INFO20003 Tutorial – Week 5 Solutions

(Tutorial: Relational algebra and translation to SQL)

Objectives:

This tutorial will cover:

I. Relational algebra (RA) review – 15 mins
II. Relational algebra and SQL statements – 35 mins

Exercises:

NOTE for students: *This is a brief summary of some of the concepts taught in lecture 7 and 8. The lectures contain detailed content related to these and many more concepts. These notes should be considered quick revision instead of a sole resource for the course material.*

1. **RA review**

Algebra, in general consists of operators and atomic operands. Relational algebra is a procedural query language for relational model and has powerful ways to provide theoretical foundation for relational databases and SQL. It consists of a collection of operators, which take instance(s) of a relation as operand(s), and returns a relation instance as an output. An operator can be either unary or binary; a unary operator is applied on a single relation whereas a binary operator requires two relations to produce an output. The procedural nature of the algebra allows us to think of an algebra expression as a recipe or a plan for evaluating a query and to represent query evaluation plans.

- Fundamental operations

There are five basic operators of Relational Algebra that can form other compound operators (described in the next section). These include Selection, Projection and set operations such as cross product, set difference and set union.

  - Removal operators: Selection (σ) and Projection (π)

These operators remove components from a relation (NOT RELATIONSHIP), selection removes rows and projection removes some columns.

Projection: $\pi_{A1, A2, ..., An}(R)$ where R is a relation and A1, …, A*n* are attributes that are 'projected'. This expression creates a new relation with a subset of attributes. All the tuples are included in the new relation, but only the attributes A1, …, A*n* are kept. For example, the table *'Person'* below contains data about some "random" people:

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

FirstName	LastName
Jon	Snow
Daenerys	Targaryen
Jamie	Lannister
Night	King

Selection: σ_C(R) where R is a relation and C is a condition used to filter rows. This expression creates a new relation consisting of those rows for which C is true. For the same Person table above the following equation will produce the relation with the same schema as Person:

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

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 operators: Set-difference (–) and Union ($\cup$)

The three most common operations on set are unions, difference and intersection (explained in next section). In this section, we will describe Union and Difference operations. Every set operation has constraint that both relations such as R and S must have the same attributes with the same domains. In addition, for clarity the ordering of attributes should be kept consistent while performing set operations.

- **Union: R ∪ S** where R and S are two relations. The result will be every row which is either in R or S. Example:

GoodGuys

FirstName	LastName
Jon	Snow
Daenerys	Targaryen

BadGuys

FirstName	LastName
Cersei	Lannister
Night	King

GoodGuys ∪ BadGuys will result in:

FirstName	LastName
Jon	Snow
Daenerys	Targaryen
Cersei	Lannister
Night	King

- **Difference: R – S** where R and S are two relations. The result will be every row which is in R but not in 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

- Combine the rows from two relations: Cross Product (×)

Cross Product: R × S where R and S are two relations. Each row of R pairs with each row of S. The resulting schema has all the attributes from both relations. If some attributes have same name, rename them by using renaming operator which we will study later. 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 × Weapon will result in:

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

These operators are not adding any computational power to the language but are useful shorthand. All these operators can be expressed using the basic operators.

  - Intersection (∩)

As intersection is also a set operator. The two relations participating in this operation should be union compatible, i.e. both should have the same number of attributes and corresponding attributes must have the same data type. Intersection can be expressed using basic operators as**:**

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

where R and S are two relations. The result is a relation containing all the tuples which are present in both relations. 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 (⋈)

The natural join (R ⋈ S) identifies attributes common to each relation R and S; it creates a new relation, pairing each tuple from R and S where the common attributes are equal. Joins in general are compound operators involving cross product, selection and occasionally projection. A natural join can be broken down into following steps:

- Compute R × S
- Select rows where attributes that appear in both relations have equal values.
- Project all unique attributes and one copy of each of the common ones.

