Relational Algebra

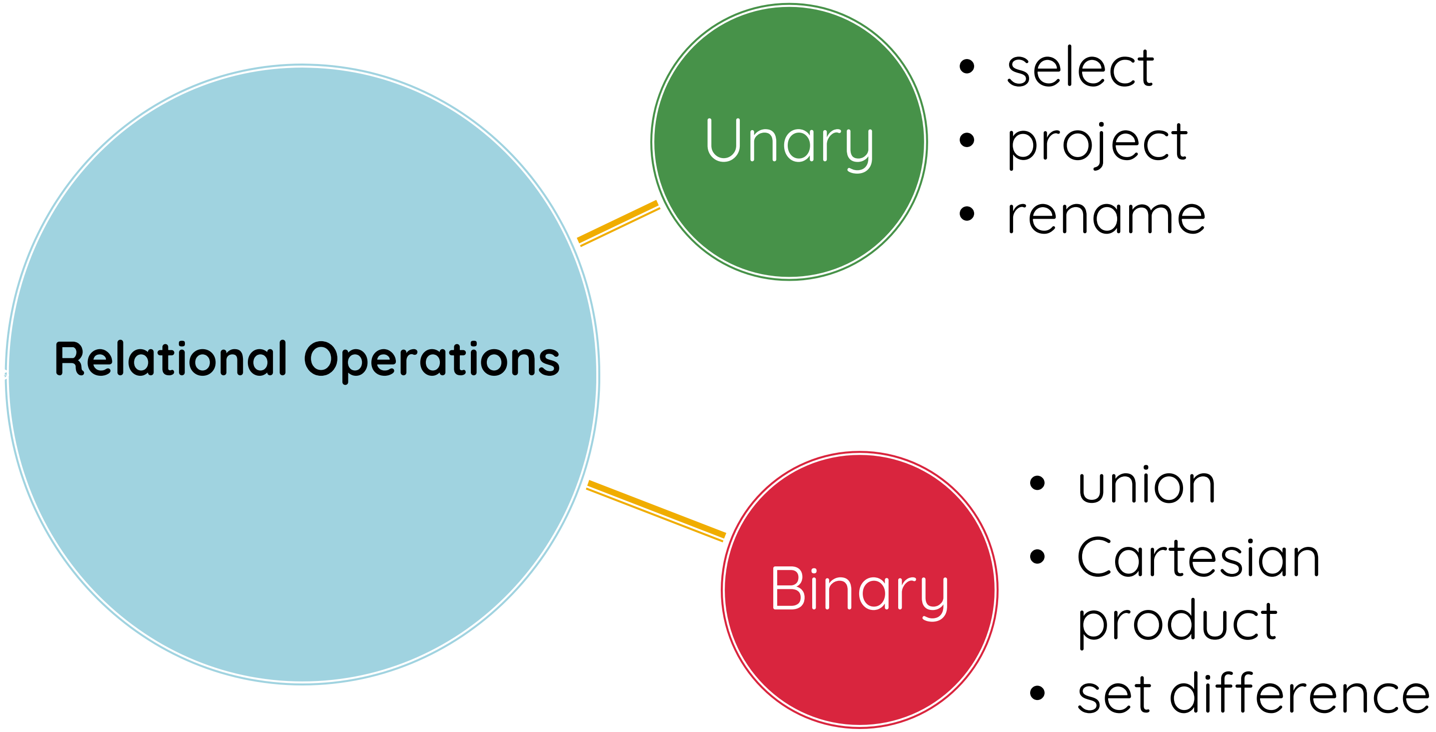
CE384: Database Design
Maryam Ramezani
Sharif University of Technology
maryam.ramezani@sharif.edu

01

Relational Formal Definitions

Introduction

- The relational algebra consists of a set of operations that take one or two relations as input and produce a new relation as their result.



Review-Formal Definitions

- A **tuple** is an ordered set of values (enclosed in angled brackets ‘< … >’)
- Each value is derived from an appropriate *domain*.
- A row in the CUSTOMER relation is a 4-tuple and would consist of four values, for example:
  CUSTOMER (Cust-id, Cust-name, Address, Phone#)
  - CUSTOMER is the relation name
  - Defined over the four attributes from domains: Cust-id: customer id values, Cust-name: customer name values, Address: address values, Phone#: phone values
  - <632895, "John Smith", "101 Main St. Atlanta, GA 30332", "(404) 894-2000">
  - This is called a 4-tuple as it has 4 values
  - A tuple (row) in the CUSTOMER relation.

Review-Formal Definitions

- A relation R is a **set** of such tuples (rows) having:
  - Heading: H_R or R(H) e.g: For R($A_1,\ldots,A_m$) R(H)={$A_1,\ldots,A_m$}
    - R(H) is constant over time.
    - Change in R(H) makes a new relation.
  - Body: r(R) set of tuples
    - It is changeable over the time.
    - Relation state r(R): a specific **state** (or "value" or "population") of relation R – this is a *set of tuples* (rows)
      - r(R) = {t1, t2, …, tn} where each ti is an n-tuple
      - ti = <v1, v2, …, vn> where each vj *element-of* dom(Aj)
- Degree: Number of heading.
- Cardinality: Number of rows.
- Think about a relation which the number of domains is smaller than degree?
  - Emp(ID,Name,ManagerID): the EmpID and ManagerID are from the same domain of EmpID.

Formal Definitions - Summary

- Formally,
  - Given R(A1, A2, ..., An)
  - $r(R) \subset dom(A1) \times dom(A2) \times \ldots \times dom(An)$
- R(A1, A2, …, An) is the **schema** of the relation
- R is the **name** of the relation
- A1, A2, …, An are the **attributes** of the relation

Formal Definitions - Example

- Let R(A1, A2) be a relation schema:
  - Let dom(A1) = {0,1}
  - Let dom(A2) = {a,b,c}

  Then: dom(A1) X dom(A2) is:

  all possible combinations: {<0,a> , <0,b> , <0,c>, <1,a>, <1,b>, <1,c> }

- The relation state r(R) $\subset$ dom(A1) X dom(A2)
  - For example: r(R) could be {<0,a> , <0,b> , <1,c> }
  - This is one possible state (or “population” or “extension”) r of the relation R, defined over A1 and A2.
  - It has three 2-tuples: <0,a> , <0,b> , <1,c>

Definition Summary

Informal Terms		Formal Terms
Table		Relation
Column Header		Attribute
All possible Column Values		Domain
Row		Tuple

Example – A relation STUDENT

Relation Name

Attributes

Tuples

STUDENT

Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	NULL	19	3.21
Chung-cha Kim	381-62-1245	375-4409	125 Kirby Road	NULL	18	2.89
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	749-1253	25	3.53
Rohan Panchal	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	NULL	19	3.25

Figure 5.1
The attributes and tuples of a relation STUDENT.

