# Backend Development– W2S1 Relational database

Cyrille Jegourel – Singapore University of Technology and Design

# References

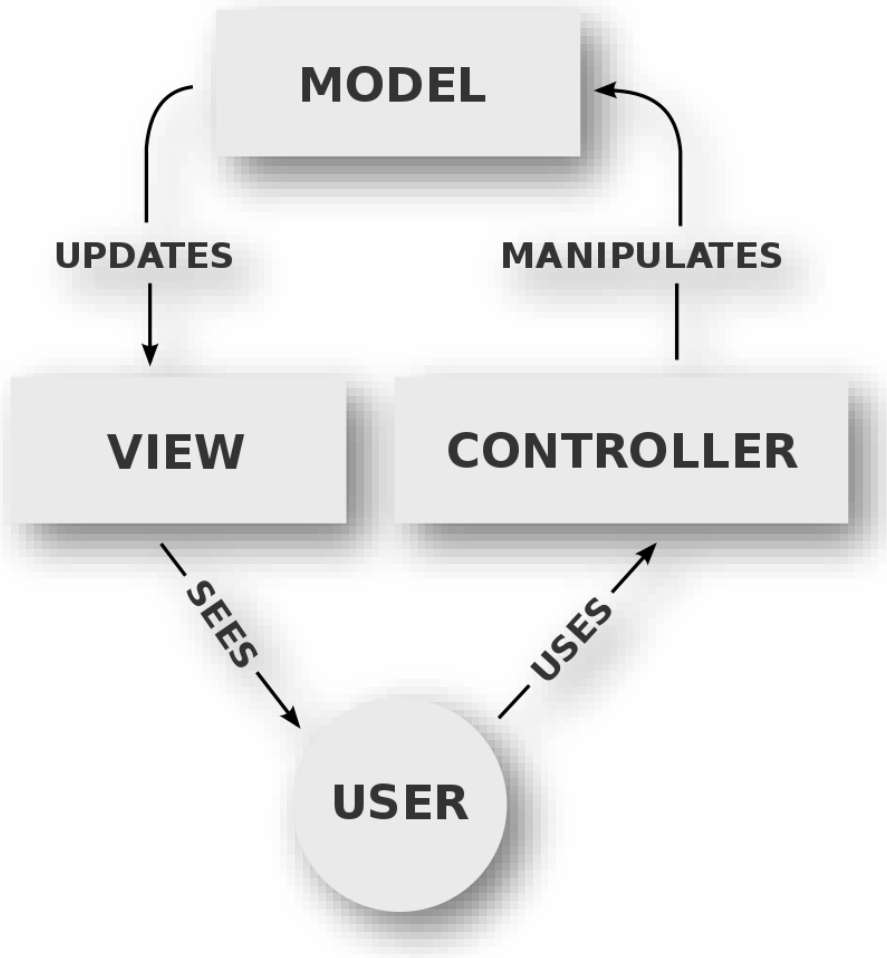
- **Mainly former slides from Ass. Pr. Anh Dinh (SUTD) and myself.**
- “JavaScript: A beginner’s guide” by John Pollock, 2020
- A beginner tutorial for Postgresql:
  - https://www.youtube.com/watch?v=qw--VYLpxG4&t=172s&ab_channel=freeCodeCamp.org

# Outline (Day 2, Sessions 1 and 2)

- Database: intro
- Relational database

# MVC pattern

- In software engineering, a **software design pattern** is a general template for how to solve a problem that can be used in many different situations.
- **Model–view–controller** (usually known as MVC) is a software design pattern commonly used for developing UI that divide the related program logic into three interconnected elements.
- The model is responsible for managing the data of the application. It receives user input from the controller.
- The view renders presentation of the model in a particular format.
- The controller responds to the user input and performs interactions on the data model objects. The controller receives the input, optionally validates it and then passes the input to the model.
- The MVC became very popular for designing web applications.
- Source:
  - https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller



# Database

What is a database?

Database is an **organized collection** of **data**

What is a database management system (DBMS)?

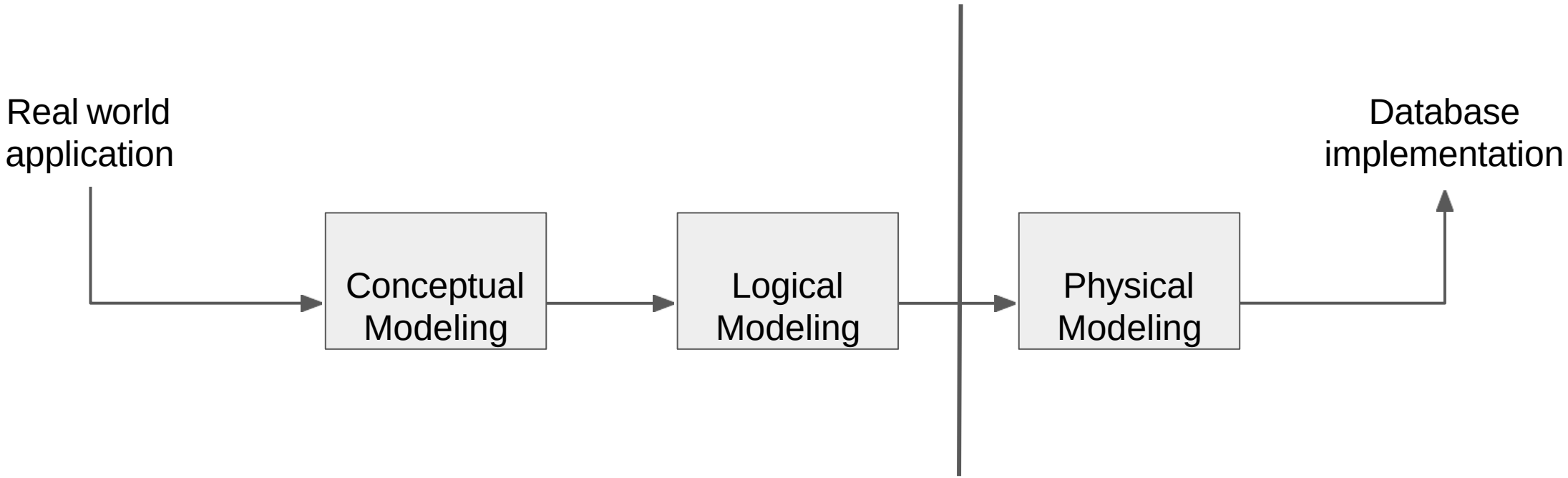
- System that **manages** the organized collection of data
  - Create, delete, store, query, analyze, etc.

# Database & DBMS

# Warning!

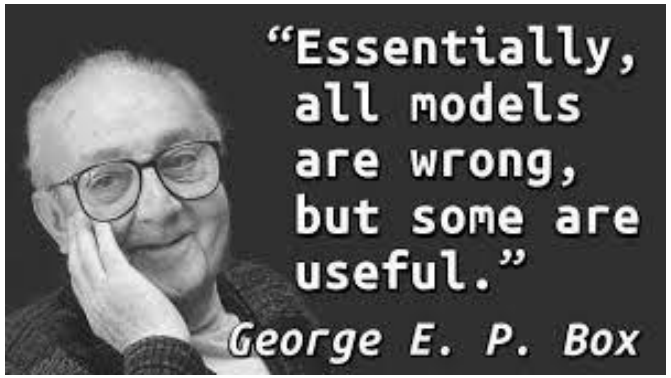
- I use the 2 terms interchangeably
- That’s me being sloppy
- If not clear from context what I meant, ASK.

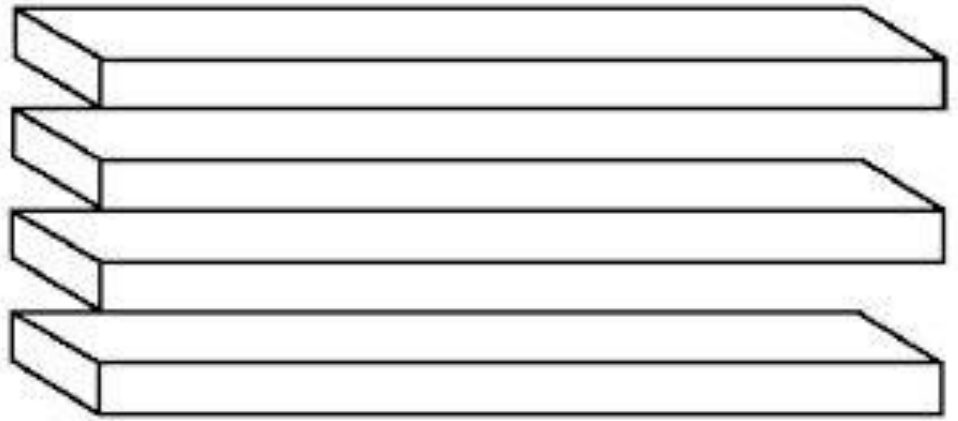
# Model

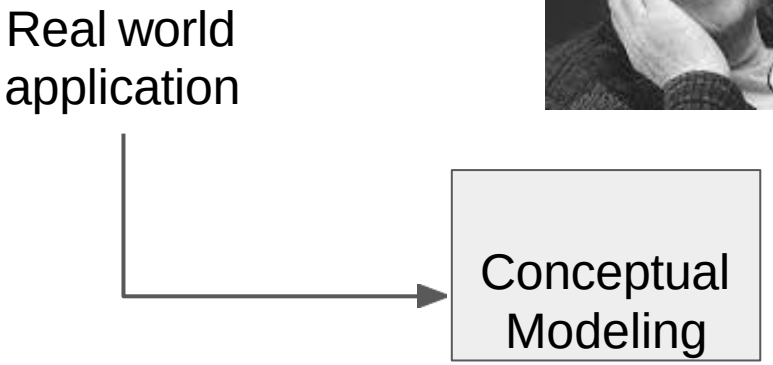

Real world application
Conceptual Modeling
Logical Modeling
Physical Modeling
Database implementation

# Conceptual Model

- How do you **describe** the application to other users?
  - Easy (for others) to understand
  - Without ambiguity (bad example: *“it stores student profiles”*)

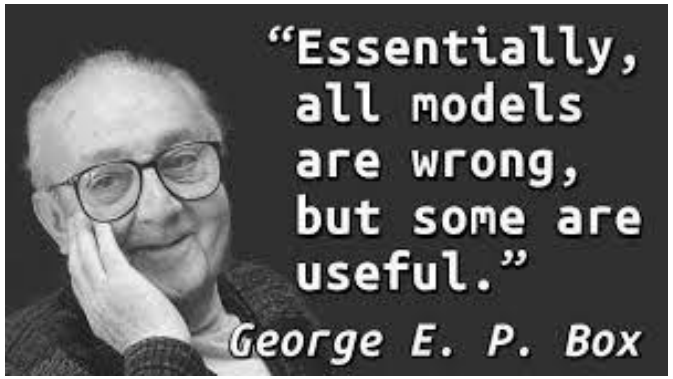


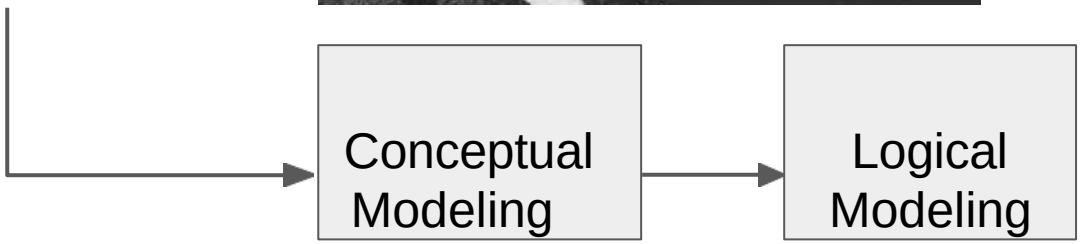


# Data Model

- How do you **describe** the data to the database?
  - In a language the database understand



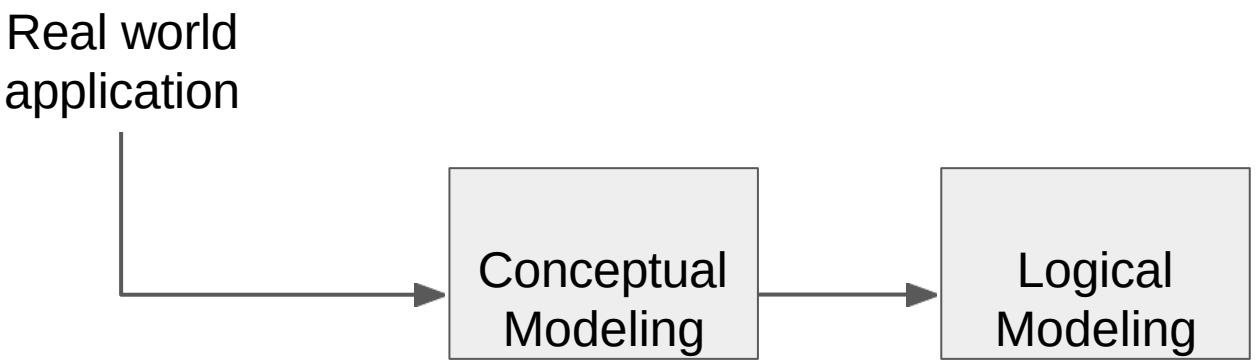

Real world application




| | |
|---|---|
| Relational | Most database |
| Key/value<br>Graph<br>Document | No-SQL |
| Array/Matrix | Machine Learning |

# Entity-Relational (ER) and Data model

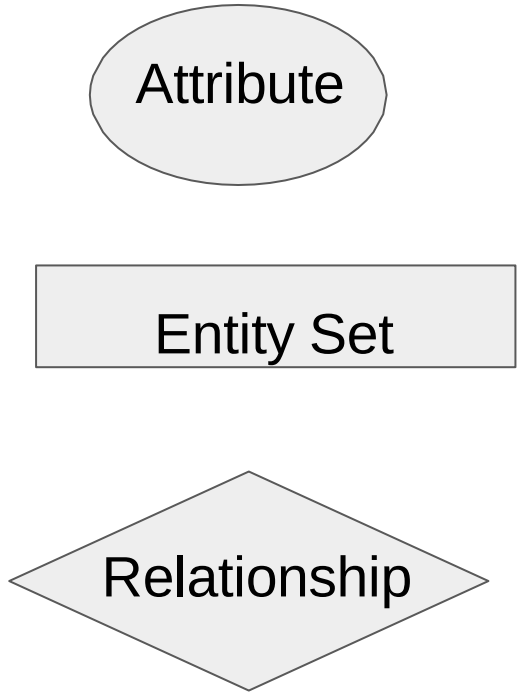
- ER Model: conceptual
  - Describe **what data** the application has
- Data Model: logical
  - Describe **what structure** the data has



