



Relational Algebra

Database Design

Department of Computer Engineering

Sharif University of Technology

Maryam Ramezani maryam.ramezani@sharif.edu



- ❑ Informally, a **relation** looks like a **table** of values.
- ❑ A relation typically contains a **set of rows**.
- ❑ The data elements in each **row** represent certain facts that correspond to a real-world **entity** or **relationship**
 - In the formal model, rows are called **tuples**
- ❑ Each **column** has a column header that gives an indication of the meaning of the data items in that column
 - In the formal model, the column header is called an **attribute name** (or just **attribute**)



❑ Key of a Relation:

- Each row has a value of a data item (or set of items) that uniquely identifies that row in the table
 - Called the *key*
- Sometimes row-ids or sequential numbers are assigned as keys to identify the rows in a table
 - Called *artificial key* or *surrogate key*



- ❑ A **tuple** is an **ordered set of values** (enclosed in angled brackets ' $\langle \dots \rangle$ ')
- ❑ 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
- $\langle 632895, \text{"John Smith"}, \text{"101 Main St. Atlanta, GA 30332"}, \text{"(404) 894-2000"} \rangle$
- This is called a 4-tuple as it has 4 values
- A tuple (row) in the CUSTOMER relation.



- ❑ A **relation** R is a **set** of such tuples (rows) having:
 - Heading: H_R or $R(H)$ e.g: For $R(A_1, \dots, A_m)$ $R(H) = \{A_1, \dots, A_m\}$
 - $R(H)$ is constant over time.
 - Change in $R(H)$ makes a new relation.
 - Body: $r(R)$ set of **tuples**
 - It is changeable over the time.
- ❑ **Degree**: Number of heading.
- ❑ **Cardinality**: Number of rows.
- ❑ Think about a relation which the number of domains is smaller than degree?
 - $\text{Emp}(\text{ID}, \text{Name}, \text{ManagerID})$: the EmpID and ManagerID are from the same domain of EmpID .



- ❑ Formally,
 - Given $R(A_1, A_2, \dots, A_n)$
 - $r(R) \subset \text{dom}(A_1) \times \text{dom}(A_2) \times \dots \times \text{dom}(A_n)$
- ❑ $R(A_1, A_2, \dots, A_n)$ is the **schema** of the relation
- ❑ R is the **name** of the relation
- ❑ A_1, A_2, \dots, A_n are the **attributes** of the relation
- ❑ **Relation state** $r(R)$: a specific **state** (or “value” or “population”) of relation R – this is a *set of tuples* (rows)
 - $r(R) = \{t_1, t_2, \dots, t_n\}$ where each t_i is an n -tuple
 - $t_i = \langle v_1, v_2, \dots, v_n \rangle$ where each v_j *element-of* $\text{dom}(A_j)$

Formal Definitions – Example



❑ Let $R(A1, A2)$ be a relation schema:

- Let $\text{dom}(A1) = \{0,1\}$
- Let $\text{dom}(A2) = \{a,b,c\}$

Then: $\text{dom}(A1) \times \text{dom}(A2)$ is all possible combinations:

$\{\langle 0,a \rangle, \langle 0,b \rangle, \langle 0,c \rangle, \langle 1,a \rangle, \langle 1,b \rangle, \langle 1,c \rangle\}$

❑ The relation state $r(R) \subset \text{dom}(A1) \times \text{dom}(A2)$

❑ For example: $r(R)$ could be $\{\langle 0,a \rangle, \langle 0,b \rangle, \langle 1,c \rangle\}$

- this is one possible state (or “population” or “extension”) r of the relation R , defined over $A1$ and $A2$.
- It has three 2-tuples: $\langle 0,a \rangle, \langle 0,b \rangle, \langle 1,c \rangle$

Definition Summary



<u>Informal Terms</u>		<u>Formal Terms</u>
Table		Relation
Column Header		Attribute
All possible Column Values		Domain
Row		Tuple

