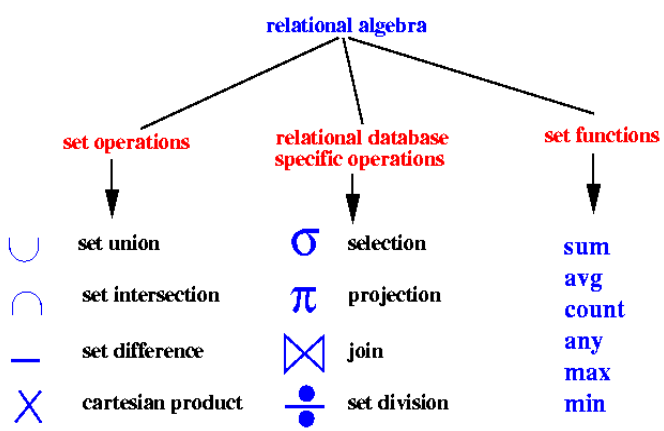
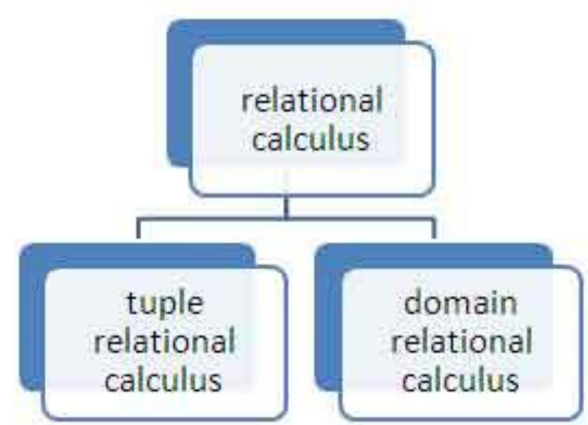
University of Bisha
Faculty Of Computing and Information Technology

## Part (7)

# The Relational Algebra & Relational Calculus




&



# Outlines of Part 7 → use Chapter 8

- Introduction

- Relational Algebra
  - Unary Relational Operations
  - Relational Algebra Operations From Set Theory
  - Binary Relational Operations
  - Additional Relational Operations

- Relational Calculus
  - Tuple Relational Calculus
  - Domain Relational Calculus

# Relational Algebra Overview

- ***Relational algebra*** is similar to high school algebra except that the variables are **tables** not numbers and the results are **tables** not numbers.
- A procedural language
- Not implemented in native form in DBMS.

- Relational algebra is the basic set of operations for the relational model
- These operations enable a user to specify **basic retrieval requests** (or **queries**)

# Relational Algebra Overview….Cont.

- The result of an operation is a *new relation*, which may have been formed from one or more *input* relations.

- 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)

