T05 Prashansa Singh

INFO20003 Tutorial 5

starting ~ 2.20 pm

Today's tutorial

- Review of Relational Algebra (RA)
- RA & SQL questions
  - group work

Github Link
https://github.com/Prashansa-Singh/INFO20003-Sem1-2022

Relational Algebra (RA) Review:

RA

- Consists of a collection of operators
- These take instances of a relation (tables) as their inputs
- Return an instance of a relation as output
- Can be unary or binary operators
  - unary: applied on a single relation
  - binary: requires two relations as input

Five fundamental basic operations of RA

- Selection
- Projection
- Cross Product
- Set Difference
- Set Union

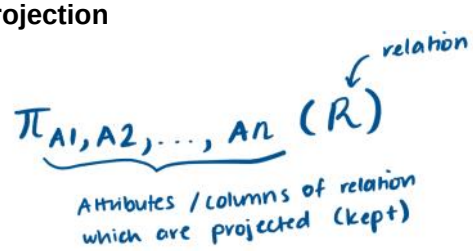
- These five basic operations can be used to form compound operations (Intersection, Natural Join and Condition Join)

Removal operations

- Selection
- Projection

- These remove components from a relation
- Selection: removes rows
- Projection: removes columns

Projection



- Creates a new relation with a subset of the columns
- All tuples from original relation are kept, but we only store projected columns

- If one of the columns which is removed is part of the PK
  - You may end up with duplicate rows in your resulting relation
  - Projection operator in relational algebra automatically removes duplicates rows

- Example:
  - Person table
FirstName	LastName	Phone	Email
Jon	Snow	0551-999-210	knowsnothing@hotmail.com
Daenerys	Targaryen	0569-988-112	bendtheknee@gmail.com
Jamie	Lannister	0531-987-654	handsfree@gmail.com
Night	King	0566-123-456	killerstare@gmail.com

    The expression $\pi_{\text{FirstName, LastName}}(\text{Person})$ will result in:

    - Result of projection example:

FirstName	LastName
Jon	Snow
Daenerys	Targaryen
Jamie	Lannister
Night	King

Selection

$\sigma_{\text{condition}}(R)$

- R is a relation
- Condition is used to filter rows
- Creates a new relation which only includes the rows for which the condition is true
- Example
  - Person table
FirstName	LastName	Phone	Email
Jon	Snow	0551-999-210	knowsnothing@hotmail.com
Daenerys	Targaryen	0569-988-112	bendtheknee@gmail.com
Jamie	Lannister	0531-987-654	handsfree@gmail.com
Night	King	0566-123-456	killerstare@gmail.com

  - $\sigma_{\text{FirstName = 'Jon' } \vee \text{ LastName = 'King'}}(\text{Person})$

$\vee$ = or

$\wedge$ = And

- Result of Selection example

FirstName	LastName	Phone	Email
Jon	Snow	0551-999-210	knowsnothing@hotmail.com
Night	King	0566-123-456	killerstare@gmail.com

We can combine the two operations in one expression as:

$\pi_{\text{FirstName, LastName}}(\sigma_{\text{FirstName = 'Jon' } \vee \text{ LastName = 'King'}}(\text{Person}))$

FirstName	LastName
Jon	Snow
Night	King

- **Set operations**
  - Cross Product
  - Set Difference
  - Set Union

  - Both Union and Set Difference require input relations to be **union-compatible**
    - Same number of attributes in same order
    - Corresponding attributes have same data type

Union

- R U S
- Result is every row which is either in R **or** S
  - (Duplicate rows are not stored)

- Example:
  -

GoodGuys

FirstName	LastName
Jon	Snow
Daenerys	Targaryen

BadGuys

FirstName	LastName
Cersei	Lannister
Night	King

GoodGuys ∪ BadGuys will result in:

GoodGuys ∪ BadGuys will result in:

FirstName	LastName
Jon	Snow
Daenerys	Targaryen
Cersei	Lannister
Night	King

Set Difference

- **R - S**
- Result is every row which is **in R but not S**
- S-R (order matters in set difference - this returns a different result to R - S)
- Example:

