# Relational Algebra

Chapter 4, Part A

# Relational Query Languages

- *Query languages*: Allow manipulation and retrieval of data from a database.
- Relational model supports simple, powerful QLs:
  - Strong formal foundation based on logic.
  - Allows for much optimization.
- Query Languages != programming languages!
  - QLs not expected to be "Turing complete".
  - QLs not intended to be used for complex calculations.
  - QLs support easy, efficient access to large data sets.

# Formal Relational Query Languages

- Two mathematical Query Languages form the basis for "real" languages (e.g. SQL), and for implementation:
  - *Relational Algebra*: More operational, very useful for representing execution plans.
  - *Relational Calculus*: Lets users describe what they want, rather than how to compute it. (Non-operational, *declarative*.)

# Preliminaries

- A query is applied to *relation instances*, and the result of a query is also a relation instance.
  - *Schemas* of input relations for a query are fixed (but query will run regardless of instance!)
  - The schema for the *result* of a given query is also fixed! Determined by definition of query language constructs.
- Positional vs. named-field notation:
  - Positional notation easier for formal definitions, named-field notation more readable.
  - Both used in SQL

# Example Instances

- "Sailors" and "Reserves" relations for our examples.
- We'll use positional or named field notation, assume that names of fields in query results are `inherited' from names of fields in query input relations.

*R1*

| sid | bid | day |
|---|---|---|
| 22 | 101 | 10/10/96 |
| 58 | 103 | 11/12/96 |

*S1*

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

*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 |

# Relational Algebra

- Basic operations:
  - *Selection* ($\sigma$) Selects a subset of rows from relation.
  - *Projection* ($\pi$) Deletes unwanted columns from relation.
  - *Cross-product* ($\times$) Allows us to combine two relations.
  - *Set-difference* ($-$) Tuples in reln. 1, but not in reln. 2.
  - *Union* ($\cup$) Tuples in reln. 1 and in reln. 2.
- Additional operations:
  - Intersection, *join*, division, renaming: Not essential, but (very!) useful.
- Since each operation returns a relation, operations can be *composed*! (Algebra is "closed".)

# Projection

- Deletes attributes that are not in *projection list*.
- *Schema* of result contains exactly the fields in the projection list, with the same names that they had in the (only) input relation.
- Projection operator has to eliminate *duplicates*! (Why??)
  - Note: real systems typically don't do duplicate elimination unless the user explicitly asks for it. (Why not?)

| sname | rating |
|---|---|
| yuppy | 9 |
| lubber | 8 |
| guppy | 5 |
| rusty | 10 |

$$\pi_{sname,rating}(S2)$$

| age |
|---|
| 35.0 |
| 55.5 |

$$\pi_{age}(S2)$$

# Selection

- Selects rows that satisfy *selection condition*.
- No duplicates in result! (Why?)
- *Schema* of result identical to schema of (only) input relation.
- *Result* relation can be the *input* for another relational algebra operation! (*Operator composition.*)

| sid | sname | rating | age |
|---|---|---|---|
| 28 | yuppy | 9 | 35.0 |
| 58 | rusty | 10 | 35.0 |

$$\sigma_{rating>8}(S2)$$

| sname | rating |
|---|---|
| yuppy | 9 |
| rusty | 10 |

$$\pi_{sname,rating}(\sigma_{rating>8}(S2))$$

# Union, Intersection, Set-Difference

- All of these operations take two input relations, which must be *union-compatible*:
  - Same number of fields.
  - `Corresponding' fields have the same type.
- What is the *schema* of result?

| sid | sname | rating | age |
|---|---|---|---|
| 22 | dustin | 7 | 45.0 |
| 31 | lubber | 8 | 55.5 |
| 58 | rusty | 10 | 35.0 |
| 44 | guppy | 5 | 35.0 |
| 28 | yuppy | 9 | 35.0 |

$S1 \cup S2$

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

$S1 - S2$

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

$S1 \cap S2$

# Cross-Product

- Each row of S1 is paired with each row of R1.
- *Result schema* has one field per field of S1 and R1, with field names `inherited' if possible.
  - *Conflict*: Both S1 and R1 have a field called *sid*.