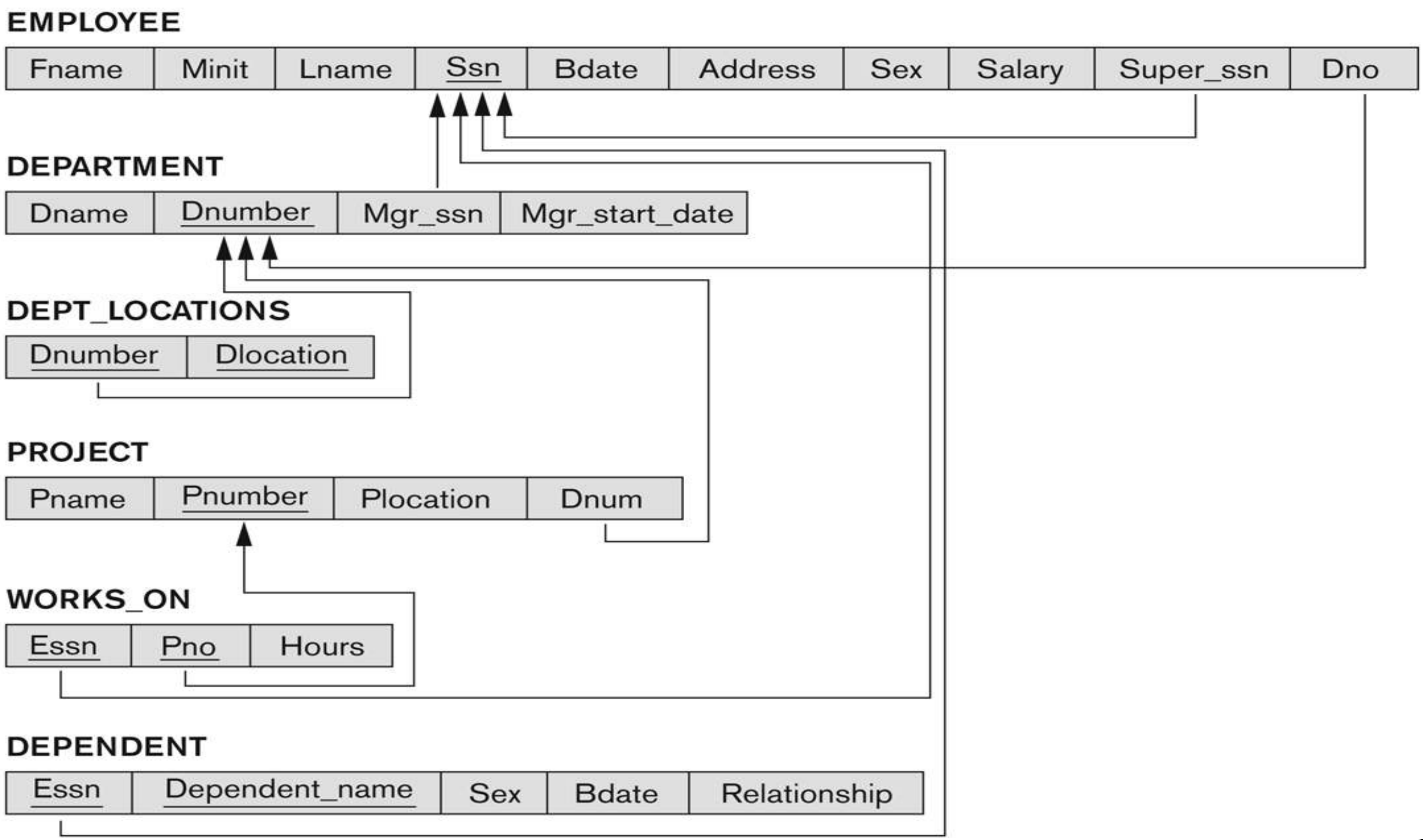
# Relational Algebra Overview

- Relational Algebra consists of several groups of operations
  - **Unary 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)

# 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.



# Unary Relational 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**. (select * from ())
  - 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*)

# Unary Relational Operations: 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, c can be =, <, ≤, >, ≥, <>
- R is the relation that we will select from.

- **Examples:**
  - Select the EMPLOYEE tuples whose department number is 4:

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

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

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

# Unary Relational Operations: SELECT…Cont.

- The SELECT operation $\sigma_{\text{<selection condition>}}(R)$ produces a relation S that has the same schema (same attributes) as R.
- The number of tuples in the result of a SELECT is less than (or equal to) the number of tuples in the input 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.

***Example:***

| SSN | Name | Salary |
|---|---|---|
| 1234545 | John | 200000 |
| 5423341 | Smith | 600000 |
| 4352342 | Fred | 500000 |

$\sigma_{Salary > 40000}$ (Employee)

| SSN | Name | Salary |
|---|---|---|
| 5423341 | Smith | 600000 |
| 4352342 | Fred | 500000 |

# Unary Relational Operations: Project

- **PROJECT** Operation is denoted by $\pi$ (pi)
- This operation keeps certain *columns* (attributes) from a relation and discards the other columns.

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

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

  - $\pi$ (pi) is the symbol used to represent the *project* operation
  - <attribute list> is the desired list of attributes from relation R.

- **Example:** To list each employee's first and last name and salary, the following is used:

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

## PROJECT Operation Properties

- The project operation ***removes any duplicate tuples***
  - The number of tuples in the result of projection $\pi_{<list>}(R)$ is always less or equal to the number of tuples in R (Why? )
    - 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

❖ ***Example:***

| SSN | Name | Salary |
|---|---|---|
| 1234545 | John | 200000 |
| 5423341 | John | 600000 |
| 4352342 | John | 200000 |

$\Pi_{Name,Salary}$ (Employee)

| Name | Salary |
|---|---|
| John | 20000 |
| John | 60000 |

# 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 (Example)

- 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:

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

# Unary Relational Operations: SELECT…Cont.
# Examples of applying SELECT and PROJECT operations

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

# Unary Relational Operations: RENAME

- Changes the schema, not the instance
- 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.

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

# Unary Relational Operations: RENAME…Cont.

## Example

**Employee**

| Name | SSN |
|---|---|
| John | 999999999 |
| Tony | 777777777 |

$$\rho_{LastName,\ SocSocNo}\ (\textbf{Employee})$$

| LastName | SocSocNo |
|---|---|
| John | 999999999 |
| Tony | 777777777 |

# Relational Algebra Operations from Set Theory : UNION

## 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 – (Data type)).

# Relational Algebra Operations from Set Theory : UNION

- Type Compatibility of operands is required for the binary set operation UNION $\cup$, (also for INTERSECTION $\cap$, and SET DIFFERENCE ).

