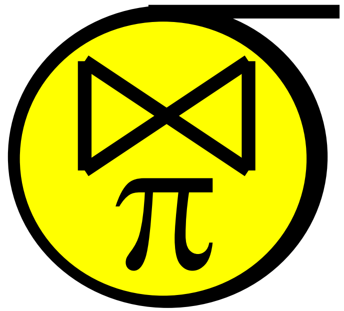
UCC

University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Information Storage and Management I

Dr. Alejandro Arbelaez



Relational Algebra

- Labs commence this week (Thursdays 3-4 PM) – **1.10**
- **Continuous Assessment (20%)**
  - In-class test (10%) – October/17/2019
  - Project Assignment (10%) – November/28/2019

shutterstock.com • 789161809

Relational Query Languages

- ***<u>Query Languages</u>:*** 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.

SELECT pname, price
FROM Product, Company
WHERE manufacturer=cname **AND**
country = 'Japan' **AND** price < 150

No details of the implementation or how to get this efficiently get this data

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*.)

Understanding Algebra & Calculus is key to understanding SQL, query processing!

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 Relational Algebra and SQL

Example Instances

Database with the following relations

- R1(<u>sid</u>, <u>bid</u>, <u>day</u>)
  - Key attributes: sid, bid, day
  - Non-key attributes: None
- S1(<u>sid</u>, sname, rating, age)
  - Key attributes: sid
  - Non-key attributes: sname, rating, age
- S2(<u>sid</u>, sname, rating, age)
  - Key attributes: sid
  - Non-key attributes

Let's assume that names of fields in query results are ***inherited*** from names of fields in query input relations.

R1

<u>sid</u>	<u>bid</u>	<u>day</u>
22	101	10/10/96
58	103	11/12/96

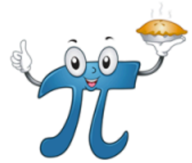
S1

<u>sid</u>	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S2

<u>sid</u>	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*** (σ)   Selects a subset of rows from relation.
  - ***Projection*** (π)   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.
  - ***Renaming*** (ρ) (for named perspective)
- Additional operations:
  - Intersection, ***join***, division, renaming: Not essential, but (very!) useful.
- Since each operation returns a relation, operations can be *composed*!

Projection -- π

- 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?)

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

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

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?)

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

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

sname	rating
yuppy	9
lubber	8
guppy	5
rusty	10

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?)

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

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

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?)

S2

<u>sid</u>	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

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

age
35.0
55.5

Selection -- σ

- 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.*)

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

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

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.*)

$S2$

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

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

sid	sname	rating	age

Same attributes

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.*)

S2

<u>sid</u>	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

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

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

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.*)

S2

<u>sid</u>	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

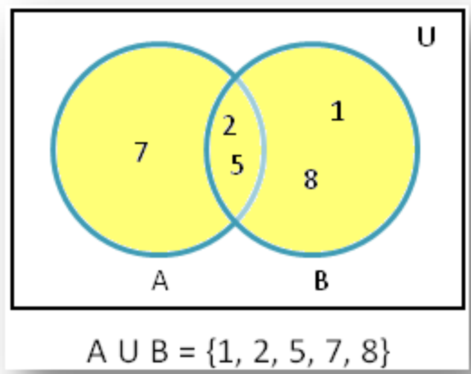
$$\pi_{sname,rating}(\sigma_{rating>8}(S2))$$

sname	rating
yuppy	9
rusty	10

Union -- ∪

- Produces a resulting relation that contains a tuple for every tuple in either or both of two input relations (duplicates only occur once)
- The Relations being combined must be **union-compatible** (type-compatible)

Set theory



Union

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

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

$S1 \cup S2$ → sid, sname, rating , age

Union

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

$S1 \cup S2$ → sid, sname, rating , age

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

$S1 \cup S2$

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

Union

SELECT * **FROM** S1

UNION

SELECT * **FROM** S2

But be careful with duplicates

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

$S1 \cup S2$

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

Implementation

- i=0
- j=0
- if T1[i] < T2[j] then print T1[i] and i+=1
- elif T1[i] > T2[j] then print T2[j] and j+=1
- elIf both are the same then print any of them and increment both I and j
- print the remaining elements of the larger array

