

# **CS2102: Database Systems**

## Lecture 6 — Relational Algebra

# Quick Recap: The Relational Model

- Basic concept: **relations**

- Unified representation of all data using tables with rows and tables
- Relation = set of tuples (row of table) filled with atomic values (or *null*)

- Structural integrity constraints

(condition that restricts what constitutes valid data)

- Domain constraints
- Key constraints
- Foreign key constraints

Table "Movies"

id	title	genre	opened
101	Aliens	action	1986
102	Logan	drama	2017
103	Heat	crime	1995
104	Terminator	action	1984

**references relation**

Table "Cast"

movie_id	actor_id	role
101	20	Ellen Ripley
101	23	Private Hudson
102	21	Logan
104	23	Punk Leader

**referencing relation**

Missing: formal method to process and query relations → **Relational Algebra**

# Quick Recap: (Structural) Integrity Constraints

- Possible misconception

- (Foreign) key constraints are not an intrinsic property of a relation
- Constraints are specified by the DB designer to define what constitutes valid data

- Example from Lecture 1

- Without any key constraints, relation on the right is perfectly valid → DBMS does not complain
- Problematic semantics from an application perspective (e.g., CS2102 gives different credits, with and without an exam???)
- Goal: avoid different values for "mc" and "exam" for the same course → Pick {code} as primary key

code	mc	exam
cs2102	2	yes
cs2102	2	no
cs2102	4	yes
cs2102	4	no
cs3223	2	yes
...	...	...
null	4	no
null	null	no
null	null	null

# Overview

- **Relation Algebra (RA)**

- Motivation & overview
- Closure property

- **Basic operators**

- Unary operators: selection, projection, renaming
- Set operators
- Cross product

- **Join operators**

- Inner joins
- Outer joins

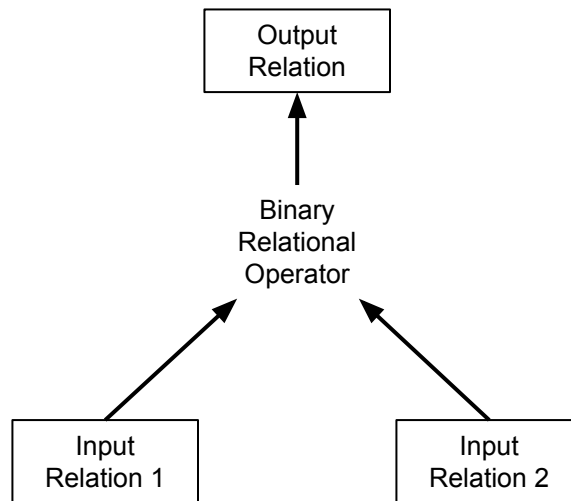
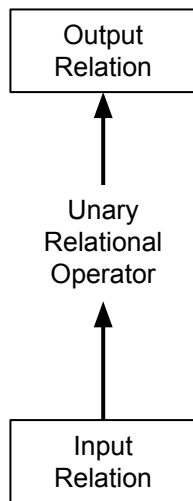
- **Complex RA expressions**

# Relational Algebra

- **Algebra** — mathematical system consisting of
    - Operands: variables or values from which new values can be constructed
    - Operators: symbols denoting procedures that construct new values from given values
  - **Relational Algebra** — procedural query language
    - Operands = relations (or variables representing relations)
    - Operators = transform one or more input relations into one output relation
  - **Basic operators of the Relational Algebra**
    - Unary operators: selection  $\sigma$ , projection  $\pi$ , (renaming  $\rho$ )
    - Binary operators: cross-product  $\times$ , union  $\cup$ , set difference  $-$
- All other RA operators can be expressed using these basic operators

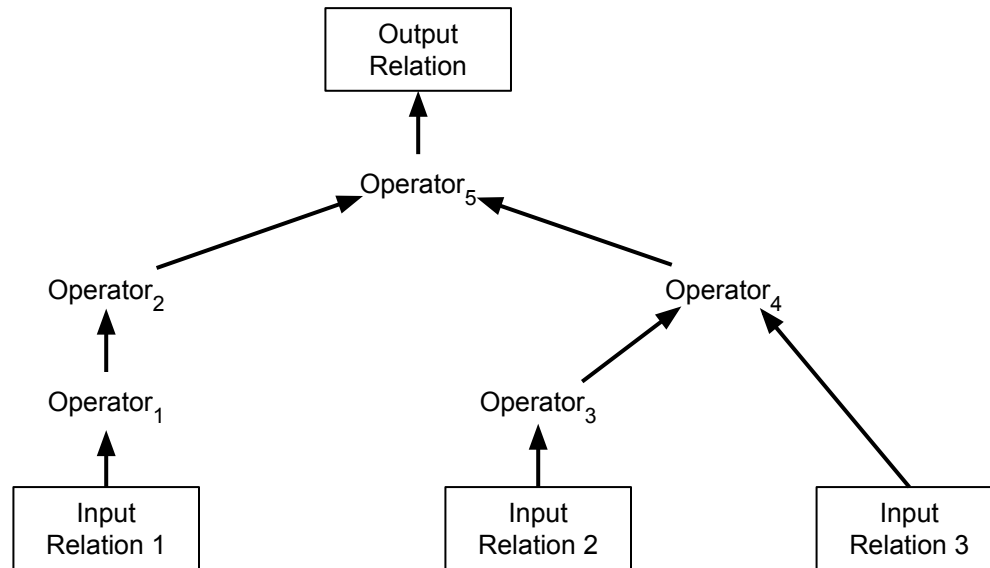
# Closure Property

- **Closure:** relations are *closed* under the Relational Algebra
  - All input operands and the outputs of all operators are relations
  - The output of one operator can serve as input for subsequent operators



# Closure Property

- Closure property allows for the nesting of relational operators  
→ **relational algebra expressions / query trees**



# Example Database

- Simplified company database schema (primary keys are underlined)

Employees (name: **text**, age: **integer**, role: **text**)

Managers (name: **text**, office: **text**)

Teams (ename: **text**, pname: **text**, hours: **integer**)

Projects (name: **text**, manager: **text**, start\_year: **integer**, end\_year: **integer**)

- Foreign key constraints

- Manager.name → Employees.name
- Teams.ename → Employees.name
- Teams.pname → Projects.name
- Projects.manager → Manager.name



# Example Database

Teams

ename	pname	hours
Sarah	BigAI	10
Sam	BigAI	5
Bill	BigAI	15
Judy	GlobalDB	20
Max	GlobalDB	5
Sarah	GlobalDB	10
Emma	GlobalDB	35
Max	CoreOS	40
Bill	CoreOS	30
Sam	CoolCoin	40
Sarah	CoolCoin	25
Emma	CoolCoin	10