- 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)

# Relational Algebra Operations from Set Theory : UNION……Cont.

## 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

**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 |

# Relational Algebra Operations from Set Theory : INTERSECTION

*INTERSECTION* is denoted by $\cap$

- The result of the operation S1 $\cap$ S2, is a relation that includes all tuples that are in both S1 and S2.
- The attribute names in the result will be the same as the attribute names in S1
- The two operand relations S1 and S2 must be "type compatible"

❖ ***Example:***

| sid | sname | rating | age |
|---|---|---|---|
| 22 | dustin | 7 | 45.0 |
| 31 | lubber | 8 | 55.5 |
| 58 | rusty | 10 | 35.0 |

**S1**

| sid | sname | rating | age |
|---|---|---|---|
| 28 | yuppy | 9 | 35.0 |
| 31 | lubber | 8 | 55.5 |
| 44 | guppy | 5 | 35.0 |
| 58 | rusty | 10 | 35.0 |

**S2**

| sid | sname | rating | age |
|---|---|---|---|
| 31 | lubber | 8 | 55.5 |
| 58 | rusty | 10 | 35.0 |

$$S1 \cap S2$$

# Relational Algebra Operations from Set Theory : SET DIFFERENCE ……Cont.

- ***SET DIFFERENCE*** (also called MINUS or EXCEPT) is denoted by –
- The result of S1 – S2, is a relation that includes all tuples that are in S1 but not in S.
- The attribute names in the result will be the same as the attribute names in S1
- The two operand relations S1 and S2 must be "type compatible".

❖ ***Examples***:

| sid | sname | rating | age |
|---|---|---|---|
| 22 | dustin | 7 | 45.0 |
| 31 | lubber | 8 | 55.5 |
| 58 | rusty | 10 | 35.0 |

S1

| sid | sname | rating | age |
|---|---|---|---|
| 22 | dustin | 7 | 45.0 |

$S1 - S2$

| sid | sname | rating | age |
|---|---|---|---|
| 28 | yuppy | 9 | 35.0 |
| 31 | lubber | 8 | 55.5 |
| 44 | guppy | 5 | 35.0 |
| 58 | rusty | 10 | 35.0 |

S2

| sid | sname | rating | age |
|---|---|---|---|
| 28 | yuppy | 9 | 35.0 |
| 44 | guppy | 5 | 35.0 |

$S2 - S1$

# Relational Algebra Operations from Set Theory : ..Cont.

## Example to illustrate the result of UNION, INTERSECT, and DIFFERENCE

(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 |

**Figure 6.4**
The set operations UNION, INTERSECTION, and MINUS. (a) Two union-compatible relations. (b) STUDENT ∪ INSTRUCTOR. (c) STUDENT ∩ INSTRUCTOR. (d) STUDENT − INSTRUCTOR. (e) INSTRUCTOR − STUDENT.

# Relational Algebra Operations from Set Theory……Cont.

## Some properties of UNION, INTERSECT, and DIFFERENCE

- Notice that both union and intersection are *commutative* operations; that is

$$R \cup S = S \cup R, \text{ and } R \cap S = S \cap R$$

- 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

$$R - S \neq S - R$$

# Relational Algebra Operations from Set Theory……Cont.
## CARTESIAN PRODUCT

- The ***cartesian product*** of two relations is the concatenation of every tuple of one relation with every tuple of a second relations. (Each tuple in R1 with each tuple in R2)

- **Notation:** R1 × R2
  - 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.
  - 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.

    - Very rare in practice; mainly used to express joins

# Relational Algebra Operations from Set Theory……Cont.
# CARTESIAN PRODUCT

***Example:***

Employee × Dependents

**Cartesian Product Example**

**Employee**

| Name | SSN |
|---|---|
| John | 999999999 |
| Tony | 777777777 |

**Dependents**

| EmployeeSSN | Dname |
|---|---|
| 999999999 | Emily |
| 777777777 | Joe |

**Employee x Dependents**

| Name | SSN | EmployeeSSN | Dname |
|---|---|---|---|
| John | 999999999 | 999999999 | Emily |
| John | 999999999 | 777777777 | Joe |
| Tony | 777777777 | 999999999 | Emily |
| Tony | 777777777 | 777777777 | Joe |

# 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

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

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

- where R and S can be any relations that result from general *relational algebra expressions*.
- Only related tuples (based on the join condition) will appear in the result

***Examples:***

student

| ID | NAME | AGE |
|---|---|---|
| 1 | AHMED | 20 |
| 2 | ASMAA | 25 |
| 3 | AYMAN | 32 |
| 4 | ALI | 19 |

marks

| ID | MARK |
|---|---|
| 1 | 80 |
| 1 | 85 |
| 2 | 90 |
| 3 | 95 |

| ID | NAME | AGE | MARK |
|---|---|---|---|
| 1 | Ahmed | 20 | 80 |
| 1 | Ahmed | 20 | 85 |
| 2 | Asmaa | 25 | 90 |
| 3 | Ayman | 32 | 95 |
| 4 | Ali | 19 | |