| (sid) | sname | rating | age | (sid) | bid | day |
|---|---|---|---|---|---|---|
| 22 | dustin | 7 | 45.0 | 22 | 101 | 10/10/96 |
| 22 | dustin | 7 | 45.0 | 58 | 103 | 11/12/96 |
| 31 | lubber | 8 | 55.5 | 22 | 101 | 10/10/96 |
| 31 | lubber | 8 | 55.5 | 58 | 103 | 11/12/96 |
| 58 | rusty | 10 | 35.0 | 22 | 101 | 10/10/96 |
| 58 | rusty | 10 | 35.0 | 58 | 103 | 11/12/96 |

- *Renaming operator*: $\rho\,(C(1 \rightarrow sid1, 5 \rightarrow sid2),\ S1 \times R1)$

# Joins

- *Condition Join*: $$R \bowtie_c S = \sigma_c (R \times S)$$

| (sid) | sname | rating | age | (sid) | bid | day |
|---|---|---|---|---|---|---|
| 22 | dustin | 7 | 45.0 | 58 | 103 | 11/12/96 |
| 31 | lubber | 8 | 55.5 | 58 | 103 | 11/12/96 |

- *Result schema* same as that of cross-product.
- Fewer tuples than cross-product, might be able to compute more efficiently
- Sometimes called a *theta-join*. $$S1 \bowtie_{S1.sid < R1.sid} R1$$

# Joins

- *Equi-Join*: A special case of condition join where the condition $c$ contains only ***equalities***.

| sid | sname | rating | age | bid | day |
|---|---|---|---|---|---|
| 22 | dustin | 7 | 45.0 | 101 | 10/10/96 |
| 58 | rusty | 10 | 35.0 | 103 | 11/12/96 |

$$S1 \bowtie_{sid} R1$$

- *Result schema* similar to cross-product, but only one copy of fields for which equality is specified.
- *Natural Join*: Equijoin on *all* common fields.

# Division

- Not supported as a primitive operator, but useful for expressing queries like:

  *Find sailors who have reserved* ***<u>all</u>*** *boats*.

- Let *A* have 2 fields, *x* and *y*; *B* have only field *y*:
  - *A/B* = $\left\{ \langle x \rangle \mid \exists \langle x, y \rangle \in A \;\; \forall \langle y \rangle \in B \right\}$
  - i.e., ***A/B*** **contains all** ***x*** **tuples (sailors) such that for** ***<u>every</u>*** ***y*** **tuple (boat) in** ***B*****, there is an** ***xy*** **tuple in** ***A*****.**
  - *Or*: If the set of *y* values (boats) associated with an *x* value (sailor) in *A* contains all *y* values in *B*, the *x* value is in *A/B*.

- In general, *x* and *y* can be any lists of fields; *y* is the list of fields in *B*, and $x \cup y$ is the list of fields of *A*.

# Examples of Division A/B

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

*A*

| pno |
|---|
| p2 |

*B1*

| pno |
|---|
| p2 |
| p4 |

*B2*

| pno |
|---|
| p1 |
| p2 |
| p4 |

*B3*

| sno |
|---|
| s1 |
| s2 |
| s3 |
| s4 |

*A/B1*

| sno |
|---|
| s1 |
| s4 |

*A/B2*

| sno |
|---|
| s1 |

*A/B3*

# Expressing A/B Using Basic Operators

- Division is not essential op; just a useful shorthand.
  - (Also true of joins, but joins are so common that systems implement joins specially.)
- *Idea*: For *A/B*, compute all *x* values that are not `disqualified' by some *y* value in *B*.
  - *x* value is *disqualified* if by attaching *y* value from *B*, we obtain an *xy* tuple that is not in *A*.

Disqualified $x$ values: $\pi_x((\pi_x(A) \times B) - A)$

*A/B*: $\pi_x(A) \; - \;$ all disqualified tuples

# Find names of sailors who've reserved boat #103

- Solution 1: $\pi_{sname}((\sigma_{bid=103} \text{Reserves}) \bowtie Sailors)$

❖ Solution 2: $\rho\ (Temp1,\ \sigma_{bid=103} \text{Reserves})$

$\rho\ (Temp2,\ Temp1 \bowtie Sailors)$

$\pi_{sname}(Temp2)$

❖ Solution 3: $\pi_{sname}(\sigma_{bid=103}(\text{Reserves} \bowtie Sailors))$

# Find names of sailors who've reserved a red boat

- Information about boat color only available in Boats; so need an extra join:

$$\pi_{sname}((\sigma_{color='red'} Boats) \bowtie \text{Re}serves \bowtie Sailors)$$

