

Relational Algebra

BBM471 Database Management Systems

Dr. Fuat Akal

akal@hacettepe.edu.tr



Today's Lecture

1. The Relational Model & Relational Algebra
2. Relational Algebra Pt. II

1. The Relational Model & Relational Algebra

What you will learn about in this section

1. The Relational Model
2. Relational Algebra: Basic Operators
3. Execution

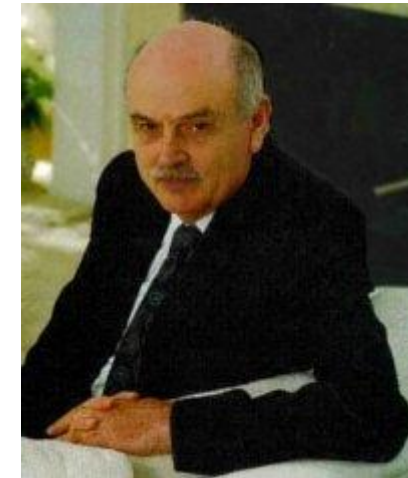
Motivation

The Relational model is **precise**,
implementable, and we can operate on it
(query/update, etc.)

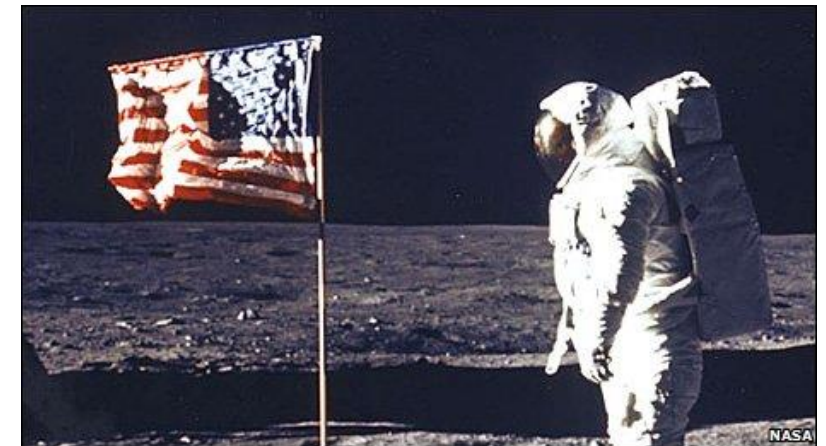
Database maps internally into this
procedural language.

A Little History

- Relational model due to Edgar “Ted” Codd, a mathematician at IBM in 1970
 - [A Relational Model of Data for Large Shared Data Banks](#)". [Communications of the ACM](#) **13** (6): 377–387
- IBM didn't want to use relational model
 - *Apparently used in the moon landing...*
 - *Google for “IMS and the Apollo program”*

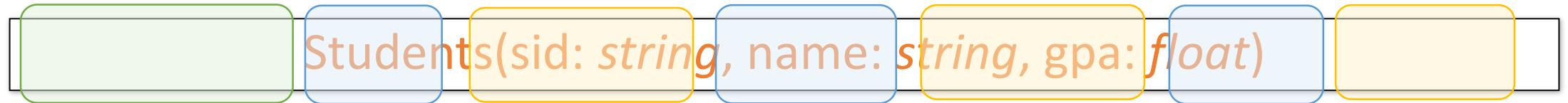


Won Turing
award 1981



The Relational Model: Schemata

- Relational Schema:



Relation name

String, float, int, etc.
are the **domains** of
the attributes

Attributes

The Relational Model: Data

An attribute (or column) is a typed data entry present in each tuple in the relation

Student

sid	name	gpa
001	Bob	3.2
002	Joe	2.8
003	Mary	3.8
004	Alice	3.5

The number of attributes is the arity of the relation

The Relational Model: Data

Student

sid	name	gpa
001	Bob	3.2
002	Joe	2.8
003	Mary	3.8
004	Alice	3.5

The number of tuples is the **cardinality** of the relation

A **tuple** or **row** (or *record*) is a single entry in the table having the attributes specified by the schema

The Relational Model: Data

Student

sid	name	gpa
001	Bob	3.2
002	Joe	2.8
003	Mary	3.8
004	Alice	3.5

Recall: In practice DBMSs relax the set requirement, and use multisets.

A relational instance is a *set* of tuples all conforming to the same *schema*

To Reiterate

- A relational schema describes the data that is contained in a relational instance

Let $R(f_1:\text{Dom}_1, \dots, f_m:\text{Dom}_m)$ be a relational schema then, an instance of R is a subset of $\text{Dom}_1 \times \text{Dom}_2 \times \dots \times \text{Dom}_n$

In this way, a relational schema R is a **total function from attribute names to types**

One More Time

- A relational schema describes the data that is contained in a relational instance

A relation R of arity t is a function:
 $R : \text{Dom}_1 \times \dots \times \text{Dom}_t \rightarrow \{0,1\}$

i.e. returns whether or not a tuple of matching types is a member of it

Then, the schema is simply the *signature* of the function

Note here that order matters, attribute name doesn't...
We'll (mostly) work with the other model (last slide) in which **attribute name matters, order doesn't!**

A relational database

- A relational database schema is a set of relational schemata, one for each relation
- A relational database instance is a set of relational instances, one for each relation