Department

| Dnumber | Dname |
|---|---|
| 1 | D1 |
| 2 | D2 |
| 3 | D3 |
| 4 | D4 |
| 5 | D5 |

Project

| Pno | Pname | Dno |
|---|---|---|
| 100 | ABC | 1 |
| 200 | CDE | 2 |
| 300 | EFG | 2 |
| 400 | YUK | 3 |

Department $\bowtie_{Dnumber = dno}$ Project

| Dnumber | Dname | Pno | Pname | Dno |
|---|---|---|---|---|
| 1 | D1 | 100 | ABC | 1 |
| 2 | D2 | 200 | CDE | 2 |
| 2 | D2 | 300 | EFG | 2 |
| 3 | D3 | 400 | YUK | 3 |

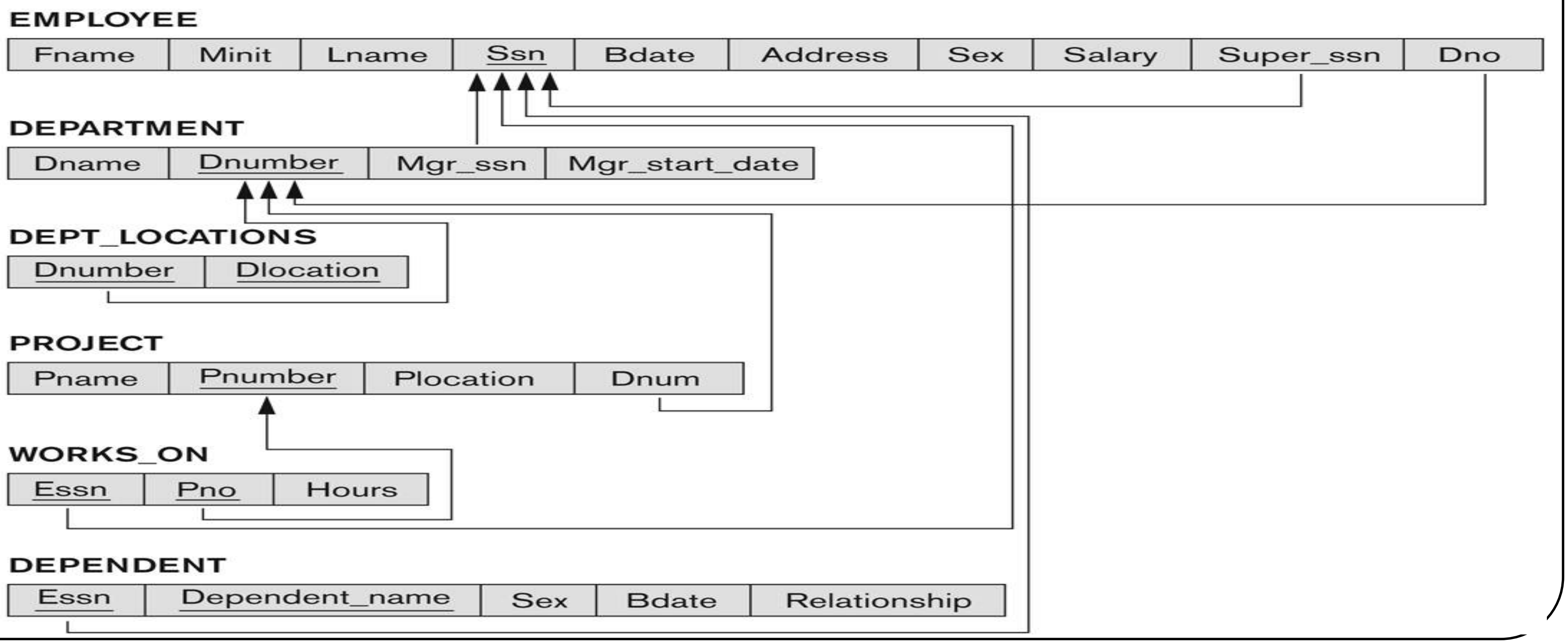
# Binary Relational Operations: JOIN….Cont.

## 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.

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



# Binary Relational Operations: JOIN….Cont.
## Example….Cont.

- We do this by using the ⋈ operation and then projecting the result over the necessary attributes, as follows:

$$DEPT\_MGR \leftarrow DEPARTMENT \bowtie_{Mgr\_ssn=Ssn} EMPLOYEE$$

$$RESULT \leftarrow \pi_{Dname,\ Lname,\ Fname}(DEPT\_MGR)$$

-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

**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 | . . . |

**Figure 6.6**
Result of the JOIN operation

# Binary Relational Operations: JOIN….Cont.

## Types of Join

| Type -Name | Notation | Description |
|---|---|---|
| **Theta-join ($\theta$)** | R1 $\bowtie_{\theta}$ R2 | ▪The join condition is denoted by the symbol **θ.**<br>▪Theta join can use all kinds of comparison operators. |
| **Equijoin (=)** | STUDENT $\bowtie_{Student.Std = Subject.Class}$ SUBJECT | ▪The only comparison operator used is =,<br>▪The JOIN seen in the previous example was an EQUIJOIN. |
| **Natural Join (*)** | ▪Q ← R(A,B,C,D) * S(C,D,E)<br>▪R.C=S.C AND R.D.S.D<br>▪Result keeps only one attribute of each such pair:<br>Q(A,B,C,D,E) | ▪Does not use any comparison operator.<br>▪If there is at least one common attribute that exists between two relations.<br>▪The attributes must have the **same name** and domain. |