RandomCombo1

FirstName	LastName
Jon	Snow
Daenerys	Targaryen
Jamie	Lannister
Night	King

RandomCombo2

FirstName	LastName
Night	King
Arya	Stark
Cersei	Lannister
Daenerys	Targaryen

RandomCombo1 – RandomCombo2 will result in:

FirstName	LastName
Jon	Snow
Jamie	Lannister

Cross Product

- R x S
- Each row in R is paired with each row in S
- Resulting relation has all attributes from both relations
  - (If some attributes have same name, rename them using the renaming operator ρ 'rho' - will learn later in tute)

- Example
  - Person

FirstName	LastName	Email
Jon	Snow	knowsnothing@hotmail.com
Night	King	killerstare@gmail.com

Weapon

Weapon	Metal
Sword	Valyrian steel
Dagger	Dragon glass

- Person

FirstName	LastName	Email
Jon	Snow	knowsnothing@hotmail.com
Night	King	killerstare@gmail.com

- Weapon

Weapon	Metal
Sword	Valyrian steel
Dagger	Dragon glass

Person × Weapon will result in:

2 x 2 = 4

- How many rows will be in the result?

FirstName	LastName	Email	Weapon	Metal
Jon	Snow	knowsnothing@hotmail.com	Sword	Valyrian steel
Jon	Snow	knowsnothing@hotmail.com	Dagger	Dragon glass
Night	King	killerstare@gmail.com	Sword	Valyrian steel
Night	King	killerstare@gmail.com	Dagger	Dragon glass

-

Compound Operations

- Can all be expressed in terms of basic operators

Intersection

- Is a set operator
- Result is a relation containing all rows which are present in **both** relations
- 2 relations should be union compatible

- Intersection in terms of basic operations:
  - $R \cap S = R - (R - S)$

- Example
  - RandomCombo1

FirstName	LastName
Jon	Snow
Daenerys	Targaryen
Jamie	Lannister
Night	King

  - RandomCombo2

FirstName	LastName
Night	King
Arya	Stark
Cersei	Lannister
Daenerys	Targaryen

RandomCombo1 ∩ RandomCombo2 will result in:

FirstName	LastName
Daenerys	Targaryen
Night	King

Natural Join

- R ⋈ S
- First, we find attributes common to both relations (have the same name)
  - Equality is implicit
- Then create a new relation, pairing each tuple from R and S where common attributes are equal

- Joins are compound operators
  - usually they are made up of the basic operators
    - cross product
    - selection
    - and projection

- Can break down a NJ into these steps:
  1) Compute R x S
  2) Select rows where attributes that appear in both relations have equal values
  3) Project all unique attributes and keep one copy of the common attributes

For example, here is a natural join between the Person and WeaponOwner relations:

Person

FirstName	LastName	Email
Jon	Snow	knowsnothing@hotmail.com
Daenerys	Targaryen	bendtheknee@gmail.com
Tyrion	Lannister	idrinkandiknow@gmail.com
Night	King	killerstare@gmail.com

WeaponOwner

Weapon	LastName	Metal
Sword	Snow	Valyrian steel
Dagger	Lannister	Dragon glass

- Result:
  - Method 1)
    - Find attributes common to both relations
    - Then create a new relation, pairing each tuple from R and S where common attributes are equal

  - Method 2)
    - Break down into steps using basic operations

Person × Weapon (intermediate result):

FirstName	LastName	Email	Weapon	LastName	Metal
Jon	Snow	knowsnothing@hotmail.com	Sword	Snow	Valyrian steel
Jon	Snow	knowsnothing@hotmail.com	Dagger	Lannister	Dragon glass
Daenerys	Targaryen	bendtheknee@gmail.com	Sword	Snow	Valyrian steel
Daenerys	Targaryen	bendtheknee@gmail.com	Dagger	Lannister	Dragon glass
Tyrion	Lannister	idrinkandiknow@gmail.com	Sword	Snow	Valyrian steel
Tyrion	Lannister	idrinkandiknow@gmail.com	Dagger	Lannister	Dragon glass
Night	King	killerstare@gmail.com	Sword	Snow	Valyrian steel
Night	King	killerstare@gmail.com	Dagger	Lannister	Dragon glass

