L21: Relational Algebra

CS3200 Database design (fa18 s2)

https://northeastern-datalab.github.io/cs3200/

Version 11/26/2018

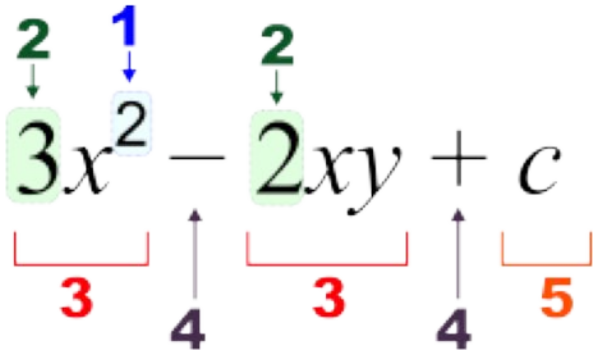
Our next focus

- The Relational Model
- Relational Algebra
- Relational Algebra Pt. II [Optional: may skip]

1. The Relational Model & Relational Algebra

Algebra

- Algebra is the study of mathematical symbols and the rules for manipulating these symbols;
- e.g., Linear Algebra
- e.g., Relational Algebra
- e.g., Boolean Algebra
- e.g., Abstract algebra (groups, rings, fields, ...)
- e.g., Elementary algebra



What you will learn about in this section

- The Relational Model
- Relational Algebra: Basic Operators
- Execution

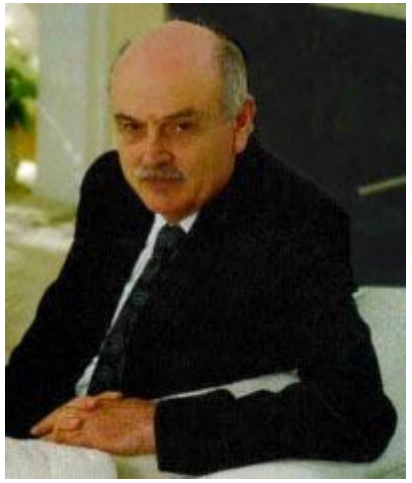
Motivation

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

Database maps internally into this *procedural language.*

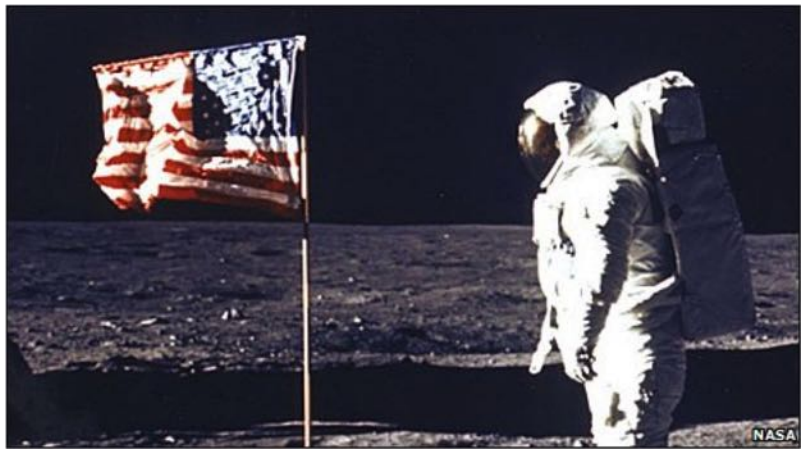
A Little History

- 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



Won Turing award 1981

- IBM didn't want to use relational model (take money from IMS)
  - Apparently used in the moon landing…



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 (or bags).

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 $R(f_1:Dom_1,\ldots,f_m:Dom_m)$ be a *relational schema* then, an *instance* of R is a subset of $Dom_1 \times Dom_2 \times \ldots \times Dom_n$

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:
$R : Dom_1 \times \ldots \times Dom_t \rightarrow \{0,1\}$

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*

A Course Management System (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"

Relation Instances

Sid	Name	Gpa
101	Bob	3.2
123	Mary	3.8

Students

sid	cid	Grade
123	564	A

Enrolled

cid	cname	credits
564	564-2	4
308	417	2

Courses

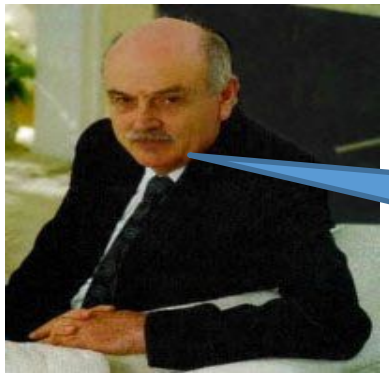
2nd Part of the Model: Querying

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

"Find names of all students with 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***

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!