Two conventions:

1. We call relational database instances as simply *databases*
2. We assume all instances are valid, i.e., satisfy the domain constraints

Remember the CMS

- *Relation DB Schema*

- Students(sid: *string*, name: *string*, gpa: *float*)
- Courses(cid: *string*, cname: *string*, credits: *int*)
- Enrolled(sid: *string*, cid: *string*, grade: *string*)

Note that the schemas impose effective domain / type constraints, i.e. Gpa can't be "Apple"

Sid	Name	Gpa
101	Bob	3.2
123	Mary	3.8

Students

Relation
Instances

sid	cid	Grade
123	564	A

Enrolled

cid	cname	credits
564	564-2	4
308	417	2

Courses

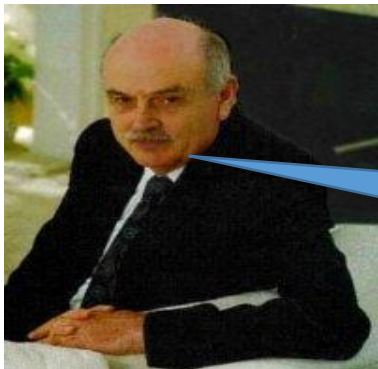
2nd Part of the Model: Querying

```
SELECT S.name  
FROM Students S  
WHERE S.gpa > 3.5;
```

We don't tell the system *how* or *where* to get the data- **just what we want**, i.e., Querying is declarative

*“Find names of all students
with GPA > 3.5”*

To make this happen, we need to translate the *declarative* query into a series of operators... we'll see this next!



Actually, I showed how to do this translation for a much richer language!

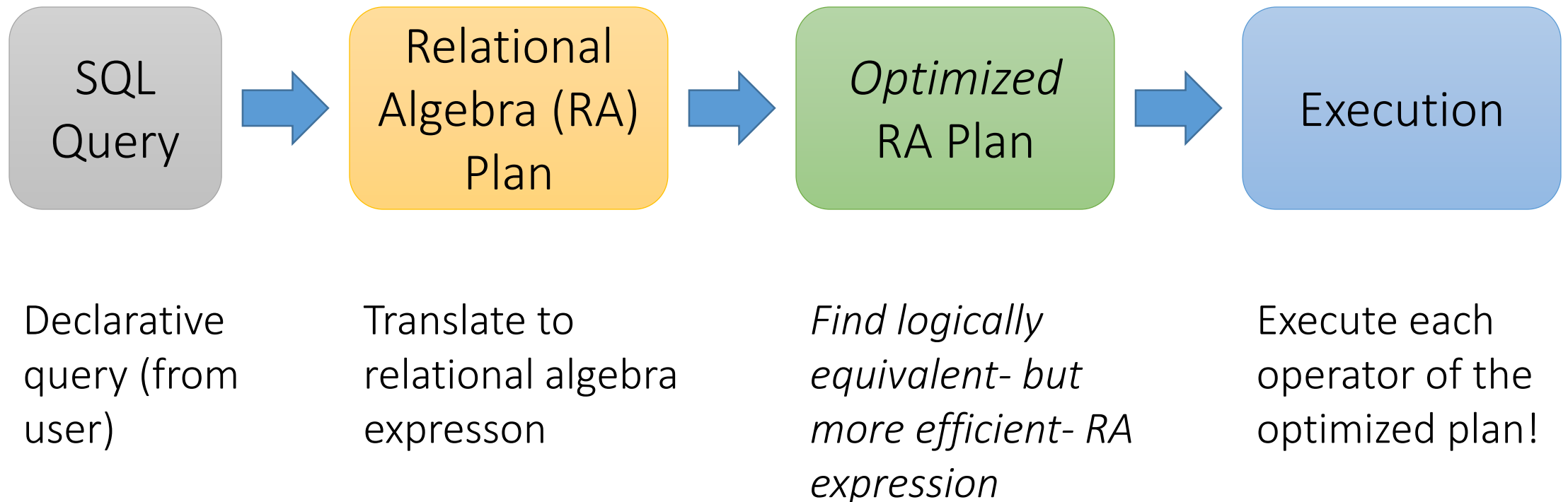
Virtues of the model

- Physical independence (logical too), Declarative
- Simple, elegant, clean: Everything is a relation
- Why did it take multiple years?
 - Doubted it could be done *efficiently*.



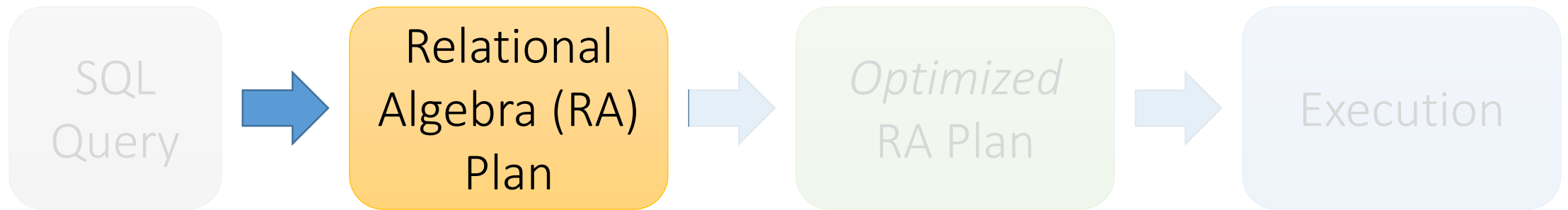
RDBMS Architecture

How does an SQL engine work ?