Characteristics Of Relations

- Ordering of tuples in a relation r(R):
  - The tuples are not considered to be ordered (because it is a set), even though they appear to be in the tabular form
- Ordering of attributes in a relation schema R (and of values within each tuple):
  - We will consider the attributes in R(A1, A2, ..., An) and the values in t=<v1, v2, ..., vn> to be ordered .
    - (However, a more general alternative definition of relation does not require this ordering).
- Because relation is a set, it does not have duplicated tuples.
- In theoretical, degree of a relation is ≥ 0

Characteristics Of Relations

- Values in a tuple:
  - All values are considered atomic (indivisible).
    - **Hence, composite and multivalued attributes are not allowed.**
  - Each value in a tuple must be from the domain of the attribute for that column
    - If tuple t = <v1, v2, …, vn> is a tuple (row) in the relation state r of R(A1, A2, …, An)
    - Then each *vi* must be a value from *dom(Ai)*

  - A special **null** value is used to represent values that are unknown or inapplicable to certain tuples.
- If m=degree of relation and n=number of domains then $m \geq n$

Characteristics Of Relations

- Notation:
  - We refer to **component values** of a tuple t by:
    - t[Ai] or t.Ai
    - This is the value vi of attribute Ai for tuple t
  - Similarly, t[Au, Av, ..., Aw] refers to the subtuple of t containing the values of attributes Au, Av, ..., Aw, respectively in t

Review: Key Constraints

- **Superkey** of R:
  - Is a set of attributes SK of R with the following condition:
    - No two tuples in any valid relation state r(R) will have the same value for SK
    - That is, for any distinct tuples t1 and t2 in r(R), t1[SK] ≠ t2[SK]
    - This condition must hold in any valid state r(R)
- **Key** of R **Candidate key** (CK):
  - A "minimal" superkey
  - That is, a key is a superkey K such that removal of any attribute from K results in a set of attributes that is not a superkey (does not possess the superkey uniqueness property)

Review: Entity Integrity

- **Entity Integrity:**
  - The Primary Key attributes **PK** of each relation schema R in S cannot have null values in any tuple of r(R).
    - This is because primary key values are used to identify the individual tuples.
    - t[PK] ≠ null for any tuple t in r(R)
    - If PK has several attributes, null is not allowed in any of these attributes
  - Note: Other attributes of R may be constrained to disallow null values, even though they are not members of the primary key.
  - If a primary key is too long, the surrogate key is used.

Relational Operations Overview

- **Relational Algebra consists of several groups of operations**
  - Unary Relational Operations
    - SELECT (symbol: σ(sigma))
    - PROJECT (symbol: π(pi))
    - RENAME (symbol: ρ (rho))
  - Relational Algebra Operations From Set Theory
    - UNION (∪), INTERSECTION (∩), DIFFERENCE (or MINUS, **-**)
    - CARTESIAN PRODUCT (**x**)
  - Binary Relational Operations
    - JOIN (several variations of JOIN exist)
    - DIVISION
  - Additional Relational Operations
    - OUTER JOINS, OUTER UNION
    - AGGREGATE FUNCTIONS (These compute summary of information: for example, SUM, COUNT, AVG, MIN, MAX)

Example

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

Figure 2.1 The *instructor* relation.

02

Unary Operations

SELECT

- The SELECT operation (denoted by **σ** (sigma)) is used to select a *subset* of the tuples from a relation based on a **selection condition**.
  - The selection condition acts as a **filter**
  - Keeps only those tuples that satisfy the qualifying condition
  - Tuples satisfying the condition are *selected* whereas the other tuples are discarded (*filtered out*)
  - We allow comparisons using =, ≠, <, ≤, >, and ≥ in the selection predicate.
  - Combine several predicates into a larger predicate by using the connectives *and* (∧), *or* (∨), and *not* (.).

SELECT

- In general, the *select* operation is denoted by

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

  - the symbol σ (sigma) is used to denote the *select* operator
  - the selection condition is a Boolean (conditional) expression specified on the attributes of relation R
  - tuples that make the condition **true** are selected
    - appear in the result of the operation
  - tuples that make the condition **false** are filtered out
    - discarded from the result of the operation

SELECT

- Example:
  - $\sigma_{dept\_name = \text{"Physics"}}(instructor)$

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
33456	Gold	Physics	87000

  - Find all instructors with salary greater than $90,000

$$\sigma_{salary > 90000}(instructor)$$

  - Find the instructors in Physics with a salary greater than $90,000

$$\sigma_{dept\_name = \text{"Physics"} \wedge salary > 90000}(instructor)$$

  - Consider the relation department. Find all departments whose name is the same as their building name. *department (dept_name, building, budget)*

$$\sigma_{dept\_name = building}(department)$$

SELECT

- SELECT Operation Properties
  - The SELECT operation $\sigma_{\text{<selection condition>}}(R)$ produces a relation S that has the same schema (same attributes) as R
  - SELECTσ is commutative:
    - $\sigma_{\text{<condition1>}}(\sigma_{\text{< condition2>}}(R)) = \sigma_{\text{<condition2>}}(\sigma_{\text{< condition1>}}(R))$
    - Because of commutativity property, a cascade (sequence) of SELECT operations may be applied in any order:
    - $\sigma_{\text{<cond1>}}(\sigma_{\text{<cond2>}}(\sigma_{\text{<cond3>}}(R)) = \sigma_{\text{<cond2>}}(\sigma_{\text{<cond3>}}(\sigma_{\text{<cond1>}}(R)))$
  - A cascade of SELECT operations may be replaced by a single selection with a conjunction of all the conditions:
    - $\sigma_{\text{<cond1>}}(\sigma_{\text{< cond2>}}(\sigma_{\text{<cond3>}}(R)) = \sigma_{\text{<cond1> AND < cond2> AND < cond3>}}(R)))$
- The cardinality in the result of a SELECT is:
  - less than (or equal to) the number of tuples in the input relation R: $0 \leq |\sigma_{c}(R)| \leq |R|$
- The degree of resulting relation from a Selection operation is:
  - same as the degree of the Relation given.

SELECT

- If $R' = \sigma_c(R)$ then what the candidate key of R'?
  - $CK_{R'} \subseteq CK_R$

- Equivalent expressions
  - commutative $\sigma_{c1}\left(\sigma_{c2}(R)\right) = \sigma_{c2}\left(\sigma_{c1}(R)\right) = \sigma_{c1 \wedge c2}(R))$

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 columns (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:

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

PROJECT

- The general form of the project operation is:

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

  - π (pi) is the symbol 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

- Example
  - 1) $\Pi_{ID,\ name,\ salary}(instructor)$
  - 2) $\Pi_{ID,name,salary/12}(instructor)$

ID	*name*	*salary*
10101	Srinivasan	65000
12121	Wu	90000
15151	Mozart	40000
22222	Einstein	95000
32343	El Said	60000
33456	Gold	87000
45565	Katz	75000
58583	Califieri	62000
76543	Singh	80000
76766	Crick	72000
83821	Brandt	92000
98345	Kim	80000

  - 3) Find the names of all instructors in the Physics department.

$$\Pi_{name}\ (\sigma_{dept\_name = \text{“Physics”}}\ (instructor))$$

Notice that, instead of giving the name of a relation as the argument of the projection operation, we give an expression that evaluates to a relation.

PROJECT

- Q1: $\pi_{Class,\ Dept}(Faculty)$

Class	Dept
5	CSE
6	EE

- Q2: $\pi_{Position}(Faculty)$

Position
Assistant Professor

- Q3: $\pi Class(Faculty)$

Class
5
6

Class	Dept	Position
5	CSE	Assistant Professor
5	CSE	Assistant Professor
6	EE	Assistant Professor
6	EE	Assistant Professor

PROJECT

- PROJECT Operation Properties
  - The cardinality in the result of projection $\pi_{<list>}(R)$ is always less or equal to the number of tuples in R $\mathbf{1} \leq |\boldsymbol{\pi_A(R)}| \leq |\boldsymbol{R}|$
    - If the list of attributes includes a *key* of R, then the number of tuples in the result of PROJECT is *equal* to the number of tuples in R
  - The degree of resulting relation from a Project operation is equal to the number of attribute in the attribute list ‘A’.
  - In SQL, SELECT DISTINCT query is exactly as same as PROJECT here.