Person ⋈ Weapon will result in:

Night	King	killerstare@gmail.com	Dagger	Lannister	Dragon glass

Person ⋈ Weapon will result in:

FirstName	LastName	Email	Weapon	Metal
Jon	Snow	knowsnothing@hotmail.com	Sword	Valyrian steel
Tyrion	Lannister	idrinkandiknow@gmail.com	Dagger	Dragon glass

Notice LastName attribute only appears once

Condition Join

- Also called Theta/Inner Join
- $R \bowtie_C S$
  - This joins rows from R and S such that the condition C is true
  - Usually, C is of type A = B (this is an equi-join, which is a type of condition join)
  - Condition joins can also involves != , <, > etc e.g. Attribute1 != 5
- Condition Join in terms of basic operations:
  - $R \bowtie_C S = \sigma_C(R \times S)$

Example:

Person

FirstName	LastName	Email
Jon	Snow	knowsnothing@hotmail.com
Daenerys	Targaryen	bendtheknee@gmail.com
Tyrion	Lannister	idrinkandiknow@gmail.com
Night	King	killerstare@gmail.com

WeaponOwner

Weapon	Name	Metal
Sword	Snow	Valyrian steel
Dagger	Lannister	Dragon glass

Person ⋈ LastName = Name **Weapon** will result in:

- *Why isn't this considered a natural join?*

  - This is very similar to a Natural Join but the difference is you can't use natural join because attribute names are different.

  - Natural Joins
    - Equality is implicit. (You don't need to specify which attributes you're joining on)
    - Attributes **must have same name** for NJ to work
    - NJ has implicit projection - only 1 copy of attribute is kept

  - Equi-joins
    - Equality is explicit, can make up any equality condition you want
    - If both attributes have same name, you need to explicitly project to keep one

Person × Weapon (intermediate result):

FirstName	LastName	Email	Weapon	Name	Metal
Jon	Snow	knowsnothing@hotmail.com	Sword	Snow	Valyrian steel
Jon	Snow	knowsnothing@hotmail.com	Dagger	Lannister	Dragon glass
Daenerys	Targaryen	bendtheknee@gmail.com	Sword	Snow	Valyrian steel
Daenerys	Targaryen	bendtheknee@gmail.com	Dagger	Lannister	Dragon glass
Tyrion	Lannister	idrinkandiknow@gmail.com	Sword	Snow	Valyrian steel
Tyrion	Lannister	idrinkandiknow@gmail.com	Dagger	Lannister	Dragon glass
Night	King	killerstare@gmail.com	Sword	Snow	Valyrian steel
Night	King	killerstare@gmail.com	Dagger	Lannister	Dragon glass

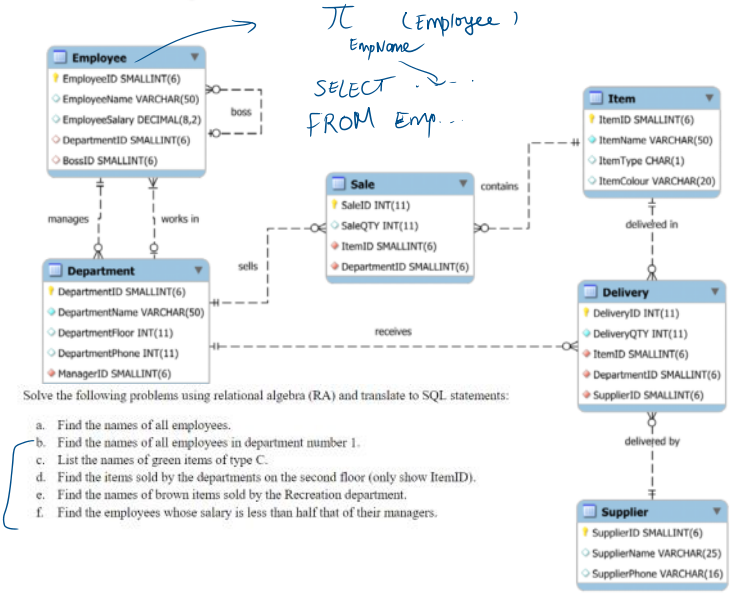
Person $\bowtie_{\text{LastName} = \text{Name}}$ **Weapon** will result in:

FirstName	LastName	Email	Weapon	Name	Metal
Jon	Snow	knowsnothing@hotmail.com	Sword	Snow	Valyrian steel
Tyrion	Lannister	idrinkandiknow@gmail.com	Dagger	Lannister	Dragon glass

Notice both the Name and LastName attributes appear in result

Q2

Consider the following schema:



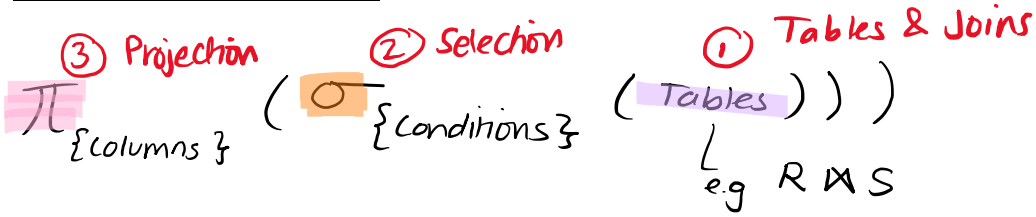
Solve the following problems using relational algebra (RA) and translate to SQL statements:

a. Find the names of all employees.
b. Find the names of all employees in department number 1.
c. List the names of green items of type C.
d. Find the items sold by the departments on the second floor (only show ItemID).
e. Find the names of brown items sold by the Recreation department.
f. Find the employees whose salary is less than half that of their managers.

Solve the following problems using relational algebra (RA) and translate to SQL statements:

a. Find the names of all employees.
b. Find the names of all employees in department number 1.
c. List the names of green items of type C.
d. Find the items sold by the departments on the second floor (only show ItemID).
e. Find the names of brown items sold by the Recreation department.
f. Find the employees whose salary is less than half that of their managers.

How to write RA statements:



- Write and read from the inside out (start with innermost bracket and work your way out)

- **STEP 1**
  - **TABLES/JOINS**
  - First figure out all the relations you need to use to access the attributes you want. These go inside the innermost bracket
    - Figure out how these must get joined
    - e.g. natural join, equijoin, condition join etc
    - You might just need 1 table and so don't need any joins

- **STEP 2**
  - **SELECTION**
  - Then after we have the table / result of joined tables we want, we do **selection on the rows** of this result. This is second inner most bracket.

  - Figure out which conditions we are doing selection on (horizontal filtering)
    - These are the **conditions we must ensure all of our returned rows in our final output result fulfil**
      - e.g. for part
        - b) dept number is 1
        - c) item colour is green and item type is c
        - d) dept floor is 2
        - e) item colour is brown and dept name is Recreation
        - f) emp.salary is < 1/2 of manager salary

- **STEP 3**
  - **PROJECTION**
    - Figure out the columns/attributes you actually want in your final result
      - Do we just want to see the employee names? or names of items? or ItemID values? etc
      - This the very last step (performed after joins and selection) and will keep only these columns in your output results
      - Done in outermost brackets

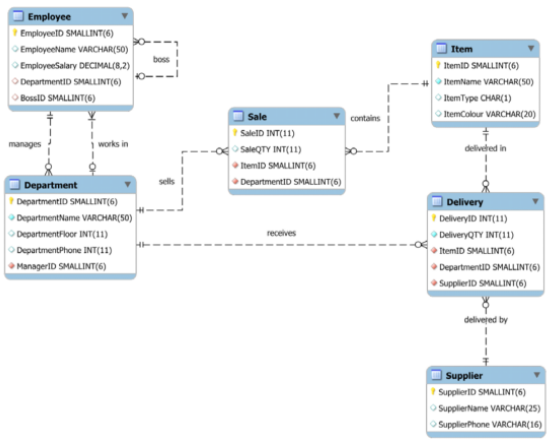
Translating from RA to SQL

- **SELECT {Columns}**
  - This statement is actually the last to get executed
  - It takes the result after the tables have been joined (FROM step) and selection on the rows of this result has occured (WHERE step), and only keeps the columns specified
  - This is equivalent to the **Projection step** (list the columns you want to keep)
- **FROM RelationA NATURAL JOIN RelationB NATURAL JOIN RelationC ...**
- **A INNER JOIN B ON LastName = Name**
  - This is equivalent to the **Tables/Joins step** and is executed first