# Entity-Relationship Model

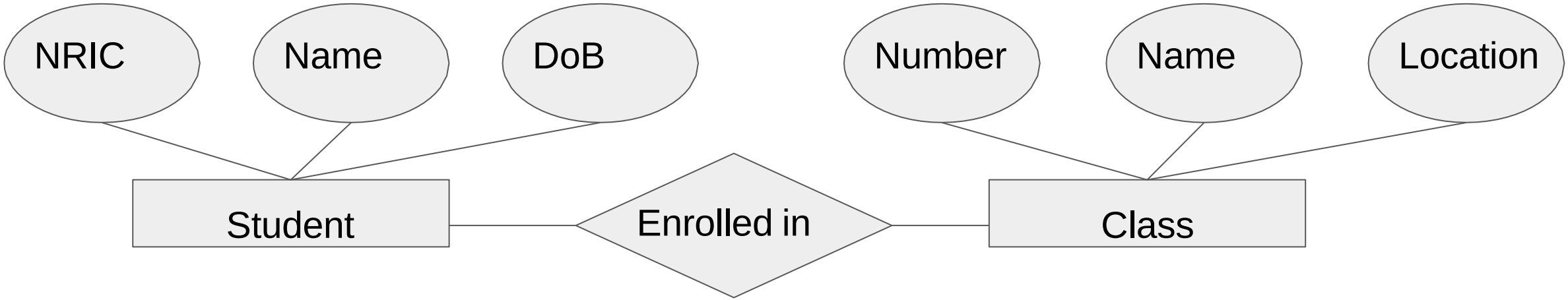
# ER Model

- A graphical diagram
  - Because a picture worth a thousand words
- ER model consists of:
  - Entity: an object
  - Entity set: a collection of similar objects
  - Attribute: property of an entity
    - Entities in the same entity set have the same set of attributes
  - Relationship: connection between entity sets



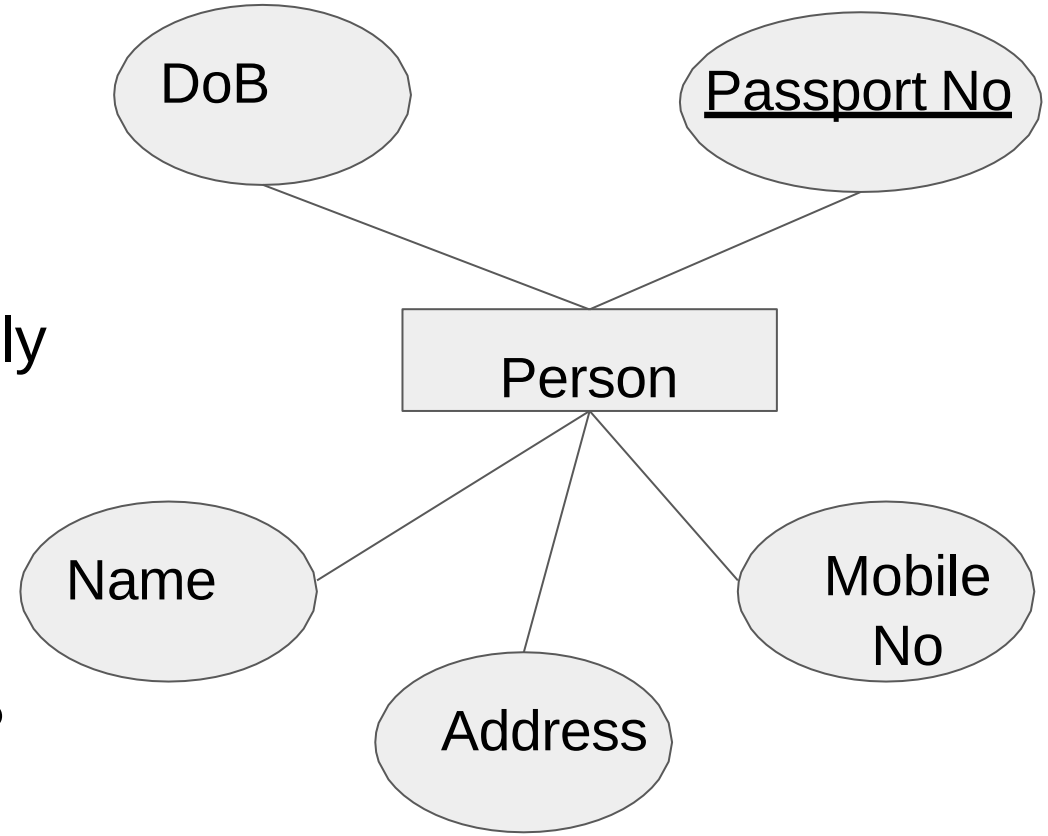
# ER Model

A student has a name, date of birth, NRIC number. A student can enrol to a class. Each class has a name, a number, and is held at a lecture theater.



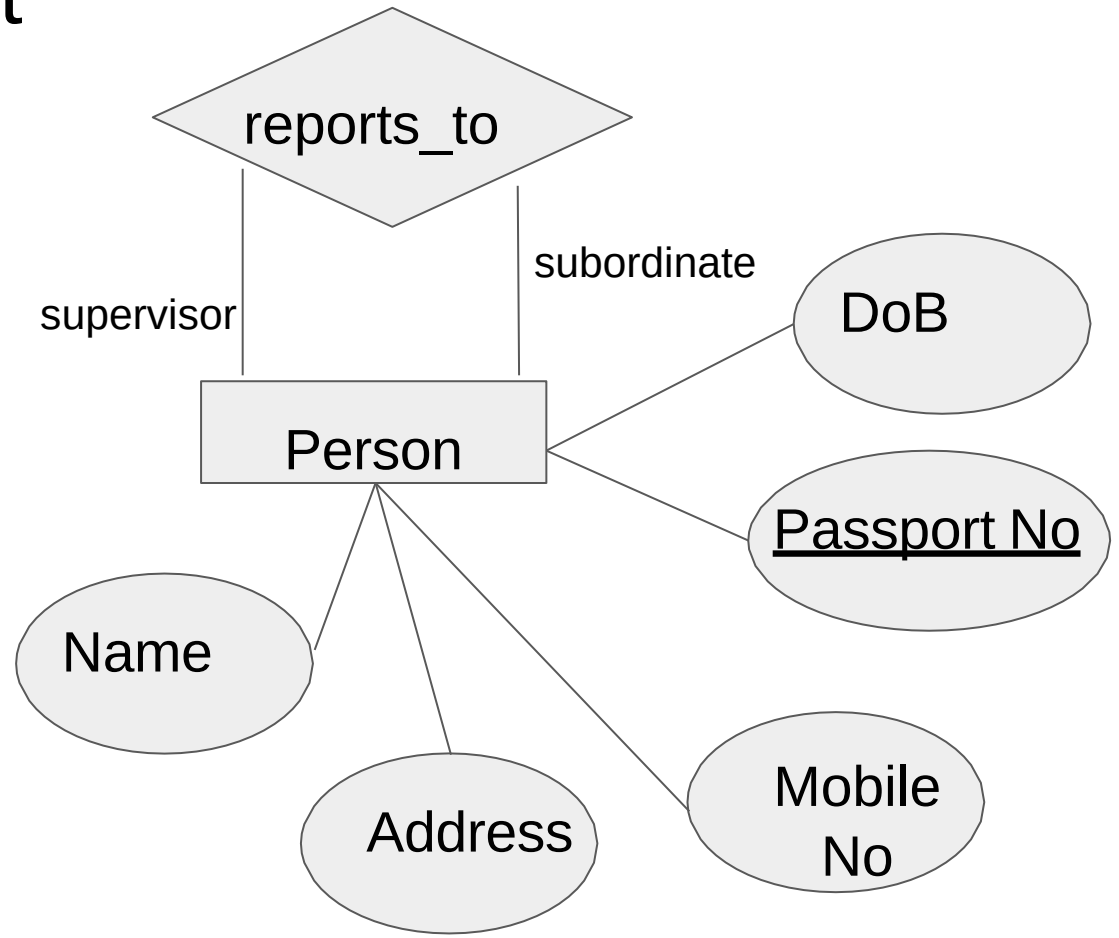
# Entity Set

- It's a **set**
  - Every entity is unique
- Primary Key:
  - *minimal set of attributes* that uniquely identifies an entity in the set
- Example:
  - Passport
  - or (Passport, Mobile, DoB, Name) ?



# Relationship

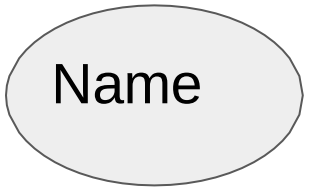
- Between entities of the same set
  - Prof / Student
  - Supervisor / Subordinate
  - etc.
- Role: model relationship in the same set



