{“Perryridge”}))\}$$

- Find the names of all customers who have an account at all branches located in Brooklyn:

$$\{ c \mid \exists\, s,n\, (< c, s, n > \in \text{customer}) \wedge \\ \forall\, x,y,z\, (< x, y, z > \in branch \wedge y = \text{“Brooklyn”}) \Rightarrow \\ \exists\, a,b\, (< a, x, b > \in account \wedge < c,a > \in depositor)\}$$

Summary

- Relational Algebra
  - Unary Relational Operations
  - Relational Algebra Operations From Set Theory
  - Binary Relational Operations
  - Additional Relational Operations
  - Examples of Queries in Relational Algebra
- Relational Calculus*
  - Tuple Relational Calculus
  - Domain Relational Calculus
- Overview of the QBE language (appendix C)*

Assignment 2

- Textbook exercises: 2.12, 2.13, 2.14, 2.15, 2.16
- Due date: Oct. 28, 2020