PROJECT

- PROJECT Operation Properties
  - PROJECT is *not* commutative
    - $\pi_{\text{Attribute List 1}}(\pi_{\text{Attribute List2}}(R)) \neq \pi_{\text{Attribute List 2}}(\pi_{\text{Attribute List1}}(R))$
    - $\pi_{\text{<list1>}}(\pi_{\text{<list2>}}(R)) = \pi_{\text{<list1>}}(R)$
      - As long as <list2> contains the attributes in <list1>.
      - Means only if Attribute List 1 is a subset of Attribute List 2.

EXTENDED PROJECT

- $\Pi_{\langle \text{STID, COID, } (1.2 * \text{GRADE}) \text{ RENAME AS G} \rangle}(\text{STCOT})$

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 **relational algebra expression** by nesting the operations, or
  - We can apply one operation at a time and create **intermediate result relations**.
- In the latter case, we must give names to the relations that hold the intermediate results.

Single expression versus sequence of relational operations

- 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 relational algebra 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:
  - $\text{DEP5\_EMPS} \leftarrow \sigma_{\text{DNO=5}}(\text{EMPLOYEE})$
  - $\text{RESULT} \leftarrow \pi_{\text{FNAME, LNAME, SALARY}}(\text{DEP5\_EMPS})$

RENAME

- 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

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

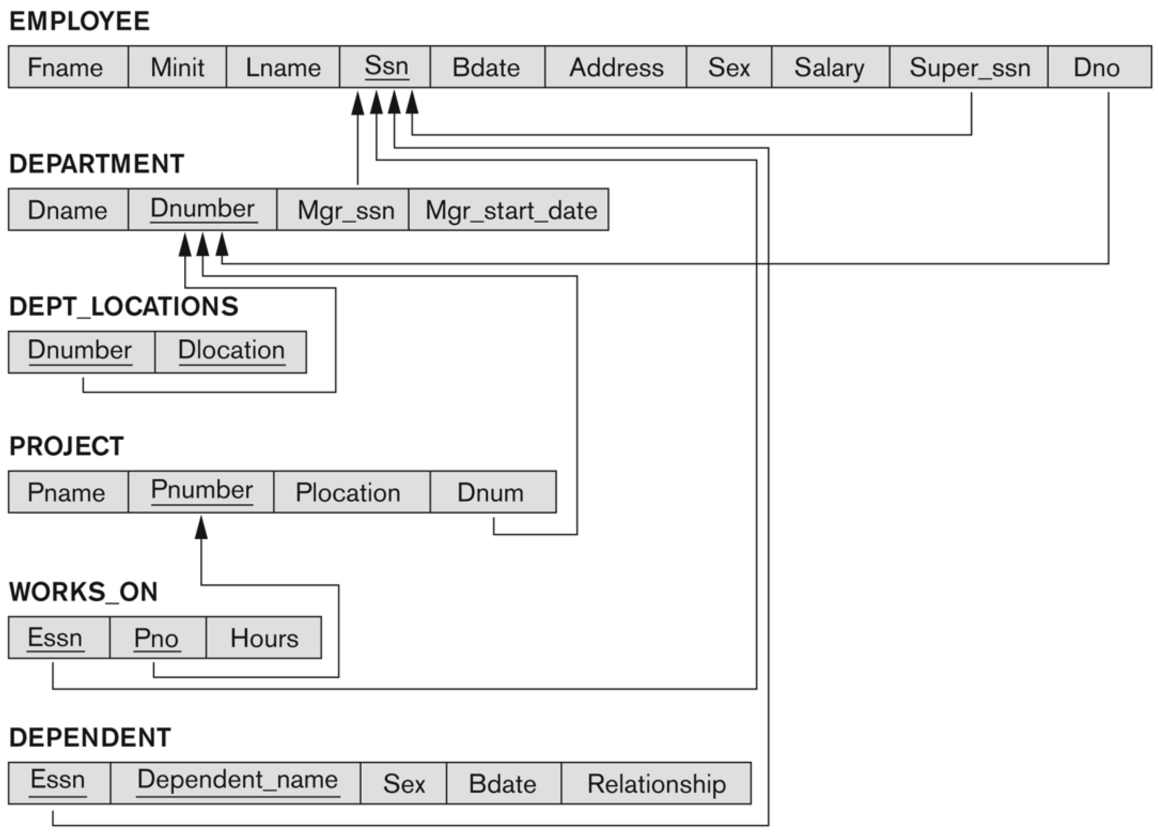
RENAME

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

Database State for COMPANY

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

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



Examples

- Query to rename the relation Student as Male Student and the attributes of Student RollNo and SName as (Sno, Name) with selecting some tuples with “Condition”.
  - $\rho_{\text{MaleStudent(Sno, Name)}}\ \pi_{\text{RollNo, SName}}(\sigma_{\text{Condition}}(\text{Student}))$
- Query to rename the attributes Name, Age of table Department to A,B.
  - $\rho_{\text{(A, B)}}\ (\pi_{\text{Name, Age}}\ \text{Department})$
- Query to rename the table name Project to Pro and its attributes to P, Q, R.
  - $\rho_{\text{Pro(P, Q, R)}}\ (\text{Project})$
- Query to rename the first attribute of the table Student with attributes A, B, C to P.
  - $\rho_{\text{(P, B, C)}}\ (\text{Student})$

Example

- Q.4: $\rho_{E\ (id,\ name,\ S)}\ \pi_{Eid,\ Ename,\ Salary}\ [\sigma_{Salary\ >10000}\ (Employee)]$

E		
id	name	S
201	P	20000

Employee		
Eid	Ename	Salary
201	P	20000
202	Q	5000
203	R	10000
204	P	7500

03

Operations From Set Theory

Relational Algebra Operations from Set Theory

- Type Compatibility of operands is required for the binary set operation UNION ∪, (also for INTERSECTION ∩, and SET DIFFERENCE –, see next slides)
- 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∪R2 (also for R1∩R2, or R1–R2, see next slides) has the same attribute names as the *first* operand relation R1 (by convention)

UNION, INTERSECTION, MINUS

- **Closedness property**: the result of evaluating any valid relational algebra expression is again a relation (which does not have duplicate tuples).
- For ∪ , ∩ , -, the operands should be Type Compatible.

$$H_{R_1} = H_{R_2}$$

$$R_3 = R_1 \ op \ R_2 \qquad op \in \{\cup, \cap, , -\}$$

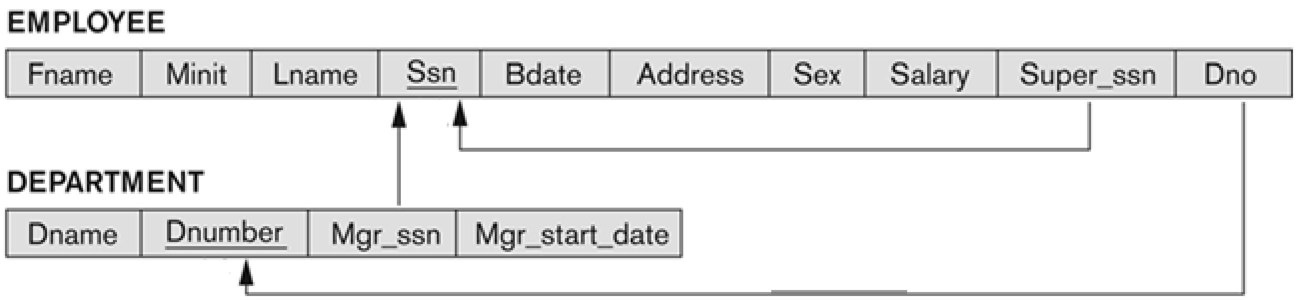
$$H_{R_3} = H_{R_1} = H_{R_2}$$

UNION

- 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 R and S
- Duplicate tuples are eliminated
- The two operand relations R and S must be “type compatible” (or UNION compatible)
  - R and S must have same number of attributes
  - Each pair of corresponding attributes must be type compatible (have same or compatible domains)
- In SQL, the operation UNION is as same as UNION operation here.
- Moreover, In SQL there is multiset operation UNION ALL.