Projects

name	manager	start_year	end_year
BigAI	Judy	2025	2030
FastCash	Judy	2023	2030
GlobalDB	Jack	2024	2028
CoreOS	Judy	2025	2025
CoolCoin	Jack	2020	2025

Employees

name	age	role
Sarah	25	dev
Judy	35	sales
Max	52	dev
Marie	36	hr
Sam	30	sales
Bernie	19	<i>null</i>
Emma	28	dev
Jack	40	dev
Bill	45	dev

Managers

name	office
Judy	#03-20
Jack	#03-10

# Overview

- Relation Algebra (RA)
  - Motivation & Overview
  - Closure Property
- Basic operators
  - **Unary operators:** renaming, selection, projection
  - Set operators
  - Cross product
- Join operators
  - Inner joins
  - Outer joins
- Complex RA expressions

# Renaming

- Motivation

- Different relations may share the same attribute  
(attributes only unique within same relation)
- RA expression often rely on multiple relations

→ How to ensure unambiguous RA expression?

- Useful: dot notation

- Specify attribute using relation and attribute name  
(e.g., Projects.name, Employees.name, Managers.name)
- Problem: insufficient when combining relations  
(we will see this more clearly when talking about joins)
- Annoyance: potentially a lot to type

Projects

name	manager	start_year	end_year
BigAI	Judy	2025	2030
FastCash	Judy	2023	2030
...	...	...	...

Employees

name	age	role
Sarah	25	dev
Judy	35	sales
...	...	.

Managers

name	office
Judy	#03-20
Jack	#03-10

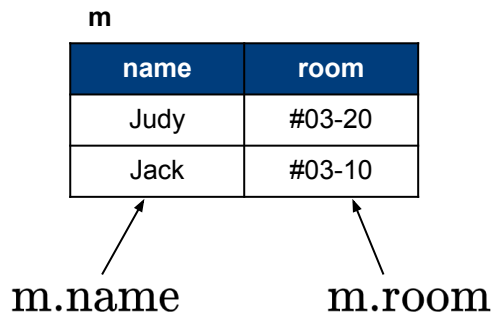
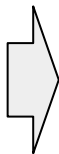
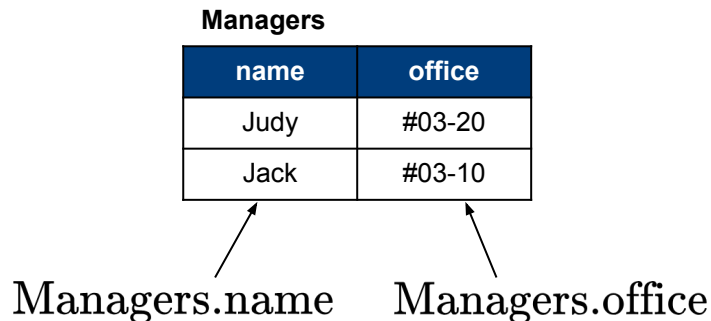
# Renaming $\rho$

- 3 ways for renaming

- Renaming a relation:  $\rho(R, S)$  (reflect aliases in SQL!)
- Renaming attributes:  $\rho(R, R(a_1 \rightarrow b_1, a_1 \rightarrow b_1, \dots))$
- Renaming a relation and attributes:  $\rho(R, S(a_1 \rightarrow b_1, a_1 \rightarrow b_1, \dots))$

} new names only valid within the scope of the RA expression!

$\rho(\text{Managers}, m(\text{office} \rightarrow \text{room}))$



# Selection $\sigma_c$

- $\sigma_c(R)$  selects all tuples from a relation  $R$  (i.e., rows from a table) that satisfy the *selection condition*  $c$ 
  - For each tuple  $t \in R$ ,  $t \in \sigma_c(R)$  iff  $c$  evaluates to **true** on  $t$
  - Input and output relation have the same schema

**Example:** Find all projects where Judy is the manager

Projects

name	manager	start_year	end_year
BigAI	Judy	2025	2030
FastCash	Judy	2023	2030
GlobalDB	Jack	2024	2028
CoreOS	Judy	2025	2025
CoolCoin	Jack	2020	2025



$\sigma_{\text{manager}='Judy'}(\text{Projects})$

name	manager	start_year	end_year
BigAI	Judy	2025	2030
FastCash	Judy	2023	2030
CoreOS	Judy	2025	2025

# Selection Conditions

- A **selection condition** is boolean expression of one of the following forms:

attribute <b>op</b> constant	$\sigma_{\text{start\_year}=2025}(\text{Projects})$	<i>constant selection</i>
attribute <sub>1</sub> <b>op</b> attribute <sub>2</sub>	$\sigma_{\text{start\_year}=\text{end\_year}}(\text{Projects})$	<i>attribute selection</i>
expr <sub>1</sub> $\wedge$ expr <sub>2</sub>	$\sigma_{\text{start\_year}=2025 \wedge \text{manager}=\text{'Judy'}}(\text{Projects})$	
expr <sub>1</sub> $\vee$ expr <sub>2</sub>	$\sigma_{\text{start\_year}=2025 \vee \text{manager}=\text{'Judy'}}(\text{Projects})$	
$\neg$ expr	$\sigma_{\neg(\text{start\_year}=2025)}(\text{Projects})$	
(expr)		

- with

- **op**  $\in \{=, <>, <, \leq, \geq, >\}$
- Operator precedence: (), **op**,  $\neg$ ,  $\wedge$ ,  $\vee$

# Selection with *null* Values

- Rules of handling *null* values

- The result of a comparison operation with *null* is ***unknown***
- The result of an arithmetic operation with *null* is ***null***

- Examples: assume that the value of *x* is *null*

$x < 2025$  → ***unknown***

$x = \text{null}$  → ***unknown***

$x <> \text{null}$  → ***unknown***

$x + 5$  → ***null***

Employees

name	age	role
...	...	...
Sam	30	sales
Bernie	19	<i>null</i>
Emma	28	dev
...	...	...