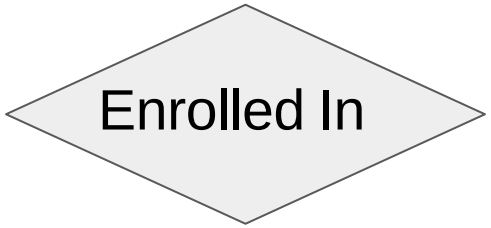
# Again: ER Diagram

Entity Set

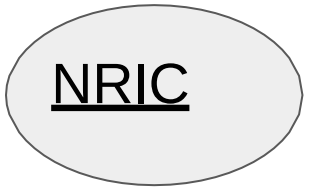

Attribute




Relationship



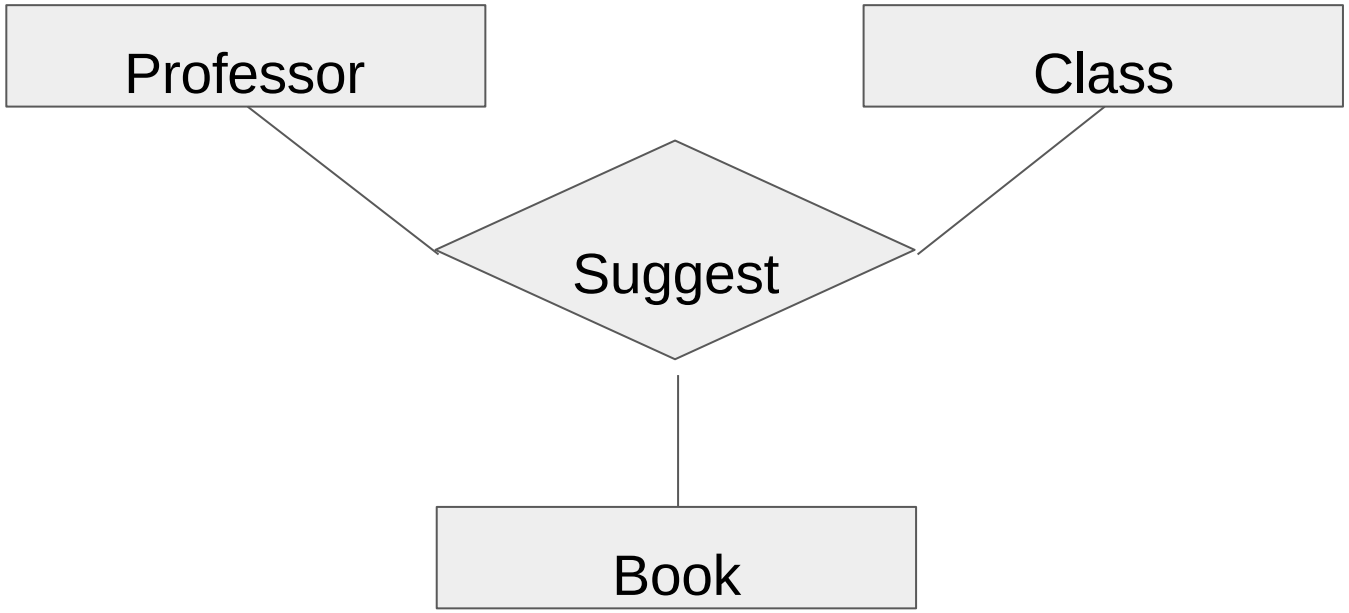

Primary Key



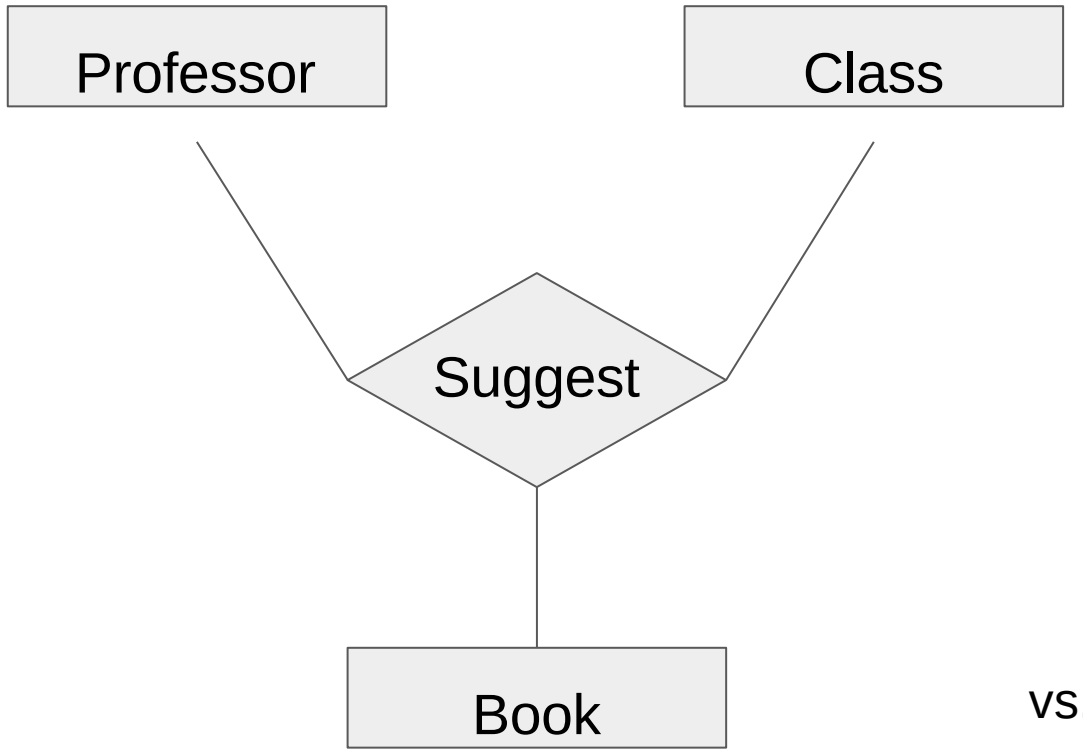

Role

# Multi-Way Relationship

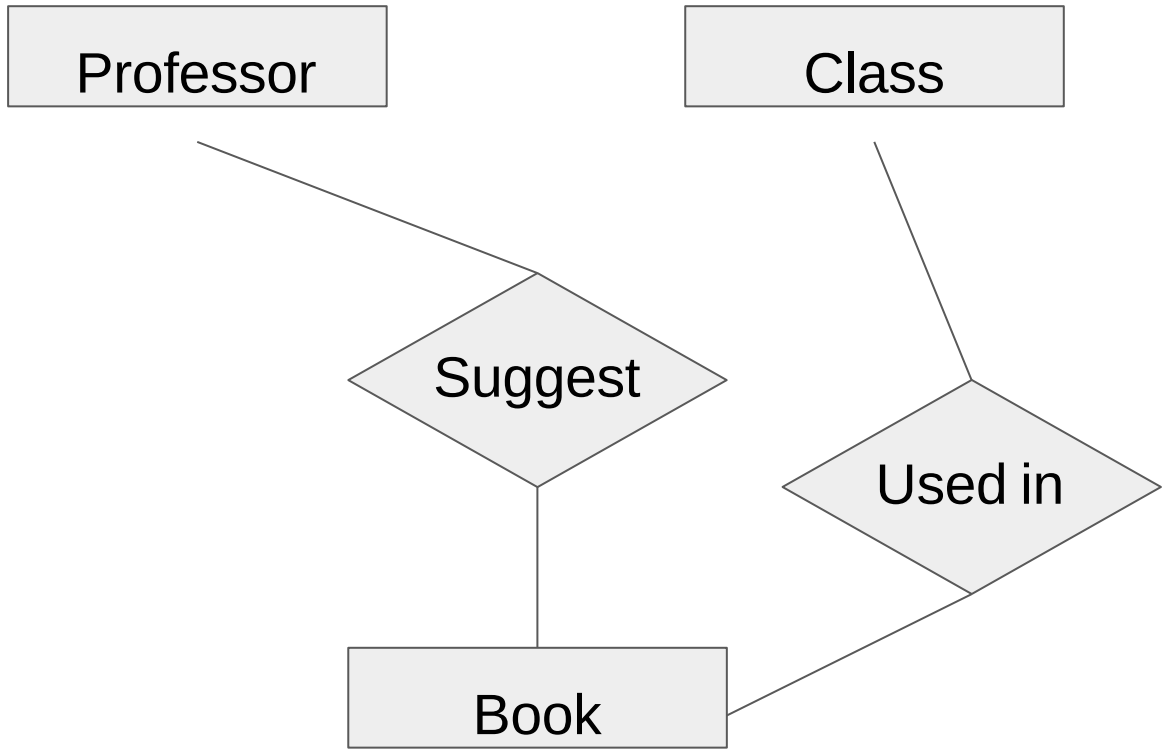
- More than 2 entity sets
  - A professor can suggest books used in a class



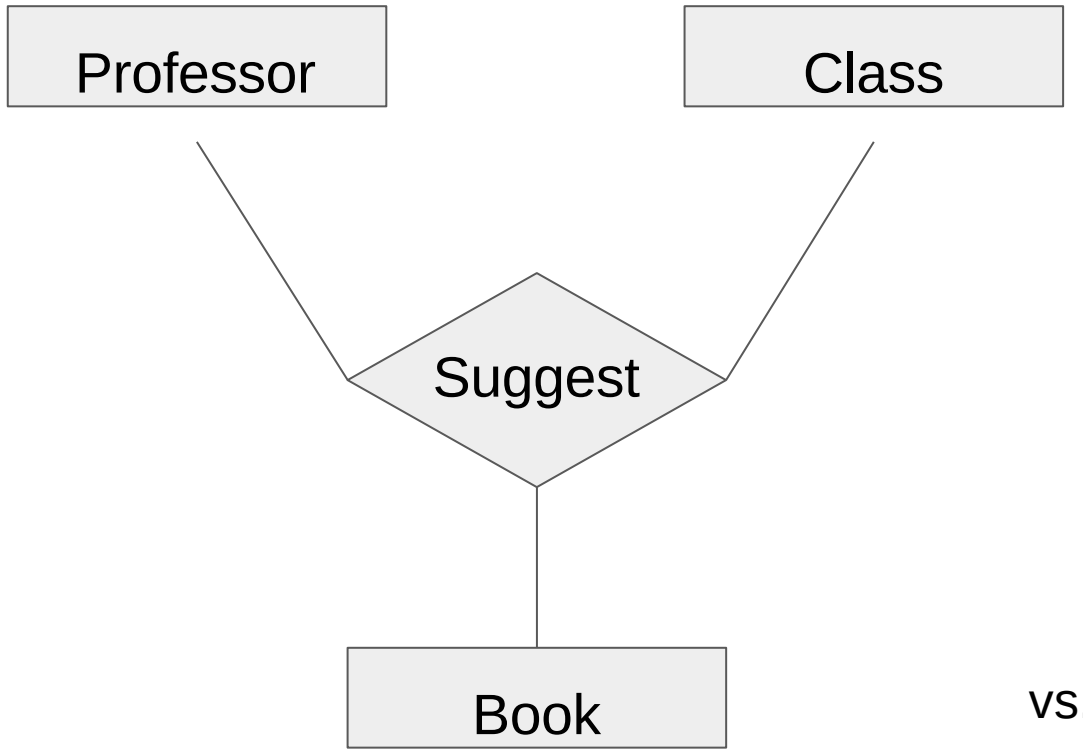
# Which Model Is Better?



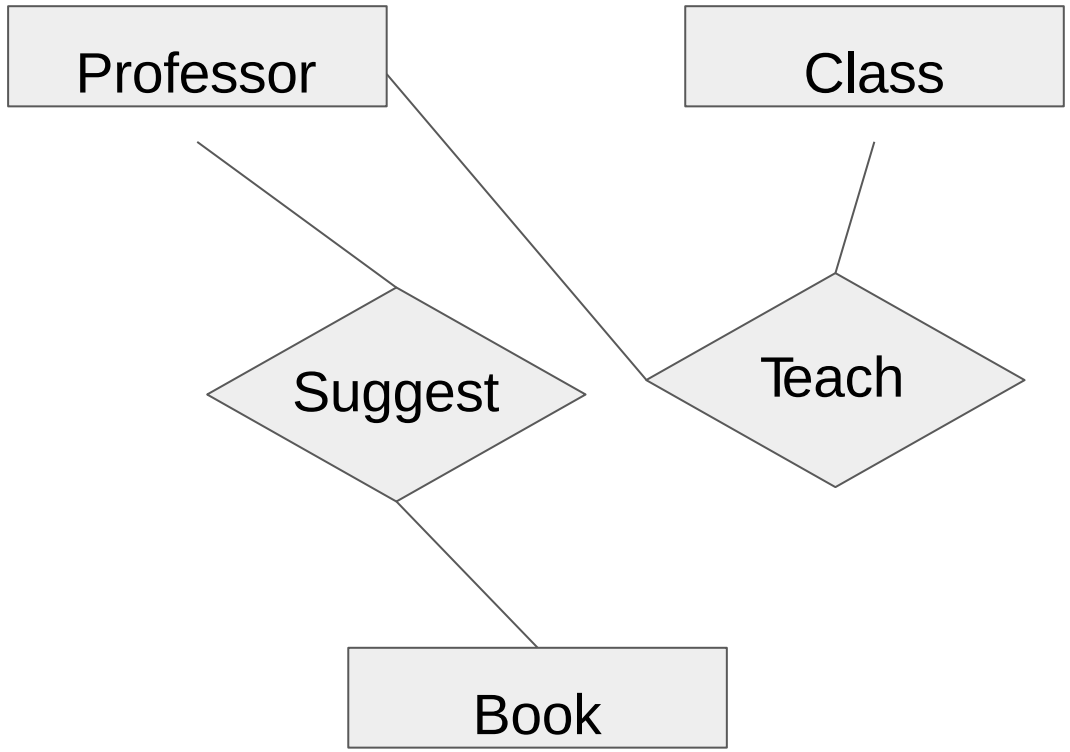

vs.



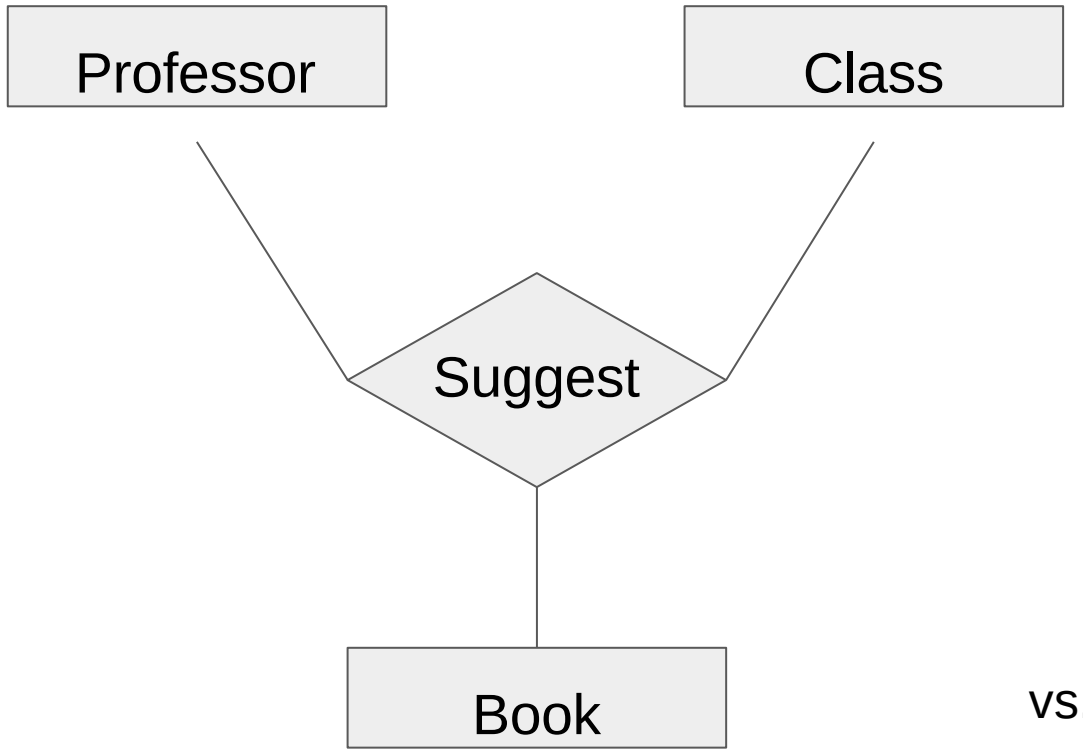
# ER Models


Professor
Class
Suggest
Book

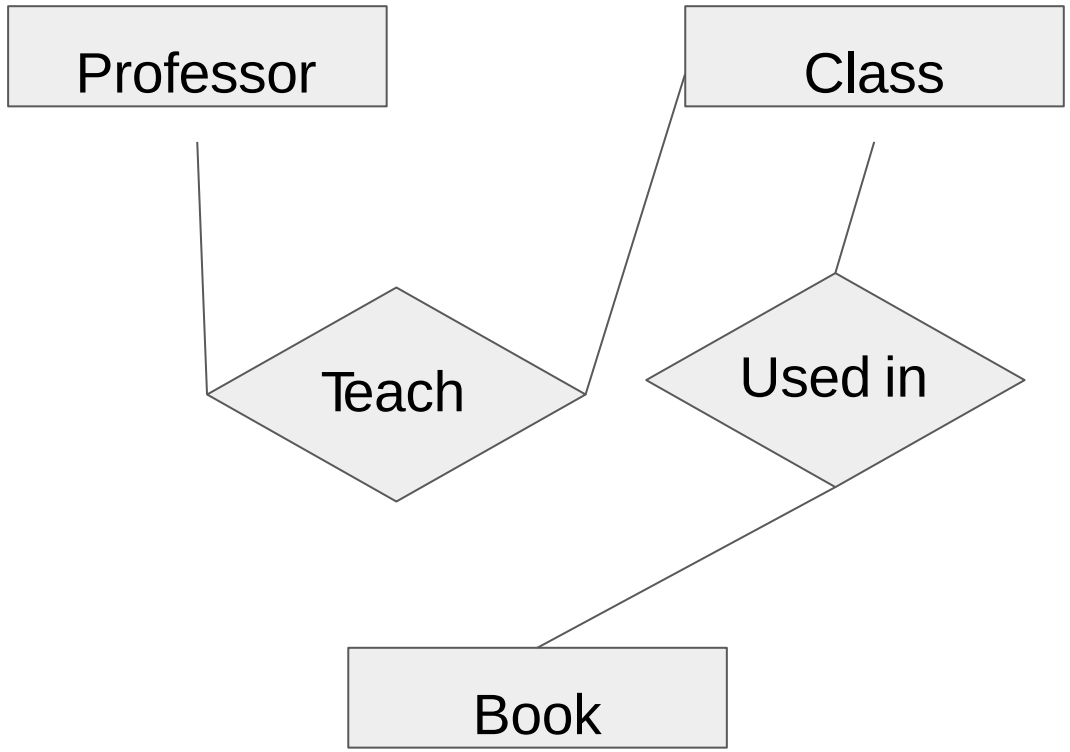
vs.


