Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

CHAI
Humanities-Centered AI

Ralf Möller

Databases

From ER Modeling to Relational Data Modeling

Summary and outlook

1. **Introduction**
   - Models and abstraction
   - Database systems
   - Applications and data independence
2. **Entity-relationship model (ER model)**
3. **Relational data model (RDM)**
   - From ER to RDM
4. **Relational design theory**
   - Relational algebra as a query language
   - Functional dependencies
   - Normal forms
5. **Structured Query Language (SQL)**
   - Data definition
   - Query specification
   - Data manipulation
6. **Query processing**
   - Architecture
   - Indexing
   - Query plans, optimization
7. **Transactions**
   - Transaction processing, schedules, locks
   - Recovery
8. **Other types of databases**
   - Vector databases
   - Data stream query languages

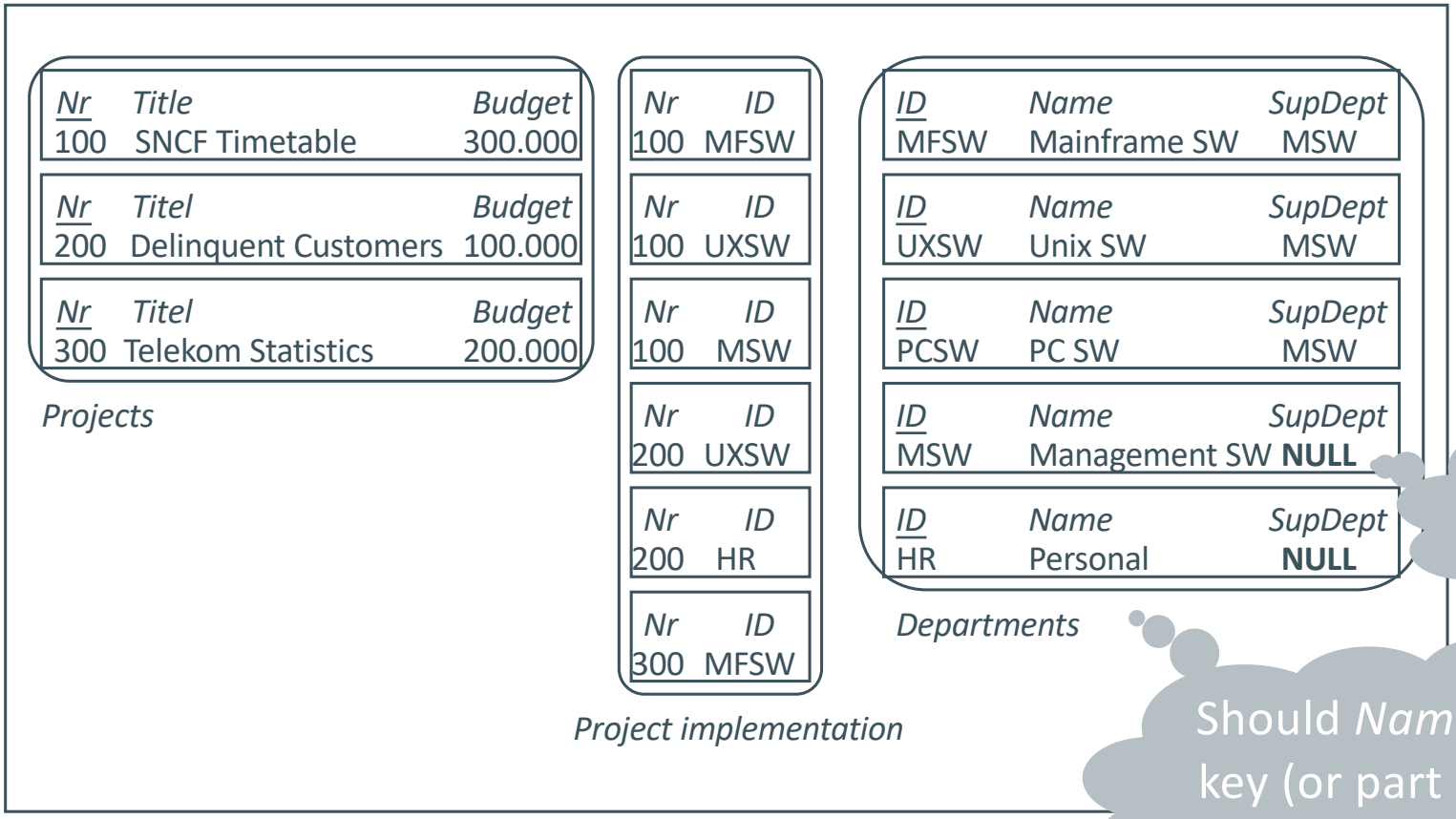
Relational Data Model (RDM)

