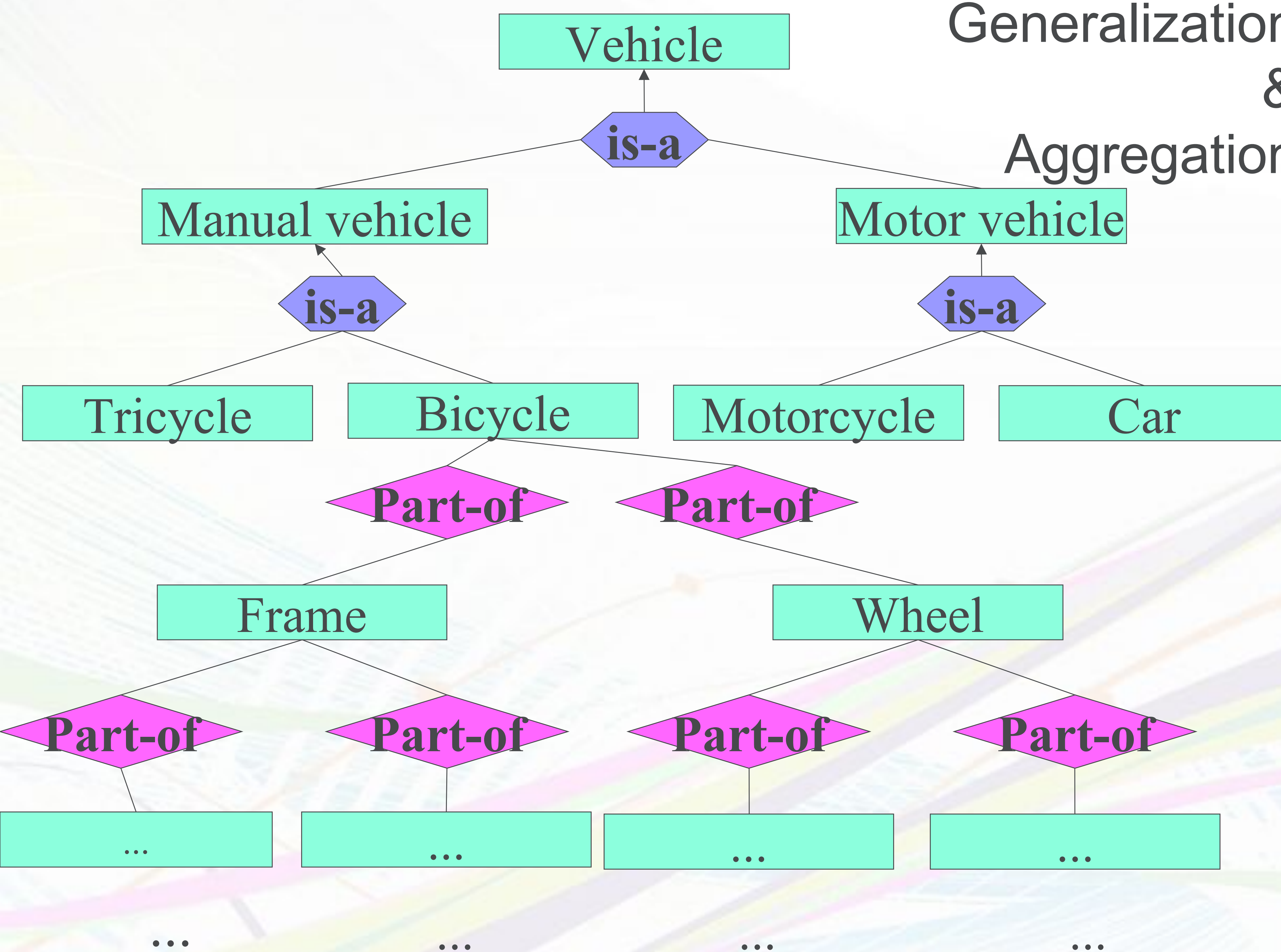
ER & Relational Models

Generalization & Aggregation



Limitations of ER

- ER has no formal semantics
  unclear whether this is a bug or a feature
  (natural language has no formal semantics either)

- No way to express relationships between sets of entities
  e.g., existence of person depends on **a set of organs**
  sets of sets are notoriously hard to model
  (more on that when we talk about 4 NF)

- No way to express negative rules
  e.g., same entity cannot be an Assistant and Professor
  again, negation notoriously hard (e.g., 2nd-order logic)

- ER has been around for 30+ years
  maybe, ER hit sweet spot of expressivity vs. simplicity
  (UML class diagrams inherit same weaknesses)

ER Modelling: Summary

- ER describes the world (the set of possible worlds)
  what it is and the laws of this world
  ER is static: it does not describe (legal) transitions

- Useful for two purposes
  build software to answer questions about the world
  judge whether something is legal or illegal

- Use Case 1: build software to answer questions
  we will learn that methodology next (ER->relational)

- Use Case 2: judge what is legal
  you need a mapping from ER to the real world
  What does "drink" mean? What does "has" mean? …
  you need to be consistent with that mapping.

Main tools of modelling: scissors   (abstraction & classification)

Data Modelling with UML

- Unified Modelling Language UML
- De-facto standard for object-orientierted design
- Data modelling is done with "class diagramms"
  - Class in UML ~ Entity in ER
  - Attribute in UML ~ Attribute in ER
  - Association in UML ~ Relationship in ER
  - Composition in UML ~ Weak Entity in ER
  - Generalization in UML ~ Generalization in ER
- Key differences between UML class diagrams and ER
  - Methods are associated to classes in UML
  - Keys are not modelled in UML
  - UML explicitly models aggregation (part-of)
  - UML supports the modelling of instances (object diagrams)
- UML has much more to offer (use cases, sequence diagr., ...)

Exercise

ER and UML modelling

- 1. An apartment is located in a house in a street in a city in a country.
- 2. Two teams play football against each other. A referee makes sure the rules are followed.
- 3. Men and women have a father and a mother each.

