Please signup to this canvas

https://canvas.instructure.com/enroll/GH6XJ7

Or using the canvas join code: **GH6XJ7**

Recap: DBMS & Data Model

- Common operations with Databases: design/define structures, create and populate DB with data, manipulate data and create report for a query

- A DBMS is a software package designed to store and manage databases.
  It provides generic functionality that otherwise would have to implemented over and over again.

- Data Model: a framework/set of concepts that can be used to describe data objects, relationships, semantics and/or constraints, while hiding details of data storage

ICCS240 Database Management

Relational Model & Relational Algebra

Many slides in this lecture are adapted from slides provided by Theodoros Rekatsinas, UW-Madison

The early days of data management …

- The very first DBMS evolved from file systems in 1960s
- Early applications: Banking system, Corporate record keeping, …
- Used several models *specific for their tasks*.
  Hierarchical model and graph-based network model are popular.
- None of them not support high-level query languages.
- Most of them requires programmers to visualize data and their behind-the-scene structures.

Emergence of Database Management System (DBMS)

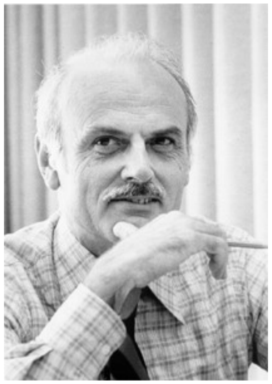
When database application was mostly ad-hoc

Need *generalization*, e.g., sorting, file maintenance, report generation

In 1970, ***Ted Codd*** proposed a database systems that should present the user with a view of data organized as tables called *relations*

AND offered an efficient "high-level" query language

Relational DB became the norm since 1990…



Edgar F. Codd

The Abstraction of Relational Model

- Store data(base) in simple data structures

Declarative query languages

- Access data through high-level language

Separate physical and logical

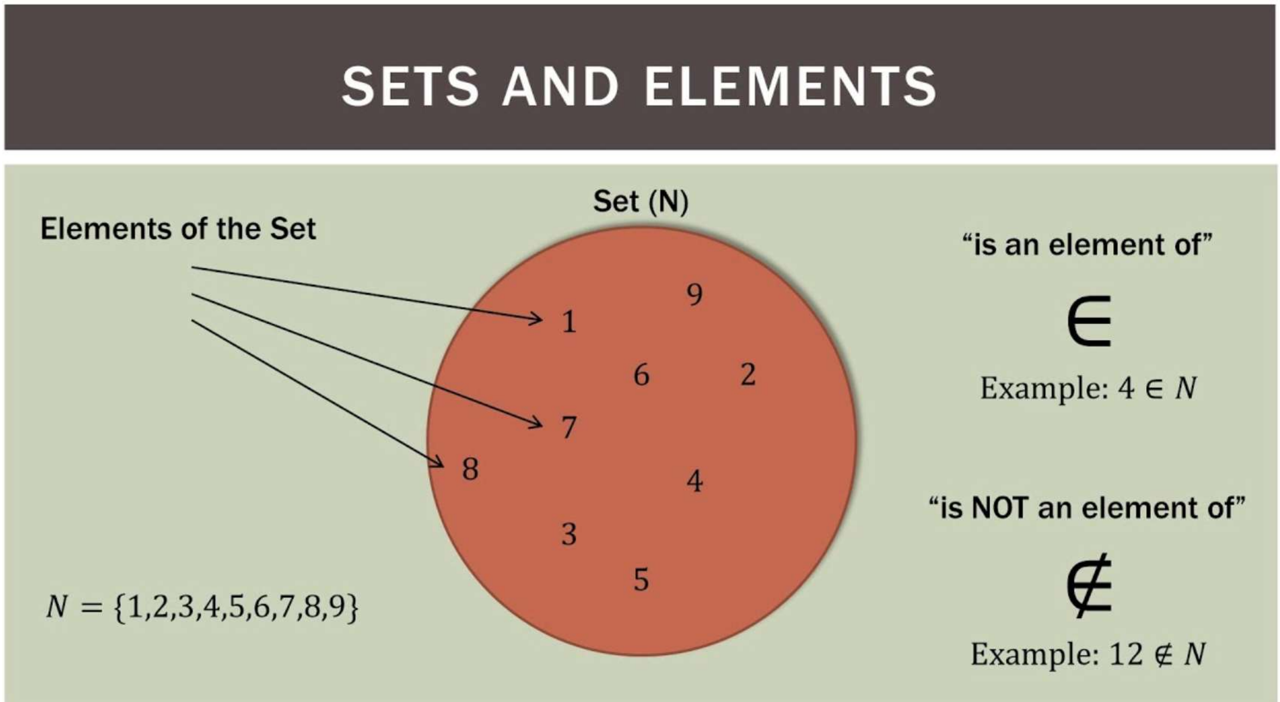
- Physical storage left up to implementation