- Mathematical data model
- Recap: A **relation** is a set of equally structured tuples
  - Structure of tuples defined by sets of attribute values D_i for $i \in \{1..k\}$ (called **domains**)
    $R \subseteq D_1 \times D_2 \times \cdots \times D_k$ (there can be a distinguished element in a domain: **NULL**)
  - k is called the **arity** of the relation (and the respective tuples: ***k*-tuples**)
  - Practical examples for domains D_i: Integers, Strings, Decimals, …
- Note that in ER we had relationship types $R \subseteq E_1 \times E_2 \times \cdots \times E_k$
  - In RDM there are no entities but just values (domains)
  - RDM does not distinguish between entities and relations, but offers just relations
- Identity of tuples defined by so-called key attributes (keys for brevity)
  - Keys are to be specified by modeler (key attributes are underlined in the following)

Relational Data Model: Database

- A **database** (DB) is a set of named relations
- The specification of all relation names of a DB together with all attribute domains per relation is called **relational schema** or **database schema**

RDM Example: Project database



Projects

Nr	*Title*	*Budget*
100	SNCF Timetable	300.000

Nr	*Titel*	*Budget*
200	Delinquent Customers	100.000

Nr	*Titel*	*Budget*
300	Telekom Statistics	200.000

Project implementation

Nr	*ID*
100	MFSW
100	UXSW
100	MSW
200	UXSW
200	HR
300	MFSW

Departments

ID	*Name*	*SupDept*
MFSW	Mainframe SW	MSW
UXSW	Unix SW	MSW
PCSW	PC SW	MSW
MSW	Management SW	**NULL**
HR	Personal	**NULL**

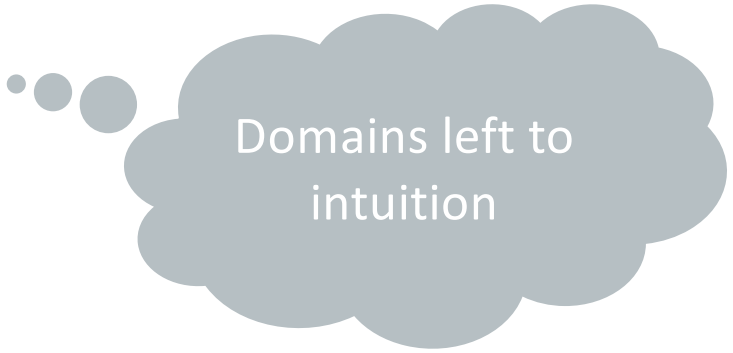
No super department

Should *Name* be a key (or part of the key)?

Project database

Relational schema: Notation

- Project: {[Nr, Title, Budget]}
- Project implementation: {[Nr, ID]}
- Departments: {[ID, Name, SupDept]}



Notation for relational schemas

- Schema specification: **Table name = {[Attr1: Type1, Attr2: Type2, ...]}**
- Square brackets [...] indicate how tuples are structured.
  Curly brackets are used to express that a relation instance is a set of tuples.
- Type specifications are optional
- Sometimes the attributes of a relation R are just needed as a set:
  - We write for the schema of table R: **R = {Attr1, Attr2, ...}**