  - This returns a table of records which is the output of all these joins
- **WHERE {Conditions}**
  - This step is performed after the FROM step and before SELECT
  - This is equivalent to the **Selection step**
  - In this step, we use the output result we got in the FROM statement and only keep the rows in this result which fulfil the conditions specified

- Beware: You can have more complicated SQL statements with more than these 3 keywords (SELECT, FROM, WHERE). This is just a template for a basic SQL statement.

a. Find the names of all employees.

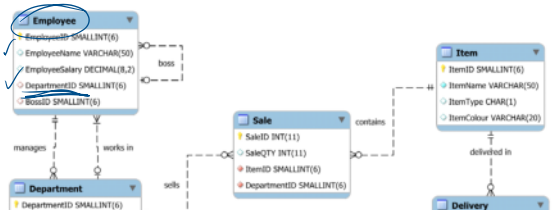


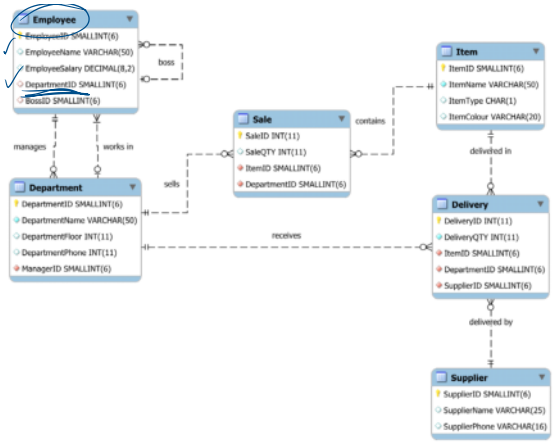
a
SELECT EmpName
FROM Employee;

Semi-colon (;) marks end of query

- SELECT in SQL means projection in RA

b. Find the names of all employees in department number 1.