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

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:

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

- Condition Join (Theta/Inner Join)

$R \bowtie_C S$ joins rows from relation R and S such that the Boolean condition C is true. Historically C was designated with a "theta", hence the name theta-join. Most commonly C is of the type A = B, making the join an "equi-join". The condition join can be written using the basic operators as below:

$$R \bowtie_C S = \sigma_C(R \times S)$$

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

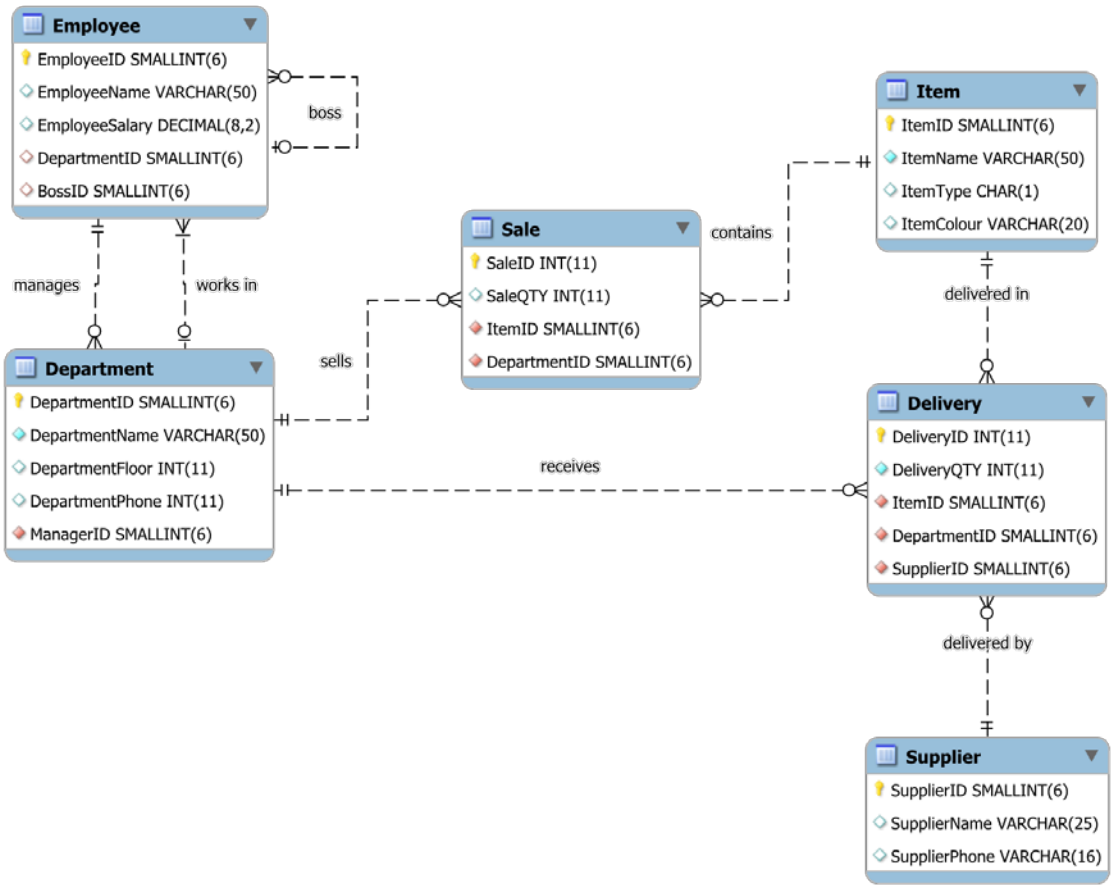
Person × Weapon (intermediate result):

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 $\bowtie_{\text{LastName = Name}}$ **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

2. Consider the following schema:



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

a. Find the names of all employees.

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

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

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

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

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

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

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

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

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

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

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

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

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:

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

Or you could use an SQL-like notation:

$\text{Emp} \coloneqq \text{Employee}$

$\text{Boss} \coloneqq \text{Employee}$

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

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