Professor
Class
Suggest
Teach
Book

# ER Models

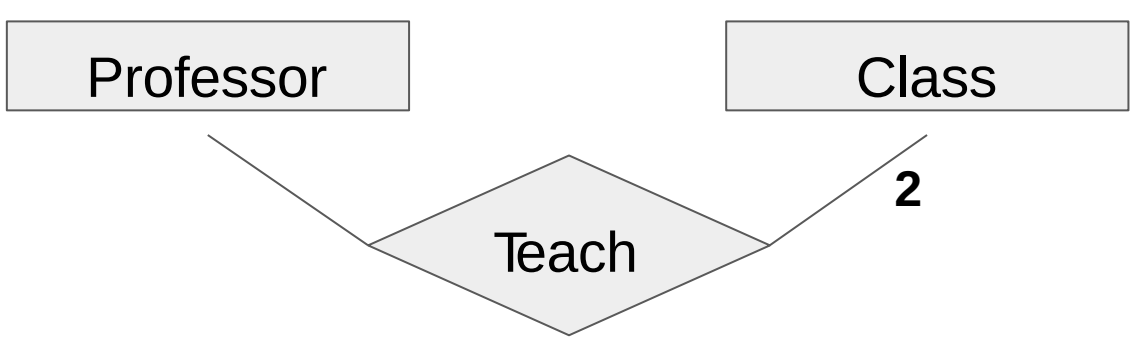

Professor
Class
Suggest
Book

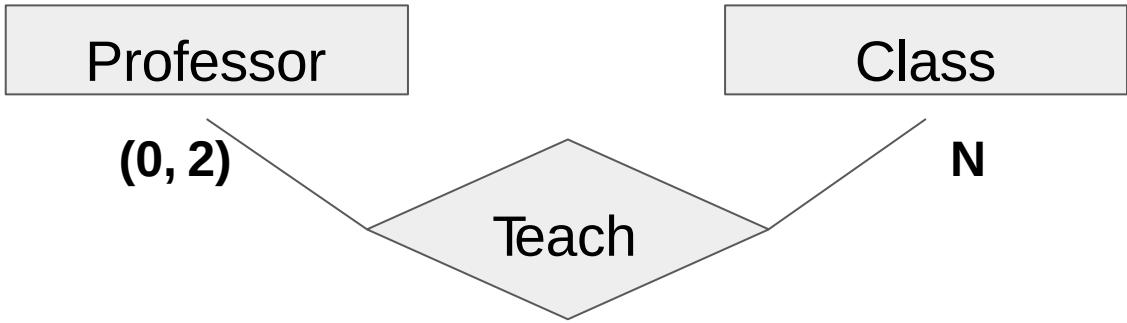
vs.


Professor
Class
Teach
Used in
Book

# ER Models

- Cardinality constraints
  - Example 1: each professor teaches up to 2 classes
  - Example 2: each professor teaches many classes, each class has at most 2 professors

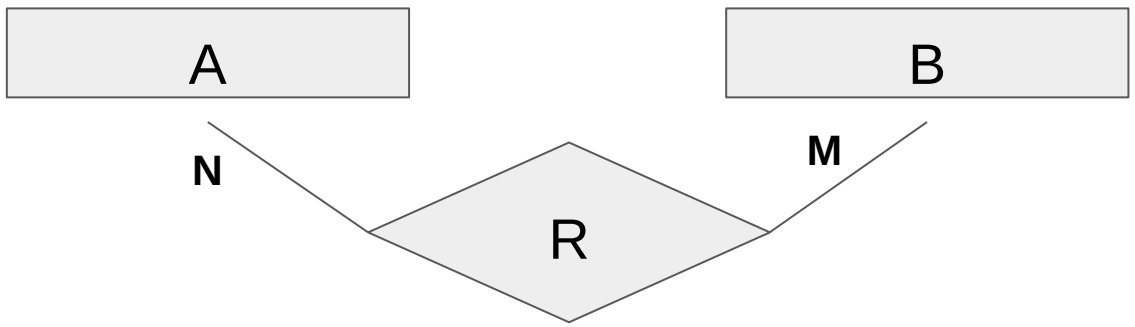


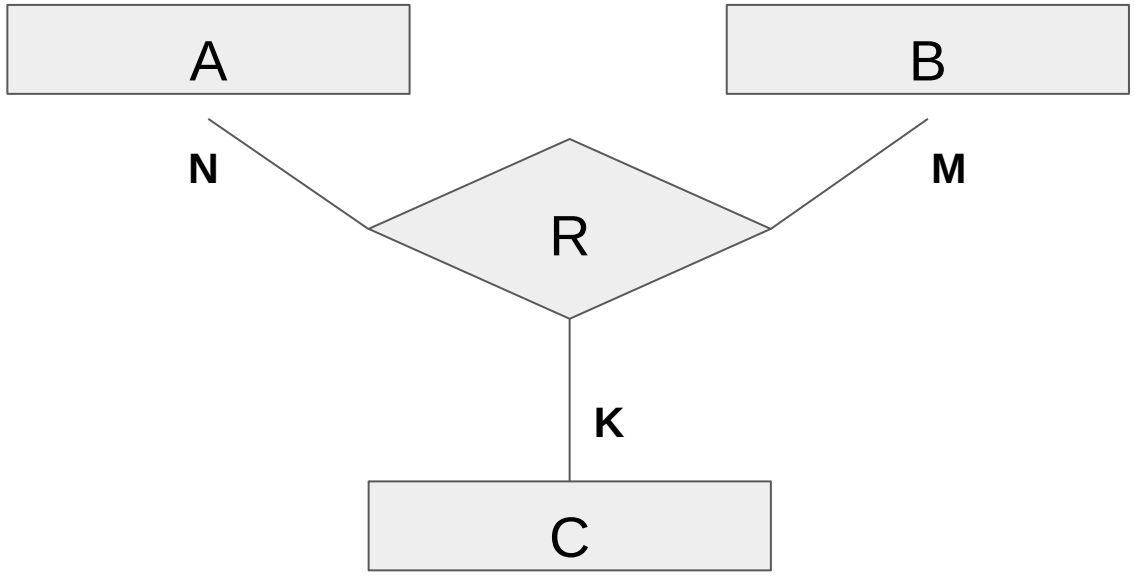
# ER Model

- Cardinality constraints

1 entity in A has relation R to **M** entities in B
1 entity in B has relation R to N entities in A

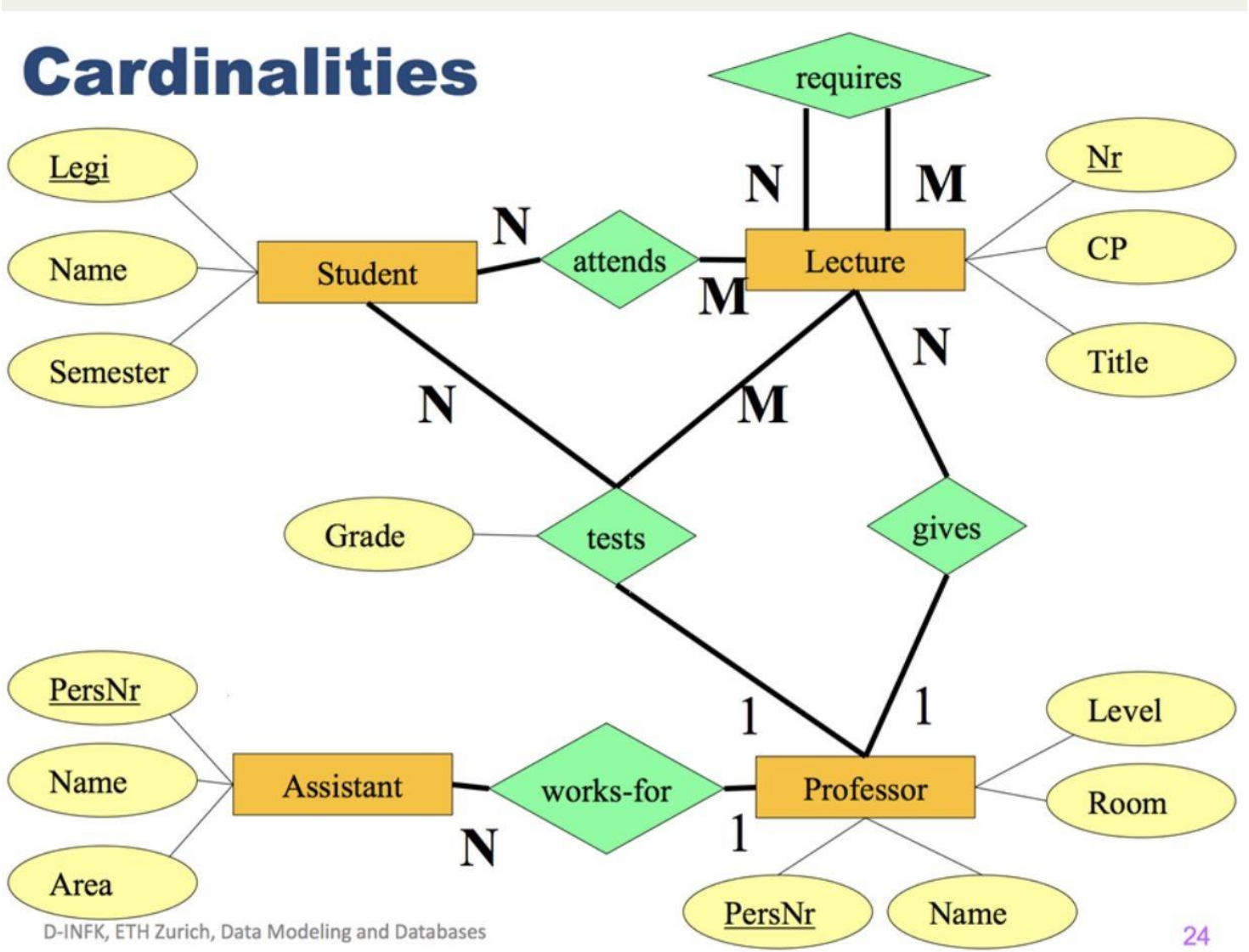



1 *pair* of entity in (A,B) has relation R to **K** entities in C
1 *pair* of entity in (A,C) has relation R to **M** entities in B
1 *pair* of entity in (B,C) has relation R to **N** entities in A



# Example of ER Model

What does this say?



# ER Recap

- ER diagram:
  - Entity Set, Attributes, Relationship, Primary Key, Role, Cardinality
- Advantages:
  - Quick to draw
  - Easy to understand
- Limitations:
  - Relationship between sets: person’s existence depends on a set of functioning organs
  - Negative relationship: Asst Prof cannot be Assoc Prof at the same time

# ER Recap

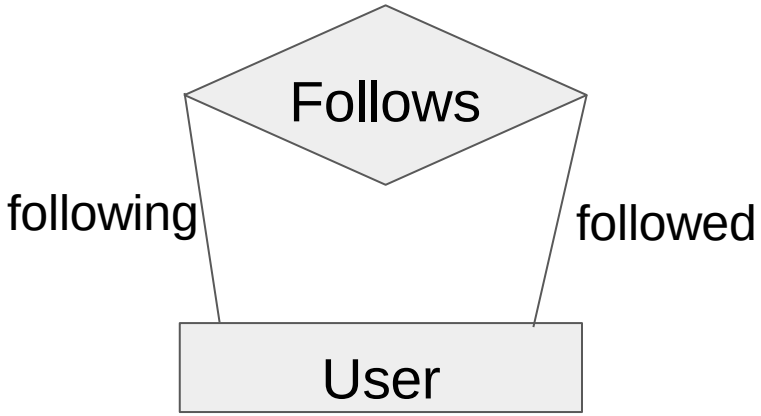
- ER in Database ~ UML in Software Engineering
- Are they useful in practice? Debatable
- Should everyone know about them? YES!

# Relational Data Model

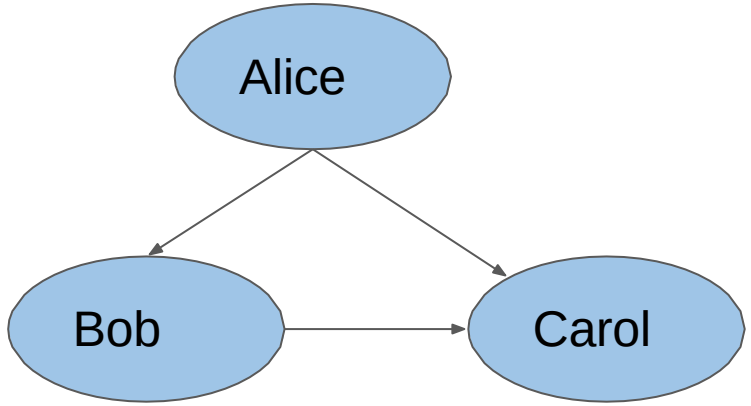
# Data Model

- ER model lets us describe *the application*
- Data model lets us describe *the data*
  - Without worrying about how it is stored or queried.

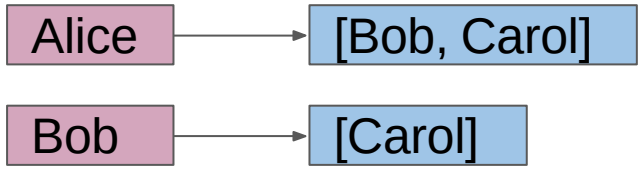
# Data Model




Graph




Key-value




Relational

| User | Following |
|---|---|
| Alice | Bob |
| Alice | Carol |
| Bob | Carol |

# Relational (SQL) vs non-Relational (no-SQL) database

- Relational database form relations between tables that store data on specific entities. Based on a structure query language.