Relational schema with types

- Project: {[Nr:integer, Title:string, Budget:integer]}
- Project implementation: {[Nr:integer, ID:integer]}
- Departments: {[ID:integer, Name:string, SupDept:string]}

RDM: Keys

- A subset of the attributes of a relation (table) can be specified as the **key**
  - Keys uniquely identify tuples in a relation, i.e.,
    there a no duplicates with reference to key attributes in a relation
  - This key is called **primary key**
- A key “is not required”
- What does that mean?
  - Then, all attributes are keys (a relation is set!)
  - Later we will explore
    how primary keys are systematically found
- Sometimes an artificial key is defined

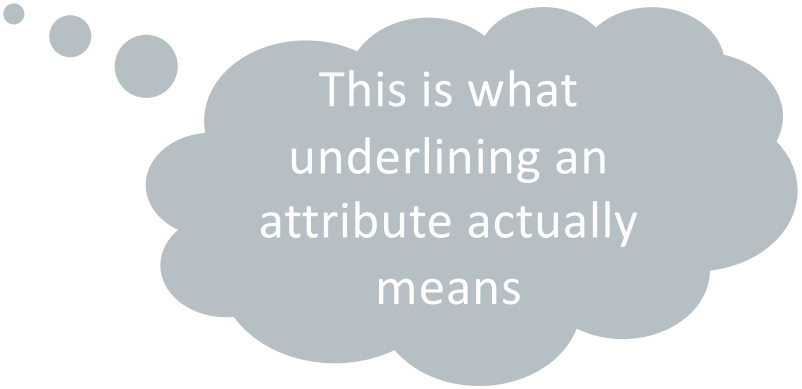
Rack	Type	Year	N_bottles
2	Franken	2009	5
1	Baden	2006	3
4	Rheinhessen	2007	10
1	Mosel	2013	2
2	Franken	2010	10

What is the meaning of keys?

The primary key of a relation defines a function from the primary key values to the values of the non-key values.

Notation:

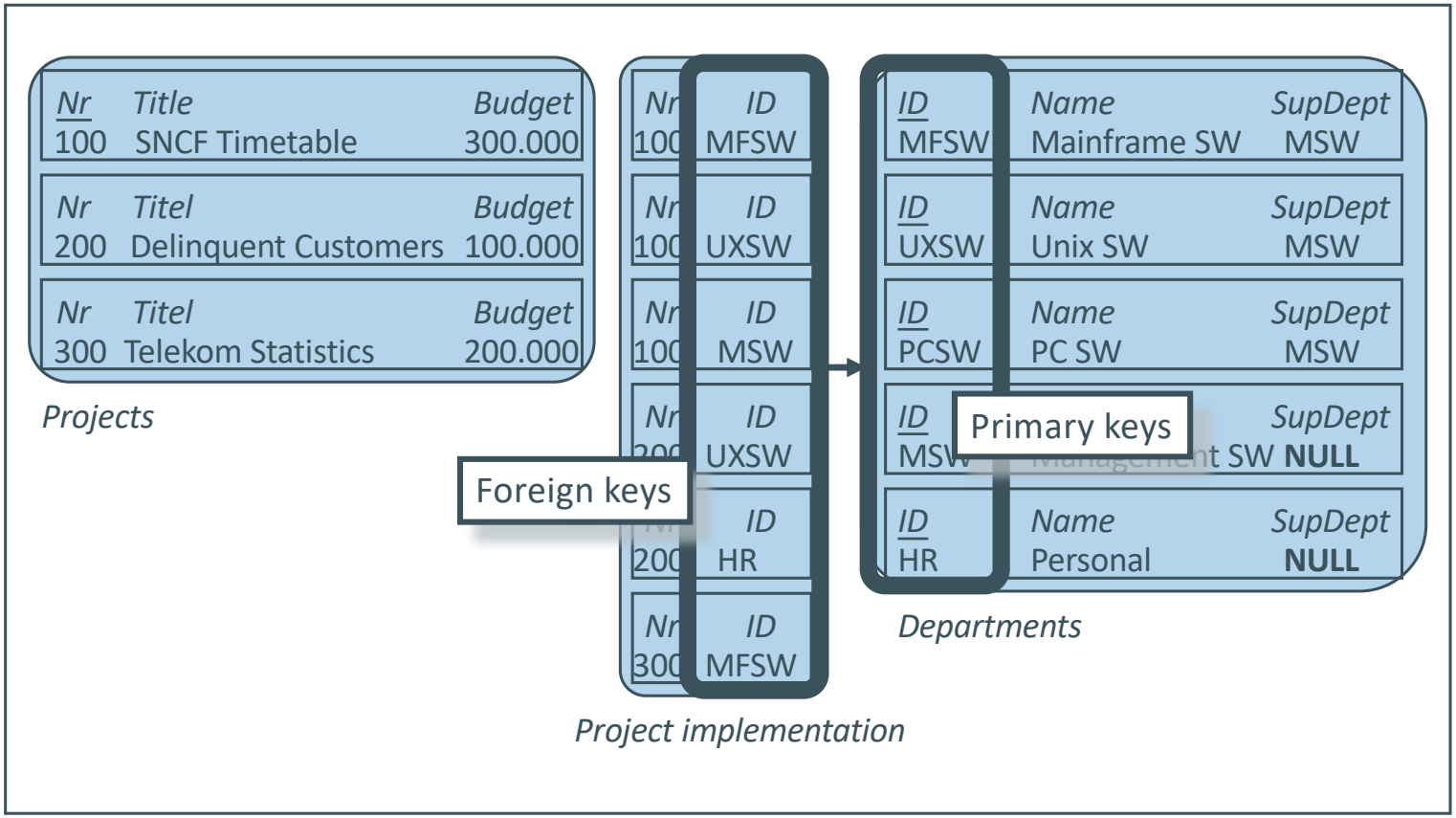
$ID \longrightarrow Name,$
$ID \longrightarrow SupDept$