Relational Data Model

- ***Relation***:

  $R \subseteq D_1 \times \ldots \times D_n$ (*a set of elementary types linked with attri*)

  $D_1, D_2, \ldots, D_n$ are domains

  Example: AddressBook $\subseteq$ *string x string x integer*

- ***Tuple***: $t \in R$ (*R:a set of tuples with defined schema*)

  Example: *t* = ("Mickey Mouse", "Main Street", 4711)

- ***Schema***: associates labels to domains

  Example:

  AddrBook: {[*Name: string, Address: string, Tel#:integer*]}

AddrBook		
Name	**Street**	**Tel#**
Mickey Mouse	Main Street	4711
Minnie Mouse	Broadway	94725
Donald Duck	Broadway	95672
...	...	...

Instance: the state of the database

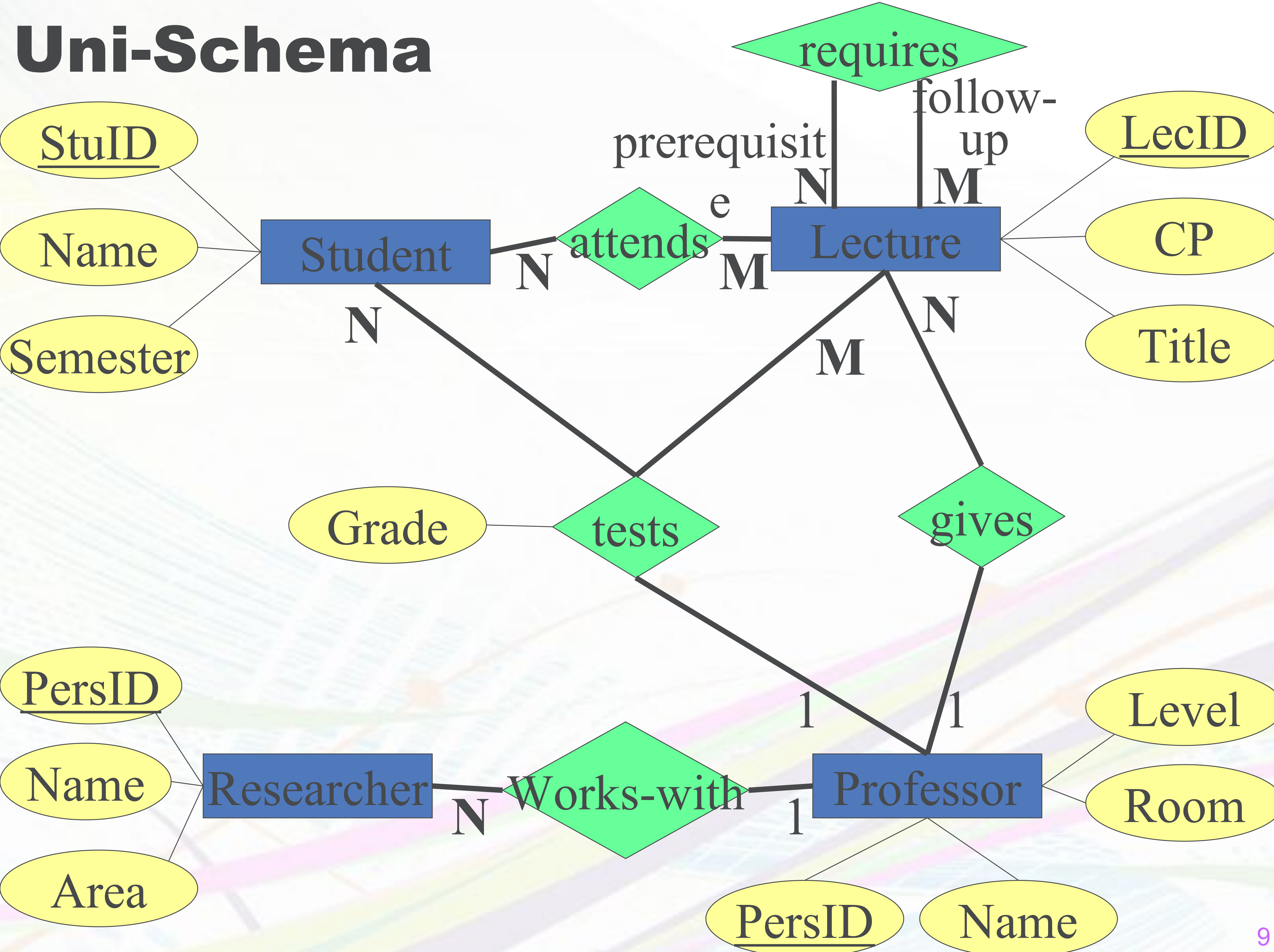
Key: **minimal** set of attributes that identify each tuple uniquely

E.g., {Tel#} or {Name, BirthDate}

Primary Key: (marked in schema by underlining)

- select one key
- use primary key for references

Uni-Schema



requires
prerequisit
e
N
follow-
up
M
StuID
Name
Semester
Student
N
attends
M
Lecture
LecID
CP
Title
N
M
N
Grade
tests
gives
1
1
PersID
Name
Area
Researcher
N
Works-with
1
Professor
Level
Room
PersID
Name