❖ A more efficient solution:

$$\pi_{sname}(\pi_{sid}((\pi_{bid}\sigma_{color='red'} Boats) \bowtie \text{Re}s) \bowtie Sailors)$$

*A query optimizer can find this, given the first solution!*

# Find sailors who've reserved a red or a green boat

- Can identify all red or green boats, then find sailors who've reserved one of these boats:

$$\rho\ (Tempboats, (\sigma_{color='red' \vee color='green'}\ Boats))$$

$$\pi_{sname}(Tempboats \bowtie \mathrm{Re}serves \bowtie Sailors)$$

- ❖ Can also define Tempboats using union! (How?)
- ❖ What happens if $\vee$ is replaced by $\wedge$ in this query?

# Find sailors who've reserved a red and a green boat

- Previous approach won't work! Must identify sailors who've reserved red boats, sailors who've reserved green boats, then find the intersection (note that *sid* is a key for Sailors):

$$\rho\ (Tempred, \pi_{sid}((\sigma_{color='red'}\ Boats) \bowtie \mathrm{Reserves}))$$

$$\rho\ (Tempgreen, \pi_{sid}((\sigma_{color='green'}\ Boats) \bowtie \mathrm{Reserves}))$$

$$\pi_{sname}((Tempred \cap Tempgreen) \bowtie Sailors)$$

# Find the names of sailors who've reserved all boats

- Uses division; schemas of the input relations to / must be carefully chosen:

$$\rho\ (Tempsids, (\pi_{sid,bid}\mathrm{Re}serves) \,/\, (\pi_{bid}\, Boats))$$

$$\pi_{sname}\,(Tempsids \bowtie Sailors)$$

❖ To find sailors who've reserved all 'Interlake' boats:

$$\ldots.. \quad /\ \pi_{bid}(\sigma_{bname='Interlake'}\ Boats)$$

# Additional information on Relational Algebra

# A
# Basic Relational
# Algebra

- A data model must include a set of operation to manipulate the database.

- This basic set of operation for relational model is Relational Algebra.

- A sequence of relational algebra operations forms a Relational Algebra expression.

# Importance of Relational Algebra:

- It provides a formal foundation for relational model operation.
- It is used as a basis for implementing and optimizing queries in RDBMS.
- Some of its concepts are incorporated into the SQL standard query language for RDBMS.

# Two groups of operations

- Set Operations:
  - Union
  - Intersection
  - Difference
  - Cartesian product
- Developed specially for relational database:
  - Select
  - Project
  - Join

- Unary operation:
  - Select
  - Project

- Binary operation:
  - All set operations
  - Join

# Select Operation

- Used to select a subset of the tuples from a relation that satisfy a selection condition.

  example

- It is a horizontal partitioning of the relation:

  - One partition satisfying the condition and will be shown as result.
  - Other partition not satisfying the condition will be discarded.

# Select Operation

- σ <selection condition> (R)
  - Where σ (sigma) is for select operation
  - selection condition is a boolean expression specified on the attributes of Relation R

Example

- Clauses of selection condition
  - <Attribute name> <comparison operator> <constant value>
  - <Attribute name> <comparison operator> <Attribute name>

- Comparison operator
  - =, <, >, ≠, ≥,≤ for numeric value or date
  - =, ≠ for strings of characters

- Constant value
  - Value from attribute domain

- Degree of resulting relation is same as that of R
- No. of tuples in resulting relation <= no. of tuples in R
  - i.e. $\sigma_c (R) <= R$
- Selection condition may be more than one
  - i.e. (Cond1 and Cond2) → true if both are true
  - i.e. (Cond1 or Cond2) → true if any one or both are true
  - Not Cond → true if cond is false

- Select operation is Commutative:
  - $\sigma_{<cond1>}(\sigma_{<cond2>}(R)) = \sigma_{<cond2>}(\sigma_{<cond1>}(R))$
  - A sequence of Selects can be applied in any order.