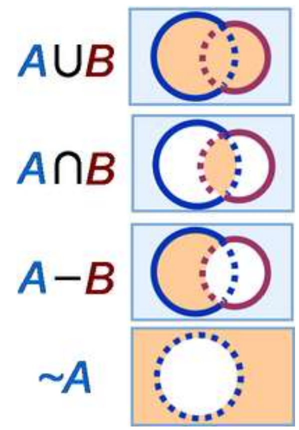
Formal semantics

- Formalized mathematically

All these facilitate "**query optimization**"
(which is the key to commercial success)

Review of Mathematical Concepts





$A \subseteq B$ means every element of *A* is in *B*.

Notions & Notations (1)

- Sets: $S,\ T,\ S_1, \dots,\ S_n,\ T_1, \dots, T_n,\ \{\},\ \emptyset$
- **Cardinality** of a set S denoted as $|S|$

- **Cartesian Product** of sets (also **cross** product):
  $S \times T$ = set of all pairs (x, y) where $x \in S$ and $y \in T$
  $S_1 \times S_2 \times \cdots \times S_n = \{(x_1, x_2, \dots, x_n) \mid \forall x_i \in S_i\}$

Example: Cartesian Product

- $R = \{1,2,3\}$
- $S = \{3,4\}$

$R \times S = \{(1,3), (1,4), (2,3), (2,4), (3,3), (3,4)\}$

$$R \times S \times R$$
$$= \{(1,3,1), (1,3,2), (1,3,3), (1,4,1), (1,4,2), (1,4,3), (2,3,1), \ldots ?\}$$

Notions & Notations (2)

- **Relation** R over S, T is a subset of $S \times T$, written $R \subseteq S \times T$
  We write $(x, y) \in R$ or, equivalently xRy

- Relation R over $S_1, \ldots, S_n$: subset $R \subseteq S_1 \times \cdots \times S_n$
  The number n is the arity of R

- **Function** f from S to T, denoted as $f : S \to T$
  Associates to every element $x \in S$ exactly one element of T, denoted as $f(x)$
  A relation $R \subseteq S \times T$ is a function if $\forall x_1, x_2 \in S$, if $x_1 = x_2$ then $f(x_1) = f(x_2)$
- A **domain** of function f is the set of input of f, denoted as $\mathrm{dom}(f)$

Notions & Notations (3)

- **Partial function** f from S to T, written $f: S \hookrightarrow T$ or $f: S \nrightarrow T$ or $f: S\sim \rightarrow T$

  is a function $f: S' \rightarrow T$, for some subset S' of S

  (We still using $f(x)$ for element in T associated with $x \in S$.)

- If $S' = S$, then f is called a **total function**

- A relation R over $S_1, \dots, S_n$ is **total** on S_i

  if for every $x_i \in S_i$, there are $x_j \in S_j, j \neq i$, such that $(x_i, \dots, x_n) \in R$

  In other words, every element of S_i occurs in some tuple of R

Some combinatory: *How Many?*

Given sets S, T where $|S| = n$ and $|T| = m$,

and $|S_i| = n_i$, for each $S_1, \dots, S_n$

- How many elements can a relation R over $S_1, \dots, S_n$ have? at least? At most?
- How many relations over S, T are there? [HW1?]
- How many functions from S to T are there? [HW1?]
- How many partial functions from S to T are there? [HW1?]
- How many relations are there over S, T that are total on S [HW1?]

The Relational Model

Normal people call it a **table**

This is called a relation in DB.

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

Motivation

The relational model is

precise,

implementable,

and we can **operate** (query/update/…) on it.