Rule #1: Implementation of Entities

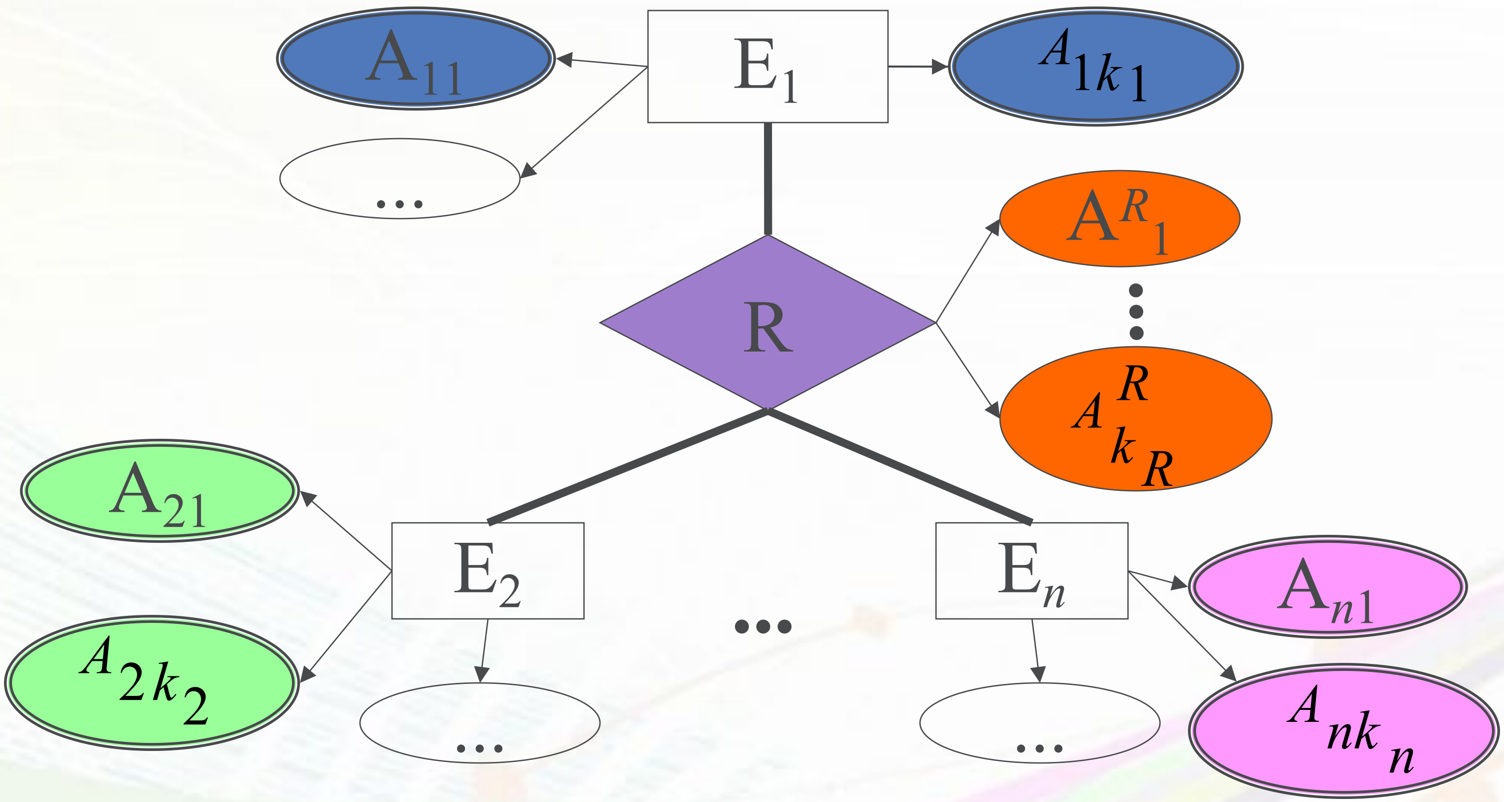
Student:{[StuID:integer, *Name*: *string, Semester*: *integer*]}

Lecture: {[LecID:integer, *Title*: *string, CP*: *integer*]}

Professor: {[PersID:integer, *Name*: *string, Level*: *string,*
Room: *integer*]}

Researcher: {[PersID:integer, *Name*: *string, Area*: *string*]}

Rule #2: Relationships



$$R:\ \{[\underbrace{A_{11},\ldots,A_{1k_1}}_{\text{Key of } E_1},\ \underbrace{A_{21},\ldots,A_{2k_2}}_{\text{Key of } E_2},\ \ldots,\ \underbrace{A_{n1},\ldots,A_{nk_n}}_{\text{Key of } E_n},\ \underbrace{A^R_1,\ldots,A^R_{k_R}}_{\text{Attributes of } R}]\}$$

Implementation of Relationships

attends : {[<u>StuID: integer, LecID: integer</u>]}

gives : {[PerID: integer, <u>LecID: integer</u>]}

works-with : {[<u>ResPersID: integer</u>, ProfPersID: integer]}

requires: {[<u>prerequisite: integer, follow-up: integer</u>]}

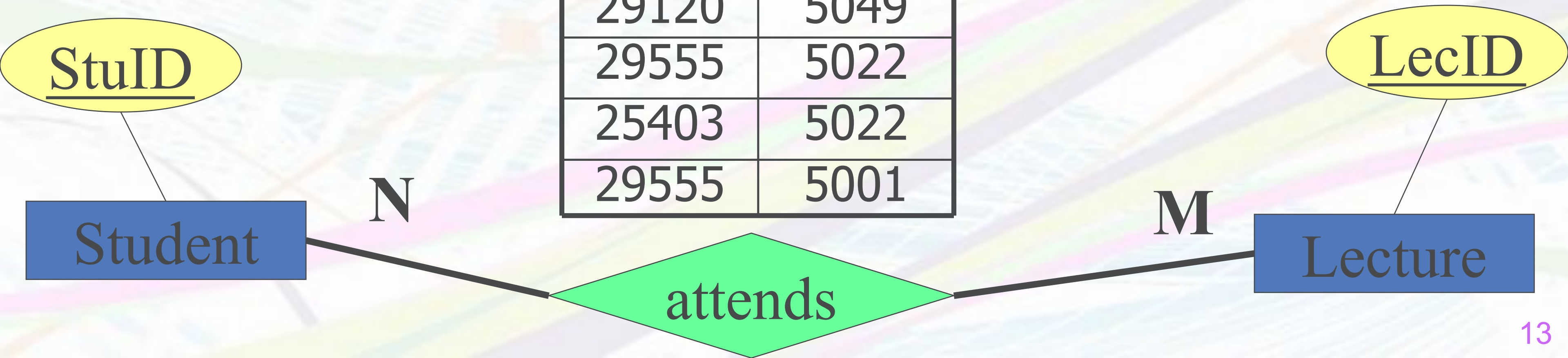
tests : {[<u>StuID: integer, LecID: integer</u>, PersID: integer, Grade: decimal]}

Instance of *attends*

Student	
StuID	...
26120	...
27550	...
...	...

attends	
StuID	LecID
26120	5001
27550	5001
27550	4052
28106	5041
28106	5052
28106	5216
28106	5259
29120	5001
29120	5041
29120	5049
29555	5022
25403	5022
29555	5001

Lecture	
LecID	...
5001	...
4052	...
...	...