Can we also write $ID \longrightarrow (Name,\ SupDept)$?

Abbreviation: $ID \longrightarrow Name,\ SupDept$

RDM: Keys as references



Project database

RDM Implementation

- Structured Query Language (SQL)
  - Data definition language
  - Data management language
  - Data query language

Example for data definitions in SQL

Relational schema of Project database in SQL

```
create table Projects
( Nr integer not null,
  Title char(30) not null,
  Budget decimal(10,2) not null,
  primary key(Nr) );
```

```
create table Departments
( ID char(4) not null,
  Name char(30) not null,
  SupDept char(4),
  primary key(ID) ),
```

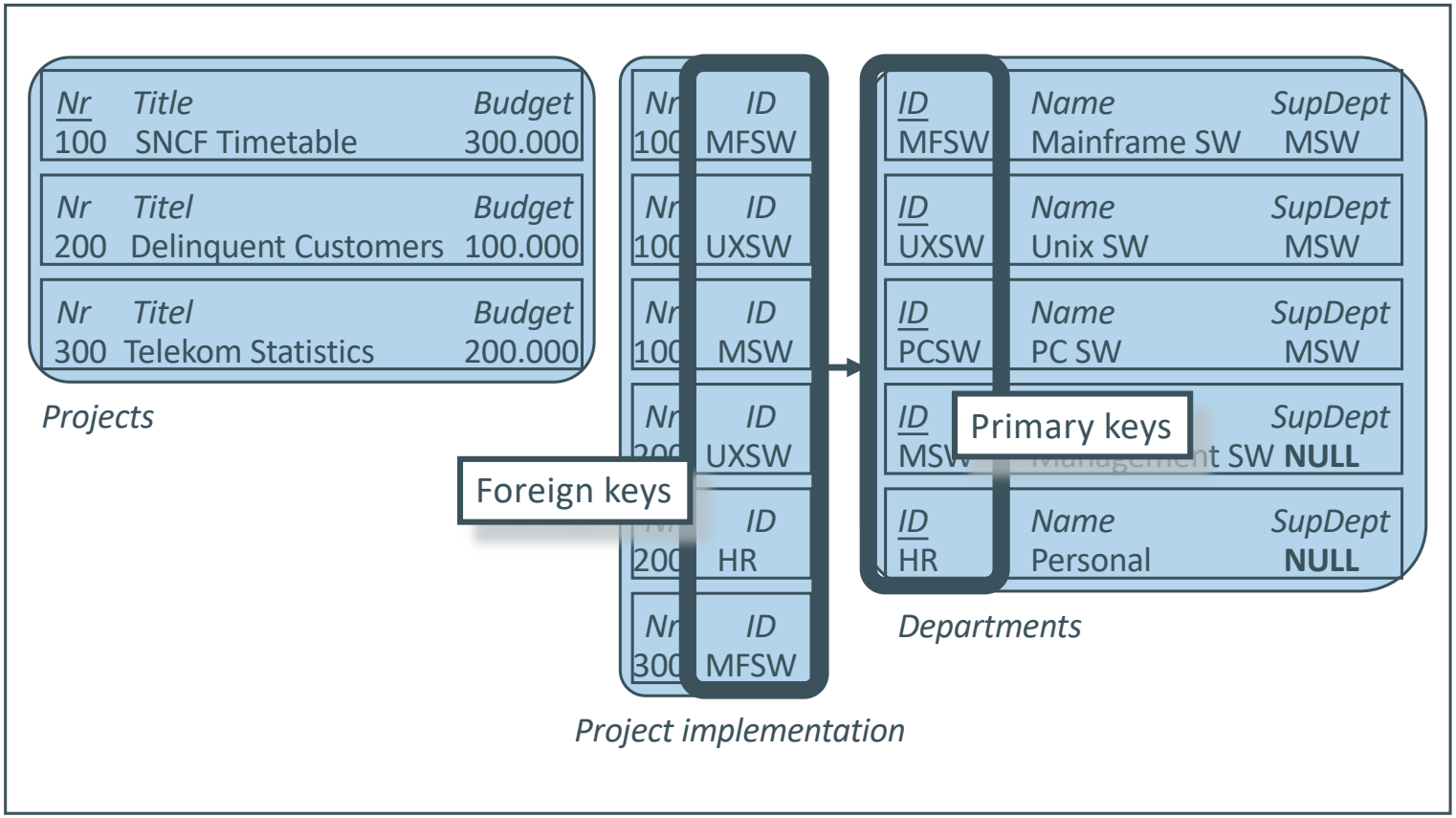
```
create table
Project implementation
( Nr integer not null,
  ID char(4) not null,
  primary key (Nr, Kurz),
  foreign key (ID) references Departments(ID),
  foreign key (Nr) references Projects(Nr) );
```

Databases in SQL

- A **set of named tables** is called **database**
- Relations are displayed as **tables**
  - with rows for **tuples** and
  - column names for **attributes**
- Number of rows is called **cardinality of the table**
- Pragmatics of SQL
  - Rows can be added and removed “easily”
  - Attributes (columns) usually not changed (although this is possible in principle)
  - Duplicate tuples are possible in SQL tables

Distinction between the mathematical relational model and SQL as a specific implementation is often blurred

RDM: Referenced tuples must exist



Project database

Data management: Referential integrity checking

For every value of a foreign key attribute to a table T,
a respective tuple with that value must exist in destination table T.

When is this checked?

- Every time a new tuple is inserted into a table with a foreign key:
  The object referred to must exist in the destination table
- Whenever a tuple in the destination table is deleted:
  Tuples referred to cannot be deleted,
  or deletion must be cascaded to tuple in referencing table
- User-defined modification context (called transaction) such that at the end all checks are performed