Example – A relation STUDENT

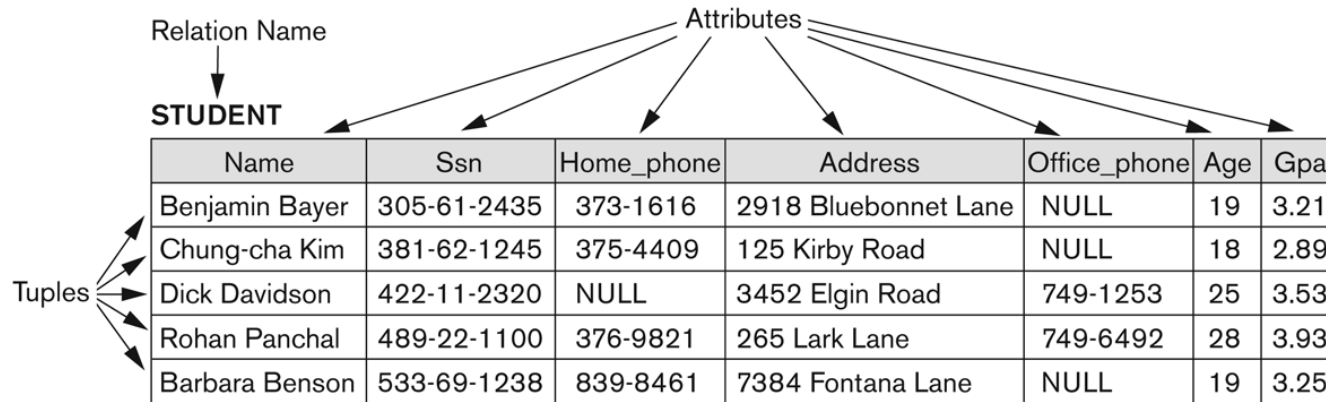


Figure 5.1

The attributes and tuples of a relation STUDENT.



- ❑ 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 $R(A_1, A_2, \dots, A_n)$ and the values in $t = \langle v_1, v_2, \dots, v_n \rangle$ 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**



❑ 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 $t = \langle v_1, v_2, \dots, v_n \rangle$ is a tuple (row) in the relation state r of $R(A_1, A_2, \dots, A_n)$
 - Then each v_i must be a value from $dom(A_i)$
- 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 $m \geq n$



□ Notation:

- We refer to **component values** of a tuple t by:
 - $t[A_i]$ or $t.A_i$
 - This is the value v_i of attribute A_i for tuple t
- Similarly, $t[A_u, A_v, \dots, A_w]$ refers to the subtuple of t containing the values of attributes A_u, A_v, \dots, A_w , respectively in t

Example



```
CREATE DOMAIN SN      CHAR(8) DEFAULT '00000000'  
CREATE DOMAIN SNAME  CHAR(20)DEFAULT 'noname'  
CREATE DOMAIN SJ     CHAR(4)  DEFAULT '?...?'  
CREATE DOMAIN SL     CHAR(3)  DEFAULT '?...?'  
CREATE DOMAIN SD     CHAR(4)  DEFAULT '?...?'  
CREATE DOMAIN CN     CHAR(6)  DEFAULT '?...?'  
CREATE DOMAIN GRADE  DEC(2, 2) DEFAULT '?...?'
```

```
CREATE RELATEION STT  
  (STID DOMAIN SN,  
   STNAME DOMAIN SNAME,  
   STJ DOMAIN SJ,  
   STL DOMAIN STL,  
   STD DOMAIN SD)
```

SQL: CREATE DOMAIN

SQL: CREATE TABLE

```
CREATE RELATION COT ....
```

```
CREATE RELATION STCOT ...
```



❑ 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 $r(R)$, $t1[SK] \neq t2[SK]$
 - 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)



❑ 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.
 - $t[PK] \neq \text{null}$ 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.