Implementation

- i=0
- j=0
- if T1[i] < T2[j] then print T1[i] and i+=1
- elif T1[i] > T2[j] then print T2[j] and j+=1
- elif both are the same then print any of them and increment both I and j
- print the remaining elements of the larger array

T1 = {1, 3, 4, 5, 7}

T2 = {2, 3, 5, 6}

Union = { }

Two-pass algorithm based on Sorted

Sort First

Implementation

- i=0
- j=0
- if T1[i] < T2[j] then print T1[i] and i+=1
- elif T1[i] > T2[j] then print T2[j] and j+=1
- elif both are the same then print any of them and increment both I and j
- print the remaining elements of the larger array

T1 = {1, 3, 4, 5, 7}

T2 = {2, 3, 5, 6}

Union = { }

Implementation

- if T1[i] < T2[j] then print T1[i] and i+=1
- elif T1[i] > T2[j] then print T2[j] and j+=1
- elif both are the same then print any of them and increment both i and j
- print the remaining elements of the larger array

T1 = {**1**, 3, 4, 5, 7}

T2 = {**2**, 3, 5, 6}

Union = { }

Implementation

- if T1[i] < T2[j] then print T1[i] and i+=1
- elif T1[i] > T2[j] then print T2[j] and j+=1
- elif both are the same then print any of them and increment both i and j
- print the remaining elements of the larger array

T1 = {1, **3**, 4, 5, 7}

T2 = {**2**, 3, 5, 6}

Union = {1}

Implementation

- if T1[i] < T2[j] then print T1[i] and i+=1
- elif T1[i] > T2[j] then print T2[j] and j+=1
- elif both are the same then print any of them and increment both i and j
- print the remaining elements of the larger array

T1 = {1, **3**, 4, 5, 7}

T2 = {**2**, 3, 5, 6}

Union = {1}

Implementation

- if T1[i] < T2[j] then print T1[i] and i+=1
- elif T1[i] > T2[j] then print T2[j] and j+=1
- elif both are the same then print any of them and increment both i and j
- print the remaining elements of the larger array

T1 = {1, **3**, 4, 5, 7}

T2 = {2, **3**, 5, 6}

Union = {1, 2}

Implementation

- if T1[i] < T2[j] then print T1[i] and i+=1
- elif T1[i] > T2[j] then print T2[j] and j+=1
- elif both are the same then print any of them and increment both i and j
- print the remaining elements of the larger array

T1 = {1, **3**, 4, 5, 7}

T2 = {2, **3**, 5, 6}

Union = {1, 2}

Implementation

- if T1[i] < T2[j] then print T1[i] and i+=1
- elif T1[i] > T2[j] then print T2[j] and j+=1
- elif both are the same then print any of them and increment both i and j
- print the remaining elements of the larger array

T1 = {1, 3, **4**, 5, 7}

T2 = {2, 3, **5**, 6}

Union = {1, 2, 3}

Implementation

- if T1[i] < T2[j] then print T1[i] and i+=1
- elif T1[i] > T2[j] then print T2[j] and j+=1
- elif both are the same then print any of them and increment both i and j
- print the remaining elements of the larger array

T1 = {1, 3, **4**, 5, 7}

T2 = {2, 3, **5**, 6}

Union = {1, 2, 3}

Implementation

- if T1[i] < T2[j] then print T1[i] and i+=1
- elif T1[i] > T2[j] then print T2[j] and j+=1
- elif both are the same then print any of them and increment both i and j
- print the remaining elements of the larger array

T1 = {1, 3, 4, **5**, 7}

T2 = {2, 3, **5**, 6}

Union = {1, 2, 3, 4}

Implementation

- if T1[i] < T2[j] then print T1[i] and i+=1
- elif T1[i] > T2[j] then print T2[j] and j+=1
- elif both are the same then print any of them and increment both i and j
- print the remaining elements of the larger array

T1 = {1, 3, 4, **5**, 7}

T2 = {2, 3, **5**, 6}

Union = {1, 2, 3, 4}

Implementation

- if T1[i] < T2[j] then print T1[i] and i+=1
- elif T1[i] > T2[j] then print T2[j] and j+=1
- elif both are the same then print any of them and increment both i and j
- print the remaining elements of the larger array

T1 = {1, 3, 4, 5, **7**}

T2 = {2, 3, 5, **6**}

Union = {1, 2, 3, 4, 5}

Implementation

- if T1[i] < T2[j] then print T1[i] and i+=1
- elif T1[i] > T2[j] then print T2[j] and j+=1
- elif both are the same then print any of them and increment both i and j
- print the remaining elements of the larger array

