# CpSc 462/662: Database Management Systems (DBMS) (TEXNH Approach)

## Relational Algebra

James Wang

## Relational Algebra

- **Relational Algebra builds the foundation for SQL Queries.**
- **Operations:**
  - Selection: $\sigma_p(R)$. p - predicates, R - relation
  - Projection: $\pi_a(R)$. a - attributes, R - relation
  - Union: R ∪ S. R, S - relations.
  - Intersection: R ∩ S. R, S - relations.
  - Difference: R – S. R, S - relations.
  - Cross-Product: R X S. R, S - relations.
  - Renaming: $\rho_x(E)$. E is a relational expression.
  - Join: $R \bowtie_p S$. p - predicates, R, S - relations
  - Division: R ÷ S. R, S – relations
  - More …



## Primitive Operations

- **The six primitive operators of relational algebra are the *selection*, the *projection*, the *Cartesian product* (also called the cross product or cross join), the *set union*, the *set difference*, and the *rename*.**
- **Constraints:**
  - For set union and set difference, the two relations involved must be union-compatible. that is, the two relations must have the same set of attributes.
  - As set intersection can be defined in terms of set difference, the two relations involved in set intersection must also be union-compatible.
  - $R \times S$ is defined as follows:

    $R \times S = \{r\, us \mid r \in R, s \in S\}$

    ***Note:*** *two relations involved in cross product must not have a common attribute name, otherwise, the result won't be a relation.*



## Set Union and Difference

- **R U S is defined as follows:**

  R U S = {x | x∈R or x∈ S}, where R and S are relations with the same attributes.
- **R – S (same times R \ S) is defined as follows:**

  R - S = {x | x∈R and x ∉ S}, where R and S are relations with the same attributes.
- **Set intersection R ∩ S can be defined by set difference.**

  $R \cap S = \{x \mid x \in R \text{ and } x \in S\}$

  $= R - (R - S)$



## PROJECTION

- **In relational algebra, a projection is a unary operation written as $\pi_{a_1, \ldots, a_n}(R)$ where $a_1,\ldots,a_n$ is a set of attribute names.**
- **The result of such projection is defined as the set that is obtained when all tuples in *R* are restricted to the set $\{a_1,\ldots,a_n\}$.**
- **Example:**

Person

| Name | Age | Weight |
|---|---|---|
| Harry | 34 | 80 |
| Sally | 28 | 64 |
| George | 29 | 70 |
| Helena | 54 | 54 |
| Peter | 34 | 80 |

$\pi_{Age, Weight}$(Person)

| Age | Weight |
|---|---|
| 34 | 80 |
| 28 | 64 |
| 29 | 70 |
| 54 | 54 |

DISTINCT

(http://en.wikipedia.org/wiki/Projection_%28relational_algebra%29)



## SELECTION

- **In relational algebra, a selection is a unary operation written as $\sigma_{a\theta b}(R)$ or $\sigma_{a\theta v}(R)$ where:**
  - *a* and *b* are attribute names
  - θ is a binary operation in the set
  - *v* is a value constant
  - *R* is a relation
- **Selection is sometimes called a restriction to avoid confusion with SQL's use of SELECT.**
- **The selection $\sigma_{a\theta b}(R)$ selects all those tuples in *R* for which θ holds between the *a* and the *b* attribute.**
- **The selection $\sigma_{a\theta v}(R)$ selects all those tuples in *R* for which θ holds between the *a* attribute and the value *v*.**

## SELECTION Examples

Person

| Name | Age | Weight |
|---|---|---|
| Harry | 34 | 80 |
| Sally | 28 | 64 |
| George | 29 | 70 |
| Helena | 54 | 54 |
| Peter | 34 | 80 |

$\sigma_{Age \geq 34}$ (Person)

| Name | Age | Weight |
|---|---|---|
| Harry | 34 | 80 |
| Helena | 54 | 54 |
| Peter | 34 | 80 |

$\sigma_{Age = Weight}$ (Person)

| Name | Age | Weight |
|---|---|---|
| Helena | 54 | 54 |

(http://en.wikipedia.org/wiki/Selection_%28relational_algebra%29)

7

## RENAME

- **In relational algebra, a rename is a unary operation written as $\rho_{a\,/\,b}(R)$ where:**
  - *a* and *b* are attribute names
  - *R* is a relation
- **The result is identical to *R* except that the *b* field in all tuples is renamed to an *a* field.**
- **Example:**