# Three-Valued Logic: true, false, unknown

$c_1$	$c_2$	$c_1 \wedge c_2$	$c_1 \vee c_2$	$\neg c_1$
false	false	false	false	true
false	unknown	false	unknown	true
false	true	false	true	true
unknown	false	false	unknown	unknown
unknown	unknown	unknown	unknown	unknown
unknown	true	unknown	true	unknown
true	false	false	true	false
true	unknown	unknown	true	false
true	true	true	true	false

Recall: For each tuple  $t \in R$ ,  $t \in \sigma_c(R)$  iff  $c$  evaluates to **true** on  $t$



# Selection — Example

**Example:** Find all employees that (a) do not work in Sales and are younger than 30 or (b) work in HR.

Employees

name	age	role
Sarah	25	dev
Judy	35	sales
Max	52	dev
Marie	36	hr
Sam	30	sales
Bernie	19	<i>null</i>
Emma	28	dev
Jack	40	dev
Bill	45	dev



$$\sigma_{(\text{role} \neq \text{'sales'} \wedge \text{age} < 30) \vee (\text{role} = \text{'hr'})}(\text{Employees})$$

name	age	role
Sarah	25	dev
Marie	36	hr
Emma	28	dev

# Projection $\pi_{\ell}$

- $\pi_{\ell}(R)$  projects all the attributes of a relation specified in list  $\ell$ 
  - i.e., projects all columns of a table specified in list  $\ell$
  - The order of attributes in  $\ell$  matters

**Example:** Find all projects and their team members

Teams

ename	pname	hours
Sarah	BigAI	10
Sam	BigAI	5
Bill	BigAI	15
Judy	GlobalDB	20
Max	GlobalDB	5
Sarah	GlobalDB	10
Emma	GlobalDB	35
Max	CoreOS	40
Bill	CoreOS	30
Sam	CoolCoin	40
Sarah	CoolCoin	25
Emma	CoolCoin	10



$\pi_{\text{pname}, \text{ename}}(\text{Teams})$

pname	ename
BigAI	Sarah
BigAI	Sam
BigAI	Bill
GlobalDB	Judy
GlobalDB	Max
GlobalDB	Sarah
GlobalDB	Emma
CoreOS	Max
CoreOS	Bill
CoolCoin	Sam
CoolCoin	Sarah
CoolCoin	Emma

# Projection $\pi_{\ell}$

- Relation = set of tuples  $\rightarrow$  duplicate tuples are removed from output relation

**Example:** Find all projects that have team members.

**Teams**

ename	pname	hours
Sarah	BigAI	10
Sam	BigAI	5
Bill	BigAI	15
Judy	GlobalDB	20
Max	GlobalDB	5
Sarah	GlobalDB	10
Emma	GlobalDB	35
Max	CoreOS	40
Bill	CoreOS	30
Sam	CoolCoin	40
Sarah	CoolCoin	25
Emma	CoolCoin	10



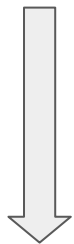
$\pi_{\text{pname}}(\text{Teams})$

pname
BigAI
GlobalDB
CoreOS
CoolCoin

# Quick Quiz

Projects

name	manager	start_year	end_year
BigAI	Judy	2025	2030
FastCash	Judy	2023	2030
GlobalDB	Jack	2024	2028
CoreOS	Judy	2025	2025
CoolCoin	Jack	2020	2025



Which **algebra expression** results in the output below?

manager	name
Judy	FastCash
Jack	GlobalDB
Jack	CoolCoin

**A**  $\sigma_{\text{start\_year} \leq 2024}(\pi_{\text{manager, name}}(\text{Projects}))$

**B**  $\pi_{\text{manager, name}}(\sigma_{\text{start\_year} < 2025}(\text{Projects}))$

**C**  $\pi_{\text{manager, name}}(\sigma_{\text{manager} = \text{'Jack'}}(\text{Projects}))$

**D**  $\pi_{\text{name, manager}}(\sigma_{\text{start\_year} = 2024}(\text{Projects}))$

# Overview

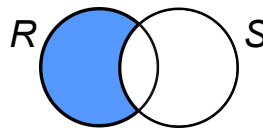
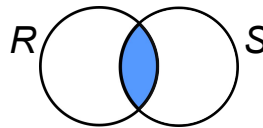
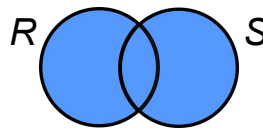
- Relation Algebra (RA)
  - Motivation & Overview
  - Closure Property
- **Basic operators**
  - Unary operators: selection, projection, renaming
  - **Set operators**
  - Cross product
- Join operators
  - Inner joins
  - Outer joins
- Complex RA expressions

# Set Operators

**Note:** The intersection operator is not fundamentally required as it can be expressed with union & difference

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

- Recall: a relation is a set of tuples
- Three set operators (given two relation  $R$  and  $S$ )
  - **Union**  $R \cup S$  returns a relation with all tuples that are in both  $R$  or  $S$
  - **Intersection**  $R \cap S$  returns a relation with all tuples that are in both  $R$  and  $S$
  - **Set difference**  $R - S$  returns a relation with all tuples that are in  $R$  but not in  $S$



- Requirement for all set operators:  $R$  and  $S$  must be **union-compatible**

# Union Compatibility (also: type compatibility)

**Note:** Just because two relations are union-compatible does not mean that a set operation is semantically meaningful.

- Two relations  $R$  and  $S$  are **union-compatible** if
  - $R$  and  $S$  have the same number of attributes and
  - The corresponding attributes have the same or compatible domains
  - But:  $R$  and  $S$  do not have to use the same attribute names

Employees (name: **text**, age: **integer**, role: **text**)

Teams (ename: **text**, pname: **text**, hours: **integer**)

Teams

ename	pname	hours
Sarah	BigAI	10
Sam	BigAI	5
...	...	...

Employees

name	age	role
Sarah	25	dev
Judy	35	sales
...	...	...

**not union-compatible**

Employees (name: **text**, role: **text**, age: **integer**)

Teams (ename: **text**, pname: **text**, hours: **integer**)

Teams

ename	pname	hours
Sarah	BigAI	10
Sam	BigAI	5
...	...	...

Employees

name	role	age
Sarah	dev	25
Judy	sales	35
...	...	...