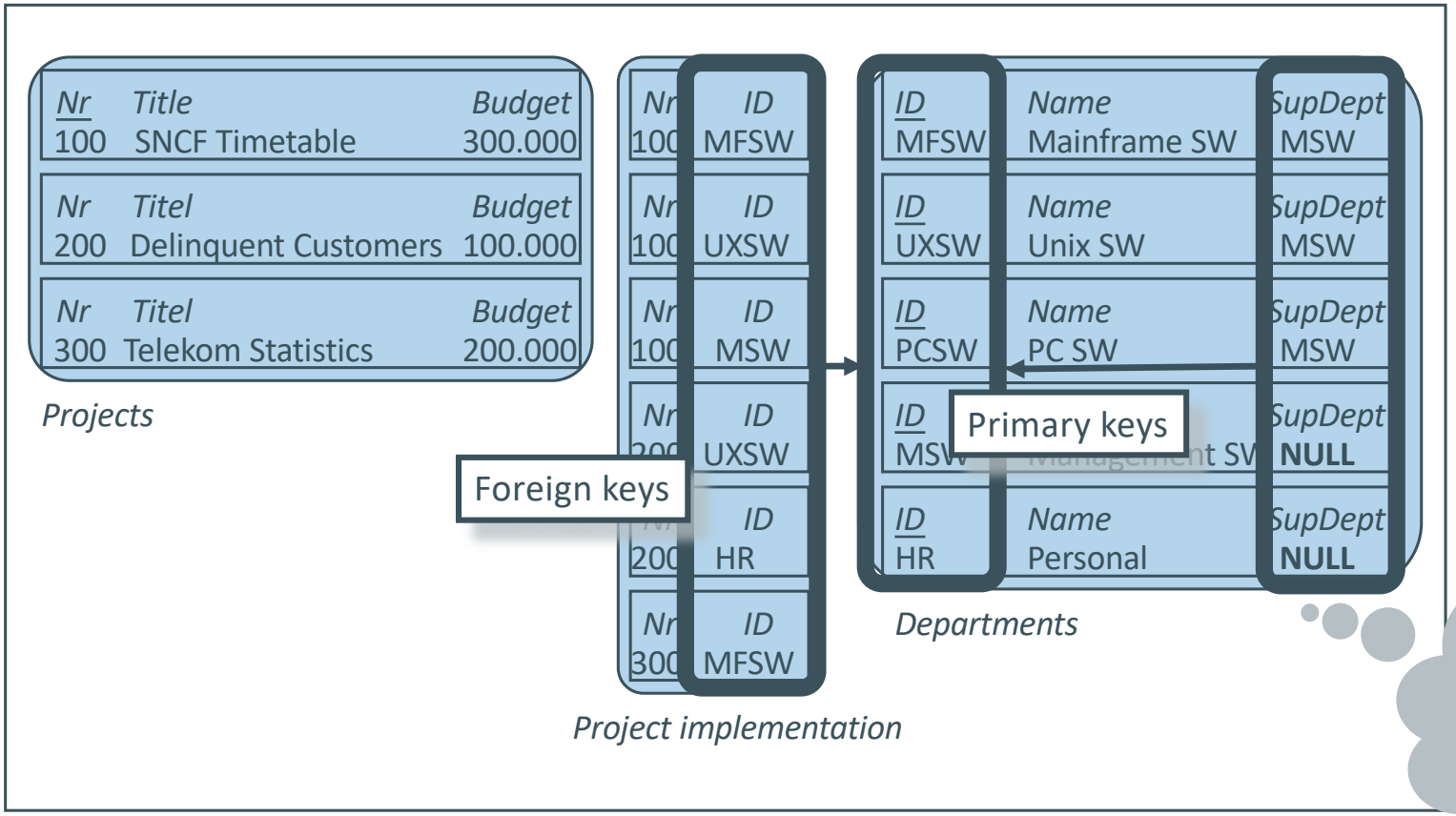
Referential integrity: Foreign keys

In general, a foreign key consists of a list of attributes of a table `T`, such that there exists a type-compatible list of attributes in `S`:

```
create table S
  (...
   foreign key S(A1, A2, ..., An) references T(B1, B2, ..., Bn) )
```