    Example
- Cascade of Selects can be used through one select using Conjunctive (AND)
  - $\sigma_{<cond1>}(\sigma_{<cond2>}(....(\sigma_{<condn>}(R))......))$

  $= \sigma_{<cond1>\ AND <cond2> AND ......\ AND <condn>}(R)$

  Example

# Project Operation

- Project operation selects certain column from relation and discards rest.

  Example

- It is a vertical partitioning of the relation:
  - One partition satisfying the condition and will be shown as result.
  - Other partition not satisfying the condition and will be discarded.

- Π <attribute list> (R) operation
  - Where Π (pi) is for project

- Attribute list is the desired list of attributes from relation R

- Resulting relation has attribute only those specified in list and in same order

- Degree of resulting relation is same as no. of attributes in list
- No. of tuples in resulting relation?
  - If attribute list has Primary key, the no. of tuples will be same as in Relation R
  - If attribute list does not have Primary key, the duplicate tuples will be removed to give valid relation
  - No. of tuples in resulting relation =< no. of tuples in relation R

- Project operation is not Commutative:
  - $\Pi_{<list1>} (\Pi_{<list2>} (R)) \neq \Pi_{<list2>} (\Pi_{<list1>} (R))$
  - $\Pi_{<list1>} (\Pi_{<list2>} (R)) = \Pi_{<list1>} (R)$

    Example

# Sequence of operations

- Several algebraic expressions:
  - As a single relational expression by nesting operations
  - $\Pi_{\text{<list1>}} (\sigma_{\text{<cond1>}} (R))$ Example

  Or
  - Name the intermediate relation and get final result as a series of operations
  - temp $\leftarrow \sigma_{\text{<cond1>}} (R)$
  - result $\leftarrow \Pi_{\text{<list1>}} (temp)$ Example

# Rename

- Attributes can be renamed in resulting relation
  - temp ← $\sigma_{<cond1>}$ (Employee)
  - Emp(empid, empname, empsalary) ← $\Pi_{<eid,\ ename,\ salary>}$ (temp)

- If no renaming is not applied, resulting relation will have same names and order of attributes as parent relation has.

# Formal Rename operation

- Renaming both relation name and attributes

  $\rho_{S(B1,\ B2,....Bn)}\ (R)$

- Renaming only relation name

  $\rho_{S}\ (R)$

- Renaming only attributes

  $\rho_{(B1,\ B2,....Bn)}\ (R)$

# Set Theoretic Operation

- Binary operation
- Both relational should be union Compatible
  - i.e. for R(A1, A2, ….An) & S(B1,B2,….Bn)
  - dom(Ai) = dom(Bi) for 1 <=i<=n
- UNION (RUS)
- INTERSECTION (R⊓S)
- MINUS (R-S)

- UNION (RUS)
  - It includes all tuples that are either in R or S or in both
  - Duplicate tuples are eliminated.

    Example
- INTERSECTION (R⊓S)
  - It includes all tuples that are in R & S both

    Example

- Set difference (MINUS)
- (R-S) example
  - It includes all tuples that are in R but not in S
- (S-R) example
  - It includes all tuples that are in S but not in R

- (R-S) ≠ (S-R)

example

- Union & Intersection are:

1. Commutative
   - $(R \cup S) = (S \cup R)$
   - $(R \cap S) = (S \cap R)$

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

example

# Cartesian Product

- Binary Operation
- Union Compatibility not required
- R x S                    example
    - no. of tuples in Resultant relation = no. of tuples in R x no. of tuples in S
    - It includes one tuple from each combination
    - No. of attributes in Resultant relation =n + m
    - R(A1, A2,.....,An) x S(B1, B2,....,Bn) =          Q(A1, A2,.....,An, B1, B2,....,Bn)
- Result will have some irrelevant tuple which can be further processed by select & project operation

# Join operation

- R $\bowtie$ S It is used to combine related tuples from two relations into single tuple.
- It allows to process relationships among relations
- R $\bowtie$ <join condition> S      example
  - Attributes in Resultant relation = n + m
  - R(A1, A2,…..,An) $\bowtie$ S(B1, B2,….,Bn) = Q(A1, A2,…..,An, B1, B2,….,Bn)
  - No. of tuples wherever join condition is satisfied.