**union-compatible**

# Set Operators — Example

**Example:** Find all projects that Bill but not Sarah are working on

Teams

ename	pname	hours
Sarah	BigAI	10
Sam	BigAI	5
Bill	BigAI	15
Judy	GlobalDB	20
Max	GlobalDB	5
Sarah	GlobalDB	10
Emma	GlobalDB	35
Max	CoreOS	40
Bill	CoreOS	30
Sam	CoolCoin	40
Sarah	CoolCoin	25
Emma	CoolCoin	10

$\pi_{\text{pname}}(\sigma_{\text{ename}='Bill'}(\text{Teams}))$

pname
BigAI
CoreOS

$\pi_{\text{pname}}(\sigma_{\text{ename}='Sarah'}(\text{Teams}))$

pname
BigAI
GlobalDB
CoolCoin

$\pi_{\text{pname}}(\sigma_{\text{ename}='Bill'}(\text{Teams})) - \pi_{\text{pname}}(\sigma_{\text{ename}='Sarah'}(\text{Teams}))$

pname
CoreOS



# Overview

- Relation Algebra (RA)
  - Motivation & Overview
  - Closure Property
- Basic operators
  - Unary operators: selection, projection, renaming
  - Set operators
  - **Cross product**
- Join operators
  - Inner joins
  - Outer joins
- Complex RA expressions

# Cross Product $\times$ (also: Cartesian Product)

- The **cross product** combines two relations  $R$  and  $S$  by forming all pairs of tuples from the two relations
- More formally, given two relations  $R(A, B, C)$  and  $S(X, Y)$ 
  - $R \times S$  returns a relation with schema  $(A, B, C, X, Y)$  defined as
  - $R \times S = \{(a, b, c, x, y) \mid (a, b, c) \in R, (x, y) \in S\}$

- Example:

$R$

A	B	C
1	0.5	m
2	2.3	f

$S$

X	Y
a	30
b	10
c	20



$R \times S$

A	B	C	X	Y
1	0.5	m	a	30
1	0.5	m	b	10
1	0.5	m	c	20
2	2.3	f	a	30
2	2.3	f	b	20
2	2.3	f	c	10

# Cross Product — Example

**Example:** Find all pairs of senior employees (age  $\geq 45$ ) and junior employees (age  $\leq 25$ )

Employees

name	age	role
Sarah	25	dev
Judy	35	sales
Max	52	dev
Marie	36	hr
Sam	30	sales
Bernie	19	<i>null</i>
Emma	28	dev
Jack	40	dev
Bill	45	dev

$$S := \pi_{\text{name}}(\sigma_{\text{age} \geq 45}(\rho(\text{Employees}, S)))$$

name
Max
Bill

$$J := \pi_{\text{name}}(\sigma_{\text{age} \leq 25}(\rho(\text{Employees}, J(\text{name} \rightarrow \text{jname}))))$$

name
Sarah
Bernie

$$S \times J$$

name	jname
Max	Sarah
Max	Bernie
Bill	Sarah
Bill	Bernie

# Cross Product — Example

**Example:** For all the projects,  
find the offices of the managers

**Managers**

name	office
Judy	#03-20
Jack	#03-10

**Projects**

name	manager	start_year	end_year
BigAI	Judy	2025	2030
FastCash	Judy	2023	2030
GlobalDB	Jack	2024	2028
CoreOS	Judy	2025	2025
CoolCoin	Jack	2020	2025



$$\pi_{\text{name,office}}(\sigma_{\text{manager}=\text{mname}}(M))$$

name	office
BigAI	#03-20
FastCash	#03-20
GlobalDB	#03-10
CoreOS	#03-20
CoolCoin	#03-10



$$M := \text{Projects} \times \rho(\text{Managers}, M(\text{name} \rightarrow \text{mname}))$$

name	manager	start_year	end_year	mname	office
BigAI	Judy	2025	2030	Judy	#03-20
BigAI	Judy	2025	2030	Jack	#03-10
FastCash	Judy	2023	2030	Judy	#03-20
FastCash	Judy	2023	2030	Jack	#03-10
GlobalDB	Jack	2024	2028	Judy	#03-20
GlobalDB	Jack	2024	2028	Jack	#03-10
CoreOS	Judy	2025	2025	Judy	#03-20
CoreOS	Judy	2025	2025	Jack	#03-10
CoolCoin	Jack	2020	2025	Judy	#03-20
CoolCoin	Jack	2020	2025	Jack	#03-10

# Cross Product — Discussion

- Observation:

- Given two relations  $R$  and  $S$ , the size of the cross product is  $|R|^*|S|$
- In practice, many to most queries requiring a cross product also require a attribute selection that removes formed pairs of tuples

$$\pi_{\text{name,office}}(\sigma_{\text{manager=mname}}(\text{Projects} \times \rho(\text{Managers}, \text{M}(\text{name} \rightarrow \text{mname}))))$$

- Goal: Make use of this observation to

- simplify Relational Algebra expressions and
- avoid generating all  $|R|^*|S|$  output tuples  
(when implementing all algebra operators within a DBMS)

→ Join operators

# Overview

- **Relation Algebra (RA)**
  - Motivation & Overview
  - Closure Property
- **Basic operators**
  - Unary operators: selection, projection, renaming
  - Set operators
  - Cross product
- **Join operators**
  - Inner joins
  - Outer joins
- **Complex RA expressions**

# Join Operators

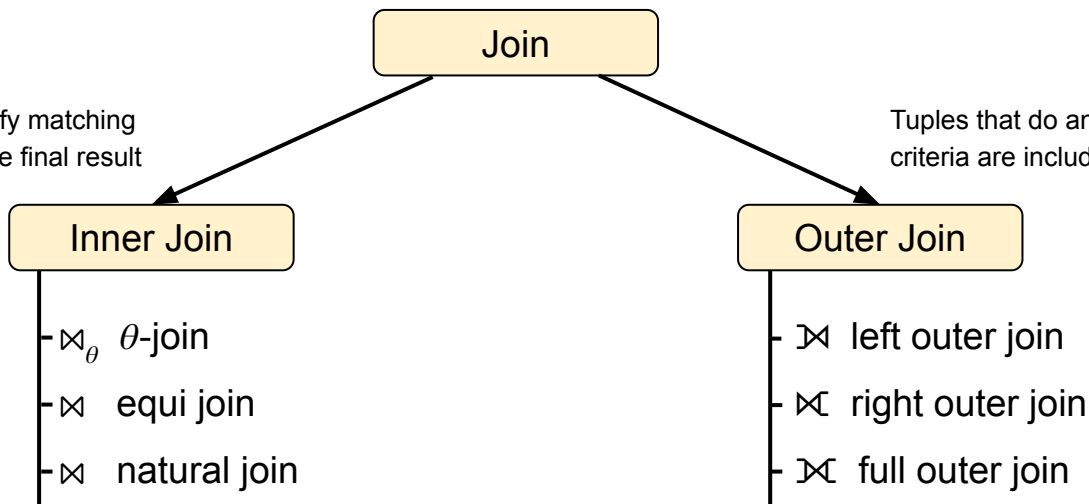
- Join operator — basic idea

- Combines cross product, attribute selection and possibly projection into a single operator
- Typically results in simpler relational algebra expressions when formulating queries