RDBMS Architecture

How does a SQL engine work ?



Relational Algebra allows us to translate declarative (SQL) queries into precise and optimizable expressions!

Relational Algebra (RA)

- Five **basic** operators:
 - Selection: σ
 - Projection: Π
 - Cartesian Product: \times
 - Union: \cup
 - Difference: $-$
- Derived or auxiliary operators:
 - Intersection, complement
 - Joins (natural, equi-join, theta join, semi-join)
 - Renaming: ρ
 - Division

Keep in mind: RA operates on sets!

- RDBMSs use *multisets*, however in relational algebra formalism we will consider **sets**!
- Also: we will consider the ***named perspective***, where every attribute must have a unique name
 - → attribute order does not matter...

Now on to the basic RA operators...

Selection (σ)

- Returns all tuples which satisfy a condition
- Notation: $\sigma_c(R)$
- Examples
 - $\sigma_{\text{Salary} > 40000}(\text{Employee})$
 - $\sigma_{\text{name} = \text{"Smith"}}(\text{Employee})$
- The condition c can be
 - $=, <, \leq, >, \geq, <>$

Students(sid,sname,gpa)

SQL:

```
SELECT *  
FROM Students  
WHERE gpa > 3.5;
```



RA:

$\sigma_{gpa > 3.5}(\text{Students})$

Another example:

SSN	Name	Salary
1234545	John	200000
5423341	Smith	600000
4352342	Fred	500000

$\sigma_{\text{Salary} > 40000}$ (Employee)



SSN	Name	Salary
5423341	Smith	600000
4352342	Fred	500000

Projection (Π)

- Eliminates columns, then removes duplicates
- Notation: $\Pi_{A1, \dots, An}(R)$
- Example: project social-security number and names:
 - $\Pi_{SSN, Name}(Employee)$
 - Output schema: *Answer (SSN, Name)*

Students(sid,sname,gpa)

SQL:

```
SELECT DISTINCT  
  sname,  
  gpa  
FROM Students;
```



RA:

$\Pi_{sname,gpa}(Students)$

Another example:

SSN	Name	Salary
1234545	John	200000
5423341	John	600000
4352342	John	200000

$\Pi_{\text{Name,Salary}}(\text{Employee})$



Name	Salary
John	200000
John	600000

Note that RA Operators are Compositional!

Students(sid,sname,gpa)

```
SELECT DISTINCT  
  sname,  
  gpa  
FROM Students  
WHERE gpa > 3.5;
```

How do we represent
this query in RA?


$$\Pi_{sname,gpa}(\sigma_{gpa>3.5}(Students))$$

$$\sigma_{gpa>3.5}(\Pi_{sname,gpa}(Students))$$

Are these logically equivalent?

Cross-Product (\times)

- Each tuple in R1 with each tuple in R2
- Notation: $R1 \times R2$
- Example:
 - Employee \times Departments
- Mainly used to express joins

```
Students(sid,sname,gpa)  
People(ssn,pname,address)
```

SQL:

```
SELECT *  
FROM Students, People;
```



RA:

Students \times People

Another example: People

ssn	pname	address
1234545	John	216 Rosse
5423341	Bob	217 Rosse

×

Students

sid	sname	gpa
001	John	3.4
002	Bob	1.3

Students × People



ssn	pname	address	sid	sname	gpa
1234545	John	216 Rosse	001	John	3.4
5423341	Bob	217 Rosse	001	John	3.4
1234545	John	216 Rosse	002	Bob	1.3
5423341	Bob	216 Rosse	002	Bob	1.3

Renaming (ρ – *Rho*)

- Changes the schema, not the instance
- A ‘special’ operator- neither basic nor derived
- Notation: $\rho_{B1,\dots,Bn}(R)$
- **Note: this is shorthand for the proper form (since names, not order matters!):**
 - $\rho_{A1 \rightarrow B1, \dots, An \rightarrow Bn}(R)$

Students(sid,sname,gpa)

SQL:

```
SELECT
  sid AS studId,
  sname AS name,
  gpa AS gradePtAvg
FROM Students;
```



RA:

$\rho_{studId,name,gradePtAvg}(Students)$

We care about this operator *because* we are working in a *named perspective*

Another example:

Students

sid	sname	gpa
001	John	3.4
002	Bob	1.3

$\rho_{studId,name,gradePtAvg}(Students)$



Students

studId	name	gradePtAvg
001	John	3.4
002	Bob	1.3

Natural Join (\bowtie)

- Notation: $R_1 \bowtie R_2$
- Joins R_1 and R_2 on *equality of all shared attributes*
 - If R_1 has attribute set A , and R_2 has attribute set B , and they share attributes $A \cap B = C$, can also be written: $R_1 \bowtie_C R_2$
- Our first example of a *derived* RA operator:
 - Meaning: $R_1 \bowtie R_2 = \Pi_{A \cup B}(\sigma_{C=D}(\rho_{C \rightarrow D}(R_1) \times R_2))$
 - Where:
 - The rename $\rho_{C \rightarrow D}$ renames the shared attributes in one of the relations
 - The selection $\sigma_{C=D}$ checks equality of the shared attributes
 - The projection $\Pi_{A \cup B}$ eliminates the duplicate common attributes