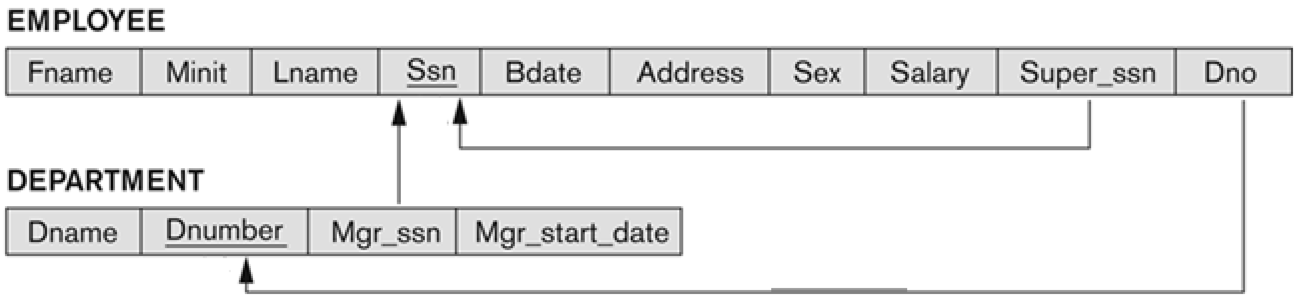
UNION

- 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)



UNION

- 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:

$$DEP5\_EMPS \leftarrow \sigma_{Dno=5}(EMPLOYEE)$$
$$RESULT1 \leftarrow \pi_{Ssn}(DEP5\_EMPS)$$
$$RESULT2 \leftarrow \rho_{SSN}(\pi_{Super\_ssn}(DEP5\_EMPS))$$
$$RESULT \leftarrow RESULT1 \cup RESULT2$$

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

UNION

- Results of Previous Question

Figure 6.3
Result of the UNION operation RESULT ← RESULT1 ∪ RESULT2.

RESULT1

Ssn
123456789
333445555
666884444
453453453

RESULT2

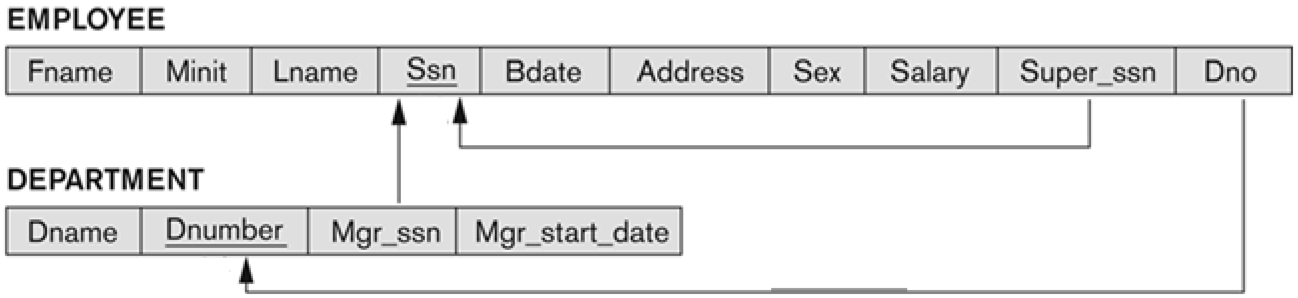
Ssn
333445555
888665555

RESULT

Ssn
123456789
333445555
666884444
453453453
888665555

UNION

- To retrieve the social security numbers of all employees who either work in department 5 or employee whose directly supervise works in department 5



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"

INTERSECTION

- The INTERSECTION operation is commutative, that is :

$$A \cap B = B \cap A$$

- The INTERSECTION is associative, that means it is applicable to any number of relation.

$$A \cap (B \cap C) = (A \cap B) \cap C$$

- In SQL, the operation INTERSECT is as same as INTERSECTION operation here.
- Moreover, In SQL there is multiset operation INTERSECT ALL.

SET DIFFERENCE (MINUS)

- 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”

MINUS

- The SET DIFFERENCE operation is not commutative, that means :

$$A - B \neq B - A$$

- In SQL, the operation EXCEPT is as same as MINUS operation here.
- Moreover, In SQL there is multiset operation EXCEPT ALL.

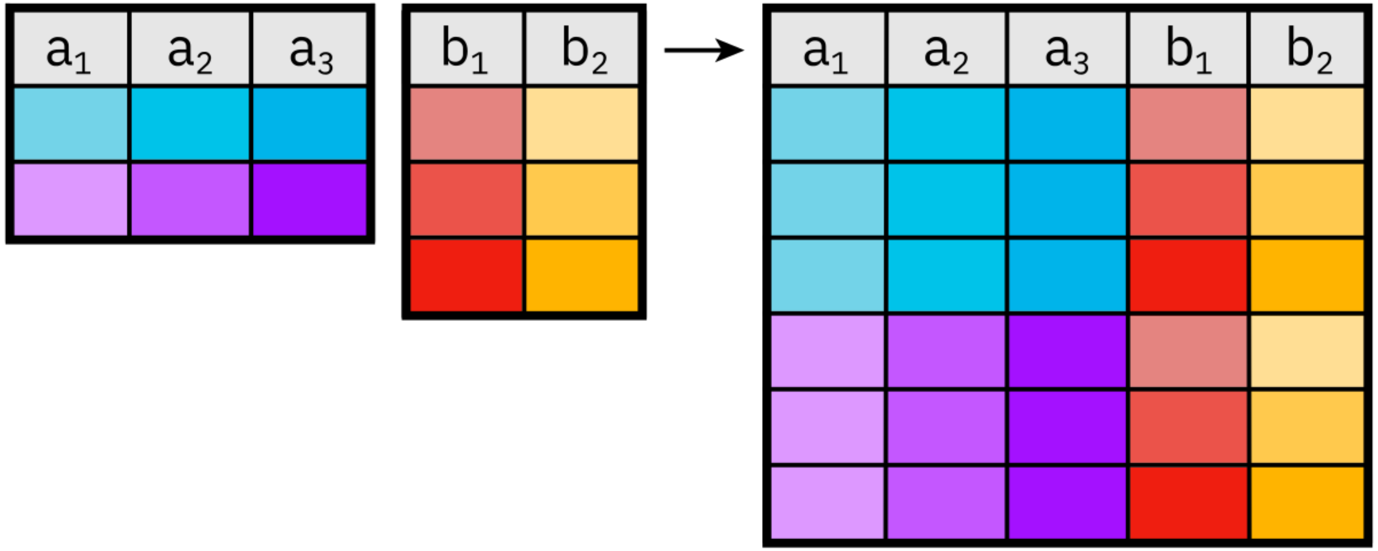
- INTERSECTION can be formed using UNION and MINUS as follows:

$$A \cap B = ((A \cup B) - (A - B)) - (B - A)$$

CARTESIAN PRODUCT/TIMES

- In theory, the two relation can not have an attribute with the same noun.