# Additional Relational Operations: ….Cont.
## 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.

❖ Computes the join and then adds tuples form one relation that do not match tuples in the other relation to the result of the join.

Uses ***null*** values:

- *null* signifies that the value is unknown or does not exist
- All comparisons involving *null* are (roughly speaking) **false** by definition.

# Additional Relational Operations: ….Cont.
## The OUTER JOIN Operation

❖An **inner join** $\bowtie$ focuses on the commonality between two tables.

❖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.

# Additional Relational Operations: ….Cont.
## Outer Join – Example

- Relation *loan*

| loan-number | branch-name | amount |
|---|---|---|
| L-170 | Downtown | 3000 |
| L-230 | Redwood | 4000 |
| L-260 | Perryridge | 1700 |

- Relation *borrower*

| customer-name | loan-number |
|---|---|
| Jones | L-170 |
| Smith | L-230 |
| Hayes | L-155 |

# Additional Relational Operations: ….Cont.
## Outer Join – Example….Cont.

- **Left Outer Join**

  loan ⟕ Borrower

| loan-number | branch-name | amount | customer-name |
|---|---|---|---|
| L-170 | Downtown | 3000 | Jones |
| L-230 | Redwood | 4000 | Smith |
| L-260 | Perryridge | 1700 | null |

- **Right Outer Join**

  *loan* ⟖ *borrower*

| loan-number | branch-name | amount | customer-name |
|---|---|---|---|
| L-170 | Downtown | 3000 | Jones |
| L-230 | Redwood | 4000 | Smith |
| L-155 | null | null | Hayes |

# Additional Relational Operations: ….Cont.
## Outer Join – Example….Cont.

- **Inner Join**

loan ⋈ Borrower

| loan-number | branch-name | amount | customer-name |
|---|---|---|---|
| L-170 | Downtown | 3000 | Jones |
| L-230 | Redwood | 4000 | Smith |

- **Full Outer Join**

*loan* ⟗ *borrower*

| loan-number | branch-name | amount | customer-name |
|---|---|---|---|
| L-170 | Downtown | 3000 | Jones |
| L-230 | Redwood | 4000 | Smith |
| L-260 | Perryridge | 1700 | null |
| L-155 | null | null | Hayes |

# Complete Set of Relational Operations

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

For example:

$$R \cap S = (R \cup S) - ((R - S) \cup (S - R))$$

$$R \bowtie_{<\text{join condition}>} S = \sigma_{<\text{join condition}>} (R \times S)$$

# Examples of Queries in Relational Algebra

- **Q1: Retrieve the name and address of all employees who work for the 'Research' department.**

RESEARCH_DEPT ← $\sigma$ DNAME='Research' (DEPARTMENT)

RESEARCH_EMPS ← (RESEARCH_DEPT ⋈ DNUMBER= DNOEMPLOYEE EMPLOYEE)

RESULT ← $\pi$ FNAME, LNAME, ADDRESS (RESEARCH_EMPS)

- **Q2: Retrieve the names of employees who have no dependents.**

ALL_EMPS ← $\pi$ SSN(EMPLOYEE)

EMPS_WITH_DEPS(SSN) ← $\pi$ ESSN(DEPENDENT)

EMPS_WITHOUT_DEPS ← (ALL_EMPS - EMPS_WITH_DEPS)

RESULT ← $\pi$ LNAME, FNAME (EMPS_WITHOUT_DEPS * EMPLOYEE)

# Binary Relational Operations: JOIN….Cont. DIVISION

## *DIVISION* Operation

- The division operation is applied to two relations

  $R(Z) \div S(X)$, where X subset Z.

- The result of DIVISION is a relation T(Y).