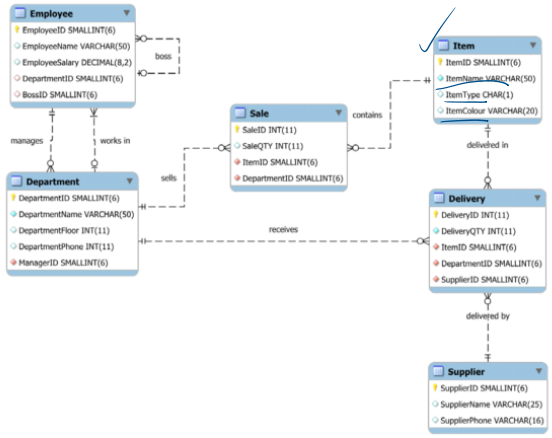




b

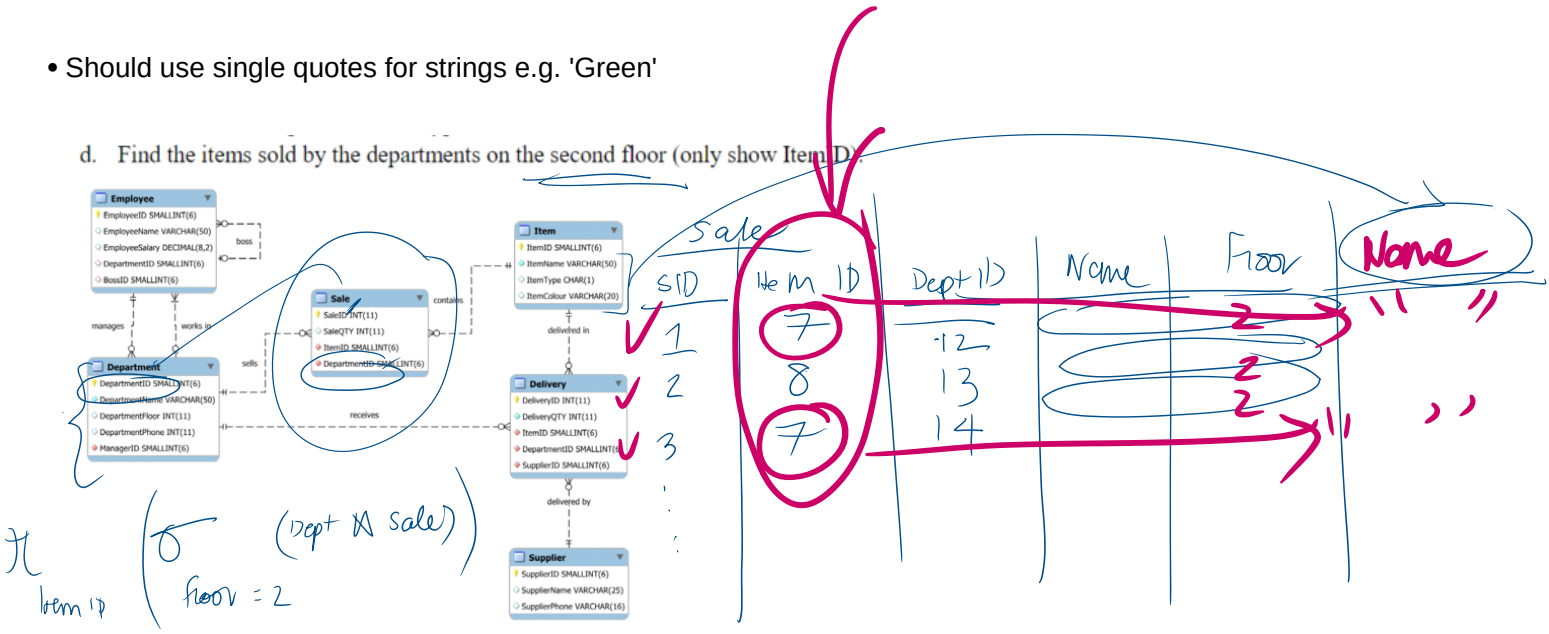
- Best to do **projection after selection!**
  - What happens if you do it the other way around?
  - If the attributes used in selection condition are not projected, then selection does not work!

c. List the names of green items of type C.



c

- Should use single quotes for strings e.g. 'Green'



d

```
SELECT DISTINCT ItemID
FROM Department NATURAL JOIN Sale
WHERE DepartmentFloor = 2;
```

- **Why do we use DISTINCT in SQL?**

- This returns only the different values
- Inside a table, columns can contain many duplicate values
- We only care about the distinct values
- MySQL doesn't discard duplicates, so we must use DISTINCT to do so

- DISTINCT
  - Sale captures each individual sale
  - A specific item can be sold many times
  - (Only SaleID is the PK)

- When you natural join Sale and Department
  - The common attribute is DepartmentID
  - The resulting schema contains all the other columns
  - (Basically you're now storing the department information for each sale)

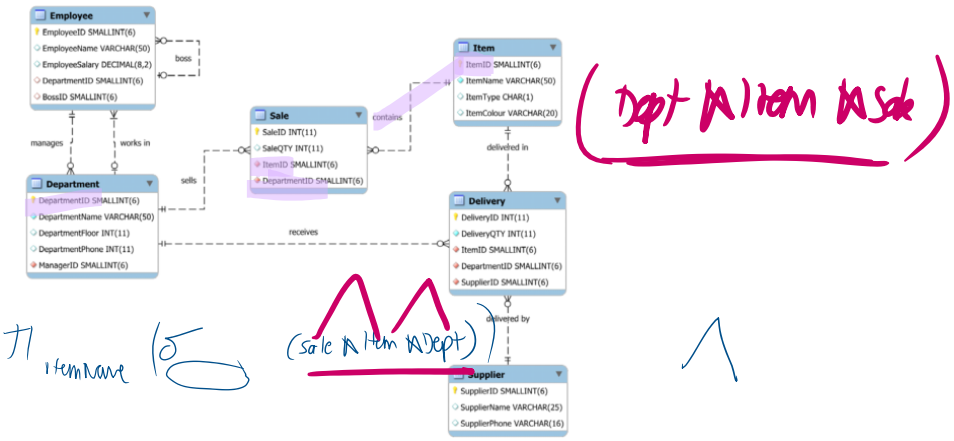
- So in the column ItemID, there may be lots of duplicate values because the same item has been sold many items
  - We don't want our list of ItemIDs to contain duplicates, we only care about the unique/distinct items sold