Rule 2: How to name the attributes

- If the ER specifies roles

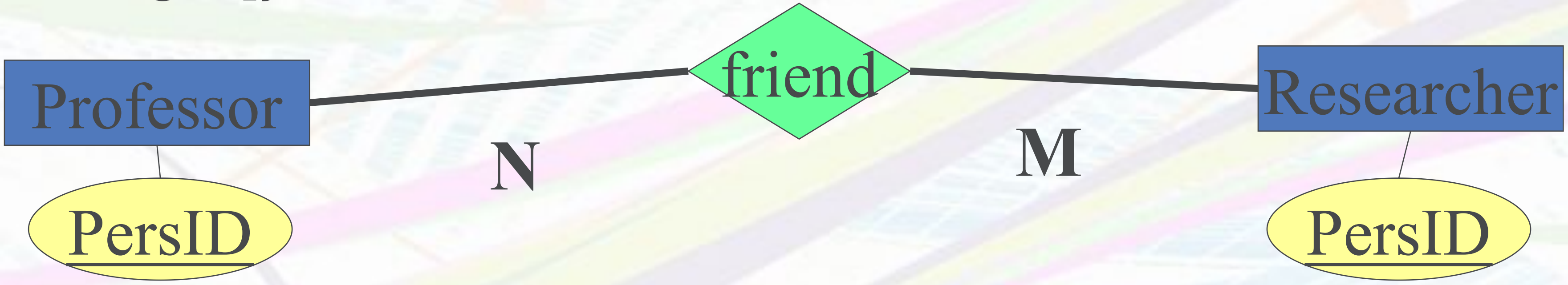
  use the names of the roles

- Otherwise

  use the names of the key attributes in the entities

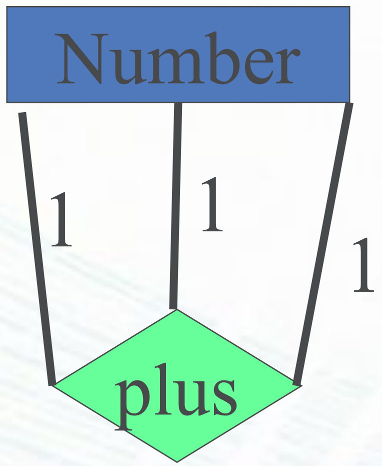
  in case of ambiguity, invent new names

- Example: **friend** : {[<u>ProfPersID: integer, ResPersID: integer</u>]}



Exercise

- Implement the following ER diagram using the relational data model



Rule #3: Merge relations with the same key

- Implementation according to Rule #2
  - ***Lecture :*** {[LecID, *Title, CP*]}
  - ***Professor :*** {[PersID, *Name, Level, Room*]}
  - ***gives:*** {[*LecID, PersID*]} *why not both as a key?*

- Merge according to Rule #3
  - ***Lecture :*** {[LecID, *Title, CP,* ***PersID***]}
  - ***Professor :*** {[*PersID Name, Level, Room*]}

Instance of *Professor* and *Lecture*

Professor			
PersID	Name	Level	Room
2125	Sokrates	FP	226
2126	Russel	FP	232
2127	Kopernikus	AP	310
2133	Popper	AP	52
2134	Augustinus	AP	309
2136	Curie	FP	36
2137	Kant	FP	7

Lecture			
LecID	Title	CP	PersID
5001	Grundzüge	4	2137
5041	Ethik	4	2125
5043	Erkenntnistheorie	3	2126
5049	Mäeutik	2	2125
4052	Logik	4	2125
5052	Wissenschaftstheorie	3	2126
5216	Bioethik	2	2126
5259	Der Wiener Kreis	2	2133
5022	Glaube und Wissen	2	2134
4630	Die 3 Kritiken	4	2137

This will NOT work

Professor				
PersID	Name	Level	Room	gives
2125	Sokrates	FP	226	5041
2125	Sokrates	FP	226	5049
2125	Sokrates	FP	226	4052
...	...	...	...	...
2134	Augustinus	AP	309	5022
2136	Curie	FP	36	??

Lecture		
LecID	Title	CP
5001	Grundzüge	4
5041	Ethik	4
5043	Erkenntnistheorie	3
5049	Maeutik	2
4052	Logik	4
5052	Wissenschaftstheorie	3
5216	Bioethik	2
5259	Der Wiener Kreis	2
5022	Glaube und Wissen	2
4630	Die 3 Kritiken	4