The Relational Model : <u>Schemata</u>

Students(sid: *string*, name: *string*, gpa: *float*)

Relation name

String, float, int, etc. are the **<u>domains</u>** of the attributes

Attributes

The Relational Model: <u>Data</u>

Student

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

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

The number of attributes is the **arity** of the relation

The Relational Model: <u>Data</u>

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 **<u>cardinality</u>** of the relation

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

The Relational Model: <u>Data</u>

Student

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

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

A **<u>relational instance</u>** is a ***set*** of tuples all conforming to the same *schema*

Relational Model Formalities

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

Let $R(A_1\!: \mathrm{Dom}_1, \dots, A_m\!: \mathrm{Dom}_m)$ be a *relational schema*, then

an *instance* of R is a subset of $\mathrm{Dom}_1 \times\ \mathrm{Dom}_2 \times \cdots \times\ \mathrm{Dom}_n$

- The relation **name** is R
- A nonempty set of **attributes** $A_1, \dots, A_n$
- A **type** or **domain**, $\mathrm{Dom}_i = \mathrm{dom}(A_i)$ = all permissible values of attribute A_i

A special value **NULL** (encoding “unknown”) is a member of every domain

Types and Domains

Type: Class of atomic values

- Integers, reals, strings
- Integers between 1 and 100
- String of (up to) 10 characters

Domain: Set of atomic values, that have specific meaning in application, e.g., Name, EmployeeAge

- Domains have a type, e.g., EmployeeAge = Integer[15,80]
- Domains may have default values

Relational Model Formalities

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

Let $R(A_1\text{: Dom}_1, \ldots, A_m\text{: Dom}_m)$ be a *relational schema*, then

an *instance* of R is a subset of $\text{Dom}_1 \times \text{Dom}_2 \times \ldots \times \text{Dom}_n$

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

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

A Concrete Example

branch	acct_no	balance
Downtown	A-101	500
Brighton	A-201	900
Brighton	A-217	500

Tables → Relations
Columns → Attributes
Rows → Tuples

Schema → e.g., **Account**(branch: Branches, acct_no: GenAccounts, balance: Balances)

Domain → set of all possible values for an attribute.

e.g., Branches = dom(branch) = { Downtown, Brighton, … }
GenAccounts = dom(acct_no) = { A-101, A-201, A-217, … }
Balances = $\mathbb{R}$ = real numbers

Account $\subseteq$ Branches $\times$ GenAccounts $\times$ Balances

```
{ (Downtown, A-101, 500),
  (Brighton, A-201, 900),
  (Brighton, A-217, 500) }
```

In the formal model,
Relations are unordered

- Order of tuples is irrelevant (tuples may be stored in an arbitrary order)
- Example: *instructor* relation with unordered tuples

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

Keys: {Super, Candidate, Primary} Keys

Kind of keys

- **Super keys** – set of attributes of table for which every row has distinct set of values
- **Candidate keys** – “minimal” superkeys
- **Primary key** – the candidate key of choice

branch	acct_no	balance
Downtown	A-101	500
Brighton	A-201	900
Brighton	A-217	500

Super keys?
Candidate keys?

Practical Use: Primary Key

A relation's **primary key** *uniquely* identifies a single tuple.

branch	acct_no	balance
Downtown	A-101	500
Brighton	A-201	900
Brighton	A-217	500

`Account(branch, acct_no, balance)`

Underline the attribute to indicate it's the primary key of that schema.

Some DBMSs automatically create an internal primary key if you don't define one.
For example, auto-generation of unique integer primary keys:

- `SEQUENCE (SQL:2003)`
- `AUTO_INCREMENT (MySQL)`

Relational Databases

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*

Example: Building a course management system

- Consider building a course management system (**CMS**):
  - Students
  - Courses
  - Professors

  Entities

  - Who takes what
  - Who teaches what

  Relationships

Example: Modeling a course management system

- Concept: the ***relation*** – essentially a table
- Every relation in a relational data model has a schema describing types
  - *Relational 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

sid	cid	Grade
123	564	A

Enrolled

cid	cname	credits
564	564-2	4
308	417	2

Courses

Rarely do we have just one table!

- A **foreign key** specifies that an attribute from one relation has to map to a tuple in another relation.