Students(sid,name,gpa)
People(ssn,name,address)

SQL:

```
SELECT DISTINCT
  ssid, S.name, gpa,
  ssn, address
FROM
  Students S,
  People P
WHERE S.name = P.name;
```



RA:

Students \bowtie *People*

Another example:

Students S

sid	S.name	gpa
001	John	3.4
002	Bob	1.3



People P

ssn	P.name	address
1234545	John	216 Rosse
5423341	Bob	217 Rosse

Students ⋈ People



sid	S.name	gpa	ssn	address
001	John	3.4	1234545	216 Rosse
002	Bob	1.3	5423341	217 Rosse

Natural Join

- Given schemas $R(A, B, C, D)$, $S(A, C, E)$, what is the schema of $R \bowtie S$?
- Given $R(A, B, C)$, $S(D, E)$, what is $R \bowtie S$?
- Given $R(A, B)$, $S(A, B)$, what is $R \bowtie S$?

Example: Converting SFW Query -> RA

Students(sid,sname,gpa)
People(ssn,sname,address)

```
SELECT DISTINCT  
  gpa,  
  address  
FROM Students S,  
     People P  
WHERE gpa > 3.5 AND  
       S.sname = P.sname;
```


$$\Pi_{gpa,address}(\sigma_{gpa>3.5}(S \bowtie P))$$

How do we represent
this query in RA?

How would you find students who have taken all classes?

Students(studentname, coursecode)
Courses(coursecode)

Division

- Notation: $r \div s$
- It has nothing to do with arithmetic division.
- Let r and s be relations on schemas R and S respectively where
 - $R = (A_1, \dots, A_m, B_1, \dots, B_n)$
 - $S = (B_1, \dots, B_n)$

The result of $r \div s$ is a relation on schema

$$R - S = (A_1, \dots, A_m)$$

$$r \div s = \{ t \mid t \in \Pi_{R-S}(r) \wedge \forall u \in s (tu \in r) \}$$

Where tu means the concatenation of tuples t and u to produce a single tuple

Division Operation - Example

□ Relations r, s

A	B
α	1
α	2
α	3
β	1
γ	1
δ	1
δ	3
δ	4
ϵ	6
ϵ	1
β	2

r

B
1
2

s

□ $r \div s$

A
α
β

Division Operation - Example

□ Relations r, s

<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>E</i>
α	a	α	a	1
α	a	γ	a	1
α	a	γ	b	1
β	a	γ	a	1
β	a	γ	b	3
γ	a	γ	a	1
γ	a	γ	b	1
γ	a	β	b	1

r

<i>D</i>	<i>E</i>
a	1
b	1

s

□ $r \div s$

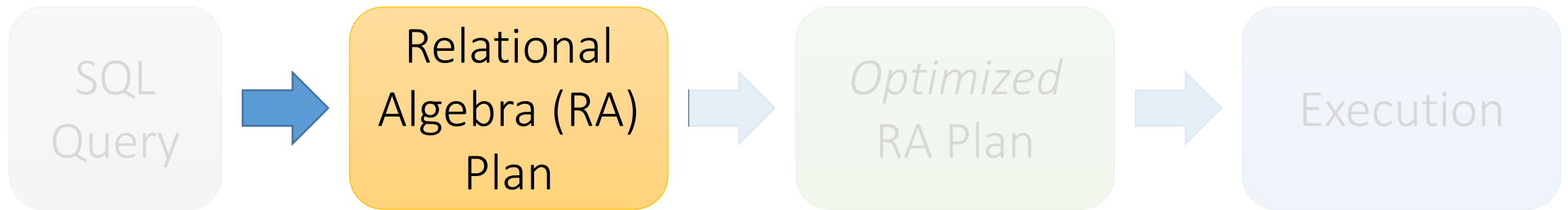
<i>A</i>	<i>B</i>	<i>C</i>
α	a	γ
γ	a	γ

Logical Equivalence of RA Plans

- Given relations $R(A,B)$:
 - Here, projection & selection commute:
 - $\sigma_{A=5}(\Pi_A(R)) = \Pi_A(\sigma_{A=5}(R))$
 - What about here?
 - $\sigma_{A=5}(\Pi_B(R)) = \Pi_B(\sigma_{A=5}(R))$???

RDBMS Architecture

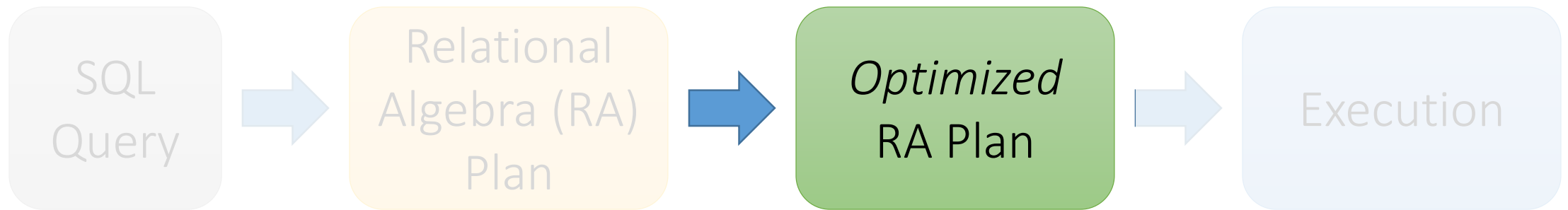
How does a SQL engine work ?