- Non-relational database use columns and rows to enter types of data and its values and identify objects with keys.

| Key | Document |
|---|---|
| 1001 | { "CustomerID": 99, "OrderItems": [ { "ProductID": 2010, "Quantity": 2, "Cost": 520 }, { "ProductID": 4365, "Quantity": 1, "Cost": 18 }], "OrderDate": "04/01/2017" } |
| 1002 | { "CustomerID": 220, "OrderItems": [ { "ProductID": 1285, "Quantity": 1, "Cost": 120 }], "OrderDate": "05/08/2017" } |

Source: https://docs.microsoft.com/en-us/azure/architecture/data-guide/big-data/non-relational-data

# Relational (SQL) database: Pros and Cons

- Pros
  - Simple and more popular.
  - Easy access to data because of the data structure.
  - Data integrity: all entries are checked on their validity => More secure.
  - Flexibility: you can create new relations between tables without violating the existing data structure.
- Cons
  - Possible performance issues
  - Takes time to set up.
  - Does not support very complex data types

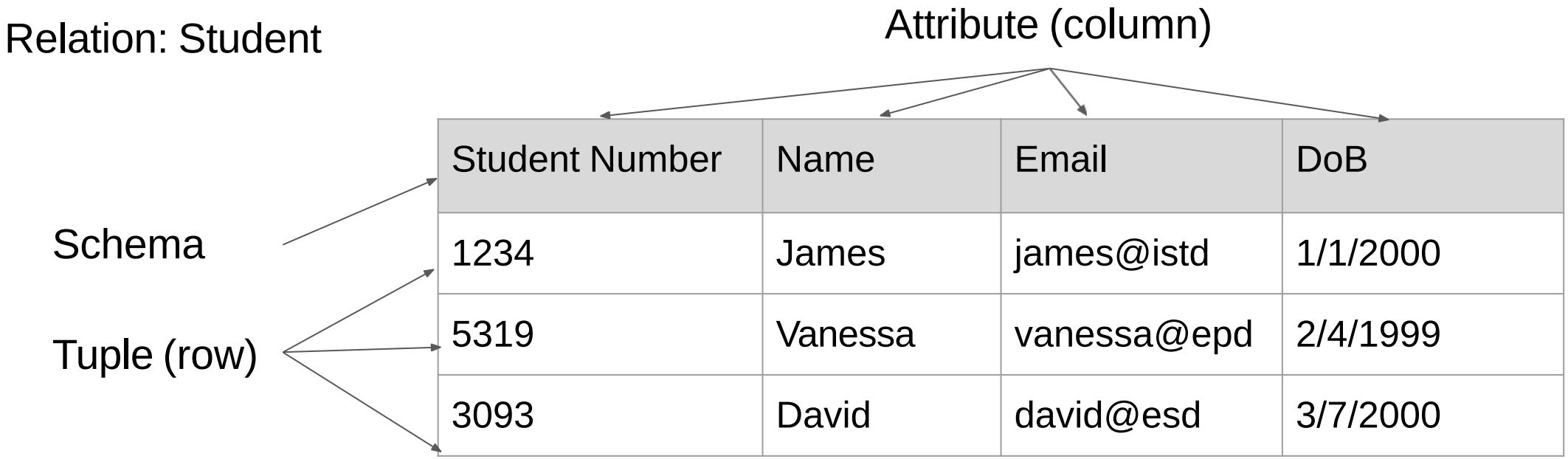
# Non-Relational (noSQL) database: Pros and Cons

- Pros
  - Support complex datatypes as it handle unstructured data
  - Agility: quicker updates can be made.
  - Readability: No need to shift between multiple tables
  - More scalable
- Cons
  - Usually dependent on a specific noSQL DBMS
  - More limited functionalities

# Relational Model

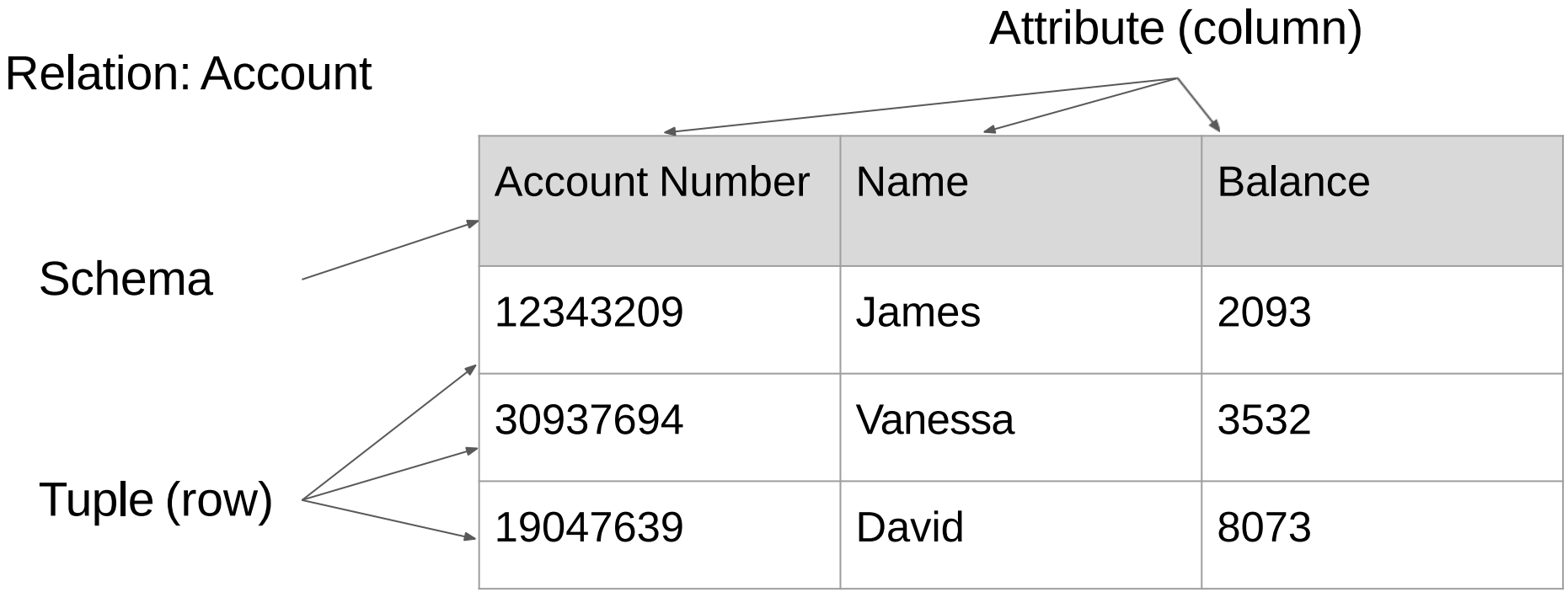
- One of the most important ideas in computer science
- Relation: an unordered set containing relationship of attributes
- Tuple: sequence of attribute values in the relation
  - Relation = {tuples}

# Relational Model




- Attribute also called Field

# Relational Model



# Relational Model

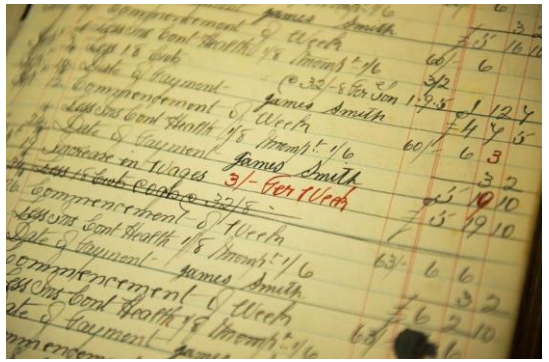
Relation: Ledger

Attribute (column)

Schema

Tuple (row)

| Account_from | Account_to | Amount | ReferenceID |
|---|---|---|---|
| 12343209 | 983620935 | 23 | 09309sdglkjwe |
| 30937694 | 800967032 | 78 | oiu093jhosgosi |
| 09094672 | 390932623 | 89 | 209sdgkljoi390 |



This is essentially what Bitcoin does!

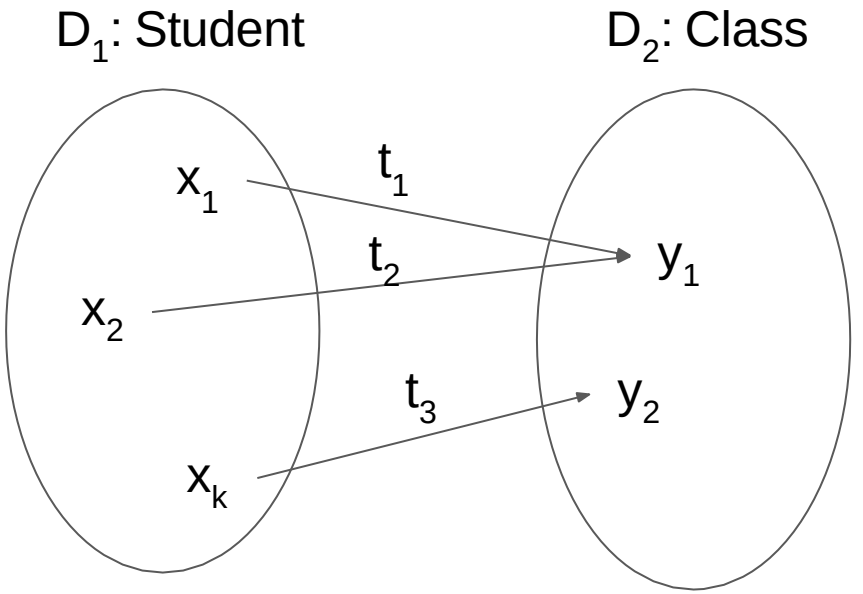
Are they just … tables?

# Relation = Table?

- Well… yes, but:
  - Not any table (next few slides)
  - Must be a set: no duplicate rows.
  - And others
- Why call it a relation? (Out of scope)
  - Because it is a *mathematical relation*

# Out of scope: Relational Model

- Relation
  - $R \subseteq D_1 \times D_2 \times D_3 \times .. \times D_N$
  - $D_1, D_2, .., D_N$ are domains, or *fields*, or *attributes*
  - Each domain has a *type*
- Tuple = record
  - $t = (d_1, d_2, .., d_n \mid d_i \in D_i) \in R$
- Schema:
  - Associate domains to name
  - $S_R = \{D_i : name_i\}$



# Database is a Set of Relations

# Out of scope: Relational Model

- Let's agree on the terminologies:
  - Relation: the entire set of all possible tuple
    - $R \subseteq D_1 \times D_2 \times D_3 \times .. \times D_N$
    - When talk about the table in general
  - Relation instance: one subset
    - $I_R \in R$
    - When refer to a specific table
- Example:
  - Student $\subseteq$ Integer $\times$ String $\times$ String $\times$ Date
  - $I_{Student}$ = {(3093, David, david@esd, 3/7/2000), (1234, James, james@istd,1/1/2000), …}

# Out of scope: Relational Model

- Mapping from relational model to table:
  - Relation → Table
  - Domain → Column
  - Tuple → Row
  - Schema → Column names
- Examples:
  - (**Student number**, Name, Email, DoB)
  - (Account number, Account number, Amount, **Reference ID**)
  - (**Class number, Student number,** Grade)

Table semantics = Relation semantics:

1. Set → must have a primary key
2. Domain has type → cannot put any value in
3. Flat → no complex object at each domain



**Not all tables are relations**

# Relational Model

- Integrity constraints:
  - Conditions that must be met if a tuple is valid
- Primary key:
  - So that the relation is a set
- Foreign key:
  - Enforce relationship between relation
  - Value in field A of $R_1$ must exist in field A of $R_2$

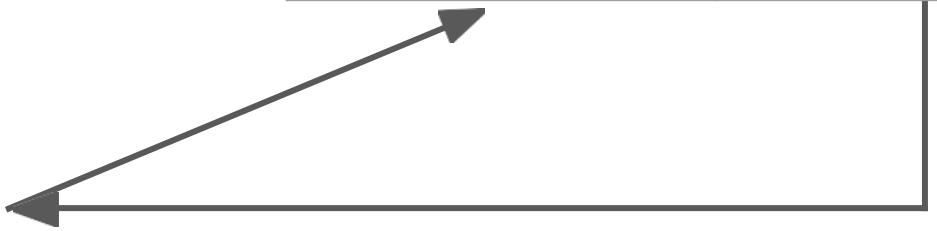
# Relational Model

- Integrity constraints:
  - Conditions that must be met if a tuple is valid
- Primary key:
  - So that the relation is a set
- Foreign key:
  - Enforce relationship between relation
  - Value in field A of $R_1$ must exist in field A of $R_2$

# Integrity Constraint

| Account number | Name | Balance |
|---|---|---|
| 12343209 | James | 2093 |
| 30937694 | Vanessa | 3532 |
| 19047639 | David | 8073 |