$$H_{R_2} \cap H_{R_1} = \varnothing$$



CARTESIAN PRODUCT

- CARTESIAN (or CROSS) PRODUCT Operation
  - This operation is used to combine tuples from two relations in a combinatorial fashion.
  - 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), in that order.
  - The resulting relation state has one tuple for each combination of tuples—one from R and one from S.
  - Hence, if R has n_R tuples (denoted as $|R| = n_R$), and S has n_S tuples, then R x S will have $n_R * n_S$ tuples.
  - The two operands do NOT have to be “type compatible”

CARTESIAN PRODUCT

- Generally, CROSS PRODUCT is not a meaningful operation
  - Can become meaningful when followed by other operations

- Example (not meaningful):
  - FEMALE_EMPS ← σ $_{SEX='F'}$(EMPLOYEE)
  - EMPNAMES ← π $_{FNAME, LNAME, SSN}$ (FEMALE_EMPS)
  - EMP_DEPENDENTS ← EMPNAMES x DEPENDENT

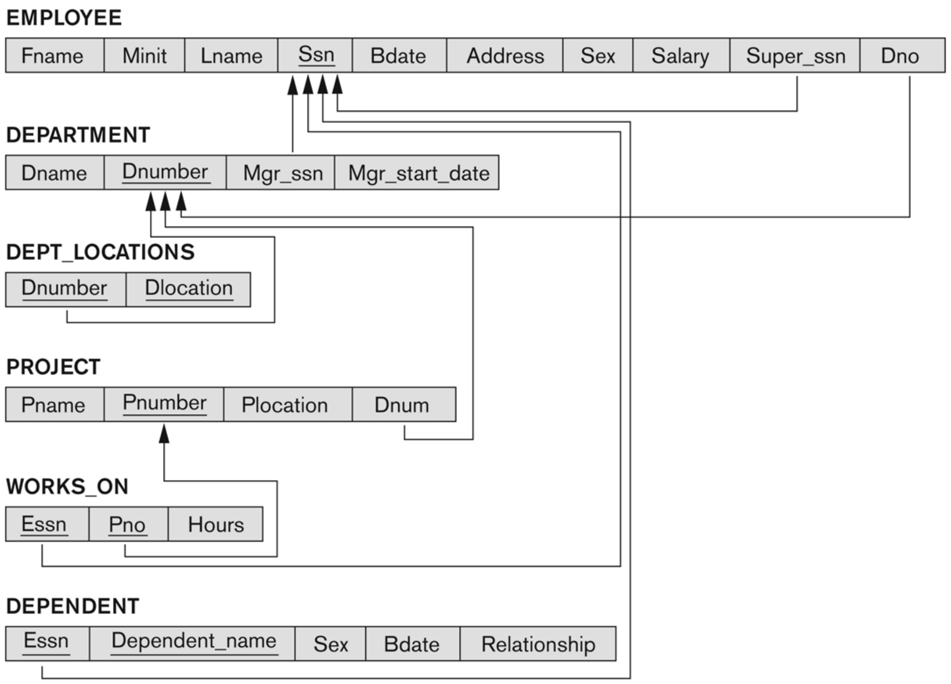
- EMP_DEPENDENTS will contain every combination of EMPNAMES and DEPENDENT
  - whether or not they are actually related

CARTESIAN PRODUCT

- To keep only combinations where the DEPENDENT is related to the EMPLOYEE, we add a SELECT operation as follows

- Example (meaningful):
  - FEMALE_EMPS ← σ $_{\text{SEX='F'}}$(EMPLOYEE)
  - EMPNAMES ← π $_{\text{FNAME, LNAME, SSN}}$ (FEMALE_EMPS)
  - EMP_DEPENDENTS ← EMPNAMES x DEPENDENT
  - ACTUAL_DEPS ← σ $_{\text{SSN=ESSN}}$(EMP_DEPENDENTS)
  - RESULT ← π $_{\text{FNAME, LNAME, DEPENDENT\_NAME}}$ (ACTUAL_DEPS)

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



CARTESIAN PRODUCT

- 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 x 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

- Write the following with MINUS, INTERSECT, UNION
  - R **WHERE** (C_1 **AND** C_2)
  - R **WHERE** (C_1 **OR** C_2)
  - R **WHERE NOT** C

Example

- Write the following with MINUS, INTERSECT, UNION
  - R **WHERE** (C_1 **AND** C_2)
    - (R **WHERE** C_1) **INTERSECT** (R **WHERE** C_2)
  - R **WHERE** (C_1 **OR** C_2)
    - (R **WHERE** C_1) **UNION** (R **WHERE** C_2)
  - R **WHERE NOT** C
    - R **MINUS** (R **WHERE** C)

04

Binary Relational Operations

JOIN

- JOIN Operation (denoted by ⋈)
  - 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
  - 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

- 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 $\bowtie_{\mathbf{condition}}$ operation.
  - DEPT_MGR $\leftarrow$ 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

- Give full details of producer-piece pairs from a city.

$R_1 := S \bowtie_{S.CITY=P.PCITY} (P \text{ RENAME CITY AS PCITY})$

S (S#, SNAME, STATUS, CITY)

S#	CITY
S1	C1
S2	C2
S3	C3
S4	C4
S5	C5
S6	C6

P (P#, . . . , W, CITY)

P#	W	CITY
P1	5	C1
P2	6	C2
P3	4	C1
P4	7	C4
P5	10	C5

R_1 (S#, . . . , CITY, P#, . . . , W, PCITY)

S#	CITY	P#	W	PCITY
S1	C1	P1	5	C1
S1	C1	P3	4	C1
S2	C2	P2	6	C2
~~S3~~ Not joinable				
S4	C4	P4	7	C4
S5	C5	P5	10	C5
~~S6~~ Not joinable				

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 tuples, and S has n_S tuples, then the join result will generally have *less than* $n_R \times n_S$ tuples.
  - Only related tuples (based on the join condition) will appear in the result. In this example:

$R.A_i = S.B_j$

Must be the same domain and not the same name.

Theta-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

Operations in join condition include:

- EQUI–JOIN $=$
- NOT EQUI–JOIN $\neq$
- LESS THAN–JOIN $<$
- LESS EQUI–JOIN $\leq$
- GREATER THAN–JOIN $>$
- GREATER EQUI–JOIN $\geq$

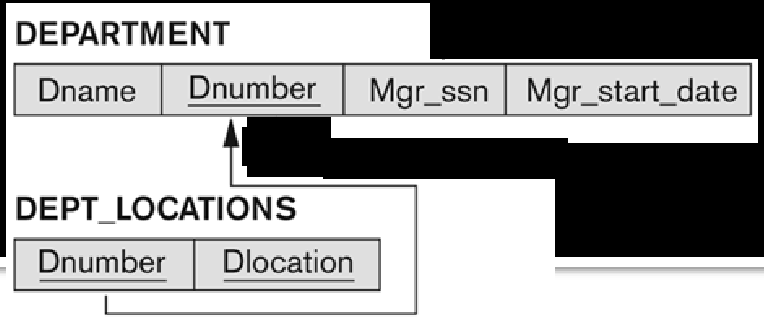
EQUIJOIN

- The most common use of join involves join conditions with only equality comparisons.
- 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.

NATURAL JOIN

- NATURAL JOIN Operation
  - Another variation of JOIN called NATURAL JOIN **— denoted by * or ⋈**) — was created to get rid of the second (superfluous) attribute in an EQUIJOIN condition.
    - Because one of each pair of attributes with identical values is superfluous
  - 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



- Example: To 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
- 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)

DIVISION

- Division operator $A \div B$ or A/B can be applied if and only if:

  A proper subset of a set A is a subset of A that is not equal to A.