- Base types

Only the tuples that satisfy matching criteria are included in the final result

Tuples that do and do not satisfy the matching criteria are included in the final result



# Inner Joins — $\theta$ -Join

- The  $\theta$ -join  $R \bowtie_{\theta} S$  of two relations  $R$  and  $S$  is defined as

$$R \bowtie_{\theta} S = \sigma_{\theta}(R \times S)$$

**Managers**

name	office
Judy	#03-20
Jack	#03-10

**Example:** For all the projects,  
find the offices of the managers

**Note:** final projection omitted  
here to show result of  $\theta$ -join

**Projects**

name	manager	start_year	end_year
BigAI	Judy	2025	2030
FastCash	Judy	2023	2030
GlobalDB	Jack	2024	2028
CoreOS	Judy	2025	2025
CoolCoin	Jack	2020	2025



Projects  $\bowtie_{\text{manager=mname}}$   $\rho(\text{Managers}, M(\text{name} \rightarrow \text{mname}))$

name	manager	start_year	end_year	mname	office
BigAI	Judy	2025	2030	Judy	#03-20
FastCash	Judy	2023	2030	Judy	#03-20
GlobalDB	Jack	2024	2028	Jack	#03-10
CoreOS	Judy	2025	2025	Judy	#03-20
CoolCoin	Jack	2020	2025	Jack	#03-10



# Inner Joins — Equi Join $\bowtie_{=}$

- Difference between  $\theta$ -join and equi join is only w.r.t. matching criteria
  - The  $\theta$ -join  $\bowtie_{\theta}$  allows arbitrary comparison operators for the attribute selection (e.g., =, <>, <, ≤, ≥, >)
  - The equi join  $\bowtie$  is a special case of  $\theta$ -join by defined over the equality operator (=) only
  - Dedicated term since attribute selections using the equality operator are most common (particularly when joining along foreign key constraints)
- Example
  - see previous slide

# Natural Join

- Same as equi join (i.e., only equality operator) but:
  - The join is performed over all attributes that R and S have in common (this means that no explicit matching criteria has to be specified)
  - The output relation contains the common attributes of R and S only once (compared with the  $\theta$ -join and equi join that contain all attributes of both relations)
- More formally, the **natural join** of two relations R and S is defined as

$$R \bowtie S = \pi_{\ell}(R \bowtie_c \rho_{b_i \leftarrow a_i, \dots, b_k \leftarrow a_k}(S))$$

- $A = \{a_i, \dots, a_k\}$  is the set of attributes that R and S have in common
- $c = ((a_i = b_i) \wedge \dots \wedge (a_k = b_k))$
- $\ell$  = list of all attributes of R + list of all attributes in S that are not in R

# Natural Join — Example

**Quick Quiz:** What would be the result of  
Projects  $\bowtie$  Managers

**Example:** For all the projects,  
find the offices of the managers

**Managers**

name	office
Judy	#03-20
Jack	#03-10

$\rho(\text{Managers}, m(\text{name} \rightarrow \text{manager}))$



manager	office
Judy	#03-20
Jack	#03-10

**Projects**

name	manager	start_year	end_year
BigAI	Judy	2020	2025
FastCash	Judy	2018	2025
GlobalDB	Jack	2019	2023
CoreOS	Judy	2020	2020
CoolCoin	Jack	2015	2020

**Note:** final projection omitted  
to show result of natural join

Projects  $\bowtie \rho(\text{Managers}, m(\text{name} \rightarrow \text{manager}))$

name	manager	start_year	end_year	office
BigAI	Judy	2020	2025	#03-20
FastCash	Judy	2018	2025	#03-20
GlobalDB	Jack	2019	2023	#03-10
CoreOS	Judy	2020	2020	#03-20
CoolCoin	Jack	2015	2020	#03-10

# Outer Joins

**Note:** This simple example can easily be solved using projection and set difference.

## • Motivation

- Inner joins eliminate all tuples that do not satisfy matching criteria (i.e., attribute selection)
- Sometimes the tuples in  $R$  or  $S$  that do not match with tuples in the other relation are of interest

→ *dangling tuples*

**Example:** Find all employees that are not assigned to any project.

An inner join can only find all employees that are assigned to at least one project.

**Employees**

name	age	role
Sarah	25	dev
Judy	35	sales
Max	52	dev
Marie	36	hr
Sam	30	sales
Bernie	19	null
Emma	28	dev
Jack	40	dev
Bill	45	dev

**Teams**

ename	pname	hours
Sarah	BigAI	10
Sam	BigAI	5
Bill	BigAI	15
Judy	GlobalDB	20
Max	GlobalDB	5
Sarah	GlobalDB	10
Emma	GlobalDB	35
Max	CoreOS	40
Bill	CoreOS	30
Sam	CoolCoin	40
Sarah	CoolCoin	25
Emma	CoolCoin	10

# Outer Joins

**Quick Quiz:** Why is there  $\pi_{\text{ename}}(\text{Teams})$  and do we really need it?

- Processing steps of an outer join between  $R$  and  $S$  (informal)

- Perform inner join  $M = R \bowtie_{\theta} S$

- To  $M$ , add dangling tuples to result of

- $R$  in case of a **left outer join**  $\bowtie_{\theta}$

- $S$  in case of a **right outer join**  $\bowtie_{\theta}$

- $R$  and  $S$  in case of a **full outer join**  $\bowtie_{\theta}$

- "Pad" missing attribute values of dangling tuples with *null*

**Example:** Find all employees that are not assigned to any project.

$\text{Employees} \bowtie_{\text{name=ename}} (\pi_{\text{ename}}(\text{Teams}))$

name	age	role	ename
...	...	...	...
Emma	28	dev	Emma
Bill	45	dev	Bill
Marie	36	hr	<i>null</i>
Bernie	19	<i>null</i>	<i>null</i>
Jack	40	dev	<i>null</i>

} inner join result

} dangling tuples with *null* padding

# Outer Joins — Formal Definitions

name	age	role	ename
...	...	...	...
Jack	40	dev	Jack
Bill	45	dev	Bill
Marie	36	hr	<i>null</i>
Bernie	19	<i>null</i>	<i>null</i>

- Auxiliary definitions

- $dangle(R \bowtie_{\theta} S)$  = set of dangling tuples in  $R$  w.r.t.  $R \bowtie_{\theta} S$

- $dangle(R \bowtie_{\theta} S) \subseteq R$

- $null(R)$  =  $n$ -component tuple of null values where  $n$  is the number of attributes of  $R$

- e.g.,  $null(Teams) = (null, null, null)$

- Definitions (outer joins)