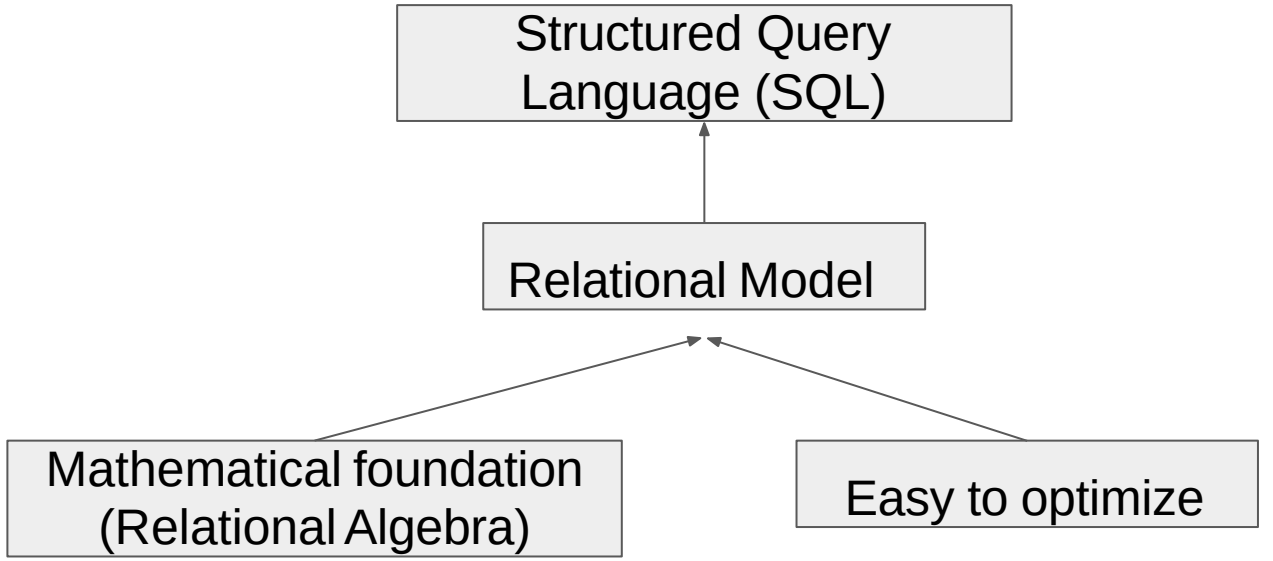
Add *(19047639, Anh, 2321)*



REJECTED

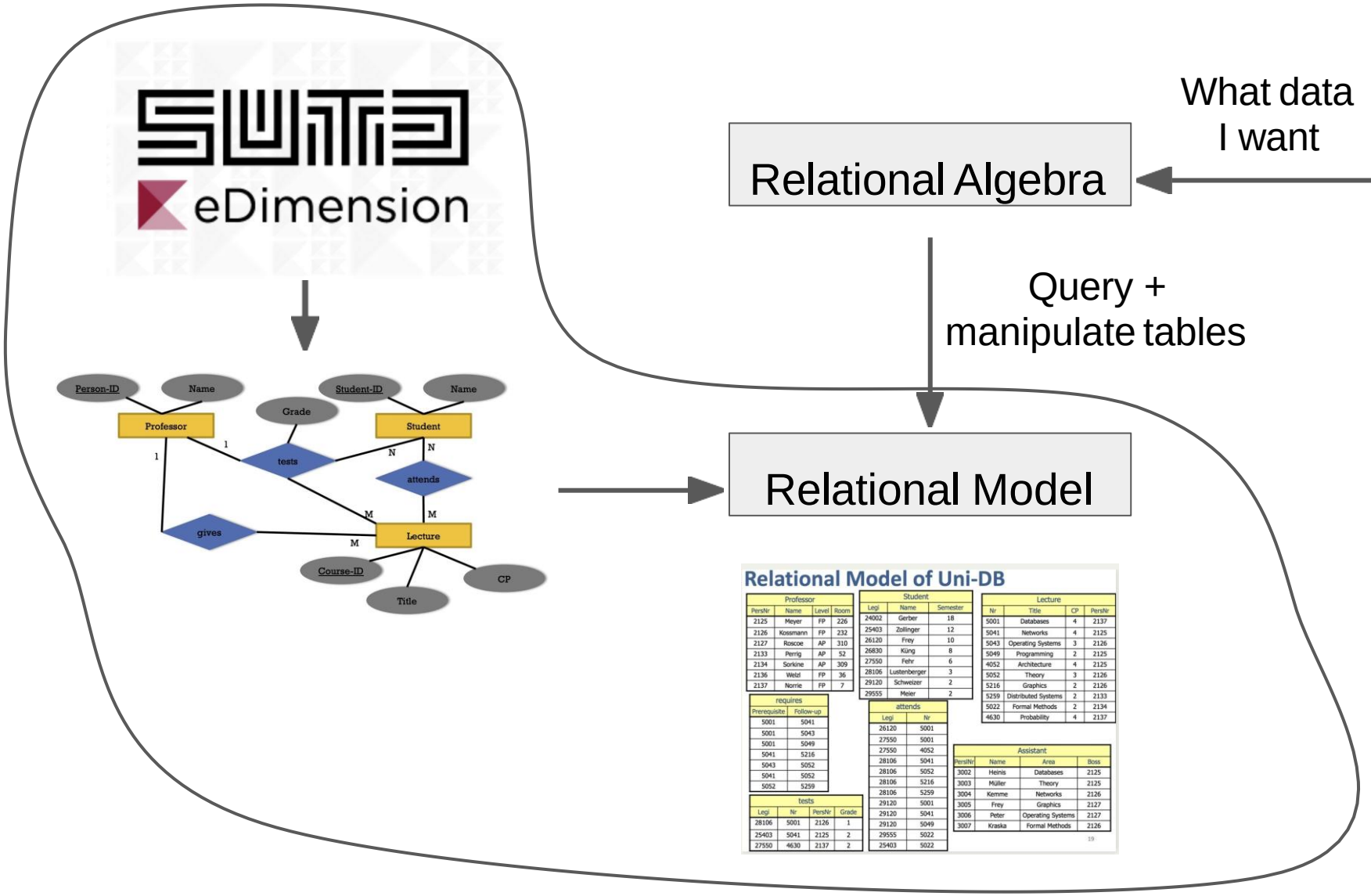
# Relational Model: Recap

- Beautifully simple:
  - Tables with some constraints
- Extremely powerful:
  - Manipulated with SQL
- Rigorous:
  - Built on strong mathematical foundation

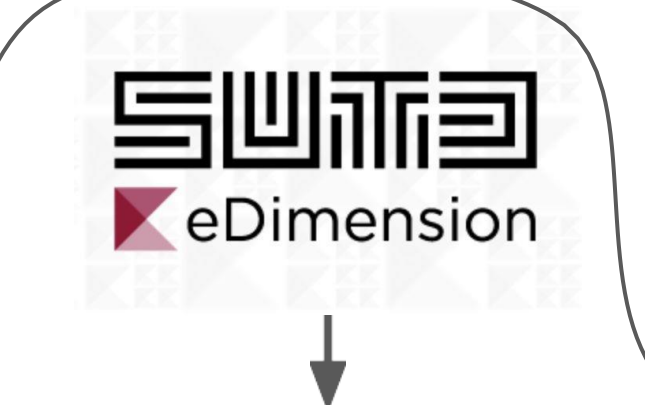


# Relational Algebra

# Data Model


eDimension
Person-ID
Name
Professor
Grade
Student-ID
Name
Student
1
1
tests
N
N
attends
M
M
gives
M
Lecture
Course-ID
CP
Title
What data
I want
Relational Algebra
SQL
Query +
manipulate tables
Relational Model
Relational Model of Uni-DB
Professor
PersNr Name Level Room
2125 Meyer FP 226
2126 Kossmann FP 232
2127 Roscoe AP 310
2133 Perrig AP 52
2134 Sorkine AP 309
2136 Weizl FP 36
2137 Norrie FP 7
Student
Legi Name Semester
24002 Gerber 18
25403 Zollinger 12
26120 Frey 10
26830 Küng 8
27550 Fehr 6
28106 Lustenberger 3
29120 Schweizer 2
29555 Meier 2
Lecture
Nr Title CP PersNr
5001 Databases 4 2137
5041 Networks 4 2125
5043 Operating Systems 3 2126
5049 Programming 2 2125
4052 Architecture 4 2125
5052 Theory 3 2126
5216 Graphics 2 2126
5259 Distributed Systems 2 2133
5022 Formal Methods 2 2134
4630 Probability 4 2137
requires
Prerequisite Follow-up
5001 5041
5001 5043
5001 5049
5041 5216
5043 5052
5041 5052
5052 5259
attends
Legi Nr
26120 5001
27550 5001
27550 4052
28106 5041
28106 5052
28106 5216
28106 5259
29120 5001
29120 5041
29120 5049
29555 5022
25403 5022
Assistant
PersNr Name Area Boss
3002 Heinis Databases 2125
3003 Müller Theory 2125
3004 Kemme Networks 2126
3005 Frey Graphics 2127
3006 Peter Operating Systems 2127
3007 Kraska Formal Methods 2126
tests
Legi Nr PersNr Grade
28106 5001 2126 1
25403 5041 2125 2
27550 4630 2137 2

# Data Model



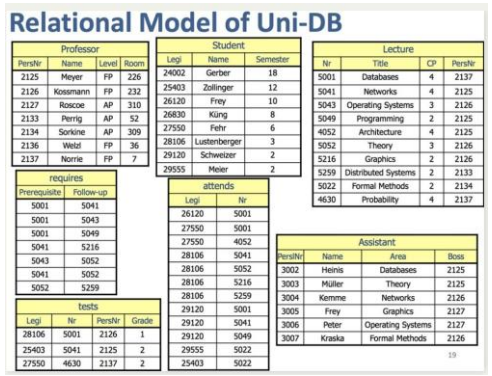

What data I want

Relational Algebra

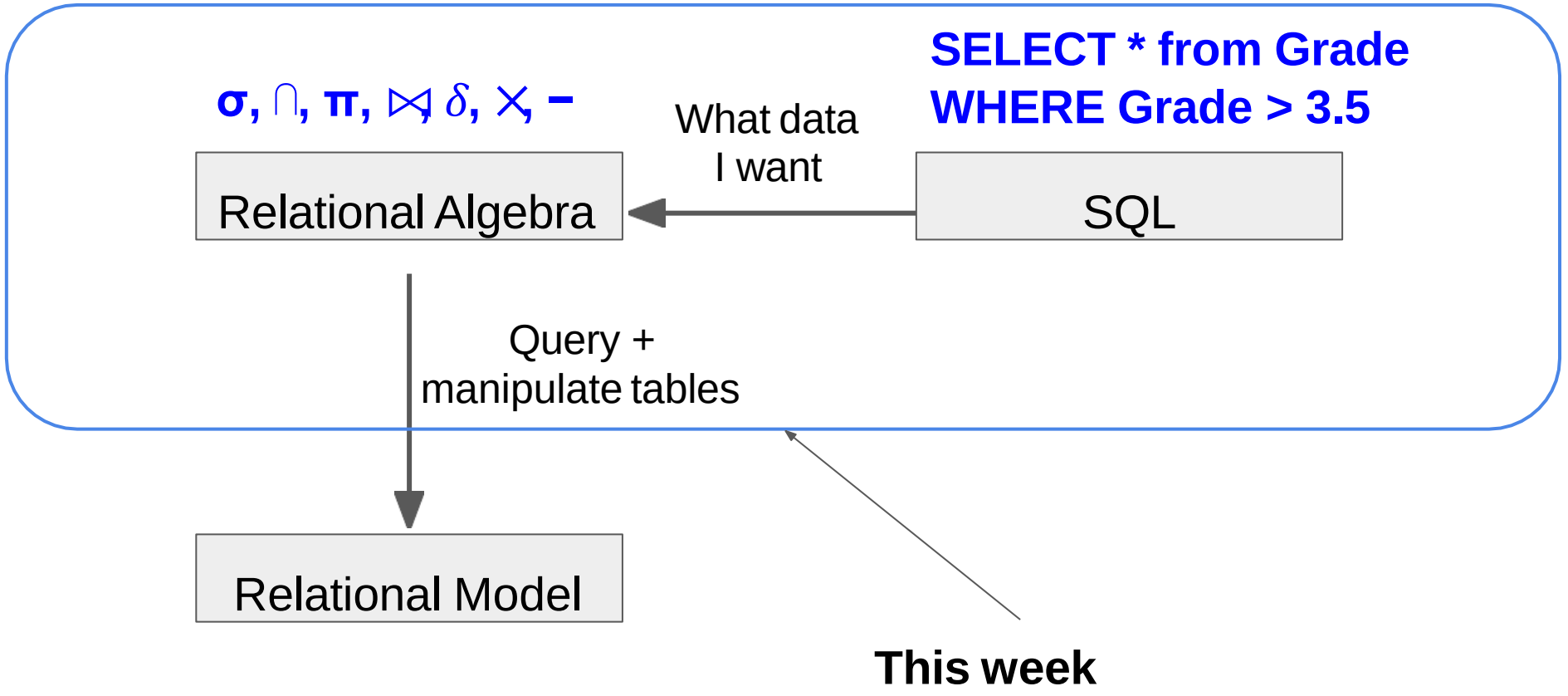
SQL

Query + manipulate tables

Relational Model



# Data Model


SELECT * from Grade
WHERE Grade > 3.5
σ, ∩, π, ⋈, δ, ×, −
What data
I want
Relational Algebra
SQL
Query +
manipulate tables
Relational Model
This week

# Example

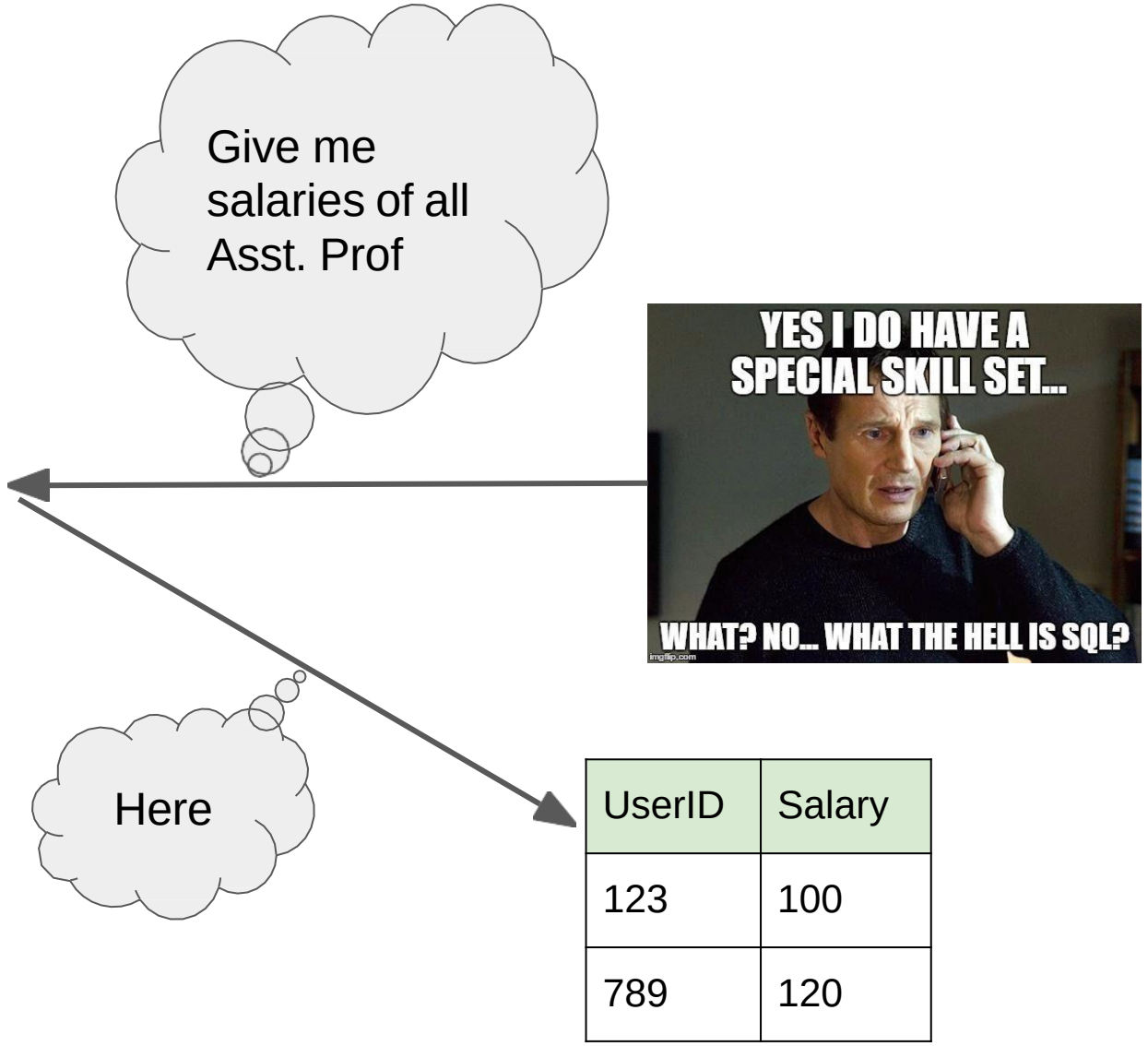
Payroll

| UserID | Name | Job | Salary |
|---|---|---|---|
| 123 | Alice | Asst. Prof | 100 |
| 456 | Bob | TA | 80 |
| 789 | Carol | Asst. Prof | 100 |
| 101 | David | Prof | 150 |