  - Attributes of B is proper subset of Attributes of A.
  - The relation returned by division operator will have attributes = (All attributes of A – All Attributes of B)
    - The relation returned by division operator will return those tuples from relation A which are associated to every B's tuple.

$$\begin{cases} R_1(\overbrace{A_1, A_2, \ldots, A_n}^{X}, \overbrace{B_1, B_2, \ldots, B_m}^{Y}) \\ R_2(B_1, B_2, \ldots, B_m) \end{cases}$$

$$R_3(X) \coloneqq R_1(X,Y) \div R_2(Y) \longrightarrow H_{R_2} \subseteq H_{R_1}$$

DIVISION

- The division operation is applied to two relations $R(Z) \div S(X)$, where X subset Z. Let $Y = Z - X$ (and hence $Z = X \cup Y$); that is, let Y be the set of attributes of R that are not attributes of S.

- The result of DIVISION is a relation T(Y) that includes a tuple t if tuples t_R appear in R with $t_R[Y] = t$, and with $t_R[X] = t_s$ for every tuple t_s in S.

- For a tuple t to appear in the result T of the DIVISION, the values in t must appear in R in combination with every tuple in S.
- For $R_3 = R_1 \div R_2$

$$0 \leq \text{card}(R_3) \leq \left\lfloor \frac{\text{card}(R_1)}{\text{card}(R_2)} \right\rfloor$$

Example of DIVISION

(a)

SSN_PNOS

Essn	Pno
123456789	1
123456789	2
666884444	3
453453453	1
453453453	2
333445555	2
333445555	3
333445555	10
333445555	20
999887777	30
999887777	10
987987987	10
987987987	30
987654321	30
987654321	20
888665555	20

SMITH_PNOS

Pno
1
2

SSNS

Ssn
123456789
453453453

(b)

R

A	B
a1	b1
a2	b1
a3	b1
a4	b1
a1	b2
a3	b2
a2	b3
a3	b3
a4	b3
a1	b4
a2	b4
a3	b4

S

A
a1
a2
a3

T

B
b1
b4

Figure 6.8
The DIVISION operation. (a) Dividing SSN_PNOS by SMITH_PNOS. (b) $T \leftarrow R \div S$.

DIVISION

- Candidate key of $R_3 = R_1 \div R_2$:
  - 1) If C.K of R1 is in R3 header, then C.K(R3)=C.K(R1)

a	b	c	d	e
1	1	4	1	2
1	2	4	1	2
2	1	5	1	2
3	2	4	1	2
4	4	6	2	1

÷

d	e
1	2

=

a	b	c
1	1	4
1	2	4
2	1	5
3	2	4

  - 2) If C.K of R1 is not in R3 header, then C.K(R3)=all keys

a	b	c	d	e
5	6	7	1	2
5	6	7	3	4
8	2	3	6	7
6	4	5	3	4
7	8	9	1	2
6	4	5	1	2
5	6	7	8	9
5	6	8	1	2
5	6	8	3	4
6	2	5	1	2
6	2	5	3	4
2	6	8	1	2
2	6	8	3	4

÷

d	e
1	2
3	4

=

a	b	c
5	6	7
6	4	5
5	6	8
6	2	5
2	6	8

Recap of Relational Algebra Operations

Operation	Purpose	Notation
SELECT	Selects all tuples that satisfy the selection condition from a relation R.	$\sigma_{\text{<selection condition>}}(R)$
PROJECT	Produces a new relation with only some of the attributes of R, and removes duplicate tuples.	$\pi_{\text{<attribute list>}}(R)$
THETA JOIN	Produces all combinations of tuples from R_1 and R_2 that satisfy the join condition.	$R_1 \bowtie_{\text{<join condition>}} R_2$
EQUIJOIN	Produces all the combinations of tuples from R_1 and R_2 that satisfy a join condition with only equality comparisons.	$R_1 \bowtie_{\text{<join condition>}} R_2$, OR $R_1 \bowtie_{(\text{<join attributes 1>}),(\text{<join attributes 2>})} R_2$
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.	$R_1 *_{\text{<join condition>}} R_2$, OR $R_1 *_{(\text{<join attributes 1>}),(\text{<join attributes 2>})} R_2$ 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 $Z = X \cup Y$.	$R_1(Z) \div R_2(Y)$

Example

- Write the Optimal Relational Algebra:

Relation "S"

S#	Sname	City
S1	Sn1	C1
S2	Sn2	C2
S3	Sn3	C2

Relation "P"

P#	Pname	Color
P1	Pn1	Red
P2	Pn2	Blue

Relation "SP"

S#	P#	QTY
S1	P1	10
S1	P2	20
S2	P1	30

  - S# of suppliers who have produced at least one product
    - Solution: $\Pi_{S\#}(SP)$
  - Specifications of suppliers who have produced at least one product
    - Solution: $\Pi_{S\#,Sname,City}(S \bowtie SP)$ The optimal version is: $S \bowtie (\Pi_{S\#} SP)$
  - S# of suppliers who have produced all products
    - Solution: $\Pi_{S\#,P\#}(SP) \div \Pi_{P\#}(P)$

05

Additional Relational Operations

Aggregate Functions and Grouping

- A type of request that cannot be expressed in the basic relational algebra is to specify mathematical **aggregate functions** on collections of values from the database.
- Examples of such functions include 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 applied to collections of numeric values include
  - SUM, AVERAGE, MAXIMUM, and MINIMUM.
- The COUNT function is used for counting tuples or values.

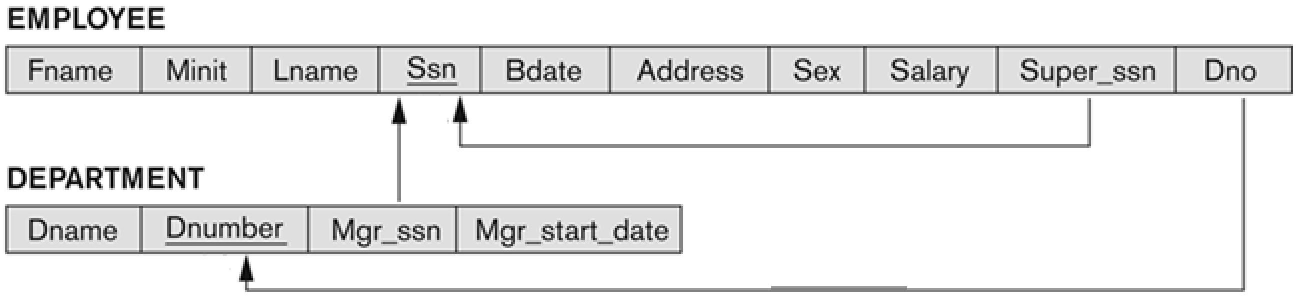
Aggregate Function Operation

- 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**

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



Using Grouping with Aggregation

- 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

OUTER JOIN

- The OUTER JOIN Operation
  - In NATURAL JOIN and EQUIJOIN, tuples without a *matching* (or *related*) tuple are eliminated from the join result
    - Tuples with null in the join attributes are also eliminated
    - This amounts to loss of information.
  - A set of operations, called OUTER joins, can be used when we want to keep all the tuples in R, or all those in S, or all those in both relations in the result of the join, regardless of whether or not they have matching tuples in the other relation.

LEFT & Right OUTER JOIN

- The left outer join operation keeps every tuple in the first or left relation R in R ⟕ S; if no matching tuple is found in S, then the attributes of S in the join result are filled or “padded” with null values.
- A similar operation, right outer join, keeps every tuple in the second or right relation S in the result of R ⟖ S
- A third operation, full outer join, denoted by ⟗ keeps all tuples in both the left and the right relations when no matching tuples are found, padding them with null values as needed.