This will NOT work

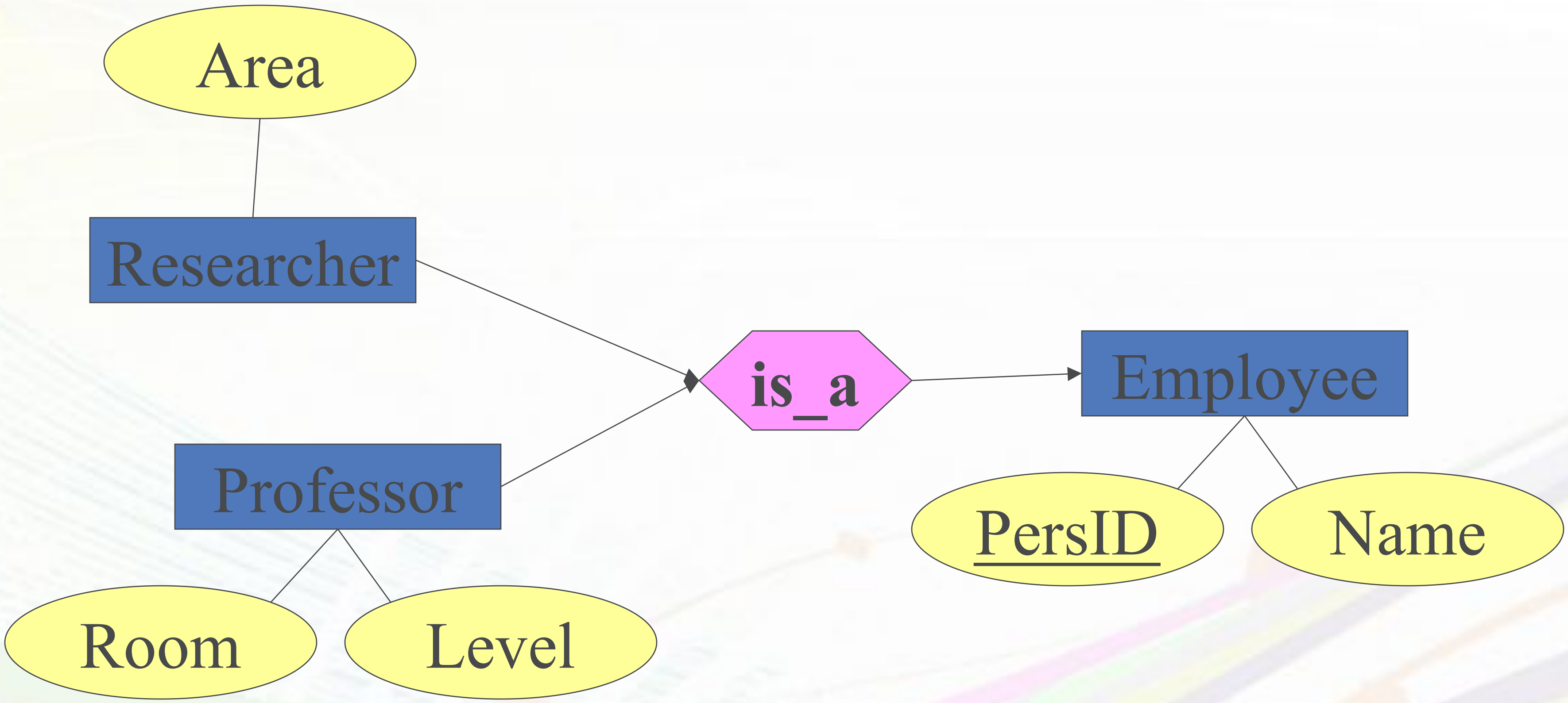
Professor				
PersID	Name	Level	Room	gives
2125	Sokrates	FP	226	5041
2125	Sokrates	FP	226	5049
2125	Sokrates	FP	226	4052
...	...	...	...	...
2134	Augustinus	AP	309	5022
2136	Curie	FP	36	??

Lecture		
LecID	Title	CP
5001	Grundzüge	4
5041	Ethik	4
5043	Erkenntnistheorie	3
5049	Mäeutik	2
4052	Logik	4
5052	Wissenschaftstheorie	3
5216	Bioethik	2
5259	Der Wiener Kreis	2
5022	Glaube und Wissen	2
4630	Die 3 Kritiken	4

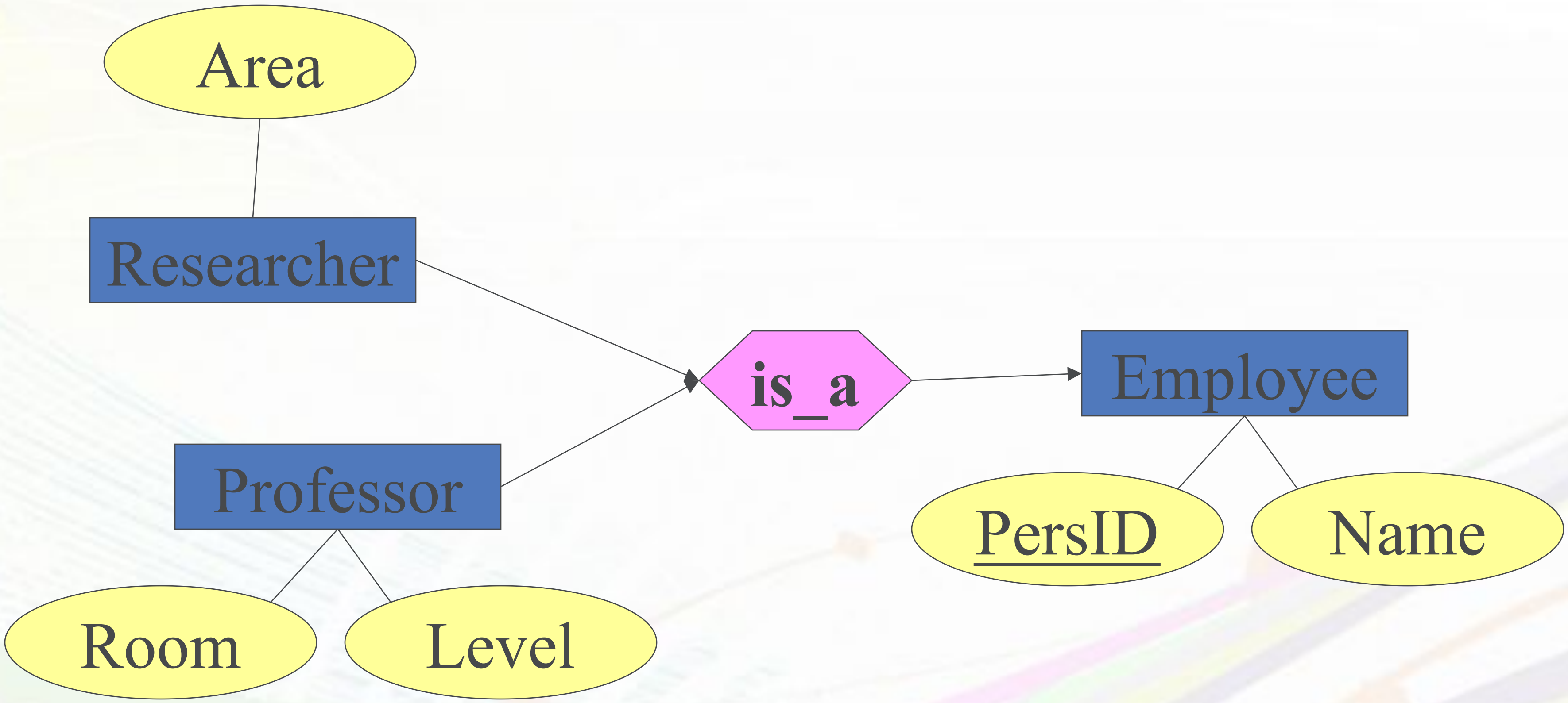
Problem: Redundancy and Anomalies
PersID is no longer key of Professor
PersID is not a key of gives