Sale NJ Dept

SaleID	ItemID	SaleQTY	DeptID	DeptName	....
1	1				
2	1				
3	1				
4	1				
...					

- In RA,
  - Projection (automatically) removes duplicate statements
  - so no need to worry about doing DISTINCT somehow in relational algebra
  - DISTINCT is only for SQL

e. Find the names of brown items sold by the Recreation department.

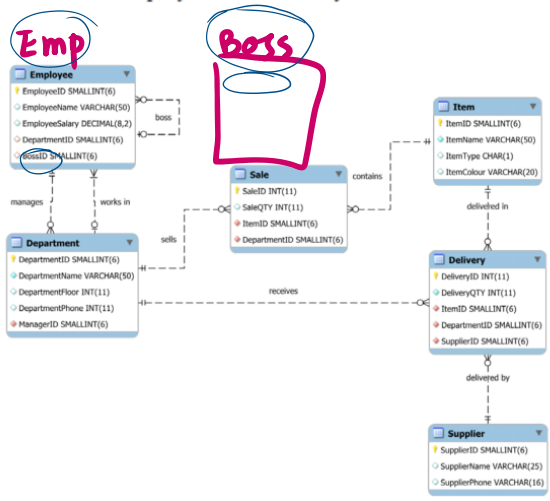


e

```
SELECT DISTINCT ItemName
FROM SALE NJ ITEM NJ DEPARTMENT
WHERE DeptName = 'Recreation' AND itemColour = 'Brown';
```

- You should also use DISTINCT in the SQL statement here
  - Natural Join gives us the Sale Table, but also storing information on the item and department of each sale
  - Items can appear many times like in d)
  - We only want the distinct item names

f. Find the employees whose salary is less than half that of their managers.



f

- Unary relationship/join occurs
- Many ways to do this - all involve some sort of renaming of tables

Solution for f)

f. Find the employees whose salary is less than half that of their managers.

Relational Algebra: *Note: The RA notation for unary joins is not agreed upon*

Here are two examples using the rename (ρ) operator:

$\rho(\text{Emp}(\text{EmployeeName} \rightarrow \text{EmpName}, \text{EmployeeSalary} \rightarrow \text{EmpSalary}, \text{BossID} \rightarrow \text{EmpBossID}), \text{Employee})$ $\rho(\text{Boss}(\text{EmployeeID} \rightarrow \text{BossEmployeeID}, \text{EmployeeSalary} \rightarrow \text{BossSalary}), \text{Employee})$ $\pi_{\text{EmpName}}(\sigma_{\text{EmpSalary} < (\text{BossSalary}/2)}(\text{Emp} \bowtie_{\text{EmpBossID} = \text{BossEmployeeID}} \text{Boss}))$	$\rho(\text{Boss}(\text{BossID} \rightarrow \text{BossBossId}, \text{EmployeeID} \rightarrow \text{BossID}, \text{Salary} \rightarrow \text{BossSalary}, \text{Name} \rightarrow \text{BossName}), \text{Employee})$ $\pi_{\text{EmployeeName}}(\sigma_{\text{EmployeeSalary} < (\text{BossSalary}/2)}(\text{Employee} \bowtie \text{Boss}))$

- LHS: renames both tables
- RHS: renames only one table
  - becomes a natural join as same attribute name

- For both LHS and RHS, you are not using notation such as table.attributeName like you are doing the the below solution using SQL notation.
  - This is because all the relevant attribute names in the two tables have been changed so they are unique.
  - So you only need to say EmployeeSalary in RHS, don't need to say Employee.EmployeeSalary because that attribute name only appears in the Employee table.