We saw how we can transform declarative SQL queries into precise, compositional RA plans

RDBMS Architecture

How does a SQL engine work ?



We'll look at how to then optimize these plans later!

RDBMS Architecture

How is the RA “plan” executed?



We will see later how to execute all the basic operators!

2. Advanced Relational Algebra

What you will learn about in this section

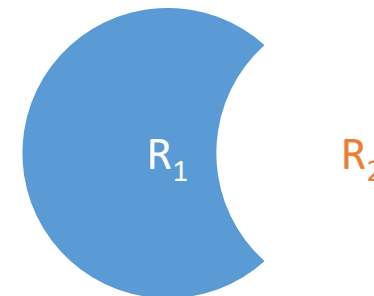
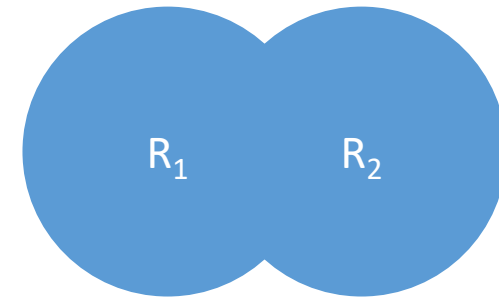
1. Set Operations in RA
2. Fancier RA
3. Extensions & Limitations

Relational Algebra (RA)

- Five **basic** operators:
 - Selection: σ
 - Projection: Π
 - Cartesian Product: \times
 - Union: \cup
 - Difference: $-$
- Derived or auxiliary operators:
 - Intersection, complement
 - Joins (natural, equi-join, theta join, semi-join)
 - Renaming: ρ
 - Division

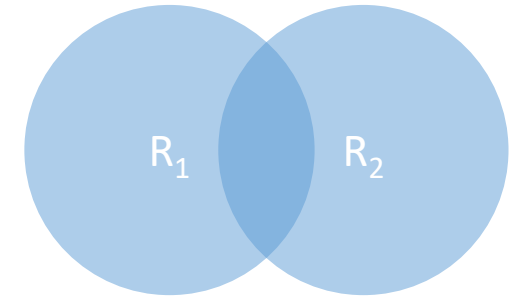
Union (\cup) and Difference ($-$)

- $R_1 \cup R_2$
- Example:
 - $\text{ActiveEmployees} \cup \text{RetiredEmployees}$
- $R_1 - R_2$
- Example:
 - $\text{AllEmployees} - \text{RetiredEmployees}$



What about Intersection (\cap) ?

- It is a derived operator
- $R1 \cap R2 = R1 - (R1 - R2)$
- Also expressed as a join!
- Example
 - $\text{UnionizedEmployees} \cap \text{RetiredEmployees}$



Theta Join (\bowtie_{θ})

- A join that involves a predicate
- $R1 \bowtie_{\theta} R2 = \sigma_{\theta} (R1 \times R2)$
- Here θ can be any condition

Note that natural join is a theta join + a projection.

```
Students(sid,sname,gpa)  
People(ssn,sname,address)
```

SQL:

```
SELECT *  
FROM  
  Students,People  
WHERE  $\theta$ ;
```



RA:

$Students \bowtie_{\theta} People$

Equi-join ($\bowtie_{A=B}$)

- A theta join where θ is an equality
- $R1 \bowtie_{A=B} R2 = \sigma_{A=B} (R1 \times R2)$
- Example:
 - $\text{Employee} \bowtie_{\text{SSN}=\text{SSN}} \text{Dependents}$

Most common join
in practice!

Students(sid,sname,gpa)
People(ssn,pname,address)

SQL:

```
SELECT *  
FROM  
  Students S,  
  People P  
WHERE sname = pname;
```



RA:

$S \bowtie_{\text{sname}=\text{pname}} P$

Semijoin (\bowtie)

- $R \bowtie S = \Pi_{A_1, \dots, A_n} (R \bowtie S)$
- Where A_1, \dots, A_n are the attributes in R
- Example:
 - Employee \bowtie Dependents

Students(sid,sname,gpa)
People(ssn,pname,address)

SQL:

```
SELECT DISTINCT  
  sid,sname,gpa  
FROM  
  Students,People  
WHERE  
  sname = pname;
```