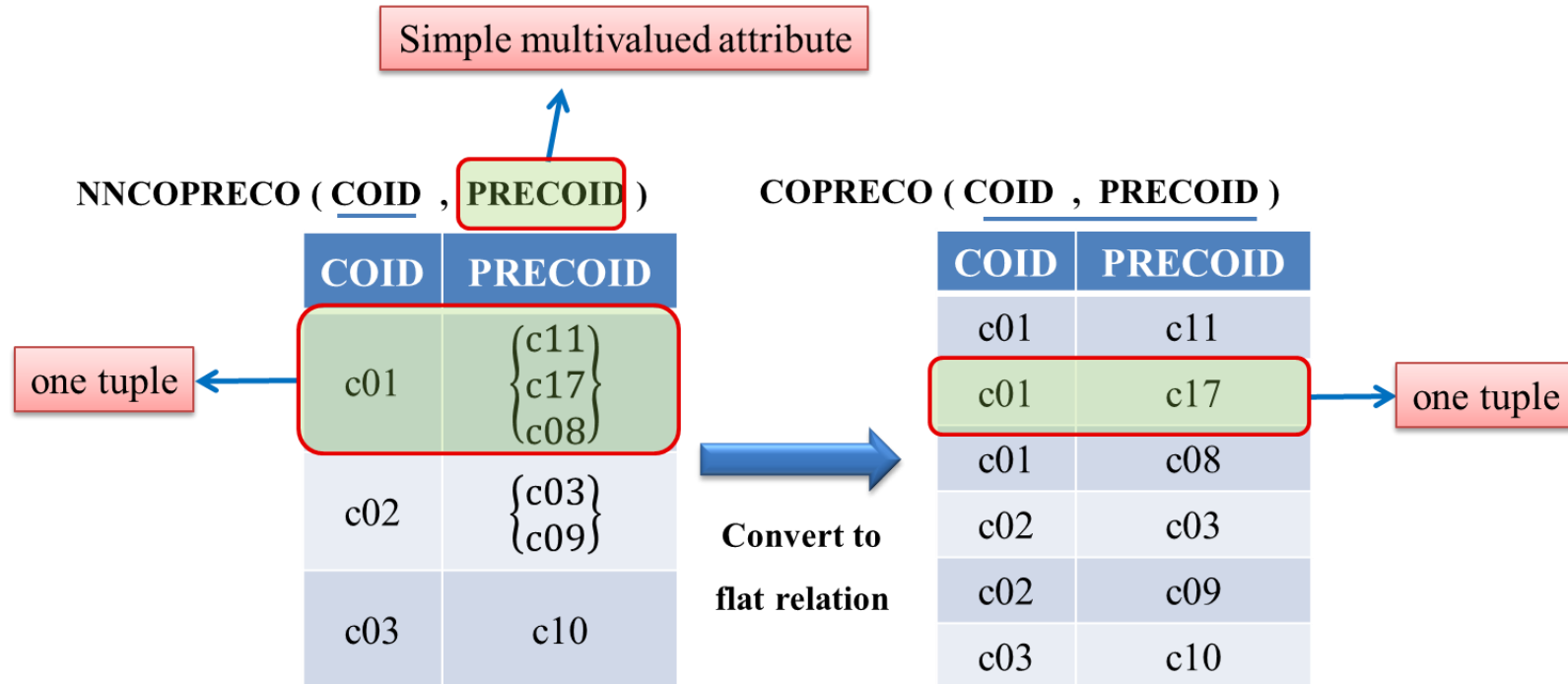


- ❑ Base relation
 - CREATE TABLE
- ❑ virtual relation:
 - CREATE VIEW
- ❑ Temporary relation
 - CREATE TEMPORARY TABLE
- ❑ Instant relation
 - CREATE MATERIALIZED VIEW
- ❑ Derivative relation (relation on other relations)

Flat and Nested Relation



- ❑ Flat Relation: a relation which has single value in all attributes.
- ❑ Nested Relation: a relation with at least one multivalued attribute.