T1 = {1, 3, 4, 5, **7**}

T2 = {2, 3, 5, 6}

Union = {1, 2, 3, 4, 5, 6}

Implementation

- if T1[i] < T2[j] then print T1[i] and i+=1
- elif T1[i] > T2[j] then print T2[j] and j+=1
- elif both are the same then print any of them and increment both i and j
- print the remaining elements of the larger array

T1 = {1, 3, 4, 5, **7**}

T2 = {2, 3, 5, 6}

Union = {1, 2, 3, 4, 5, 6}

Implementation

- if T1[i] < T2[j] then print T1[i] and i+=1
- elif T1[i] > T2[j] then print T2[j] and j+=1
- elif both are the same then print any of them and increment both i and j
- print the remaining elements of the larger array

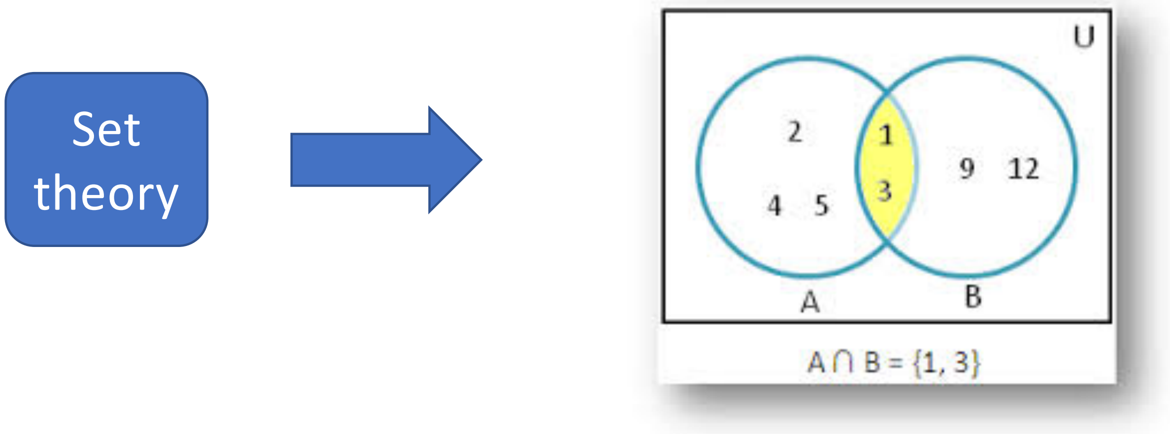
T1 = {1, 3, 4, 5, 7}

T2 = {2, 3, 5, 6}

Union = {1, 2, 3, 4, 5, 6, 7}

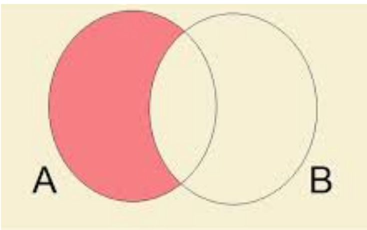
Intersection

- Produces a resulting relation that contains a tuple for every tuple in BOTH of the two input relations
- The relations being combined must be **compatible** (type-compatible)



Set-Difference (Minus)

- Produces a resulting relation that contains a tuple for every tuple in the first of two input relations and not in the second.
- The Relations being combined must be union-compatible (type-compatible)



A = {1, 2, 3, 4} **B = {2, 4, 6}**

A – B = {1, 2, 3, 4} - {2, 4, 6} = { 1, 3 }

B – A = {2, 4, 6} – {1, 2,3,4} = { 6 }

Set-Difference (Minus)

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

$S1 - S2$ → sid, sname, rating , age

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

Set-Difference (Minus)

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

$S1 - S2$ → sid, sname, rating , age

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

$S1 - S2$

sid	sname	rating	age

Set-Difference (Minus)

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

$S1 - S2$ → sid, sname, rating , age

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

$S1 - S2$

sid	sname	rating	age
22	dustin	7	45.0

Set-Difference (Minus)

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

$S1 - S2$ → sid, sname, rating , age

Unfortunately, MySQL doesn't support MINUS operator

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

$S1 - S2$

sid	sname	rating	age
22	dustin	7	45.0