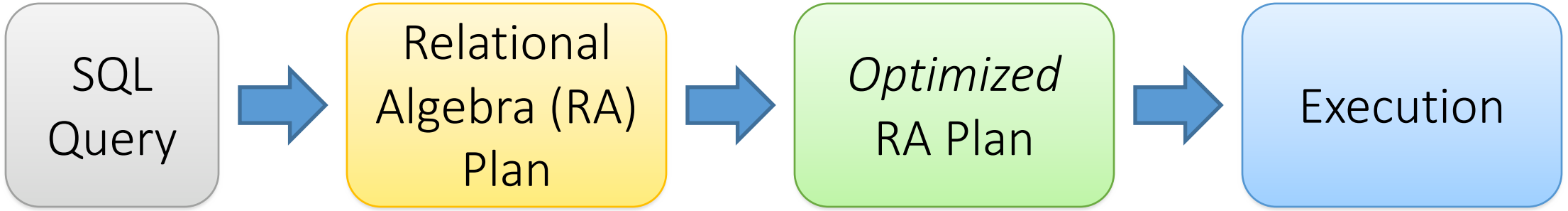
Virtues of the model

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

2. Relational Algebra

RDBMS Architecture

- How does a SQL engine work ?



Declarative query (from user)

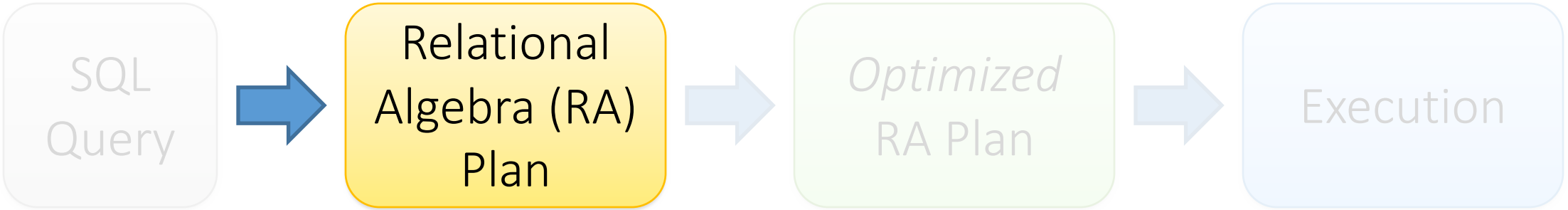
Translate to relational algebra expression

Find logically equivalent- but more efficient- RA expression

Execute each operator of the optimized plan!

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:
  1. Selection: σ
  2. Projection: Π
  3. Cartesian Product: $\times$
  4. Union: $\cup$
  5. Difference: -
- Derived or auxiliary operators:
  - Intersection, complement
  - Joins (natural, equi-join, theta join, semi-join)
  - Renaming: ρ
  - Division

We'll look at these first!

And also at one example of a derived operator (natural join) and a special operator (renaming)

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…

1. Selection (σ)

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

```
Students(sid,sname,gpa)
```

SQL:

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

RA:

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

Another example:

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

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

SSN	Name	Salary
5423341	Smith	600000
4352342	Fred	500000

2. Projection (Π)

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

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

Π_{SSN} (Employee)

SSN
1234545
5423341
4352342

Another example:

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

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

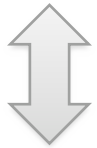
Name	Salary
John	200000
John	600000

Note that RA Operators are Compositional!

```
Students(sid,sname,gpa)
```

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

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



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

How do we represent this query in RA?

Are these logically equivalent?

3. Cross-Product (×)

- Each tuple in R1 with each tuple in R2
- Notation: R1 × R2
- Example:
  - Employee × Dependents
- Rare in practice; 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

$Students \times People$

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

4. Renaming (ρ)

- Changes the schema, not the instance
- A 'special' operator- neither basic nor derived
- Notation: $\rho_{B1,\ldots,Bn}(R)$

- Note: this is <u>shorthand</u> for the proper form (since <u>names, not order</u> matters!):
  - $\rho_{A1 \rightarrow B1,\ldots,An \rightarrow Bn}(R)$

```
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

5. Natural Join (⋈)

- 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: $R_1 \bowtie R_2 = \Pi_{A \cup B}(\sigma_{C=D}(\rho_{C \to D}(R_1) \times R_2))$
  - Where:
    - The rename $\rho_{C \to D}$ 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

$\bowtie$

People P

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

$$Students \bowtie People$$

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

Natural Join practice

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

- Given R(A, B, C), S(D, E), what is R ⋈ S ?

- Given R(A, B), S(A, B), what is R ⋈ S ?

Example: Converting SFW Query -> RA

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

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

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

How do we represent this query in RA?