(issue will be revisited when we talk about normal forms)

Rule #4: Generalization



Rule #4: Generalization

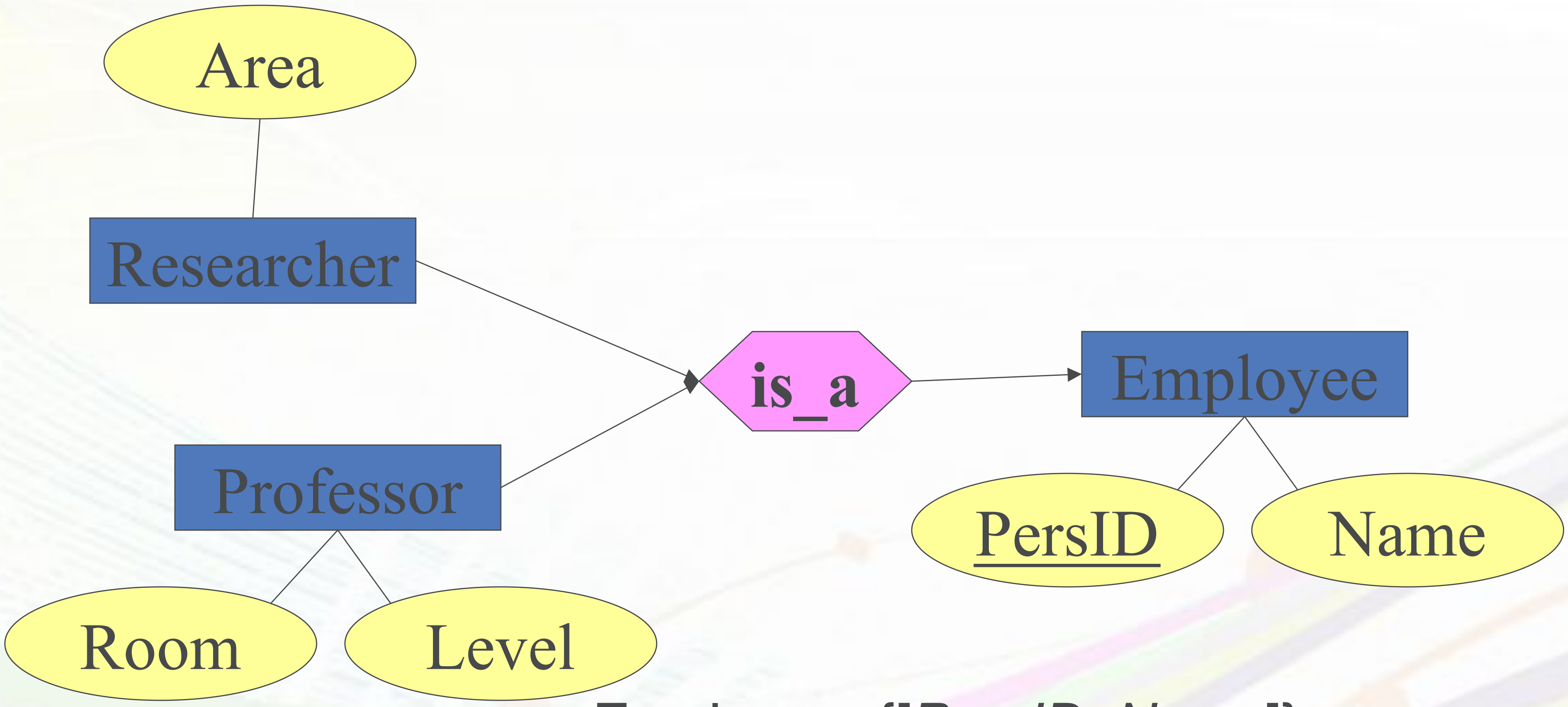


Employee: {[<u>*PersID*</u>, *Name*]}

Professor: {[<u>*PersID*</u>, *Level*, *Room*]}

Researcher: {[<u>*PersID*</u>, *Area*]}

Rule #4: Generalization-alternative



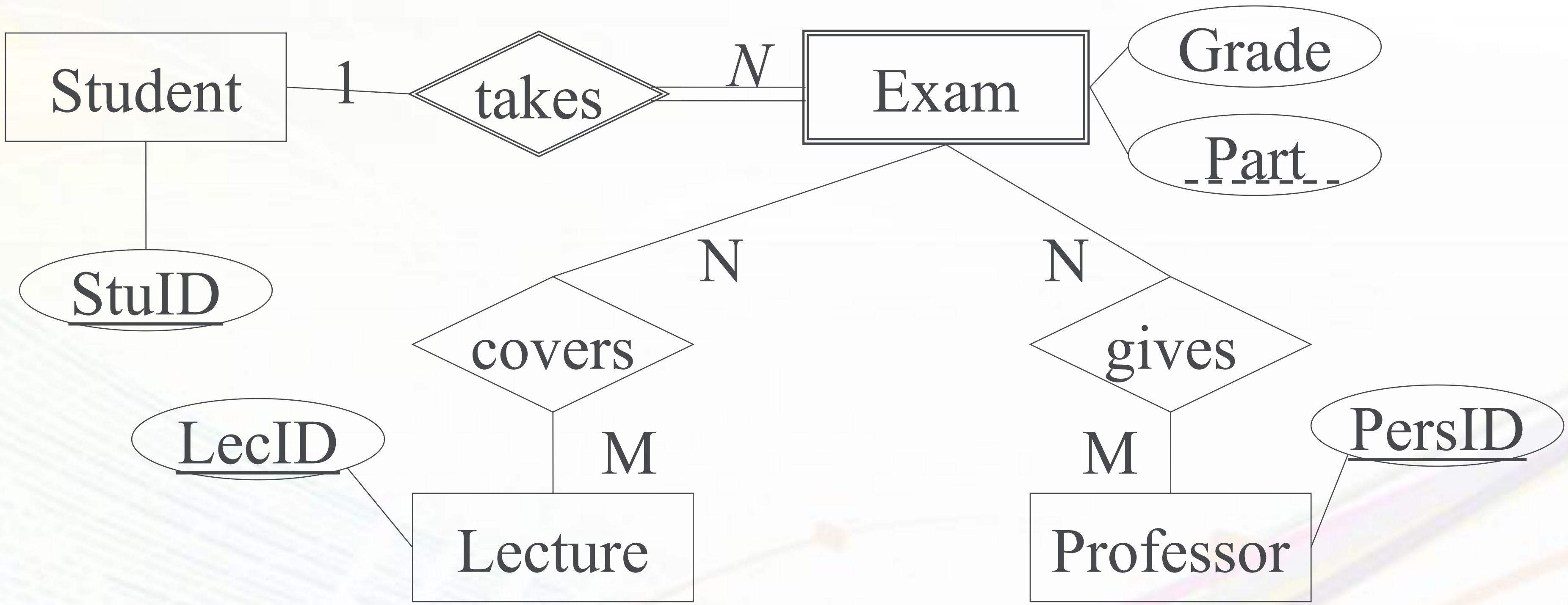
Employee: {[*PersID*, *Name*]}

Professor: {[*PersID*, *Name*, *Level*, *Room*]}

Researcher: {[*PersID*, *Name*, *Area*]}

What is better?

Rule #5: Weak Entities



Exam: {[StuID: integer, Part: string, Grade: integer]}

covers: {[StuID: integer, Part: string, LecID: integer]}

gives: {[StuID: integer, Part: string, PersID: integer]}

Weak Entities in detail: "takes"

- takes: {[StuID: int, ExamID: int, Part: string, LecID: int]}

- What is/are the key(s) of the "takes" relation?
- Why can it be merged with the "Exam" relation (Rule #3)?
- What happened to the "StuID" column as part of this merge?

Food for Thought: OO vs. Relations

- How do Java and C++ implement ER?

  Are they a better match than the relational model?

- Specifically, how do Java and C++ implement Generalization?

  Is it good or bad to have several possible ways?

- Concept of Reference: Compare Java and Relational Model

  Which one is better?

- Life-time of objects: Compare Java and Relational Model

  Why different?

Relational Model of Uni-DB

Professor

PersID	Name	Level	Room
2125	Sokrates	FP	226
2126	Russel	FP	232
2127	Kopernikus	AP	310
2133	Popper	AP	52
2134	Augustinus	AP	309
2136	Curie	FP	36
2137	Kant	FP	7

Student

StuID	Name	Semester
24002	Xenokrates	18
25403	Jonas	12
26120	Fichte	10
26830	Aristoxenos	8
27550	Schopenhauer	6
28106	Carnap	3
29120	Theophrastos	2
29555	Feuerbach	2

Lecture

LecID	Title	CP	PersID
5001	Grundzüge	4	2137
5041	Ethik	4	2125
5043	Erkenntnistheorie	3	2126
5049	Mäeutik	2	2125
4052	Logik	4	2125