Another way:

Or you could use an SQL-like notation:

Emp := Employee

Boss := Employee

$\pi_{Emp.EmployeeName}(\sigma_{Emp.EmployeeSalary < (Boss.EmployeeSalary / 2)}($
Emp $\bowtie_{Emp.BossID = Boss.EmployeeID}$ Boss))

```
SQL: SELECT Emp.EmployeeName
     FROM Employee AS Emp
       INNER JOIN Employee AS Boss
       ON Emp.BossID = Boss.EmployeeID
     WHERE Emp.EmployeeSalary < (Boss.EmployeeSalary / 2);
```

Inner join in SQL is equi-join in RA

Solutions

a. Find the names of all employees.

Relational Algebra: $\pi_{EmployeeName}(Employee)$

```
SQL: SELECT EmployeeName
     FROM Employee;
```

b. Find the names of all employees in department number 1.

Relational Algebra: $\pi_{EmployeeName}(\sigma_{DepartmentID = 1}(Employee))$

```
SQL: SELECT EmployeeName
     FROM Employee
     WHERE DepartmentID = 1;
```

c. List the names of green items of type C.

Relational Algebra: $\pi_{ItemName}(\sigma_{ItemColour = 'Green' \wedge ItemType = 'C'}(Item))$

```
SQL: SELECT ItemName
     FROM Item
     WHERE ItemType = 'C' AND ItemColour = 'Green';
```

- We don't need distinct here in (c)
  - All ItemNames are expected to be distinct since ItemID is the PK
  - But can add it in and it won't change the result

d. Find the items sold by the departments on the second floor (only show ItemID).

Relational Algebra: $\pi_{ItemID}(\sigma_{DepartmentFloor = 2}(Sale \bowtie Department))$

```
SQL: SELECT DISTINCT ItemID
     FROM Sale NATURAL JOIN Department
     WHERE DepartmentFloor = 2;
```

e. Find the names of brown items sold by the Recreation department.

Relational Algebra: $\pi_{ItemName}(\sigma_{DepartmentName = 'Recreation' \wedge ItemColour = 'Brown'}(Item \bowtie Sale \bowtie Department))$

```
SQL: SELECT ItemName
     FROM Item NATURAL JOIN Sale NATURAL JOIN Department
     WHERE DepartmentName = 'Recreation'
       AND ItemColour = 'Brown';
```

** I would use DISTINCT in part e) as well
** SELECT DISTINCT ItemName

f. Find the employees whose salary is less than half that of their managers.

Relational Algebra: *Note: The RA notation for unary joins is not agreed upon*

Here are two examples using the rename (ρ) operator:

ρ(Emp(EmployeeName → EmpName, EmployeeSalary → EmpSalary, BossID → EmpBossID), Employee) ρ(Boss(EmployeeID → BossEmployeeID, EmployeeSalary → BossSalary), Employee) π_{EmpName} ($\sigma_{\text{EmpSalary} < (\text{BossSalary} / 2)}$ (Emp $\bowtie_{\text{EmpBossID} = \text{BossEmployeeID}}$ Boss))	ρ(Boss(BossID → BossBossId, EmployeeID → BossID, Salary → BossSalary, Name → BossName), Employee) $\pi_{\text{EmployeeName}}$ ($\sigma_{\text{EmployeeSalary} < (\text{BossSalary} / 2)}$ (Employee ⋈ Boss))

Or you could use an SQL-like notation:

Emp ≔ Employee
Boss ≔ Employee
$\pi_{\text{Emp.EmployeeName}}$ ($\sigma_{\text{Emp.EmployeeSalary} < (\text{Boss.EmployeeSalary} / 2)}$ (
Emp $\bowtie_{\text{Emp.BossID} = \text{Boss.EmployeeID}}$ Boss))

SQL:

```
SELECT Emp.EmployeeName
FROM Employee AS Emp
  INNER JOIN Employee AS Boss
  ON Emp.BossID = Boss.EmployeeID
WHERE Emp.EmployeeSalary < (Boss.EmployeeSalary / 2);
```