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

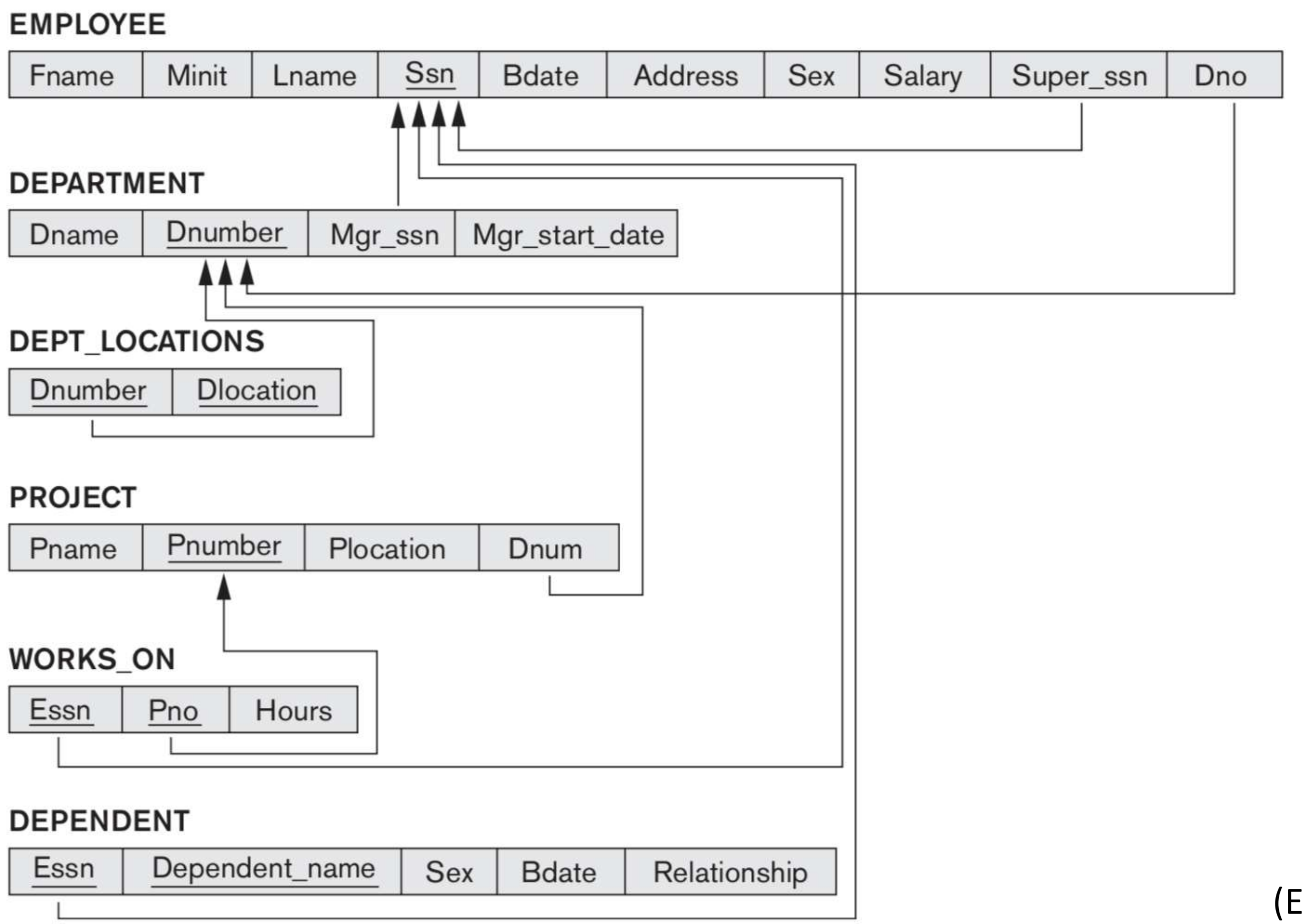
In a diagram

`Student(`**`sid`**`, name, gpa)`

`Enrolled(`**`sid`**`, `**`cid`**`, grade)`

`Courses(`**`cid`**`,cname, credits)`

In a real database ☺



(Elmasri & Navathe, 7th Ed.)

Note: Choosing your keys carefully

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

```
Students(sid, name, gpa)
Courses(cid, cname, credits)
Enrolled(sid, cid, grade)
```

Note: Choosing your keys carefully

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

`Students(sid, name, gpa)`
`Courses(cid, cname, credits)`
`Enrolled(sid, cid, grade)`

?

Example: creating a table

```
create table instructor (
  ID char(5),
  name varchar(20) not null,
  dept_name varchar(20),
  salary numeric(8,2),
  primary key (ID)
);
```

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

Instructor(**ID**,name,dept_name,salary)

Integrity Constraints in creating a table

Instructor(**ID**,name,dept_name,salary)

```
create table instructor (
  ID char(5),
  name varchar(20) not null,
  dept_name varchar(20),
  salary numeric(8,2),
  primary key (ID)
);
```

Integrity constraint

Primary key declaration ensures that attribute cannot be NULL.