5052	Wissenschaftstheorie	3	2126
5216	Bioethik	2	2126
5259	Der Wiener Kreis	2	2133
5022	Glaube und Wissen	2	2134
4630	Die 3 Kritiken	4	2137

requires

Prerequisite	Follow-up
5001	5041
5001	5043
5001	5049
5041	5216
5043	5052
5041	5052
5052	5259

attends

StuID	LecID
26120	5001
27550	5001
27550	4052
28106	5041
28106	5052
28106	5216
28106	5259
29120	5001
29120	5041
29120	5049
29555	5022
25403	5022

Researcher

PersID	Name	Area	Supervisor
3002	Platon	Ideenlehre	2125
3003	Aristoteles	Syllogistik	2125
3004	Wittgenstein	Sprachtheorie	2126
3005	Rhetikus	Planetenbewegung	2127
3006	Newton	Keplersche Gesetze	2127
3007	Spinoza	Gott und Natur	2126

tests

StuID	LecID	PersID	Grade
28106	5001	2126	1
25403	5041	2125	2
27550	4630	2137	2

Formal Definition of Relational Algebra

Atoms (basic expressions)

- Define a relation in the database
- A constant relation

Operators (composite expressions)

- Selection: $\sigma_p (E_1)$
- Projection: $\Pi_S (E_1)$
- Cartesian Product: $E_1 \times E_2$
- Rename: $\rho_V (E_1)$, $\rho_{A \leftarrow B} (E_1)$
- Union: $E_1 \cup E_2$
- Minus: $E_1 - E_2$

Relational Algebra

σ	Selection
π	Projection
X	Cartesian Product
$\bowtie$	Join
ρ	Rename
$-$	Set Minus
$\div$	Relational Division
$\cup$	Union
$\cap$	Intersection
$\ltimes$	Semi-Join (left)
$\rtimes$	Semi-Join (right)
⟕	left outer Join
⟖	right outer Join

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

Selection and Projection

Selection

$\sigma_{\text{Semester} > 10}$ (Student)

$\sigma_{\text{Semester} > 10}$ (Student)		
StuID	Name	Semester
24002	Xenokrates	18
25403	Jonas	12

Projection

Π_{Level}(Professor)

Π_{Rang}(Professor)
Level
FP
AP

Cartesian Product

L		
A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

x

R	
D	E
d_1	e_1
d_2	e_2

=

Result				
A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
a_1	b_1	c_1	d_2	e_2
a_2	b_2	c_2	d_1	e_1
a_2	b_2	c_2	d_2	e_2

Cartesian Product (ctd.)