**Left outer join**      $R \Join_{\theta} S = R \bowtie_{\theta} S \cup (dangle(R \bowtie_{\theta} S) \times \{null(S)\})$

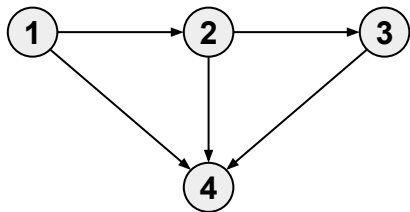
**Right outer join**      $R \Join_{\theta} S = R \bowtie_{\theta} S \cup (\{null(R)\} \times dangle(S \bowtie_{\theta} R))$

**Full outer join**      $R \Join_{\theta} S = R \bowtie_{\theta} S \cup (dangle(R \bowtie_{\theta} S) \times \{null(S)\}) \cup (\{null(R)\} \times dangle(S \bowtie_{\theta} R))$

inner join

dangling tuples with *null* padding

# Full Outer Join — Example



**Example:** Find all nodes with no incoming or outgoing edge

$$\text{Edges} \bowtie_{t=\text{in}} \rho(\text{Edges}, e(s \rightarrow \text{in}, t \rightarrow \text{out}))$$

Edges

s	t
1	2
1	4
2	3
2	4
3	4

s	t	in	out
null	null	1	2
null	null	1	4
1	2	2	3
1	2	2	4
2	3	3	4
1	4	null	null
2	4	null	null
3	4	null	null

# Natural Outer Joins

- Analog to natural (inner) join
  - Only the equality operator is used for the join condition
  - The join is performed over all attributes that  $R$  and  $S$  have in common
  - The output relations contains the common attributes of  $R$  and  $S$  only once

Natural left outer join      $R \bowtie\!\!\!\bowtie S$

Natural right outer join      $R \ltimes S$

Natural full outer join      $R \Join S$

Edges

s	t
1	2
1	4
2	3
2	4
3	4

**Quick Quiz:** What is the result of the expression:

Edges  $\Join$  Edges



# Quick Quiz

Teams

ename	pname	hours
Sarah	BigAI	10
Sam	BigAI	5
Bill	BigAI	15
Judy	GlobalDB	20
Max	GlobalDB	5
Sarah	GlobalDB	10
Emma	GlobalDB	35
Max	CoreOS	40
Bill	CoreOS	30
Sam	CoolCoin	40
Sarah	CoolCoin	25
Emma	CoolCoin	10

Managers

name	office
Judy	#03-20
Jack	#03-10

How many **rows & columns** has the result of the algebra expression below?

$\sigma_{\text{ename} \equiv \text{null}}(\text{Managers} \bowtie_{\text{name}=\text{ename}} \text{Teams})$



A

1 row, 5 cols

B

2 rows, 5 cols

C

1 row, 3 cols

D

2 rows, 3 cols

**Note:** We use  $\equiv$  to check if a value is null or not; recall that the basic comparison with  $=$  will yield "unknown" and thus not return the desired result. SQL has the special comparison "... IS (NOT) NULL" (cf. later lectures) which the basic Relational Algebra lacks.

# Overview

- **Relation Algebra (RA)**
  - Motivation & Overview
  - Closure Property
- **Basic operators**
  - Unary operators: selection, projection, renaming
  - Set operators
  - Cross product
- **Join operators**
  - Inner joins
  - Outer joins
- **Complex RA expressions**

# Complex Relational Expressions (Queries)

**Example:** Find all managers (with their offices) of projects that started 2025 or later, where at least one member of the project team has to work 30h or more on that project per week!

$$P := \sigma_{\text{start\_year} \geq 2025}(\text{Projects})$$

name	manager	start_year	end_year
BigAI	Judy	2025	2030
CoreOS	Judy	2025	2030

$$M := \pi_{\text{name,manager,office}}(P \bowtie \rho(\text{Managers}, M(\text{name} \rightarrow \text{manager})))$$

name	manager	office
BigAI	Judy	#03-20
CoreOS	Judy	#03-20

$$W := \sigma_{\text{hours} \geq 30}(\text{Teams})$$

ename	pname	hours
Emma	GlobalDB	35
Max	CoreOS	40
Bill	CoreOS	30
Sam	CoolCoin	40

$$T := W \bowtie_{\text{pname=name}} M$$

ename	pname	hours	name	manager	office
Max	CoreOS	40	CoreOS	Judy	#03-20
Bill	CoreOS	30	CoreOS	Judy	#03-20