Employ

| Name | EmployeeID |
|---|---|
| Harry | 3415 |
| Sally | 3345 |

$\rho_{EmplloyeeName\,/\,Name}$ (Employ)

| EmployeeName | EmployeeID |
|---|---|
| Harry | 3415 |
| Sally | 3345 |

8

## Natural join

- **Natural join is a binary operator that is written as $R \bowtie S$ where *R* and *S* are relations. The result of the natural join is the set of all combinations of tuples in *R* and *S* that are equal on their common attribute names.**
- **Example:**

**Employee**

| Name | EmpId | DeptName |
|---|---|---|
| Harry | 3415 | Finance |
| Sally | 2241 | Sales |
| George | 3401 | Finance |
| Harriet | 2202 | Sales |

**Dept**

| DeptName | Manager |
|---|---|
| Finance | George |
| Sales | Harriet |
| Production | Charles |

**Employee $\bowtie$ Dept**

| Name | EmpId | DeptName | Manager |
|---|---|---|---|
| Harry | 3415 | Finance | George |
| Sally | 2241 | Sales | Harriet |
| George | 3401 | Finance | George |
| Harriet | 2202 | Sales | Harriet |

9

## θ-join and equijoin

- **If we want to combine tuples from two relations where the combination condition is not simply the equality of shared attributes, θ-join (or theta-join) is necessary.**
- **The θ-join is a binary operator that is written as $R \underset{a\,\theta\,b}{\bowtie} S$ or $R \underset{a\,\theta\,v}{\bowtie} S$, where *a* and *b* are attribute names, θ is a binary relation in the set {<, ≤, =, >, ≥}, *v* is a value constant, and *R* and *S* are relations.**
- **The result of this operation consists of all combinations of tuples in *R* and *S* that satisfy the relation θ. The result of the θ-join is defined only if the headers of *S* and *R* are disjoint, that is, do not contain a common attribute.**
- **In case the operator θ is the equality operator (=) then this join is also called an equijoin.**

10

## Example of Θ-join

- **Consider tables *Car* and *Boat* which list models of cars and boats and their respective prices. Suppose a customer wants to buy a car and a boat, but she doesn't want to spend more money for the boat than for the car. The θ-join on the relation *CarPrice* ≥ *BoatPrice* produces a table with all the possible options.**

**Car**

| CarModel | CarPrice |
|---|---|
| CarA | 20'000 |
| CarB | 30'000 |
| CarC | 50'000 |

**Boat**

| BoatModel | BoatPrice |
|---|---|
| Boat1 | 10'000 |
| Boat2 | 40'000 |
| Boat3 | 60'000 |

Car $\underset{CarPrice \geq BoatPrice}{\bowtie}$ Boat

| CarModel | CarPrice | BoatModel | BoatPrice |
|---|---|---|---|
| CarA | 20'000 | Boat1 | 10'000 |
| CarB | 30'000 | Boat1 | 10'000 |
| CarC | 50'000 | Boat1 | 10'000 |
| CarC | 50'000 | Boat2 | 40'000 |

11

## Semijoin

- **The semijoin is joining similar to the natural join and written as $R \ltimes S$ where *R* and *S* are relations. The result of the semijoin is only the set of all tuples in *R* for which there is a tuple in *S* that is equal on their common attribute names.**
- **Example:**

**Employee**

| Name | EmpId | DeptName |
|---|---|---|
| Harry | 3415 | Finance |
| Sally | 2241 | Sales |
| George | 3401 | Finance |
| Harriet | 2202 | Production |

**Dept**

| DeptName | Manager |
|---|---|
| Sales | Harriet |
| Production | Charles |

**Employee $\ltimes$ Dept**

| Name | EmpId | DeptName |
|---|---|---|
| Sally | 2241 | Sales |
| Harriet | 2202 | Production |

12

## Antijoin

- The antijoin, written as $R \triangleright S$ where $R$ and $S$ are relations, is similar to the natural join, but the result of an antijoin is only those tuples in $R$ for which there is NOT a tuple in $S$ that is equal on their common attribute names.
- Example:

**Employee**

| Name | EmpId | DeptName |
|---|---|---|
| Harry | 3415 | Finance |
| Sally | 2241 | Sales |
| George | 3401 | Finance |
| Harriet | 2202 | Production |