Professor x attends

Professoren				attends	
PersID	Name	Level	Raum	StuID	LecID
2125	Sokrates	FP	226	26120	5001
...	...	...	...	...	...
2125	Sokrates	FP	226	29555	5001
...	...	...	...	...	...
2137	Kant	FP	7	29555	5001

Huge result set (n * m)

Usually only useful in combination with a selection (-> Join)

Natural Join

Two relations:

$R(A_1,\ldots, A_m, B_1,\ldots, B_k)$

$S(B_1,\ldots, B_k, C_1,\ldots, C_n)$

$$R \bowtie S = \Pi_{A1,\ldots, Am,\ R.B1,\ldots,\ R.Bk,\ C1,\ldots,\ Cn}(\sigma_{R.B1=S.B1\ \wedge\ldots\wedge\ R.Bk = S.Bk}(RxS))$$

$R \bowtie S$											
$R - S$				$R \cap S$				$S - R$			
A_1	A_2	…	A_m	B_1	B_2	…	B_k	C_1	C_2	…	C_n
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Three-way natural Join

$(\text{Student} \bowtie \text{attends}) \bowtie \text{Lecture}$

$(\text{Student} \bowtie \text{attends}) \bowtie \text{Lecture}$						
StuID	Name	Semester	LecID	Title	CP	PersID
26120	Fichte	10	5001	Professor	4	2137
27550	Jonas	12	5022	Researcher	2	2134
28106	Carnap	3	4052	Administrator	3	2126
...	...	...	...	...	...	...

Theta-Join

Two Relations:

R(A1, . . . , An)

S(B1, . . . , Bm)

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

$$R \bowtie_{\theta} S$$

$R \bowtie_{\theta} S$							
R				S			
A_1	A_2	…	A_n	B_1	B_2	…	B_m
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Theta Join Example

- Exercise, write an example

Join Variants

natural join

L		
A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

$\bowtie$

R		
C	D	E
c_1	d_1	e_1
c_3	d_2	e_2

=

Result				
A	B	C	D	E
a_1	b_1	c_1	d_1	e_1

left outer join

L		
A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

⟕

R		
C	D	E
c_1	d_1	e_1
c_3	d_2	e_2

=

Result				
A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
a_2	b_2	c_2	-	-

Join Variants

right outer join

L		
A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

$\rtimes$

R		
C	D	E
c_1	d_1	e_1
c_3	d_2	e_2

=

Resultat				
A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
-	-	c_3	d_2	e_2

Join Variants

(full) outer join

L		
A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

$⟗$

R		
C	D	E
c_1	d_1	e_1
c_3	d_2	e_2

=

Resultat				
A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
a_2	b_2	c_2	-	-
-	-	c_3	d_2	e_2

left semi join

L		
A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

$\ltimes$

R		
C	D	E
c_1	d_1	e_1
c_3	d_2	e_2

=

Resultat		
A	B	C
a_1	b_1	c_1

Join Variants

right semi join

L		
A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

$\rtimes$

R		
C	D	E
c_1	d_1	e_1
c_3	d_2	e_2

=

Resultat		
C	D	E
c_1	d_1	e_1

Rename Operator

Rename operator ρ

- Renaming of relation names
  - Needed to process self-joins and recursive relationships
  - E.g., two-level dependencies of lectures ("grandparents")

$$\Pi_{L1.Prerequisite}(\sigma_{L2.\ Follow\text{-}up=5216\ \wedge\ L1.Follow\text{-}up\ =\ L2.Prerequisite}$$

$$(\ \rho_{L1}\ (requires)\ x\ \ \rho_{L2}\ (requires)))$$

- Renaming of attribute names

$$\rho_{Requirement \leftarrow Prerequisite}\ (requires)$$

Intersection

$$\Pi_{PersID}(\text{Lecture}) \cap \Pi_{PersID}(\sigma_{Level=FP}(\text{Professor}))$$

- Only works if both relations have the same schema

  Same attribute names and attribute domains

- Intersection can be simulated with minus:

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

Relational Division

Find students who attended **all** lectures with 4CP.

$$\text{attends} \div \Pi_{\text{LecID}}(\sigma_{\text{CP=4}}(\text{Lecture}))$$

Definition of Division

- $t \in R \div S$, iff for each $ts \in S$ exists a $tr \in R$ such that:

$tr.S = ts.S$

$tr.$

R	
M	V
m_1	v_1
m_1	v_2
m_1	v_3
m_2	v_2
m_2	v_3

$\div$

S
V
v_1
v_2

$=$

$R \div S$
M
m_1

$$R \div S = \Pi_{(R-S)}(R) - \Pi_{(R-S)}((\Pi_{(R-S)}(R) \times S) - R)$$

Division: Example

$$R \div S = \Pi_{(R-S)}(R) - \Pi_{(R-S)}((\Pi_{(R-S)}(R) \times S) - R)$$

- R = attends; S = Lecture
- **Π_{StuID}(attends)**
  All students (who attend at least one lecture)
- **Π_{StuID}(attends) x Lecture**
  All students attend all lectures
- **(Π_{StuID}(attends) x Lecture) – attends**
  Lectures a student does not attend
- **Π_{StuID}((Π_{StuID}(attends) x Lecture) – attends):**
  Students who miss at least one lecture

What happens if there are no lectures or no attendance?

Codd`s Theorem

Impact of Codd`s theorem

- SQL is based on the relational calculus
- SQL implementation is based on relational algebra
- Codd`s theorem shows that SQL implementation is correct and complete.