**Managers**

name	office
Judy	#03-20
Jack	#03-10

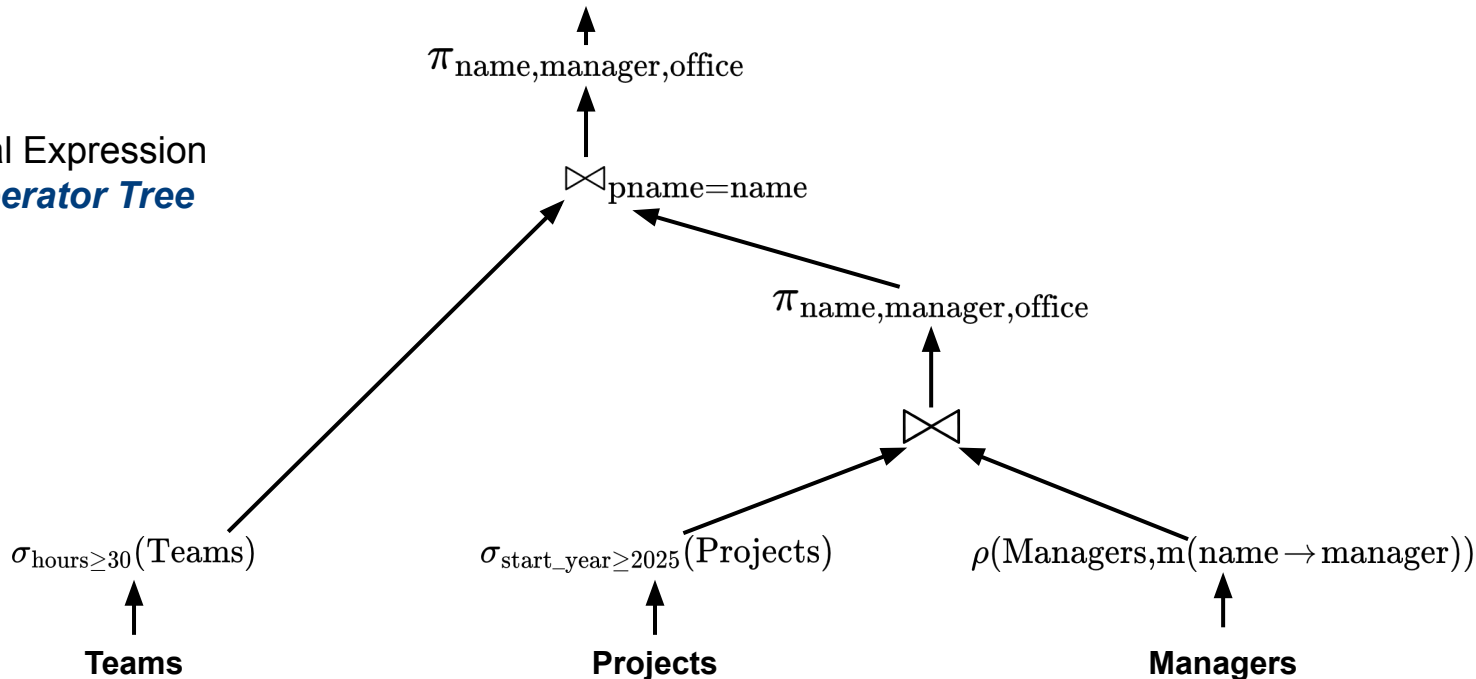
$$\pi_{\text{name,manager,office}}(T)$$

name	manager	office
CoreOS	Judy	#03-20

# Complex Relational Expressions (Queries)

**Example:** Find all managers (with their offices) of projects that started 2025 or later, where at least one member of the project team has to work more 30h or more on that project per week!

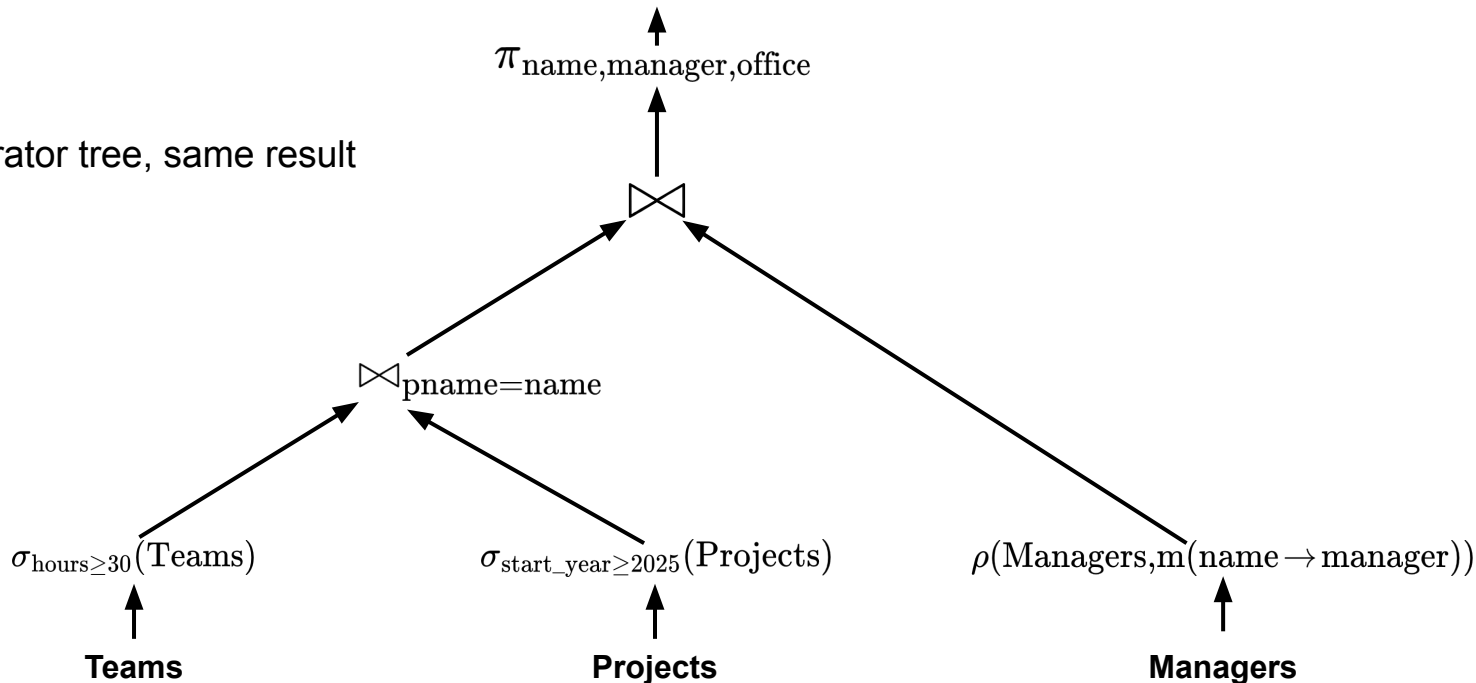
Relational Expression  
as an *Operator Tree*



# Complex Relational Expressions (Queries)

**Example:** Find all managers (with their offices) of projects that started 2025 or later, where at least one member of the project team has to work more 30h or more on that project per week!

Different operator tree, same result



# Complex Relational Expressions — Observation

- In general, multiple ways to formulate a query to get the same result, e.g.,
  - Order in which join operations are performed
  - Order in which selection operations are performed (e.g., before or after join operators)
  - Inserting additional projections to minimize intermediate results
  - ...and many more
- Finding the "best" operator tree → **query optimization**
  - Handled by the DBMS transparent to the user
  - Covered in, e.g., CS3223

# Invalid Relational Expressions — Examples

- Attribute no longer available after projection

$$\sigma_{\text{role}='dev'}(\pi_{\text{name,age}}(\text{Employees}))$$

- Attribute no longer available after renaming

$$\sigma_{\text{role}='dev'}(\rho(\text{Employees}, \text{emp}(\text{role} \rightarrow \text{position})))$$

- Incompatible attribute types

$$\sigma_{\text{age}=\text{role}}(\text{Employees})$$

# Valid but not "Smart" Expressions — Examples

- Cross product + attribute selection instead join

$$\sigma_{\text{manager}=\text{m.name}}(\text{Projects} \times \rho(\text{Managers}, \text{m})) \rightarrow \text{Projects} \bowtie_{\text{manager}=\text{m.name}} \rho(\text{Managers}, \text{m})$$

- Unnecessary operators

$$\pi_{\text{name}}(\pi_{\text{name}, \text{age}}(\text{Employees})) \rightarrow \pi_{\text{name}}(\text{Employees})$$

- Query optimization (performance)

$$\sigma_{\text{start\_year}=2025}(\text{Projects} \bowtie_{\text{manager}=\text{m.name}} \rho(\text{Managers}, \text{m}))$$

$$\rightarrow (\sigma_{\text{start\_year}=2025}(\text{Projects})) \bowtie_{\text{manager}=\text{m.name}} \rho(\text{Managers}, \text{m})$$

**Note:** query optimization is beyond the scope of CS2102 and covered in other courses (e.g., CS3223). A solid grasp of the Relational Algebra is very important for this topic.