❑ Useful for expressing "for all" queries like:

*Find sids of sailors who have reserved **all** boats.*

# Binary Relational Operations: JOIN….Cont.
# DIVISION-Schema for Student Registration System

**Example:** Given the following Schema for Student Registration System:

Student (*Id, Name, Addr, Status*)

Professor (*Id, Name, DeptId*)

Course (*DeptId, CrsCode, CrsName, Descr*)

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

Teaching (*ProfId, CrsCode, Semester*)

Department (*DeptId, Name*)

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

Numerator: *StudId* and *CrsCode* for every course passed by every student

$$X \longleftarrow \pi_{StudId,\ CrsCode}(\sigma_{Grade \neq 'F'}(\text{Transcript}))$$

Denominator: *CrsCode* of all courses taught in spring 2000

$$Y \longleftarrow \Pi_{CrsCode}(\sigma_{Semester = 'S2000'}(\text{Teaching}))$$

**Result** is *numerator / denominator* = **X / Y**

# Binary Relational Operations: JOIN….Cont.
# DIVISION - Example

**Example:** ***Retrieve the names of employees who work on all the projects that 'John Smith' works on.***

To express this query using the DIVISION operation, proceed as follows.

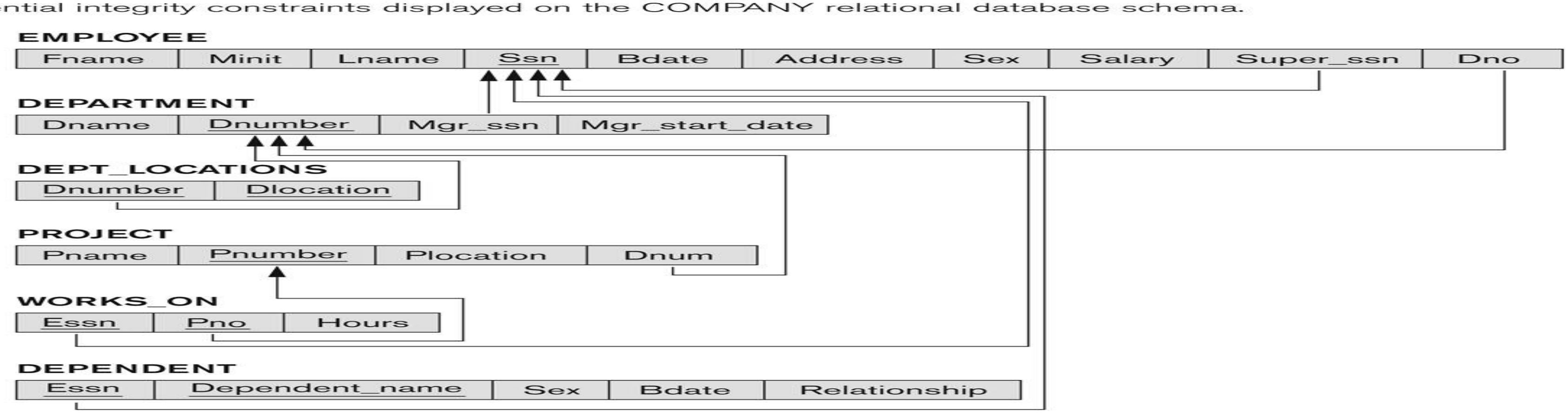
- First, retrieve the list of project numbers that 'John Smith' works on in the intermediate relation

SMITH_PNOS:

$$SMITH \leftarrow \sigma_{Fname='John' \text{ AND } Lname='Smith'}(EMPLOYEE)$$

$$SMITH\_PNOS \leftarrow \pi_{Pno}(WORKS\_ON \bowtie_{Essn=Ssn} SMITH)$$

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



# Binary Relational Operations: JOIN….Cont. DIVISION

- Next, create a relation that includes a tuple <Pno, Essn> whenever the employee whose Ssn is Essn works on the project whose number is Pno in the intermediate relation SSN_PNOS:

$$SSN\_PNOS \leftarrow \pi_{Essn,\ Pno}(WORKS\_ON)$$

- Finally, apply the DIVISION operation to the two relations, which gives the desired employees' Social Security numbers:

$$SSNS(Ssn) \leftarrow SSN\_PNOS \div SMITH\_PNOS$$

$$RESULT \leftarrow \pi_{Fname,\ Lname}(SSNS * EMPLOYEE)$$

The preceding operations are shown in Figure 8.8(a).

# Binary Relational Operations: JOIN….Cont.
# DIVISION - Example of DIVISION

**Figure 8.8**
The DIVISION operation. (a) Dividing SSN_PNOS by SMITH_PNOS. (b) $T \leftarrow R \div S$.

**(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 |

# Recap of Relational Algebra Operations

**Table 8.1** Operations of Relational Algebra

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

# 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.

# Additional Relational Operations: ….Cont.
## Aggregate Function Operations

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

# Additional Relational Operations: ….Cont.
## 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

$$_{\text{DNO}}\mathcal{F}_{\text{COUNT SSN, AVERAGE Salary}}\ (\text{EMPLOYEE})$$

Above operation groups employees by DNO (department number) and computes the count of employees and average salary per department

# Additional Relational Operations: ….Cont.
## Examples of applying aggregate functions and grouping

R

(a)

| 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.