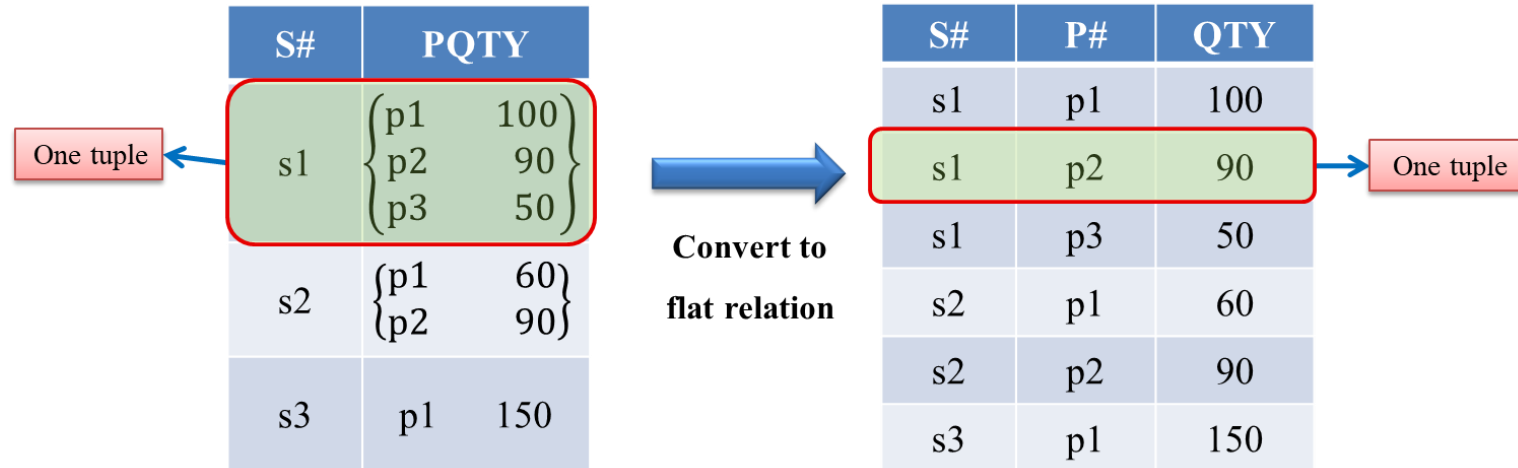
Flat and Nested Relation



Composite multi-valued attribute
P# , QTY

NNSP (S# , **PQTY**)

SP (S# , P# , QTY)



❑ What is difference between two following INSERTs in the above tables?

- $I_1 : < s4 , p4 , 40 >$
- $I_2 : < s2 , p3 , 30 >$



- ❑ Much of the theory behind the relational model was developed with this assumption in mind, which is called the **first normal form** assumption.

- ❑ **Advances of flat relation:**
 - Good representation
 - Running DDL, DML, and DCL commands
 - Simple conditions in where clause `SELECT ... FROM ... WHERE A<(=)>` 'Single Value'

- ❑ **Disadvantages of flat relation:**
 - Longer primary key
 - Data redundancy
 - Slow query speed for complex queries



❑ Relational Algebra consists of several groups of operations

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

Relational Algebra Operations



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

$$\begin{aligned} H_{R_1} &= H_{R_2} \\ R_3 &= R_1 \text{ op } R_2 \quad \text{op} \in \{\cup, \cap, -, \} \\ H_{R_3} &= H_{R_1} = H_{R_2} \end{aligned}$$



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

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

- ❑ What is the SQL form?

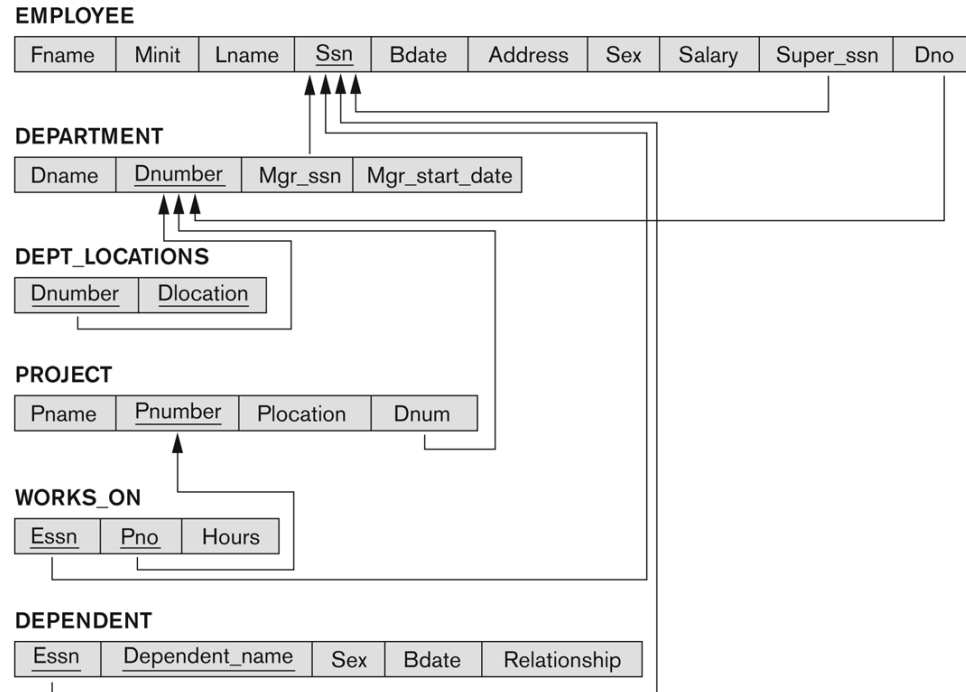
- **select** * **from** A, B
- **select** * **from** A **cross join** B



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

Figure 5.7

Referential integrity constraints displayed on the COMPANY relational database schema.