- Join operation can be stated as Cartesian product operation followed by Select operation

- Theta join: example
  - Ai θ Bi
  - dom(Ai) = dom(Bi)
  - θ is one of the {=,<,>,≥,≤,≠}
  - Tuples whose join attributes are null do not appear in the result

# Equi join:

- Most commonly used
- Operator is =
- One or more pairs of attributes that have identical values
- E.g. Mgrssn = ssn                    example

# Natural join:

- R * S [example]

- No condition is required

- Both the tables should have one same attribute with same name

- If attribute is same but name is different (first perform rename than natural join)

# Join selectivity

- If join condition is not satisfied → no tuple in resultant relation

- Tuples are in between zero and $n_R$ x $n_S$

- Join selectivity ratio = expected size of join result / maximum size

# m-way join

- $((\text{Project} \bowtie_{\text{DNUM = DNUMBER}} \text{Department}) \bowtie_{\text{MGRSSN = SSN}} \text{Employee})$

## A complete set of relational algebra operation

- {σ, π, U, -, x) is a complete set
- R ⊓ S = (RUS) – ((R-S) U (S-R))
- R $\bowtie$ <condition> S = σ <condition> (R x S)
- Natural join = π…(σ…(ρ….(R x S)))

- But for convenience, different operations are used.

# Division operation

- Denoted by ÷
- Operation is applied to two relations $R(Z) \div S(X)$

example

# Summary

- The relational model has rigorously defined query languages that are simple and powerful.
- Relational algebra is more operational; useful as internal representation for query evaluation plans.
- Several ways of expressing a given query; a query optimizer should choose the most efficient version.

# Examples for relational algebra in the following slides

Employee

| Empid | Name | Salary |
| --- | --- | --- |
| 101 | Ram | 20,000 |
| 102 | Shyam | 22,000 |
| 103 | Geeta | 24,000 |
| 104 | Geeta | 25,000 |
| 105 | Pam | 18,000 |

Resultant table:

Employee

| Empid | Name | Salary |
| --- | --- | --- |
| 102 | Shyam | 22,000 |
| 103 | Geeta | 24,000 |
| 104 | Geeta | 25,000 |

Employee

| Empid | Name | Salary |
| --- | --- | --- |
| 101 | Ram | 20,000 |
| 102 | Shyam | 22,000 |
| 103 | Geeta | 24,000 |
| 104 | Geeta | 25,000 |
| 105 | Pam | 18,000 |

$\sigma$ salary>20,000 (Employee)

Resultant table:

Employee

| Empid | Name | Salary |
| --- | --- | --- |
| 102 | Shyam | 22,000 |
| 103 | Geeta | 24,000 |
| 104 | Geeta | 25,000 |

Employee

σ salary>20,000

(σ Dno=2
(Employee))

| Empid | Name | Dno | Salary |
|---|---|---|---|
| 101 | Ram | 1 | 20,000 |
| 102 | Shyam | 2 | 22,000 |
| 103 | Geeta | 3 | 24,000 |
| 104 | Geeta | 2 | 25,000 |
| 105 | Pam | 2 | 18,000 |

Resultant table:

Employee

| Empid | Name | Dno | Salary |
|---|---|---|---|
| 102 | Shyam | 2 | 22,000 |
| 104 | Geeta | 2 | 25,000 |

Employee

σ Dno=2

(σ salary>20,000 (Employee))

| Empid | Name | Dno | Salary |
|---|---|---|---|
| 101 | Ram | 1 | 20,000 |
| 102 | Shyam | 2 | 22,000 |
| 103 | Geeta | 3 | 24,000 |
| 104 | Geeta | 2 | 25,000 |
| 105 | Pam | 2 | 18,000 |

Resultant table:

Employee

| Empid | Name | Dno | Salary |
|---|---|---|---|
| 102 | Shyam | 2 | 22,000 |
| 104 | Geeta | 2 | 25,000 |

Employee

σ Dno=2 and

σ salary>20,000 (Employee)

| Empid | Name | Dno | Salary |
|---|---|---|---|
| 101 | Ram | 1 | 20,000 |
| 102 | Shyam | 2 | 22,000 |
| 103 | Geeta | 3 | 24,000 |
| 104 | Geeta | 2 | 25,000 |
| 105 | Pam | 2 | 18,000 |

Resultant table:

Employee

| Empid | Name | Dno | Salary |
|---|---|---|---|
| 102 | Shyam | 2 | 22,000 |
| 104 | Geeta | 2 | 25,000 |

Employee

| Empid | Name | Salary |
|---|---|---|
| 101 | Ram | 20,000 |
| 102 | Shyam | 22,000 |
| 103 | Geeta | 24,000 |
| 104 | Geeta | 25,000 |

$\Pi$ empid, salary (Employee)

Resultant table:

Employee

| Empid | Salary |
|---|---|
| 101 | 20,000 |
| 102 | 22,000 |
| 103 | 24,000 |
| 104 | 25,000 |

Employee

| <u>Empid</u> | Name | Salary |
|---|---|---|
| 101 | Ram | 20,000 |
| 102 | Shyam | 22,000 |
| 103 | Geeta | 25,000 |
| 104 | Geeta | 25,000 |

$\Pi$ name, salary (Employee)

Resultant table:

Employee

| Name | Salary |
|---|---|
| Ram | 20,000 |
| Shyam | 22,000 |
| Geeta | 25,000 |

Employee

| Empid | Name | Salary |
|---|---|---|
| 101 | Ram | 20,000 |
| 102 | Shyam | 22,000 |
| 103 | Geeta | 24,000 |
| 104 | Geeta | 25,000 |

Π empid ( Π empid, salary (Employee))

Resultant table:

Employee

| Empid |
|---|
| 101 |
| 102 |
| 103 |
| 104 |

Π salary ( Π empid, salary (Employee))

Employee →

| Salary |
|---|
| 20,000 |
| 22,000 |
| 24,000 |
| 25,000 |

Employee

| Empid | Name | Salary |
|---|---|---|
| 101 | Ram | 20,000 |
| 102 | Shyam | 22,000 |
| 103 | Geeta | 24,000 |
| 104 | Geeta | 25,000 |

Π empid ( Π empid, salary (Employee))

Resultant table:

Employee

| Empid |
|---|
| 101 |
| 102 |
| 103 |
| 104 |

Π empid (Employee))

Employee →

| Empid |
|---|
| 101 |
| 102 |
| 103 |
| 104 |

Employee

| Empid | Name | Salary |
|---|---|---|
| 101 | Ram | 20,000 |
| 102 | Shyam | 22,000 |
| 103 | Geeta | 24,000 |
| 104 | Geeta | 25,000 |

$\Pi$ empid ($\sigma$ salary>20,000 (Employee))

Resultant table:

Employee

| Empid |
|---|
| 102 |
| 103 |
| 104 |

Employee

Π empid (σ salary>20,000 (Employee))

| Empid | Name | Salary |
| --- | --- | --- |
| 101 | Ram | 20,000 |
| 102 | Shyam | 22,000 |
| 103 | Geeta | 24,000 |
| 104 | Geeta | 25,000 |

Π empid (temp)

Employee

temp

temp ← σ salary>20,000 (Employee)

| Empid | Name | Salary |
| --- | --- | --- |
| 102 | Shyam | 22,000 |
| 103 | Geeta | 24,000 |
| 104 | Geeta | 25,000 |

| Empid |
| --- |
| 102 |
| 103 |
| 104 |

Employee

| Empid | Name | Salary |
|---|---|---|
| 101 | Ram | 20,000 |
| 102 | Shyam | 22,000 |
| 103 | Geeta | 24,000 |
| 104 | Geeta | 25,000 |

EMP(EID) ← Π empid (temp)

EMP

| EID |
|---|
| 102 |
| 103 |
| 104 |

Employee

temp ← σ salary>20,000 (Employee)

| Empid | Name | Salary |
|---|---|---|
| 101 | Ram | 20,000 |
| 102 | Shyam | 22,000 |
| 103 | Geeta | 24,000 |
| 104 | Geeta | 25,000 |

Student

| Stu_id | Name |
|---|---|
| 101 | Ram |
| 102 | Shyam |
| 103 | Geeta |
| 104 | Rita |
| 105 | John |

Instructor

| Empid | Name |
|---|---|
| 102 | Shyam |
| 106 | Smith |
| 107 | Nita |

Student U Instructor

| Stu_id | Name |
|---|---|
| 101 | Ram |
| 102 | Shyam |
| 103 | Geeta |
| 104 | Rita |
| 105 | John |
| 106 | Smith |
| 107 | Nita |