# Example

Payroll

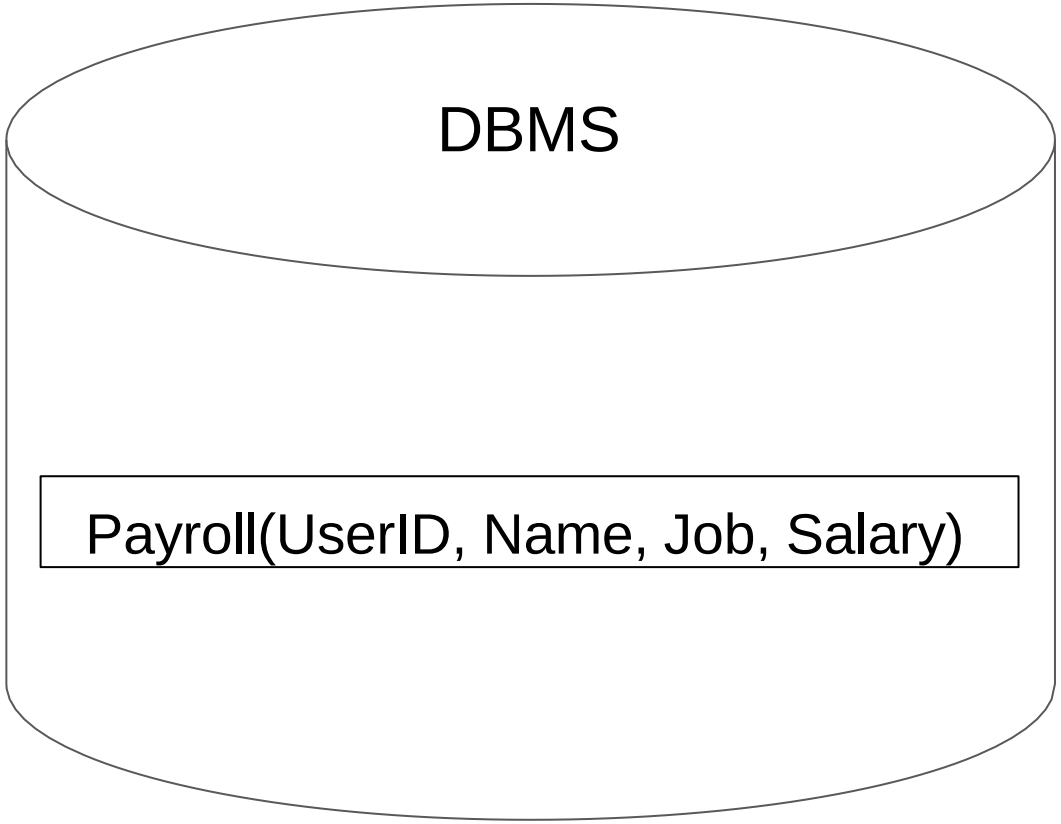
| UserID | Name | Job | Salary |
|---|---|---|---|
| 123 | Alice | Asst. Prof | 100 |
| 456 | Bob | TA | 80 |
| 789 | Carol | Asst. Prof | 120 |
| 101 | David | Prof | 150 |




| UserID | Salary |
|---|---|
| 123 | 100 |
| 789 | 120 |

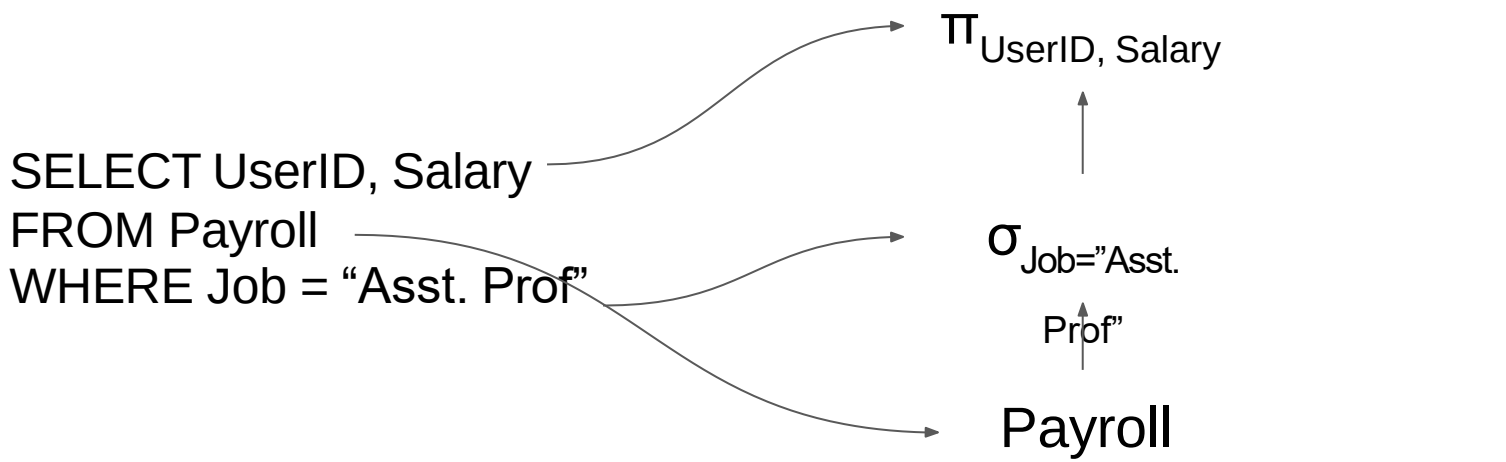
# Example

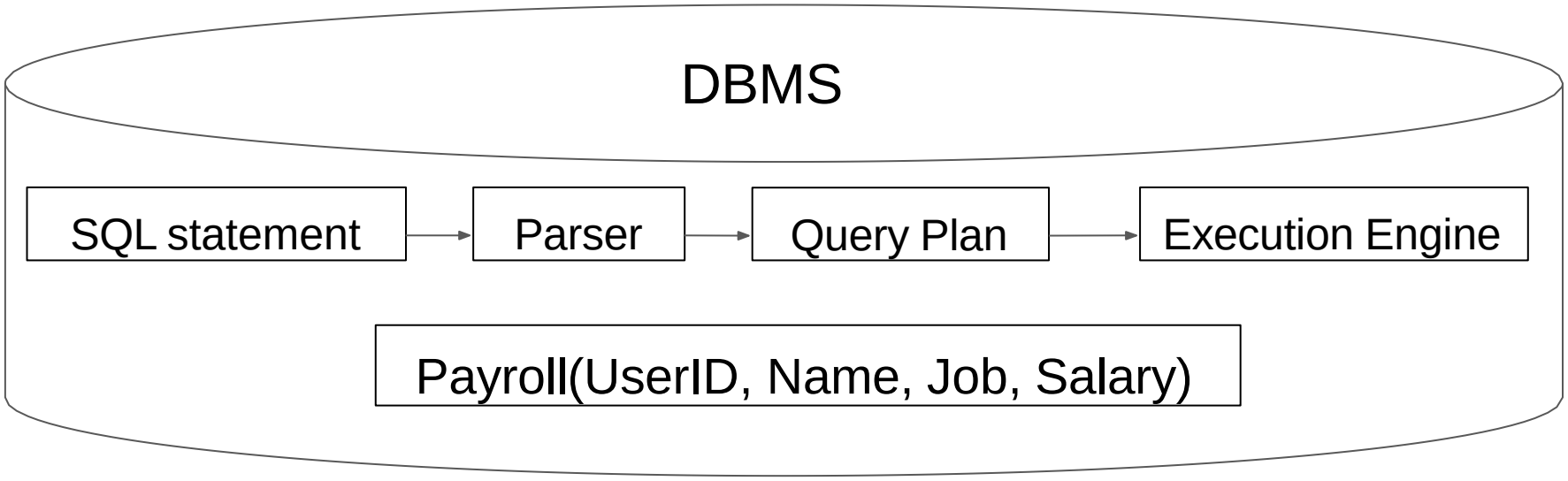
SQL query




```
SELECT UserID, Salary
FROM Payroll
WHERE Job = “Asst. Prof”
```

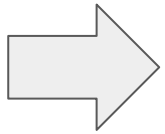
# Example

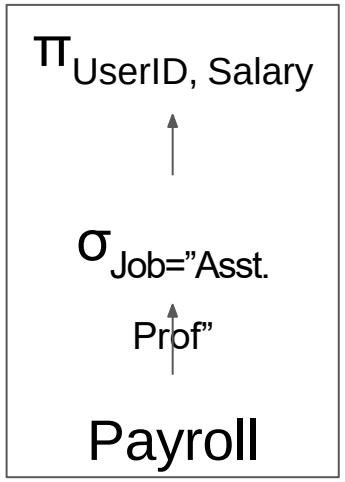

SELECT UserID, Salary
FROM Payroll
WHERE Job = “Asst. Prof”
π UserID, Salary
σ Job=”Asst. Prof”
Payroll

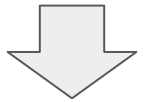

DBMS
SQL statement
Parser
Query Plan
Execution Engine
Payroll(UserID, Name, Job, Salary)

# Example

| UserID | Name | Job | Salary |
|---|---|---|---|
| 123 | Alice | Asst. Prof | 100 |
| 456 | Bob | TA | 80 |
| 789 | Carol | Asst. Prof | 120 |
| 101 | David | Prof | 150 |

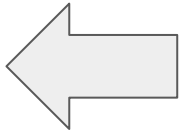




Execution Engine

```
foreach row in Payroll:
  if (row.Job == "Asst. Prof")
    output (row.UseID, row.Salary)
```



| UserID | Salary |
|---|---|
| 123 | 100 |
| 789 | 120 |

# Relational Algebra

- Algebra:
  - Study of symbols
    - Their meanings
    - Their relationships
- Relational Algebra
  - Study of symbols that manipulates relations
  - Symbols = **operators**

**Algebra** (from Arabic *"al-jabr"*, literally meaning "reunion of broken parts"[1]) is one of the broad parts of mathematics, together with number theory, geometry and analysis. In its most general form, algebra is the study of mathematical symbols and the rules for manipulating these symbols;[2] it is a unifying thread of almost all of mathematics.[3] It includes everything from elementary equation solving to the study of abstractions such as groups, rings, and

$$x^2 - 2x - 4 = 0$$

$$\sigma_{A>3}(\pi_{X,Y}(R) \bowtie_X \pi_{X,Z}(T)) = S$$

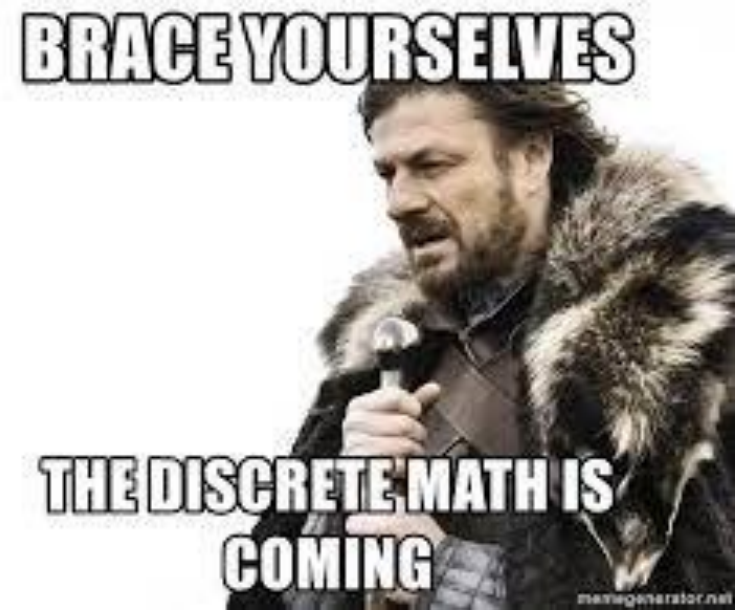
# Relational Algebra

- Fundamental operators to:
  - Retrieve
  - Manipulate relations
- Each operator:
  - Take one or more relations as input
  - Output a new relation

| | |
|---|---|
| σ | Selection |
| π | Projection |
| − | Difference |
| ∪ | Union |
| ∩ | Intersection |
| ⋈ | Join |
| × | Product |

# Relational Algebra

- We focus on 7 operators
  - They can be chained (or composed) to create more complex operators
- Based on set algebra
  - Because relation is a set




| | |
|---|---|
| σ | Selection |
| π | Projection |
| − | Difference |
| ∪ | Union |
| ∩ | Intersection |
| ⋈ | Join |
| × | Product |

# Selection

- Syntax: $\sigma_{predicate}(R)$
- Choose a subset of tuples from R that satisfy the predicate
  - Like a filter
  - Predicate can be complex, with conjunction (AND) and disjunction (OR)

```
select * from R
where A='a2' and B > 102;
```

R(A,B)

| A | B |
|---|---|
| a1 | 101 |
| a2 | 102 |
| a2 | 103 |
| a3 | 104 |

$\sigma_{A=="a2"}(R)$

| A | B |
|---|---|
| a2 | 102 |
| a2 | 103 |

$\sigma_{A=="a2" \text{ AND } B>102}(R)$

| A | B |
|---|---|
| a2 | 103 |

# Projection

- Syntax: $\pi_{A1,A2,..}(R)$
- Generate a relation with tuples containing only the specified attributes
  - Can rearrange attribute order
  - Can transform values

```
select B-100, A
from R where A='a2';
```