❑ 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*)

❑ Examples:

- Select the EMPLOYEE tuples whose department number is 4:

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

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

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

$$\sigma_{Place = 'Mumbai' \text{ or } Salary \geq 1000000} (Citizen)$$

$$\sigma_{Department = 'Analytics'} (\sigma_{Location = 'NewYork'} (Manager))$$

This is called nested expression, here, as usual, we evaluate the inner expression first (which results in relation say Manager1), then we calculate the outer expression on Manager1 (the relation we obtained from evaluating the inner expression), which results in relation again, which is an instance of a relation we input.



□ In general, the *select* operation is denoted by $\sigma_{\langle \text{selection condition} \rangle}(R)$ 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 Operation Properties

- The SELECT operation $\sigma_{\langle \text{selection_condition} \rangle}(R)$ produces a relation S that has the same schema (same attributes) as R
- SELECT is commutative:
 - $\sigma_{\langle \text{condition1} \rangle}(\sigma_{\langle \text{condition2} \rangle}(R)) = \sigma_{\langle \text{condition2} \rangle}(\sigma_{\langle \text{condition1} \rangle}(R))$
- Because of commutativity property, a cascade (sequence) of SELECT operations may be applied in any order:
 - $\sigma_{\langle \text{cond1} \rangle}(\sigma_{\langle \text{cond2} \rangle}(\sigma_{\langle \text{cond3} \rangle}(R))) = \sigma_{\langle \text{cond2} \rangle}(\sigma_{\langle \text{cond3} \rangle}(\sigma_{\langle \text{cond1} \rangle}(R)))$
- A cascade of SELECT operations may be replaced by a single selection with a conjunction of all the conditions:
 - $\sigma_{\langle \text{cond1} \rangle}(\sigma_{\langle \text{cond2} \rangle}(\sigma_{\langle \text{cond3} \rangle}(R))) = \sigma_{\langle \text{cond1} \rangle \text{ AND } \langle \text{cond2} \rangle \text{ AND } \langle \text{cond3} \rangle}(R)$

❑ The cardinality (number of tuples) 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 (number of attributes) of resulting relation from a Selection operation is same as the degree of the Relation given.



- ❑ If $R' = \sigma_c(R)$ then what the candidate key of R' ?
 - $CK_{R'} \subseteq CK_R$
- ❑ Equivalent expressions ($\sigma_{c_1}(\sigma_{c_2}(R)) = \sigma_{c_2}(\sigma_{c_1}(R)) = \sigma_{c_1 \wedge c_2}(R)$)
- ❑ Write the following with MINUS, INTERSECT, UNION
 - $R \text{ WHERE } (C_1 \text{ AND } C_2)$
 - $(R \text{ WHERE } C_1) \text{ INTERSECT } (R \text{ WHERE } C_2)$
 - $R \text{ WHERE } (C_1 \text{ OR } C_2)$
 - $(R \text{ WHERE } C_1) \text{ UNION } (R \text{ WHERE } C_2)$
 - $R \text{ WHERE NOT } C$
 - $R \text{ MINUS } (R \text{ WHERE } C)$



- ❑ PROJECT Operation is denoted by $\pi(\rho)$
- ❑ 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:

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



- ❑ The general form of the project operation is:

$$\pi_{\langle \text{attribute list} \rangle}(R)$$

- π (pi) is the symbol used to represent the project operation
 - $\langle \text{attribute list} \rangle$ 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.

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

Class	Dept
5	CSE
6	EE

❑ Q2: $\pi_{\text{Position}}(\text{Faculty})$

Position
Assistant Professor

❑ Q3: $\pi_{\text{Class}}(\text{Faculty})$

Class
5
6

Relation of this table!