Student

| Stu_id | Name |
|---|---|
| 101 | Ram |
| 102 | Shyam |
| 103 | Geeta |
| 104 | Rita |
| 105 | John |

Instructor

| Empid | Name |
|---|---|
| 102 | Shyam |
| 106 | Smith |
| 107 | Nita |

Student ∩ Instructor

| Stu_id | Name |
|---|---|
| 102 | Shyam |

Student

| Stu_id | Name |
|---|---|
| 101 | Ram |
| 102 | Shyam |
| 103 | Geeta |
| 104 | Rita |
| 105 | John |

Instructor

| Empid | Name |
|---|---|
| 102 | Shyam |
| 106 | Smith |
| 107 | Nita |

Student - Instructor

| Stu_id | Name |
|---|---|
| 101 | Ram |
| 103 | Geeta |
| 104 | Rita |
| 105 | John |

Student

| Stu_id | Name |
|---|---|
| 101 | Ram |
| 102 | Shyam |
| 103 | Geeta |
| 104 | Rita |
| 105 | John |

Instructor

| Empid | Name |
|---|---|
| 102 | Shyam |
| 106 | Smith |
| 107 | Nita |

Instructor - Student

| Stu_id | Name |
|---|---|
| 106 | Smith |
| 107 | Nita |

Student - Instructor

| Stu_id | Name |
| --- | --- |
| 101 | Ram |
| 103 | Geeta |
| 104 | Rita |
| 105 | John |

Instructor - Student

| Stu_id | Name |
| --- | --- |
| 106 | Smith |
| 107 | Nita |

Student

| Stu_id | Name |
|---|---|
| 101 | Ram |
| 102 | Shyam |
| 103 | Geeta |

Player

| Play_id | Name |
|---|---|
| 103 | Geeta |
| 102 | Shyam |

Instructor

| Empid | Name |
|---|---|
| 102 | Shyam |
| 106 | Smith |
| 107 | Nita |

Student U (Instructor U Player )

| Stu_id | Name |
|---|---|
| 101 | Ram |
| 103 | Geeta |
| 104 | Rita |
| 105 | John |
| 102 | Shyam |

# Cartesian Product

Employee

| Emp_id | Name |
|---|---|
| 101 | Ram |
| 102 | Shyam |
| 103 | Geeta |

Dependant

| Dep_name | Bdate |
|---|---|
| Meena | 23-02-1988 |
| Raju | 23-02-1990 |

Employee x dependant

| Emp_id | Name | Dep_name | Bdate |
|---|---|---|---|
| 101 | Ram | Meena | 23-02-1988 |
| 101 | Ram | Raju | 23-02-1990 |
| 102 | Shyam | Meena | 23-02-1990 |
| 102 | Shyam | Raju | 23-02-1990 |
| 103 | Geeta | Meena | 23-02-1990 |
| 103 | Geeta | Raju | 23-02-1990 |

# Join operation

Employee

| emp_id | Name |
|---|---|
| 101 | Ram |
| 102 | Shyam |
| 103 | Geeta |

Dependant

| Dep_name | eid |
|---|---|
| Meena | 101 |
| Raju | 102 |

Employee $\bowtie_{emp\_id=eid}$ Dependant

| emp_id | eid | Name | Dep_name |
|---|---|---|---|
| 101 | 101 | Ram | Meena |
| 102 | 102 | Shyam | Raju |

# Join operation

Employee

| emp_id | Name |
| --- | --- |
| 101 | Ram |
| 102 | Shyam |
| 103 | Geeta |

Dependant

| Dep_name | emp_id |
| --- | --- |
| Meena | 101 |
| Raju | 102 |

Employee * Dependant

| emp_id | Name | Dep_name |
| --- | --- | --- |
| 101 | Ram | Meena |
| 102 | Shyam | Raju |

# Division

R

| ESSN | PNO |
| --- | --- |
| 12345 | 1 |
| 12345 | 2 |
| 23456 | 2 |
| 23456 | 1 |
| 35346 | 3 |
| 21336 | 4 |

S

| PNO |
| --- |
| 1 |
| 2 |

R ÷ S

| ESSN |
| --- |
| 12345 |
| 23456 |