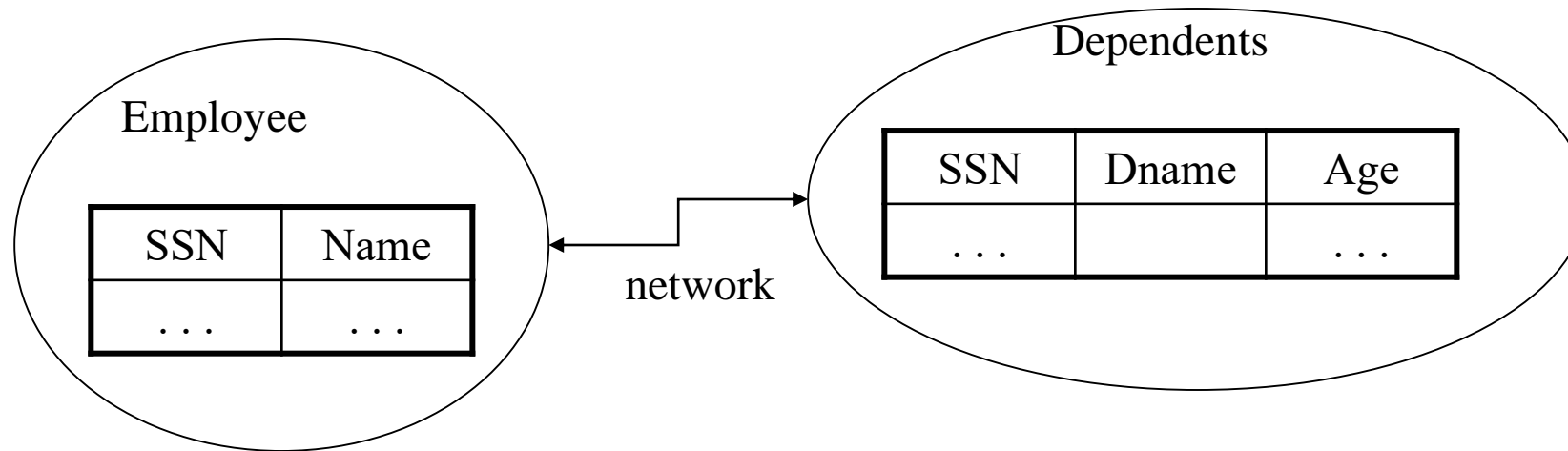


RA:

Students \bowtie People

Semijoins in Distributed Databases

- Semijoins are often used to compute natural joins in distributed databases



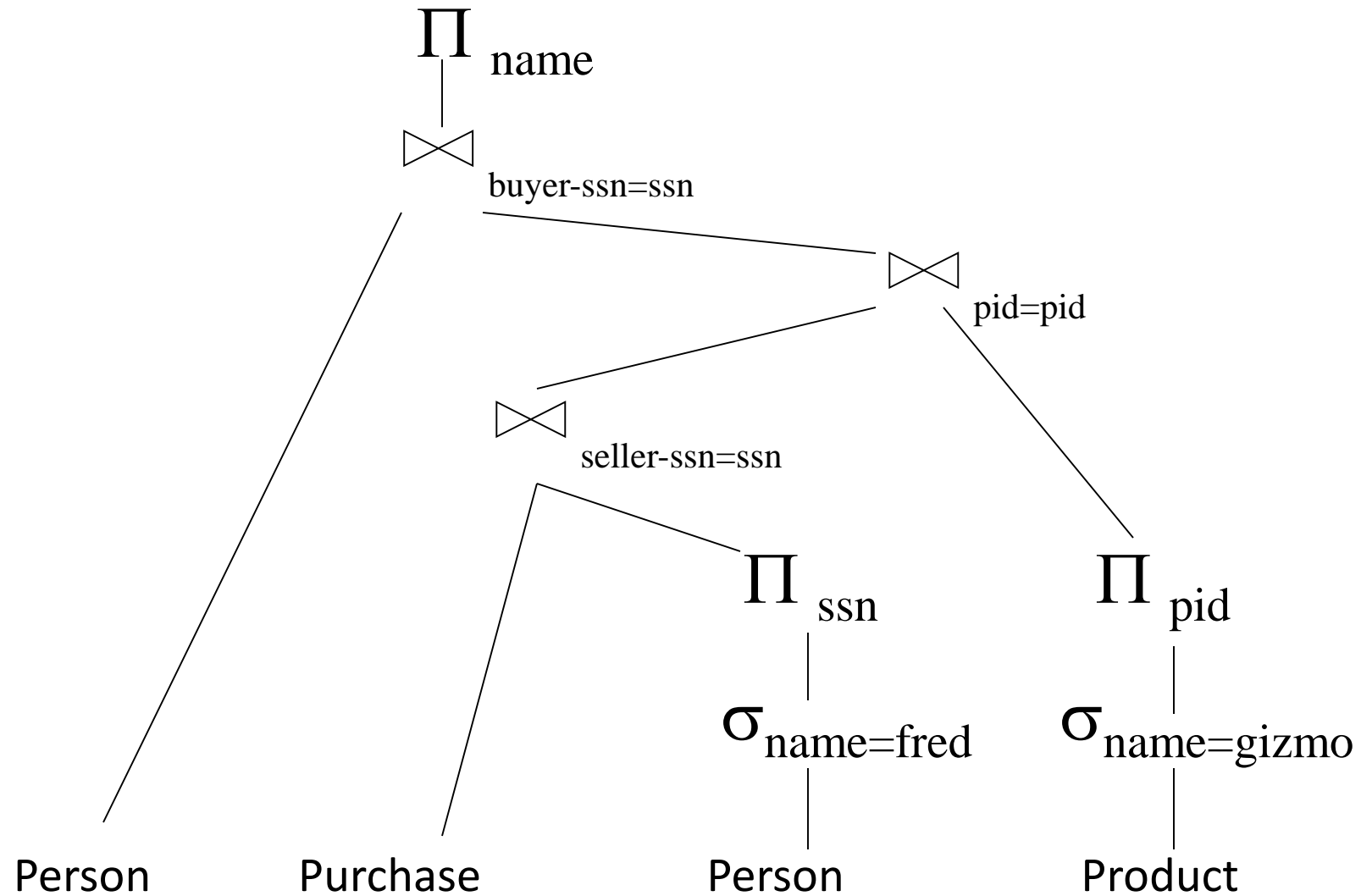
Send less data to
reduce network
bandwidth!