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



❑ PROJECT Operation Properties

- The cardinality (number of tuples) in the result of projection $\pi_{\langle \text{list} \rangle}(R)$ is always less or equal to the number of tuples in R $1 \leq \pi A(R) \leq |R|$
 - 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 (number of attributes) 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 Operation Properties

- PROJECT is *not commutative*

- $\pi_{\text{Attribute List 1}}(\pi_{\text{Attribute List 2}}(R)) \neq \pi_{\text{Attribute List 2}}(\pi_{\text{Attribute List 1}}(R))$
- $\pi_{\langle \text{list1} \rangle}(\pi_{\langle \text{list2} \rangle}(R)) = \pi_{\langle \text{list1} \rangle}(R)$
 - as long as $\langle \text{list2} \rangle$ contains the attributes in $\langle \text{list1} \rangle$.
 - Means only if Attribute List 1 is a subset of Attribute List 2.



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



- ❑ 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.



- ❑ 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:
 - $\text{DEP5_EMPS} \leftarrow \sigma_{\text{DNO}=5}(\text{EMPLOYEE})$
 - $\text{RESULT} \leftarrow \pi_{\text{FNAME, LNAME, SALARY}}(\text{DEP5_EMPS})$



- ❑ 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)



□ The general RENAME operation ρ can be expressed by any of the following forms:

- $\rho_{S(B_1, B_2, \dots, B_n)}(R)$ changes both:
 - the relation name to S , *and*
 - the column (attribute) names to B_1, B_1, \dots, B_n
- $\rho_S(R)$ changes:
 - the *relation name* only to S
- $\rho_{(B_1, B_2, \dots, B_n)}(R)$ changes:
 - the *column (attribute) names* only to B_1, B_1, \dots, B_n



- ❑ For convenience, we also use a *shorthand* for renaming attributes in an intermediate relation:

- If we write:

- $\text{RESULT} \leftarrow \pi_{\text{FNAME, LNAME, SALARY}} (\text{DEP5_EMPS})$
- RESULT will have the *same attribute names* as DEP5_EMPS

- If we write:

- $\text{RESULT (F, M, L, S, B, A, SX, SAL, SU, DNO)} \leftarrow \pi_{\text{FNAME, LNAME, SALARY}} (\text{DEP5_EMPS})$
- The 10 attributes of DEP5_EMPS are *renamed* to F, M, L, S, B, A, SX, SAL, SU, DNO, respectively.



- ❑ 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”.
 - $\rho_{\text{MaleStudent}(\text{Sno}, \text{Name})} \pi_{\text{RollNo}, \text{SName}}(\sigma_{\text{Condition}}(\text{Student}))$
- ❑ Query to rename the attributes Name, Age of table Department to A,B.
 - $\rho_{(A, B)}(\pi_{\text{Name}, \text{Age}} \text{Department})$
- ❑ 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.
 - $\rho_{(P, B, C)}(\text{Student})$

❑ 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



□ UNION Operation

- Binary operation, denoted by \cup
- The result of $R \cup S$, 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.



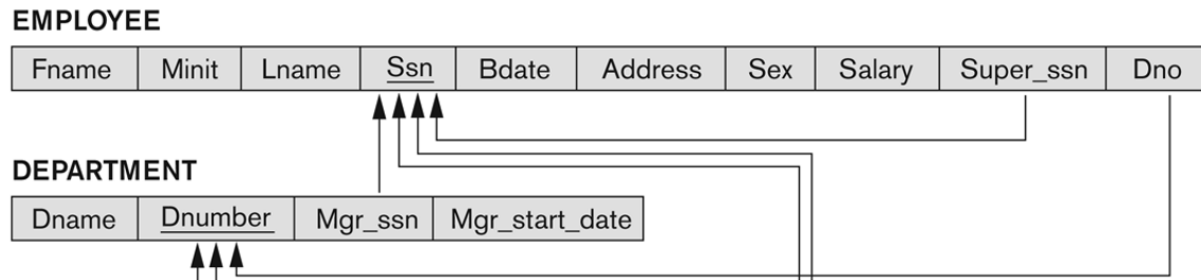
□ Example:

- 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:

$$\begin{aligned} \text{DEP5_EMPS} &\leftarrow \sigma_{\text{DNO}=5}(\text{EMPLOYEE}) \\ \text{RESULT1} &\leftarrow \pi_{\text{SSN}}(\text{DEP5_EMPS}) \\ \text{RESULT2} &\leftarrow \rho_{\text{SSN}}(\pi_{\text{SUPERSSN}}(\text{DEP5_EMPS})) \\ \text{RESULT} &\leftarrow \text{RESULT1} \cup \text{RESULT2} \end{aligned}$$

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





❑ UNION Example

Figure 6.3

Result of the
UNION operation
 $\text{RESULT} \leftarrow \text{RESULT1} \cup \text{RESULT2}$.