**Dept**

| DeptName | Manager |
|---|---|
| Sales | Harriet |
| Production | Charles |

**Employee ▷ Dept**

| Name | EmpId | DeptName |
|---|---|---|
| Harry | 3415 | Finance |
| George | 3401 | Finance |

13

## Division

- The division is a binary operation that is written as $R \div S$. The result consists of the restrictions of tuples in $R$ to the attribute names unique to $R$, i.e., in the header of $R$ but not in the header of $S$, for which it holds that all their combinations with tuples in $S$ are present in $R$.
- Example:

**Completed**

| Student | Task |
|---|---|
| Fred | Database1 |
| Fred | Database2 |
| Fred | Compiler1 |
| Eugene | Database1 |
| Eugene | Compiler1 |
| Sara | Database1 |
| Sara | Database2 |

**DBProject**

| Task |
|---|
| Database1 |
| Database2 |

**Completed ÷ DBProject**

| Student |
|---|
| Fred |
| Sara |

14

## Left outer join

- The left outer join is written as $R =\!\!\times S$ where $R$ and $S$ are relations. The result of the left outer join is the set of all combinations of tuples in $R$ and $S$ that are equal on their common attribute names, in addition (loosely speaking) to tuples in $R$ that have no matching tuples in $S$.
- Example:

**Employee**

| Name | EmpId | DeptName |
|---|---|---|
| Harry | 3415 | Finance |
| Sally | 2241 | Sales |
| George | 3401 | Finance |
| Harriet | 2202 | Sales |
| Tim | 1123 | Executive |

**Dept**

| DeptName | Manager |
|---|---|
| Sales | Harriet |
| Production | Charles |

**Employee =X Dept**

| Name | EmpId | DeptName | Manager |
|---|---|---|---|
| Harry | 3415 | Finance | NULL |
| Sally | 2241 | Sales | Harriet |
| George | 3401 | Finance | NULL |
| Harriet | 2202 | Sales | Harriet |
| Tim | 1123 | Executive | NULL |

15

## Right outer join

- The right outer join is written as $R =\!\!\times S$ where $R$ and $S$ are relations. The result of the right outer join is the set of all combinations of tuples in $R$ and $S$ that are equal on their common attribute names, in addition to tuples in $S$ that have no matching tuples in $R$.
- Example:

**Employee**

| Name | EmpId | DeptName |
|---|---|---|
| Harry | 3415 | Finance |
| Sally | 2241 | Sales |
| George | 3401 | Finance |
| Harriet | 2202 | Sales |
| Tim | 1123 | Executive |

**Dept**

| DeptName | Manager |
|---|---|
| Sales | Harriet |
| Production | Charles |

**Employee =X Dept**

| Name | EmpId | DeptName | Manager |
|---|---|---|---|
| Sally | 2241 | Sales | Harriet |
| Harriet | 2202 | Sales | Harriet |
| NULL | NULL | Production | Charles |

16

## Outer join (Full outer join)

- The full outer join is written as $R =\!\!\times\!\!= S$ where $R$ and $S$ are relations. The result of the full outer join is the set of all combinations of tuples in $R$ and $S$ that are equal on their common attribute names, in addition to tuples in $S$ that have no matching tuples in $R$ and tuples in $R$ that have no matching tuples in $S$ in their common attribute names.
- Example:

**Employee**

| Name | EmpId | DeptName |
|---|---|---|
| Harry | 3415 | Finance |
| Sally | 2241 | Sales |
| George | 3401 | Finance |
| Harriet | 2202 | Sales |
| Tim | 1123 | Executive |

**Dept**

| DeptName | Manager |
|---|---|
| Sales | Harriet |
| Production | Charles |

| Name | EmpId | DeptName | Manager |
|---|---|---|---|
| Harry | 3415 | Finance | NULL |
| Sally | 2241 | Sales | Harriet |
| George | 3401 | Finance | NULL |
| Harriet | 2202 | Sales | Harriet |
| Tim | 1123 | Executive | NULL |
| NULL | NULL | Production | Charles |

17

## References


- An Introduction to Database Systems, Eighth Edition, C. J. Date, Addison Wesley, 2004, ISBN: 0-321-19784-4.
- http://en.wikipedia.org/wiki/Relational_algebra


18