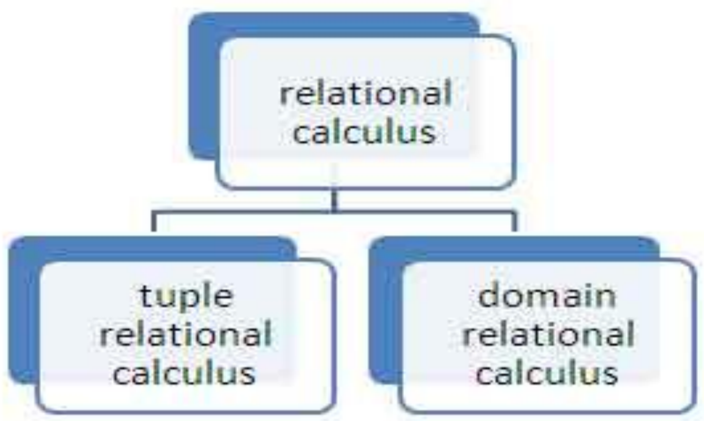
a. $\rho_{R(\text{Dno, No\_of\_employees, Average\_sal})}(_{\text{Dno}}\,\Im_{\text{COUNT Ssn, AVERAGE Salary}}(\text{EMPLOYEE}))$.

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

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

# 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**).




Relational calculus is considered to be a **nonprocedural** 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.

# Relational Calculus….Cont.
## Tuple Relational Calculus (TRC)

- 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 individual tuple from that relation.

- A simple tuple relational calculus query is of the form

$$\{t \mid COND(t)\}$$

- 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**).

# Relational Calculus….Cont.

## Tuple Relational Calculus - Example

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

- we can write the following tuple calculus expression:

**{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 $\pi_{FNAME,\ LNAME}$)** of each EMPLOYEE tuple **t** that satisfies the condition **t**.SALARY>50000 (SELECTION $\sigma_{SALARY > 50000}$) will be retrieved.

# Relational Calculus….Cont.
## Tuple Relational Calculus..Cont.

- **Example:** Select all female students

$$\{ t \mid students(t) \land t.sex = 'f' \}$$

Range = relation "students"

Condition for result tuples

students

| matNr | firstName | lastName | sex |
|---|---|---|---|
| 1005 | Clark | Kent | m |
| 2832 | Louise | Lane | f |
| 4512 | Lex | Luther | m |
| 5119 | Charles | Xavier | m |
| 6676 | Erik | Magnus | m |
| 8024 | Jeanne | Gray | f |
| 9876 | Logan | | m |

students

| matNr | firstName | lastName | sex |
|---|---|---|---|
| 1005 | Clark | Kent | m |
| 2832 | Louise | Lane | f |
| 4512 | Lex | Luther | m |
| 5119 | Charles | Xavier | m |
| 6676 | Erik | Magnus | m |
| 8024 | Jeanne | Gray | f |
| 9876 | Logan | | m |


This type of expression resembles relational algebra's selection!

# Relational Calculus….Cont.
## Tuple Relational Calculus..Cont.

It is possible to retrieve only a subset of attributes

–The **request attributes (**This type of expression resembles relational algebra‟s projection! )

•**Example:**

"Retrieve first name and last name of all female students"

$\pi_{firstName,\ lastName}\ (\sigma_{sex='f'}(students))$

$\{\ t.firstName,\ t.lastName \mid students(t) \land t.sex='f'\ \}$

students

| matNr | firstName | lastName | sex |
|---|---|---|---|
| 1005 | Clark | Kent | m |
| 2832 | Louise | Lane | f |
| 4512 | Lex | Luther | m |
| 5119 | Charles | Xavier | m |
| 6676 | Erik | Magnus | m |
| 8024 | Jeanne | Gray | f |
| 9876 | Logan | | m |

$\{\ t \mid students(t) \land t.sex='f'\ \}$

| matNr | firstName | lastName | sex |
|---|---|---|---|
| 2832 | Louise | Lane | f |
| 8024 | Jeanne | Gray | f |

$\{\ t.firstName,\ t.lastName \mid students(t) \land t.sex='f'\ \}$

| firstName | lastName |
|---|---|
| Loise | Lane |
| Jeanne | Gray |

# Relational Calculus….Cont.
## Tuple Relational Calculus..Cont.

- **Example**: All male students with student number greater than 6000
  - $\{ t \mid students(t) \land t.matNr > 6000 \land t.sex = \text{'m'} \}$
  - Evaluate formula for every tuple in students

students

| matNr | firstName | lastName | sex |
|---|---|---|---|
| 1005 | Clark | Kent | m |
| 2832 | Louise | Lane | f |
| 4512 | Lex | Luther | m |
| 5119 | Charles | Xavier | m |
| 6676 | Erik | Magnus | m |
| 8024 | Jeanne | Gray | f |
| 9876 | Logan | | m |

true ∧ false ∧ true = false

true ∧ false ∧ false = false

Result tuples

true ∧ true ∧ true = true

true ∧ true ∧ false = false

# Relational Calculus….Cont.
## The Existential and Universal Quantifiers

Two special symbols called quantifiers can appear in formulas; these are: the ***universal*** quantifier **(∀)** and the ***existential*** quantifier **(∃)**.