**R(A,B)**

| A | B |
|---|---|
| a1 | 101 |
| a2 | 102 |
| a2 | 103 |
| a3 | 104 |

$\pi_{B-100, A}(\sigma_{A==”a2”}(R))$

| B-100 | A |
|---|---|
| 2 | a2 |
| 3 | a2 |

# Union

- Syntax: **(R ∪ S)**
- Generate a relation with tuples appearing in any of the two relations
  - Exactly like set union

```
(select * from R)
UNION
(select * from S);
```

**R(A,B)**

| A | B |
|---|---|
| a1 | 101 |
| a2 | 102 |
| a3 | 103 |

**S(A,B)**

| A | B |
|---|---|
| a3 | 103 |
| a4 | 104 |
| a5 | 105 |

**R ∪ S**

| A | B |
|---|---|
| a1 | 101 |
| a2 | 102 |
| a3 | 103 |
| a4 | 104 |
| a5 | 105 |

# Intersection

- Syntax: **(R ∩ S)**
- Generate a relation with tuples appearing in both relations
  - Exactly like set intersection

```
(select * from R)
INTERSECT
(select * from S);
```

**R(A,B)**

| A | B |
|---|---|
| a1 | 101 |
| a2 | 102 |
| a3 | 103 |

**S(A,B)**

| A | B |
|---|---|
| a3 | 103 |
| a4 | 104 |
| a5 | 105 |

**R ∩ S**

| A | B |
|---|---|
| a3 | 103 |

# Difference

- Syntax: **(R − S)**
- Generate a relation with tuples appearing in R but not in S
  - Exactly like set difference

```
(select * from R)
EXCEPT
(select * from S);
```

**R(A,B)**

| A | B |
|---|---|
| a1 | 101 |
| a2 | 102 |
| a3 | 103 |

**S(A,B)**

| A | B |
|---|---|
| a3 | 103 |
| a4 | 104 |
| a5 | 105 |

**R − S**

| A | B |
|---|---|
| a1 | 101 |
| a2 | 102 |

# Product

- Syntax: **(R ×S)**
- Generate a relation with all possible combination of tuples from R and S
  - Exactly like set Cartesian product
  - R, S can have different schema

```
select * from R cross join S;
```

**R(A,B)**

| A | B |
|---|---|
| a1 | 101 |
| a2 | 102 |

**S(C,D)**

| C | D |
|---|---|
| a3 | 103 |
| a4 | 104 |

**R ×S**

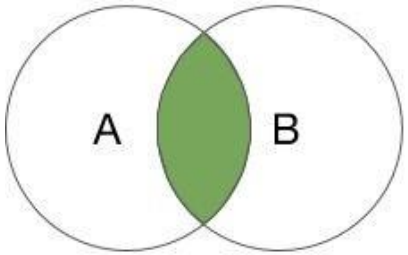
| R.A | R.B | S.C | S.D |
|---|---|---|---|
| a1 | 101 | a3 | 103 |
| a1 | 101 | a4 | 104 |
| a2 | 102 | a3 | 103 |
| a2 | 102 | a4 | 104 |

# Join

- Bread and butter! Very important
- Already seen cross-join: $\times$
- We focus on three variants:
  - Inner Join (Equi-Join)
  - Natural Join
  - Left/Right/Full Outer Join

# Inner Join

- Syntax: **$(R \bowtie_{R.A = S.D, R.B=S.E, ..} S)$**
- Generate a relation with tuples appearing in R ×S and satisfying condition
  - Product followed by Selection
  - Can join on multiple columns




INNER JOIN

```
select * from R, S
where R.A = S.D;
```

**R(A,B,C)**

| A | B | C |
|---|---|---|
| a1 | 101 | 0 |
| a2 | 102 | 1 |
| a3 | 103 | 0 |

**S(D,E,F)**

| D | E | F |
|---|---|---|
| a3 | 103 | ‘a’ |
| a1 | 107 | ‘b’ |
| a5 | 105 | ‘c’ |

**$R \bowtie_{R.A = S.D} S$**

| R.A | R.B | R.C | S.D | S.E | S.F |
|---|---|---|---|---|---|
| a1 | 101 | 0 | a1 | 107 | ‘b’ |
| a3 | 103 | 0 | a3 | 103 | ‘a’ |

# Natural Join

- Syntax: **(R ⋈ S)**
- Like Inner Join, but:
  - Automatically detect **all common attributes** (by names), and use them as join condition
  - Automatically remove duplicate column

```
select * from R natural join S;
```

**R(A,B,C)**

| A | B | C |
|---|---|---|
| a1 | 101 | 0 |
| a2 | 102 | 1 |
| a3 | 103 | 0 |

**S(A,E,F)**

| A | E | F |
|---|---|---|
| a3 | 103 | ‘a’ |
| a1 | 107 | ‘b’ |
| a5 | 105 | ‘c’ |

**R ⋈ S**

| R.A | R.B | R.C | S.E | S.F |
|---|---|---|---|---|
| a1 | 101 | 0 | 107 | ‘b’ |
| a3 | 103 | 0 | 103 | ‘a’ |

# Left Outer Join

- Syntax: **(R $\bowtie_{R.A = S.B}$ S)**
- Same as Inner Join, except:
  - All tuples of R appear in the result

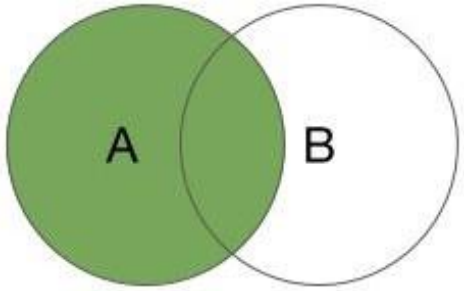
**R(A,B,C)**

| A | B | C |
|---|---|---|
| a1 | 101 | 0 |
| a2 | 102 | 1 |
| a3 | 103 | 0 |

**S(D,E,F)**

| D | E | F |
|---|---|---|
| a3 | 103 | ‘a’ |
| a1 | 107 | ‘b’ |
| a5 | 105 | ‘c’ |

```
select * from R left outer join S
where R.A = S.D;
```



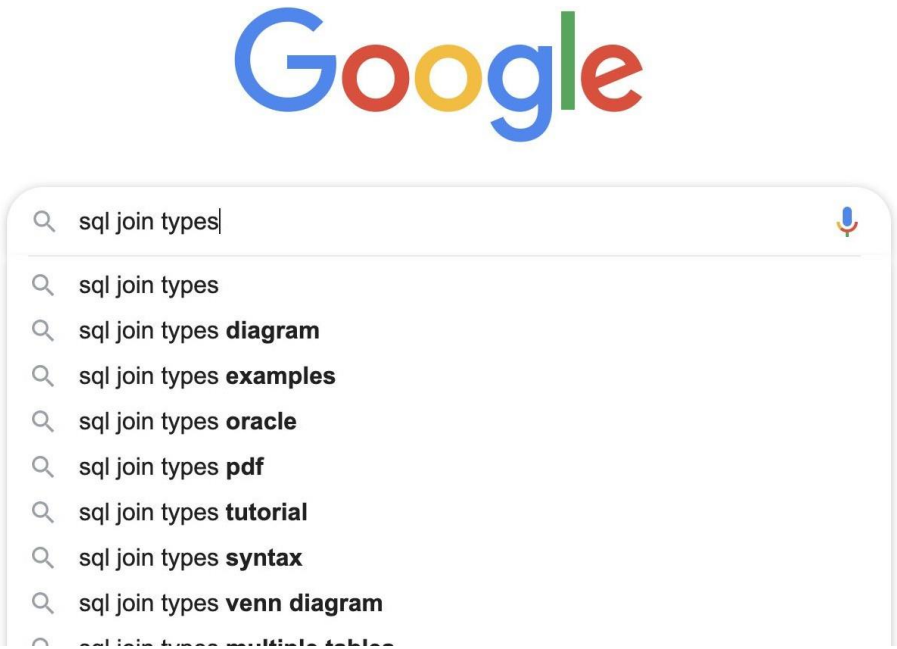

LEFT OUTER JOIN

**R $\bowtie_{R.A = S.D}$ S**

| R.A | R.B | R.C | S.D | S.E | S.F |
|---|---|---|---|---|---|
| a1 | 101 | 0 | a1 | 107 | ‘b’ |
| a3 | 103 | 0 | a3 | 103 | ‘a’ |
| a2 | 102 | 1 | *NULL* | *NULL* | *NULL* |

# Other Operators

- Many other operators
- Check them out yourself




| | |
|---|---|
| $\rho$ | Rename |
| $\delta$ | Duplicate Elimination |
| $\gamma$ | Aggregation |
| $\tau$ | Sorting |
| $\rtimes$ | Right Outer Join |
| $\bowtie$ | Full Outer Join |

# Relational Algebra

- Observation:

$$\sigma_{B=102}(R \bowtie S) = R \bowtie (\sigma_{B=102}(S))$$

- Given input relations, there are > 1 ways to get the desired output
- Good design = only specify the output
  - Let the machine select the best way

# Activity 1: Size of results

**R(A,B)**

| A | B |
|---|---|
| 1 | x |
| 2 | y |
| 2 | z |
| 3 | x |
| 9 | a |

**S(B,C,D)**

| B | C | D |
|---|---|---|
| x | 0 | 3 |
| y | 2 | 1 |
| y | 3 | 3 |
| w | 3 | 0 |
| y | 2 | 0 |

| Expression | Size of results |
|---|---|
| $R \times S$ | |
| $R \bowtie S$ | |
| $R \bowtie_{A=D} S$ | |
| $\pi_B(R) - \pi_B(\sigma_{C<3}(S))$ | |

# Activity 2: A bit of theory

Given the following relations used in a e-commerce website

Product(**pid**, pname, price)

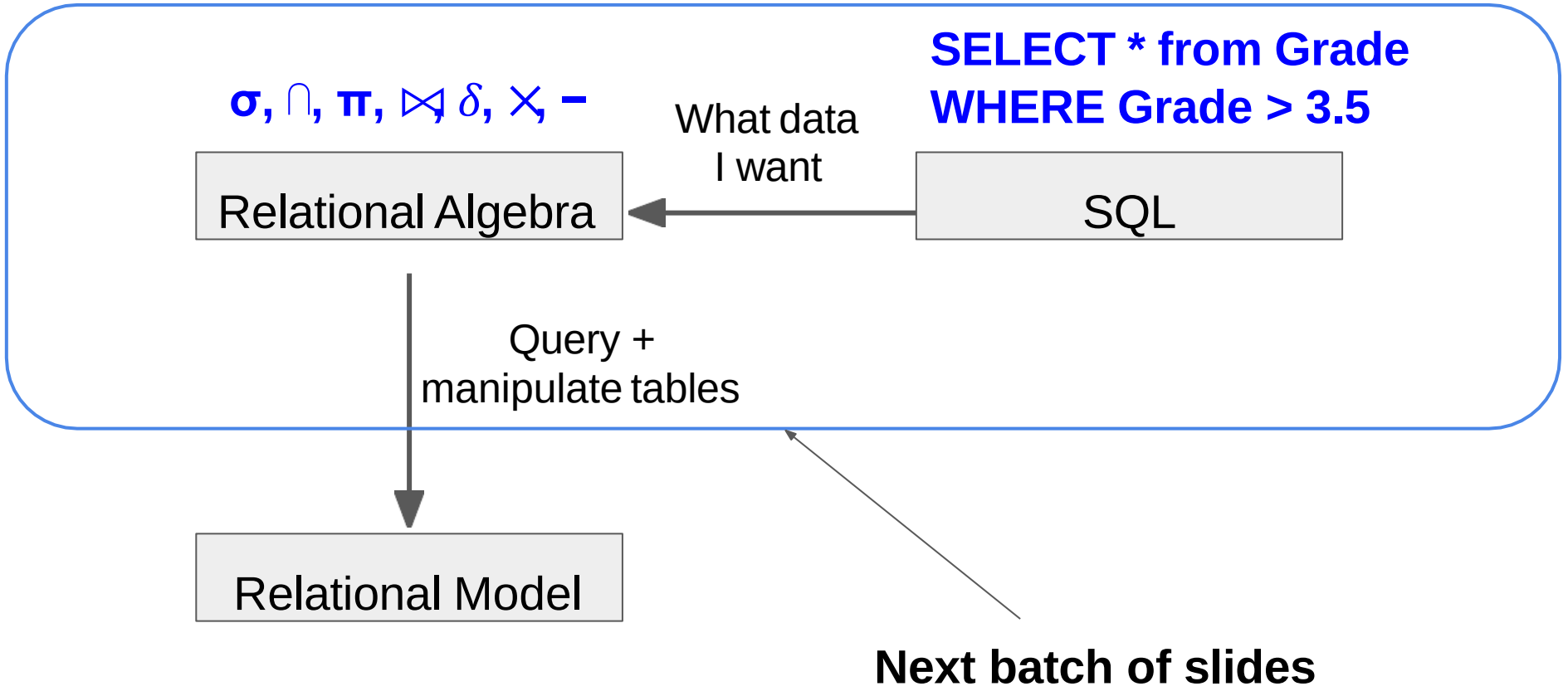
Orders(**oid**, pid, cid, qnt, date)

Customer(**cid**, cname, city)

Explain what the following relational algebra expression returns.

$$\Pi_{\text{city}}(\text{Customer} \bowtie (\Pi_{\text{cid}}(\text{Customer}) - \Pi_{\text{cid}}(\sigma_{\text{qnt}<100}(\text{Orders}))))$$

# Summary


σ, ∩, π, ⋈, δ, ×, −
What data
I want
SELECT * from Grade
WHERE Grade > 3.5
Relational Algebra
SQL
Query +
manipulate tables
Relational Model
Next batch of slides

# Conclusion

- Database: introduction
- Entity-Relationship model
- Relational Data model
- Relational Algebra

# For training only, exercise

**Reader** (ReaderID, FirstName, LastName)
**Book** (ISBN, Title, Author, PublicationDate, PublisherName)
**Publisher** (PublisherName, PublisherCity)
**Loan** (ReaderID, ISBN, Copy, ReturnDate)

Who are the readers who borrow more than 10 copies of a book at a time.

Which books (Author, Title) are from publishers in New York or London?

Which books did “Cyrille Jegourel” borrowed?