$$\text{Employee} \bowtie_{\text{ssn}=\text{ssn}} (\sigma_{\text{age}>71} (\text{Dependents}))$$

$$R = \text{Employee} \bowtie T$$

$T = \Pi_{\text{SSN}} \sigma_{\text{age}>71} (\text{Dependents})$
 $\text{Answer} = R \bowtie \text{Dependents}$

RA Expressions Can Get Complex!



Operations on Multisets

All RA operations need to be defined carefully on bags

- $\sigma_C(R)$: preserve the number of occurrences
- $\Pi_A(R)$: no duplicate elimination
- Cross-product, join: no duplicate elimination

This is important- relational engines work on multisets, not sets!

RA has Limitations !

- Cannot compute “transitive closure”

Name1	Name2	Relationship
Fred	Mary	Father
Mary	Joe	Cousin
Mary	Bill	Spouse
Nancy	Lou	Sister

- Find all direct and indirect relatives of Fred
- Cannot express in RA !!!
 - Need to write C program, use a graph engine, or modern SQL...

Example: Banking Database

branch (branch_name, branch_city, assets)

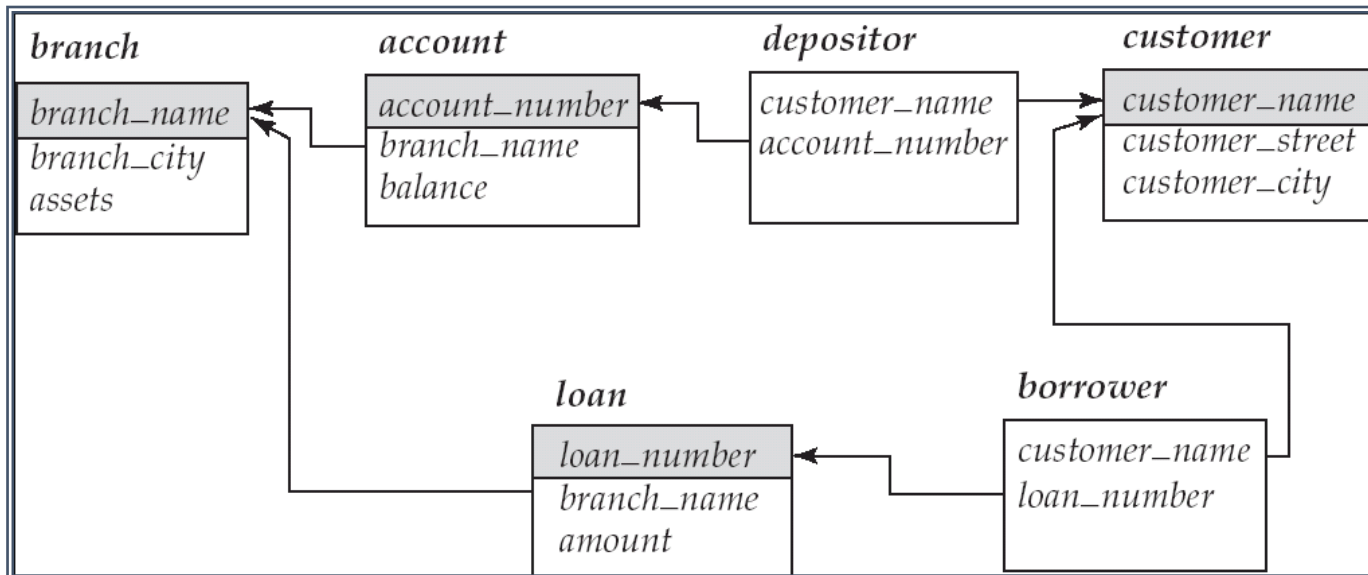
customer (customer_name, customer_street, customer_city)

account (account_number, branch_name, balance)

loan (loan_number, branch_name, amount)

depositor (customer_name, account_number)

borrower (customer_name, loan_number)



branch (branch_name, branch_city, assets)