Informally, a tuple variable **t** is bound if it is quantified, meaning that it appears in an (∀ t) or (∃ t) clause; otherwise, it is free.

If **F** is a formula, then so are (∃ t)(F) and (∀ t)(F), where **t** is a tuple variable:

- ∀ 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 (***at least one***) that exists in "the universe of" tuples may make **F** true to make the quantified formula true.(**v**)

# Relational Calculus….Cont.

## Example Query Using Existential Quantifier

❖Retrieve the name and address of all employees who work for the 'Research' department. The query can be expressed as :

**{t.FNAME, t.LNAME, t.ADDRESS | EMPLOYEE(t) and (∃ d) (DEPARTMENT(d) and d.DNAME='Research' and d.DNUMBER=t.DNO) }**

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.

# Relational Calculus….Cont.

## 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.

# Relational Calculus….Cont.
## Example Query Using Existential Quantifier

"List the names of all students that took some exam"

$\pi_{firstName}(\text{students} \ltimes \text{exams})$

$$\{ t_1.firstName \mid students(t_1) \land \exists t_2(exams(t_2) \land t_2.matNr = t_1.matNr) \}$$

students

| matNr | firstName | lastName | sex |
|---|---|---|---|
| 1005 | Clark | Kent | m |
| 2832 | Louise | Lane | f |
| 4512 | Lex | Luther | m |
| 5119 | Charles | Xavier | m |

exams

| matNr | crsNr | result |
|---|---|---|
| 9876 | 100 | 3.7 |
| 2832 | 102 | 2.0 |
| 1005 | 101 | 4.0 |
| 1005 | 100 | 1.3 |

# Relational Calculus….Cont.
## Example Query Using Universal Quantifier

❖Find the names of employees who work on ***all*** the projects controlled by department number 5. The query can be:

**{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.

# Relational Calculus….Cont.
## 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)**

# Relational Calculus….Cont.
## Languages Based on Tuple Relational Quantifier

The language **SQL** is based on tuple calculus. It uses the basic block structure to express the queries in tuple calculus:

**SELECT <list of attributes>**
**FROM <list of relations>**
**WHERE <conditions>**

SELECT clause mentions the attributes being projected, the FROM clause mentions the relations needed in the query, and the WHERE clause mentions the selection as well as the join conditions.

# Relational Calculus….Cont.

## Languages Based on Tuple Relational Quantifier…Cont.

Another language which is based on tuple calculus is **QUEL** which actually uses the range variables as in tuple calculus. Its syntax includes:

**RANGE OF** <variable name> **IS** <relation name>

Then it uses

**RETRIEVE** <list of attributes from range variables>

**WHERE** <conditions>

***Example*** : Compute salary divided by age-18 for employee Jones.

range of E is EMPLOYEE

retrieve into W

(COMP = E.Salary / (E.Age - 18))

where E.Name = "Jones"

Here E is a tuple variable which ranges over the EMPLOYEE relation, and all tuples in that relation are found which satisfy the qualification E.Name = "Jones." The result of the query is a new relation W, which has a single domain COMP that has been calculated for each qualifying tuple.

# Relational Calculus….Cont.
## The Domain Relational Calculus (DRC)

- 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 called QBE (Query-By-Example) that is related to domain calculus was developed almost concurrently to SQL at IBM Research, Yorktown Heights, New York.

- In Domain relational calculus filtering of records is done based on the domain of the ***attributes*** rather than tuple values.

- A domain is nothing but the set of allowed values in the column of a table

# Relational Calculus….Cont.

## The Domain Relational Calculus…..Cont.

An expression of the domain calculus is of the form

$$\{ x_1, x_2, \ldots, x_n \mid COND(x_1, x_2, \ldots, x_n, x_{n+1}, x_{n+2}, \ldots, x_{n+m})\}$$

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.

# Relational Calculus….Cont.

## Example Query Using Domain Calculus

**Example (1):**

Retrieve the names and ages of the students in the table Student who not greater than 21 years old

$$\{ <name, age> \mid \in Student \land age < 21 \}$$

**Example (2):**

Retrieve the Fname and Emp_ID values for all the rows in the employee table where salary is greater than 10000.

$$\{ <Fname, Emp\_ID> \mid \in Employee \land Salary > 10000 \}$$

# Relational Calculus….Cont.
## Chapter Summary

Relational Algebra

- Unary Relational Operations
- Relational Algebra Operations From Set Theory
- Binary Relational Operations
- Additional Relational Operations

Relational Calculus

- Tuple Relational Calculus
- Domain Relational Calculus

Thank you