# Relational Algebra vs. SQL

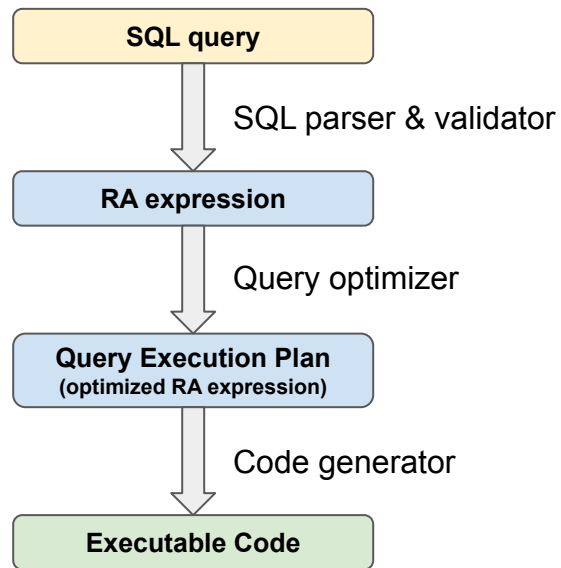
- **Relation Algebra (RA)**

- Procedural query language using operators to perform queries
- Query = relational algebra expression (operator tree)

- **SQL** (more precisely: the DQL part of SQL)


- **Declarative** query language built on top of RA  
(Focus on *what* to compute, not on *how* to compute)
- Multiset / bag semantics (unlike sets in RA!)
- Query = SELECT statement

<b>SELECT</b>	[ <b>DISTINCT</b> ] target-list
<b>FROM</b>	relation-list
[ <b>WHERE</b>	conditions ]



# Relational Algebra vs. SQL

- Mapping between basic SQL query to corresponding RA expression
  - Conceptual mapping; no consideration of performance

<b>SELECT</b>	<b>DISTINCT</b>	$a_1, a_2, \dots, a_m$		$\pi_{a_1, a_2, \dots, a_m} (\sigma_c(r_1 \times r_2 \times \dots \times r_n))$
<b>FROM</b>		$r_1, r_2, \dots, r_n$		
<b>WHERE</b>		$c$		

- Practical use when working with database
  - Database tuning (e.g., if different queries yield the same result, which runs faster)
  - Relevant command: **EXPLAIN** (shows query plan with operators and specific implementations)

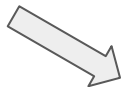
# Recall: Flexibility of SQL

*Find all names that refer to both a city and a country.*

(**SELECT** name **FROM** cities)

**INTERSECT**

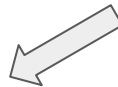
(**SELECT** name **FROM** countries);



**SELECT** n.name

**FROM** countries n

**WHERE EXISTS** (**SELECT** c.name  
**FROM** cities c  
**WHERE** c.name = n.name);

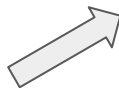


**SELECT DISTINCT**(name)

**FROM** (**SELECT** name **FROM** cities) t1

**NATURAL JOIN**

(**SELECT** name **FROM** countries) t2;



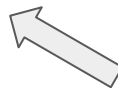
name
Singapore
Mexico
Peru
Monaco
Mali
El Salvador
China
Poland
...

29 tuples

**SELECT** name

**FROM** countries


**WHERE** name **IN** (**SELECT** name  
**FROM** cities);



# Recall: Flexibility of SQL

```
EXPLAIN SELECT n.name  
FROM countries n  
WHERE EXISTS (SELECT c.name  
               FROM cities c  
               WHERE c.name = n.name);
```

```
EXPLAIN SELECT name  
FROM countries  
WHERE name IN (SELECT name  
               FROM cities);
```

	QUERY PLAN	
	text	
1	Nested Loop Semi Join (cost=0.29..549.36 rows=196 width=9)	
2	-> Seq Scan on countries n (cost=0.00..3.96 rows=196 width=9)	
3	-> Index Only Scan using cities_pkey on cities c (cost=0.29..2.82 rows=1 width=9)	
4	Index Cond: (name = (n.name)::text)	


# Recall: Flexibility of SQL

## EXPLAIN

(**SELECT** name **FROM** cities)

## INTERSECT

(**SELECT** name **FROM** countries);


	QUERY PLAN text	
1	HashSetOp Intersect (cost=0.00..1358.19 rows=196 width=520)	
2	-> Append (cost=0.00..1257.35 rows=40334 width=520)	
3	-> Subquery Scan on "*SELECT* 2" (cost=0.00..5.92 rows=196 width=13)	
4	-> Seq Scan on countries (cost=0.00..3.96 rows=196 width=9)	
5	-> Subquery Scan on "*SELECT* 1" (cost=0.00..1049.76 rows=40138 width=10)	
6	-> Seq Scan on cities (cost=0.00..648.38 rows=40138 width=10)	

**EXPLAIN SELECT DISTINCT**(name)

**FROM** (**SELECT** name **FROM** cities) t1

## NATURAL JOIN

(**SELECT** name **FROM** countries) t2;

	QUERY PLAN text	
1	HashAggregate (cost=558.70..560.73 rows=203 width=10)	
2	Group Key: cities.name	
3	-> Nested Loop (cost=0.29..558.19 rows=203 width=10)	
4	-> Seq Scan on countries (cost=0.00..3.96 rows=196 width=9)	
5	-> Index Only Scan using cities_pkey on cities (cost=0.29..2.82 rows=1 width=10)	
6	Index Cond: (name = (countries.name)::text)	

# Summary

- Relational Algebra

- Formal method to query relational data
- Closure property for arbitrarily complex relational expressions
- Basis for DB query languages such as SQL
- Important for query (automated) optimization

- Most common operators

- Unary operators: selection, projection, renaming
- Binary operators: set operators, (cross product), joins

# Quick Quiz Solutions

## Quick Quiz (Slide 20)





## Quick Quiz (Slide 35)



## Quick Quiz (Slide 37)



## Quick Quiz (Slide 40)



## Quick Quiz (Slide 41)