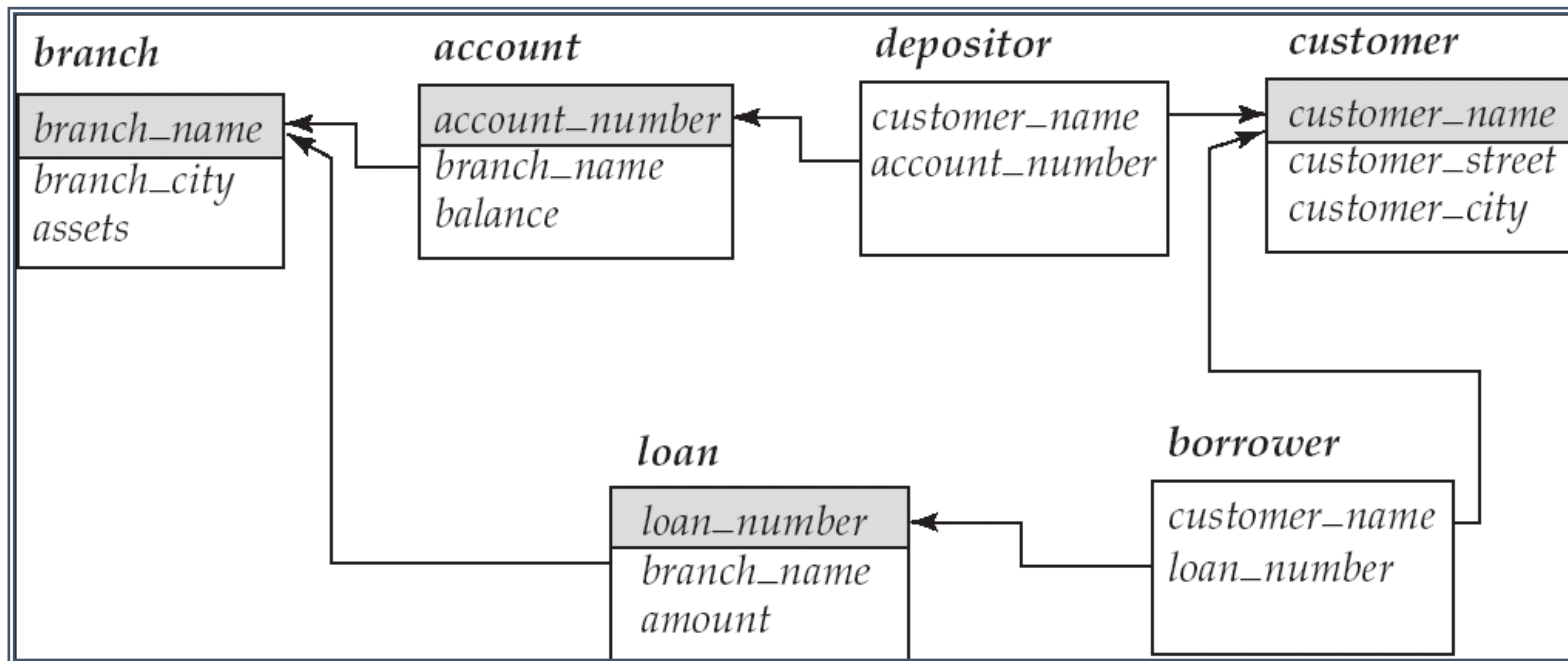
customer (customer_name, customer_street, customer_city)

account (account_number, branch_name, balance)

loan (loan_number, branch_name, amount)

depositor (customer_name, account_number)

borrower (customer_name, loan_number)



Example Queries

- Find all loans of over \$1200

$$\sigma_{amount > 1200} (loan)$$

- Find the loan number for each loan of an amount greater than \$1200

$$\Pi_{loan_number} (\sigma_{amount > 1200} (loan))$$

Example Queries

- Find the names of all customers who have a loan, an account, or both, from the bank

$$\Pi_{customer_name}(borrower) \cup \Pi_{customer_name}(depositor)$$

- Find the names of all customers who have a loan and an account at bank.

$$\Pi_{customer_name}(borrower) \cap \Pi_{customer_name}(depositor)$$

Example Queries

- Find the names of all customers who have a loan at the Perryridge branch.

$$\Pi_{customer_name} (\sigma_{branch_name = "Perryridge"} \\ (\sigma_{borrower.loan_number = loan.loan_number} (borrower \times loan)))$$

- Find the names of all customers who have a loan at the Perryridge branch but do not have an account at any branch of the bank.

$$\Pi_{customer_name} (\sigma_{branch_name = "Perryridge"} \\ (\sigma_{borrower.loan_number = loan.loan_number} (borrower \times loan))) - \Pi_{customer_name} (depositor)$$

Example Queries

- Find the names of all customers who have a loan at the Perryridge branch.

- **Query 1**

$$\Pi_{\text{customer_name}} (\sigma_{\text{branch_name} = \text{"Perryridge"}} (\sigma_{\text{borrower.loan_number} = \text{loan.loan_number}} (\text{borrower} \times \text{loan})))$$

- **Query 2**

$$\Pi_{\text{customer_name}} (\sigma_{\text{loan.loan_number} = \text{borrower.loan_number}} (\sigma_{\text{branch_name} = \text{"Perryridge"}} (\text{loan})) \times \text{borrower}))$$

Example Queries

- Find all customers who have an account at all branches located in Brooklyn city.

$$\Pi_{\text{customer name, branch name}}(\text{ depositor} \bowtie \text{ account}) \\ \div \Pi_{\text{branch name}}(\sigma_{\text{branch city} = \text{"Brooklyn"}}(\text{ branch}))$$

Acknowledgements

The course material used for this lecture is mostly taken and/or adopted from the course materials of the *CS145 Introduction to Databases* lecture given by *Christopher Ré* at *Stanford University* (<http://web.stanford.edu/class/cs145/>).