LEFT & Right OUTER JOIN

- Example

$$R_3 = R_1 =\!\bowtie R_2$$

a	b
1	2
3	4
5	6
7	2

$=\!\bowtie$

b	c	d
2	5	1
6	3	2
2	8	3
9	6	7
7	4	5

=

a	b	c	d
1	2	5	1
1	2	8	3
5	6	3	2
7	2	5	1
7	2	8	3
3	4	NULL	NULL

LEFT & Right OUTER JOIN

- For $R_3 = R_1 ⟕ R_2$ with *k* same columns:
  - Degree: $\deg(R_3) = (\deg(R_1) + \deg(R_2)) - k$
  - Cardinality: $card(R_1 \bowtie R_2) \leq card(R_3) \leq card(R_1) \times card(R_2)$

LEFT SEMI-JOIN

- A Left Semi-join returns rows from the left table for which there are corresponding matching rows in the right table.
- Unlike regular joins which include the matching rows from both tables, a left semi-join only includes columns from the left table in the result.
  - $R_3 \coloneqq R_1 \ltimes_C R_2 = \Pi_{\langle H_{R_1} \rangle}(R_1 \bowtie_C R_2)$
  - Example:

Customers table:

Customer_ID	Customer_Name
01	Alice
02	Bob
03	Charlie
04	David

Orders Table

Customer_ID	Order_ID	Order_Name
02	101	Stationery
01	102	Books
04	103	Pens

Output:

Customer_ID	Customer_Name
01	Alice
02	Bob
04	David

RIGHT SEMI-JOIN

- A Right Semi-join returns rows from the right table for which there are corresponding matching rows in the left table.
  - $R_3 := R_1 \rtimes R_2 = \Pi_{\langle H_{R_2} \rangle}(R_1 \bowtie_C R_2)$
  - Example:

a	b
1	2
3	4
5	6
7	2

$\rtimes$

b	c	d
2	5	1
6	3	2
2	8	3
9	6	7
7	4	5

=

b	c	d
2	5	1
2	8	3
6	3	2

SEMI-JOIN

- $\deg(R_1 \ltimes R_2) = \deg(R_1)$
- $\text{card}(R_1 \ltimes R_2) \leq \text{card}(R_1)$
- $\text{card}(R_1 \ltimes R_2) \leq \text{card}(R_1 \bowtie R_2)$
- $C.K(R_1 \ltimes R_2) = C.K(R_1)$
- $R_1 \ltimes R_2 \neq R_2 \ltimes R_1$

- $\deg(R_1 \rtimes R_2) = \deg(R_2)$
- $\text{card}(R_1 \rtimes R_2) \leq \text{card}(R_2)$
- $\text{card}(R_1 \rtimes R_2) \leq \text{card}(R_1 \bowtie R_2)$
- $C.K(R_1 \rtimes R_2) = C.K(R_2)$
- $R_1 \rtimes R_2 \neq R_2 \rtimes R_1$

- If $H_{R1} = H_{R2}$ then $R_1 \rtimes R_2 = R_1 \ltimes R_2 = R_1 \bowtie R_2 = R_1 \cap R_2$

ANTI-JOIN

- R ▷ S
- It is Exactly the opposite to semi-join.
- An Anti-join returns rows from the left table for which there are no corresponding matching rows in the right table.
- Example:

Output:

Customer_ID	Customer_Name
03	Charlie

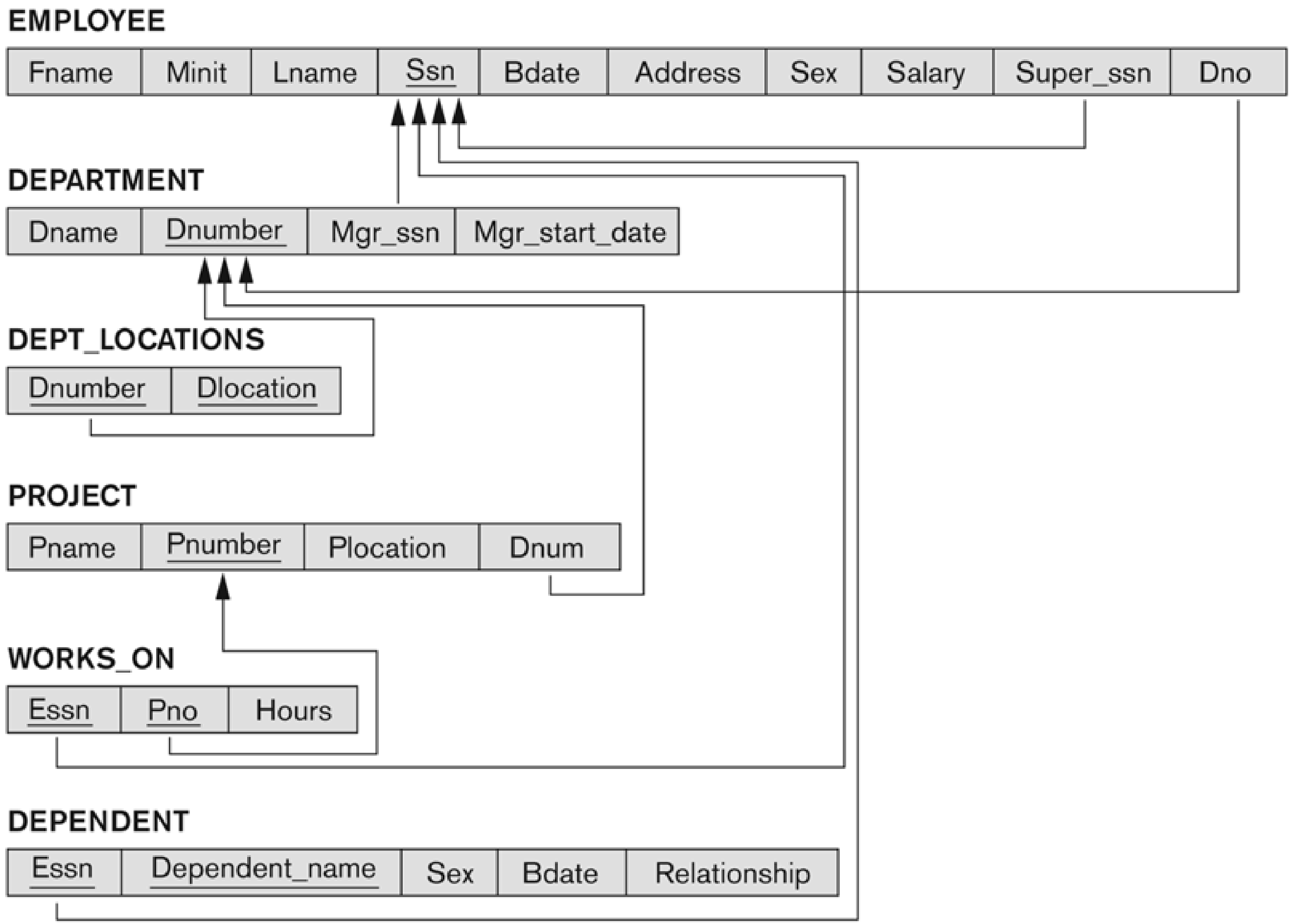
SEMI-MINUS

- R_3 **=** R_1 **SEMIMINUS** $R_2 = R_1$ **MINUS** $(R_1$ **SEMIJOIN** $R_2)$
- $H_{R_3} = H_{R_1}$

Examples of Queries in Relational Algebra

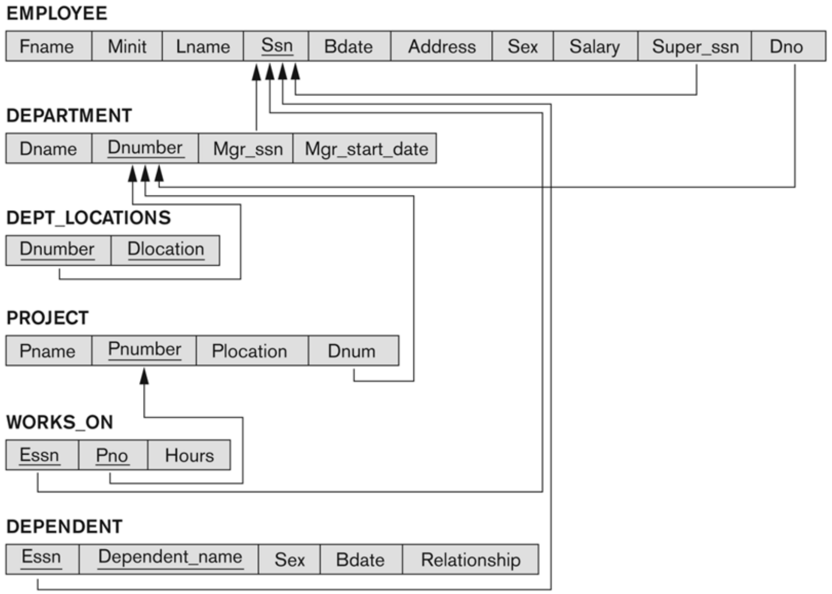
- Retrieve the name and address of all employees who work for the 'Research' department.

- Retrieve the names of employees who have no dependents.



Examples of Queries in Relational Algebra

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



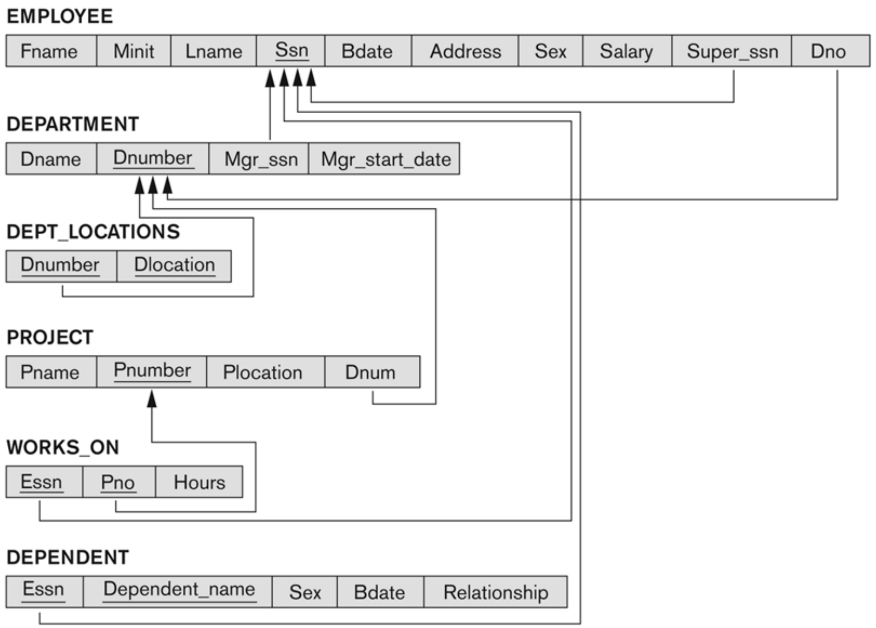
$RESEARCH\_DEPT \leftarrow \sigma_{DNAME='Research'}(DEPARTMENT)$

$RESEARCH\_EMPS \leftarrow (RESEARCH\_DEPT \bowtie_{DNUMBER=DNO} EMPLOYEE)$

$RESULT \leftarrow \pi_{FNAME, LNAME, ADDRESS}(RESEARCH\_EMPS)$

Examples of Queries in Relational Algebra

- Retrieve the names of employees who have no dependents.



$$ALL\_EMPS \leftarrow \pi_{SSN}(EMPLOYEE)$$

$$EMPS\_WITH\_DEPS(SSN) \leftarrow \pi_{ESSN}(DEPENDENT)$$

$$EMPS\_WITHOUT\_DEPS \leftarrow (ALL\_EMPS - EMPS\_WITH\_DEPS)$$

$$RESULT \leftarrow \pi_{LNAME,\ FNAME}\ (EMPS\_WITHOUT\_DEPS * EMPLOYEE)$$

05

Complete Set of Relational Operations

Complete Set of Relational Operations

- The set of operations including SELECT σ, PROJECT π , UNION ∪, DIFFERENCE – , RENAME ρ, and CARTESIAN PRODUCT × is called a complete set because any other relational algebra expression can be expressed by a combination of these five operations.
- For example:
  - $R \cap S = (R \cup S) - \big((R - S) \cup (S - R)\big) = R - (R - S)$
  - $R \bowtie_{condition} S = \sigma_{condition} (R \times S)$

DIVISION

- Division can be expressed in terms of Cross Product , Set Difference and Projection. How??

DIVISION

- **Division can be expressed in terms of Cross Product , Set Difference and Projection. How??**

Let R(A,B) and S(A), we want to do R÷S.

take $T_1 = \pi_B(R)$ using project operator
take $T_2 = S \times T_2$ (cross product)
take $T_3 = T_2 - R$
take $T_4 = \pi_B(T_3)$ using project operator
$Result = T_1 - T_4$

Thus, we implemented Division operator using Project, Difference and Cross product which are all present in Minimal set of operators

NATURAL JOIN

- Natural join can be expressed in terms of Cross Product , Select and Projection. How??

Example

a	b
1	2
3	4
5	6
7	2

r

b	c	d
2	5	1
6	3	2
2	8	3
9	6	7
7	4	5

s

$$r \bowtie s = \Pi_{R \cup S}(\sigma_{r.b=s.b}(r \times s)) = \Pi_{a,s.b,c,d}(\sigma_{r.b=s.b}(r \times s))$$

a	b	c	d
1	2	5	1
1	2	8	3
5	6	3	2
7	2	5	1
7	2	8	3

Natural Join

a	b
1	2
3	4
5	6
7	2

×

b	c	d
2	5	1
6	3	2
2	8	3
9	6	7
7	4	5

=

a	r.b	s.b	c	d
1	2	2	5	1
1	2	2	8	3
5	6	6	3	2
7	2	2	5	1
7	2	2	8	3

→

a	s.b	c	d
1	2	5	1
1	2	8	3
5	6	3	2
7	2	5	1
7	2	8	3

cross product, select & project

LEFT-OUTER JOIN

- Left outer join can be expressed in terms of complete set. How??

Example

a	b
1	2
3	4
5	6
7	2

r

b	c	D
2	5	1
6	3	2
2	8	3
9	6	7
7	4	5

s

$$r ⟕ s = \left(\Pi_{a,s.b,c,d}\left(\sigma_{r.b = s.b}(r \times s)\right)\right) \cup \left(\left(r - \Pi_{a,r.b}\left(\sigma_{r.b = s.b}(r \times s)\right)\right) \times \{NULL, NULL\}\right)$$

a	b	c	d
1	2	5	1
1	2	8	3
5	6	3	2
7	2	5	1
7	2	8	3
3	4	NULL	NULL

Left Outer Join

Any Question?