RESULT1

Ssn
123456789
333445555
666884444
453453453

RESULT2

Ssn
333445555
888665555

RESULT

Ssn
123456789
333445555
666884444
453453453
888665555



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



- ❑ INTERSECTION is denoted by \cap
- ❑ The result of the operation $R \cap S$, 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”



- ❑ 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.

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

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

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

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



- ❑ 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”



- ❑ 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.



- ❑ 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 minus operation is not commutative; that is, in general
 - $R - S \neq S - R$



❑ CARTESIAN (or CROSS) PRODUCT Operation

- This operation is used to combine tuples from two relations in a combinatorial fashion.
- Denoted by $R(A_1, A_2, \dots, A_n) \times S(B_1, B_2, \dots, B_m)$
- Result is a relation Q with degree $n + m$ attributes:
 - $Q(A_1, A_2, \dots, A_n, B_1, B_2, \dots, B_m)$, 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 $R \times S$ will have $n_R * n_S$ tuples.
- The two operands do NOT have to be “type compatible”



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



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



❑ 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
- The general form of a join operation on two relations $R(A_1, A_2, \dots, A_n)$ and $S(B_1, B_2, \dots, B_m)$ is:

$$R \bowtie_{\langle \text{join condition} \rangle} S$$

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



- ❑ 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.
 - $\text{DEPT_MGR} \leftarrow \text{DEPARTMENT} \bowtie_{\text{MGRSSN=SSN}} \text{EMPLOYEE}$
- ❑ 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 $\text{DEPARTMENT.MGRSSN} = \text{EMPLOYEE.SSN}$

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)

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

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

P1	5	C1
P2	6	C2
P3	4	C1
P4	7	C4
P5	10	C5

R₁ (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:

- $R(A_1, A_2, \dots, A_n) \bowtie S(B_1, B_2, \dots, B_m)$

$$R.A_i = S.B_j$$

- Result is a relation Q with degree $n + m$ attributes:
 - $Q(A_1, A_2, \dots, A_n, B_1, B_2, \dots, B_m)$, 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* $r[A_i] = s[B_j]$
- Hence, if R has n_R tuples, and S has n_S tuples, then the join result will generally have *less than* $n_R * n_S$ tuples.
- Only related tuples (based on the join condition) will appear in the result

$R_1.A_i$ theta $R_2.B_j$



Must be the same domain and not the same name.

Some properties of JOIN



- ❑ The general case of JOIN operation is called a Theta-join: $R \bowtie_{\text{theta}} S$
- ❑ The join condition is called *theta*
- ❑ *Theta* can be any general boolean expression on the attributes of R and S; for example:
 - $R.A_i < S.B_j \text{ AND } (R.A_k = S.B_l \text{ OR } R.A_p < S.B_q)$
- ❑ Most join conditions involve one or more equality conditions “AND”ed together; for example:
 - $R.A_i = S.B_j \text{ AND } R.A_k = S.B_l \text{ AND } R.A_p = S.B_q$

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



- ❑ **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

- Another variation of JOIN called NATURAL JOIN — denoted by $*$ — 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.



- ❑ Example: To apply a natural join on the DNUMBER attributes of DEPARTMENT and DEPT_LOCATIONS, it is sufficient to write:
 - $DEPT_LOCS \leftarrow DEPARTMENT * DEPT_LOCATIONS$
- ❑ 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: $Q \leftarrow R(A,B,C,D) * S(C,D,E)$
 - The implicit join condition includes *each pair* of attributes with the same name, “AND”ed together:
 - $R.C = S.C \text{ AND } R.D = S.D$
 - Result keeps only one attribute of each such pair:
 - $Q(A,B,C,D,E)$



- ❑ The set of operations including **SELECT σ** , **PROJECT π** , **UNION \cup** , **DIFFERENCE $-$** , **RENAME ρ** , and **CARTESIAN PRODUCT \times** is called a **complete set** because any other relational algebra expression can be expressed by a combination of these five operations.