Virtues of the Relational Model

- Physical independence, Declarative
- Simple, elegant clean: Everything is a relation

SQL: Querying is *declarative*

```
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…

RDBMS Architecture

How does a SQL engine work ?

SQL Query → Relational Algebra (RA) Plan → *Optimized* RA Plan → Execution

SQL Query	Relational Algebra (RA) Plan	*Optimized* RA Plan	Execution
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 (RA)
Plan

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

Intro to Relational Algebra

Intro to Relational Algebra

- Fundamental operations to retrieve and manipulate tuples in a relation.

  → Based on set algebra.

- Each operator takes one or more relations as its inputs and outputs a new relation.

  → Can chain operators to form more complex expressions.

TODAY: 2 "natural" operators

σ SELECT

π PROJECTION

Keep in mind that

- RDBMS **use multiset,**

  but RA formalism consider **set**s.

- We consider *named perspective*, where every attribute must have a **unique name**.

Selection ($\boldsymbol{\sigma}$)

- Returns all tuples which satisfy a condition
- Notation: $\sigma_c(R)$
- Examples
  - $\sigma_{\text{Salary > 40000}}$ (Employee)
  - $\sigma_{\text{name = “Smith”}}$ (Employee)
- The condition c can be =, <, $\leq$, >, $\geq$, <> or logical operators such as $\vee$ or $\wedge$

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	20000
5423341	Smith	600000
4352342	Fred	500000

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

SSN	Name	Salary
5423341	Smith	600000
4352342	Fred	500000

RAs are compositional

SSN	Name	Salary
1234545	John	20000
5423341	Smith	600000
4352342	Fred	500000

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

SSN	Name	Salary
5423341	Smith	600000
4352342	Fred	500000

$\sigma_{\text{Name = 'Fred'}}$ (T)

SSN	Name	Salary
4352342	Fred	500000

Some algebraic properties (for queries optimization)

- Selection is **idempotent** (multiple applications of same selection have no effect beyond the first application)

$$\sigma_C\left(\sigma_C(R)\right) = \sigma_C(R)$$

- Selection is **commutative**

$$\sigma_{C_1}\left(\sigma_{C_2}(R)\right) = \sigma_{C_2}\left(\sigma_{C_1}(R)\right)$$

Some algebraic properties (for queries optimization)

- We can break a **conjunction** of conditions in a selection into a sequence of those individual conditions.

$$\sigma_{C_1 \wedge C_2}(R) = \sigma_{C_1}\left(\sigma_{C_2}(R)\right) = \sigma_{C_2}\left(\sigma_{C_1}(R)\right)$$

- We can break a **disjunction** of conditions in a selection into a union of selection of those individual conditions.

$$\sigma_{C_1 \vee C_2}(R) = \sigma_{C_1}(R) \cup \sigma_{C_2}(R)$$

Projection ($\boldsymbol{\Pi}$)

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

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

Name	Salary
John	200000
John	600000

`A(a_id, b_id)`

a_id	b_id
a1	101
a2	102
a2	103
a3	104

- Projection can also be used to modify attributes’ values
- Projection can also be used to rearrange attributes

$$\pi_{b\_id-100, a\_id}(A)$$

b_id-100	a_id
1	a1
2	a2
3	a2
4	a3

```
SELECT b_id - 100, a_id from A
```

Some algebraic properties (for queries optimization)

- Projection is **idempotent** (a sequence of valid projections is equivalent to the projection of outermost projection)

$$\pi_{A_1,\dots,A_n}\left(\pi_{B_1,\dots,B_n}(R)\right) = \pi_{A_1,\dots,A_n}(R)$$

where $\{A_1, \dots, A_n\} \subseteq \{B_1, \dots, B_n\}$

Selection & Projection

`A(a_id, b_id)`

a_id	b_id
a1	101
a2	102
a2	103
a3	104

a_id	b_id
a2	102
a2	103

$B = \sigma_{\text{a\_id}='a2'}(A)$

b_id	a_id
2	a2
3	a2

$\pi_{b\_id-100, a\_id}(B)$

But which representation then?

Students(sid,sname,gpa)

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

How do we represent this query in RA?

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

or?

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

Are these (always) logically equivalent?