- ❑ For example:

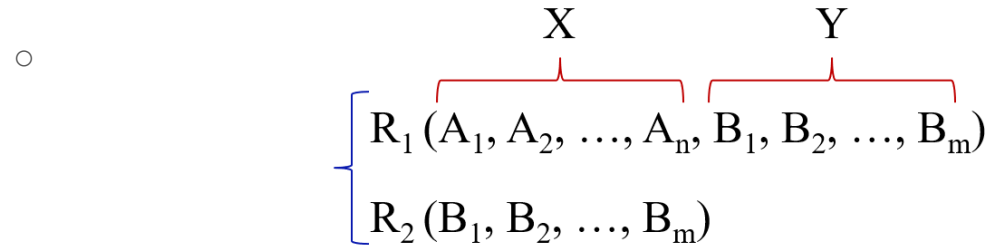
- $R \cap S = (R \cup S) - ((R - S) \cup (S - R)) = R - (R - S)$
- $R \bowtie_{\langle \text{join condition} \rangle} S = \sigma_{\langle \text{join condition} \rangle} (R \times S)$





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

- 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.



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



- ❑ 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 .
- ❑

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

Let $R(A,B)$ and $S(A)$, we want to do $R \div S$.

take $T1=R(B)$ using project operator

take $T2=S(A) \times T1$ (cross product)

take $T3=T2-T1$

take $T4=T3(B)$ using project operator

Result= $T1-T4$

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



Table 6.1
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 $Z = X \cup Y$.	$R_1(Z) \div R_2(Y)$



- ❑ 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.



□ 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
- ❑ 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






❑ 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.

— —
— —



- ❑ The **left outer join** operation keeps every tuple in the first or left relation R in $R \bowtie S$; 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 $R \bowtie S$

- ❑ A third operation, **full outer join**, denoted by \boxplus keeps all tuples in both the left and the right relations when no matching tuples are found, padding them with null values as needed.


- ❑ A 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 semi-join only includes columns from the left table in the result.
- ❑ $R_3 := R_1 \bowtie_C R_2 = \Pi_{\langle H_{R_1} \rangle}(R_1 \bowtie_C R_2)$
- ❑ 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



❑ IN SQL?

using `EXISTS`:

```
SELECT column1, column2, ...FROM table1  
WHERE EXISTS (SELECT 1 FROM table2 WHERE table1.column = table2.column);
```

using `IN`:

```
SELECT column1, column2, ...FROM table1WHERE column IN (SELECT column FROM table2);
```



- ❑ $R \supset S$
- ❑ 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

- ❑ IN SQL?

```
SELECT column1, column2,...FROM table1
```

```
LEFT JOIN table2 ON table1.column_name = table2.column_name
```

```
WHERE table2.column_name IS NULL;
```



- ❑ $R_5 = R_1 \text{ SEMIMINUS } R_2 = R_1 \text{ MINUS } (R_1 \text{ SEMIJOIN } R_2)$
- ❑ $H_{R_5} = H_{R_1}$



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

$\text{RESEARCH_DEPT} \leftarrow \sigma_{\text{DNAME}='Research'}(\text{DEPARTMENT})$

$\text{RESEARCH_EMPS} \leftarrow (\text{RESEARCH_DEPT} \bowtie_{\text{DNUMBER}=\text{DNOEMPLOYEE}} \text{EMPLOYEE})$

$\text{RESULT} \leftarrow \pi_{\text{FNAME}, \text{LNAME}, \text{ADDRESS}}(\text{RESEARCH_EMPS})$

- ❑ Retrieve the names of employees who have no dependents.

$\text{ALL_EMPS} \leftarrow \pi_{\text{SSN}}(\text{EMPLOYEE})$

$\text{EMPS_WITH_DEPS}(\text{SSN}) \leftarrow \pi_{\text{ESSN}}(\text{DEPENDENT})$

$\text{EMPS_WITHOUT_DEPS} \leftarrow (\text{ALL_EMPS} - \text{EMPS_WITH_DEPS})$

$\text{RESULT} \leftarrow \pi_{\text{LNAME}, \text{FNAME}}(\text{EMPS_WITHOUT_DEPS} * \text{EMPLOYEE})$