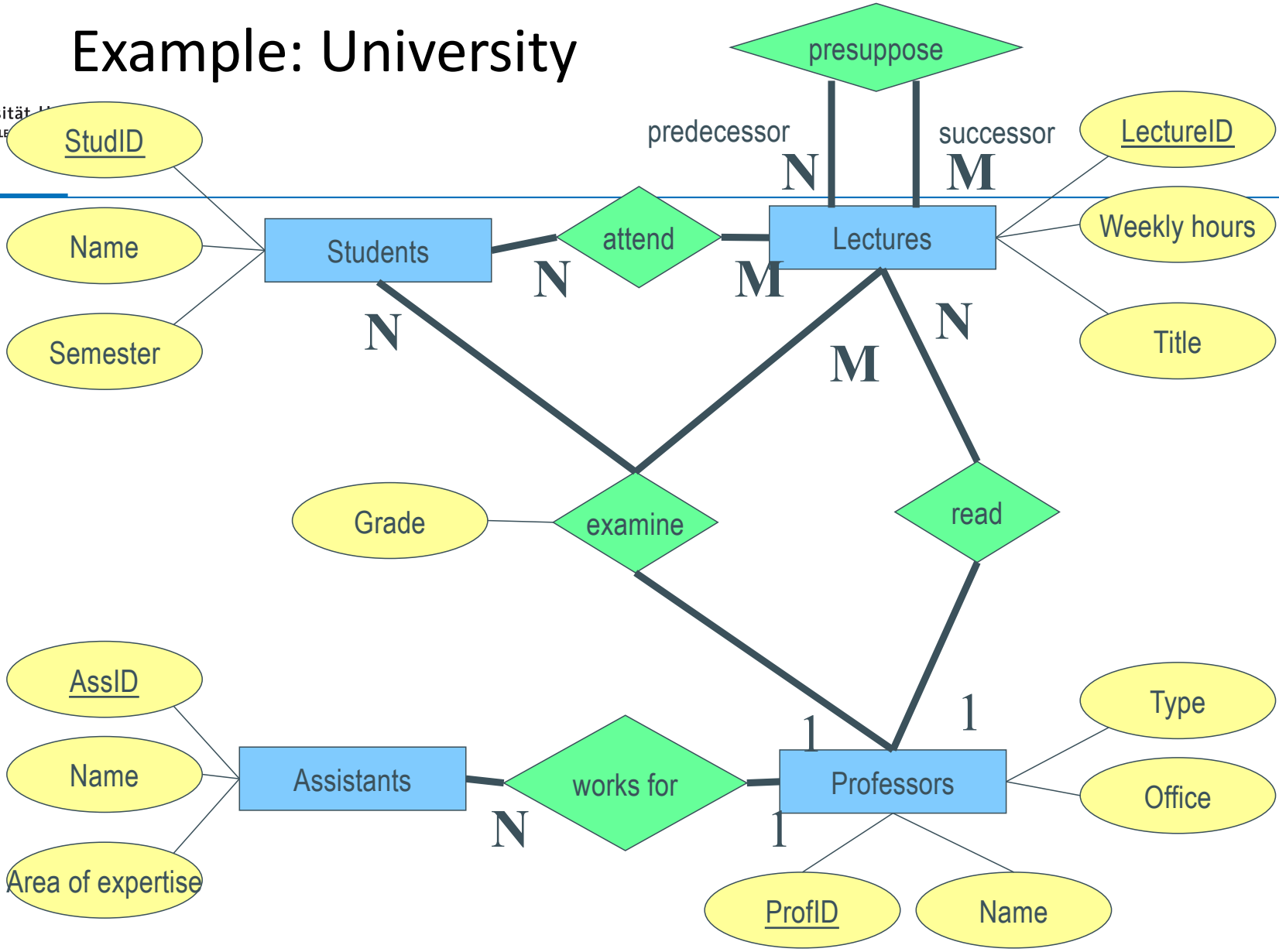
Recursive foreign keys (`S` = `T`) are indeed possible (reflexive keys).

RDM: Keys as references



Project database

Example: University



Relational representation of entity types

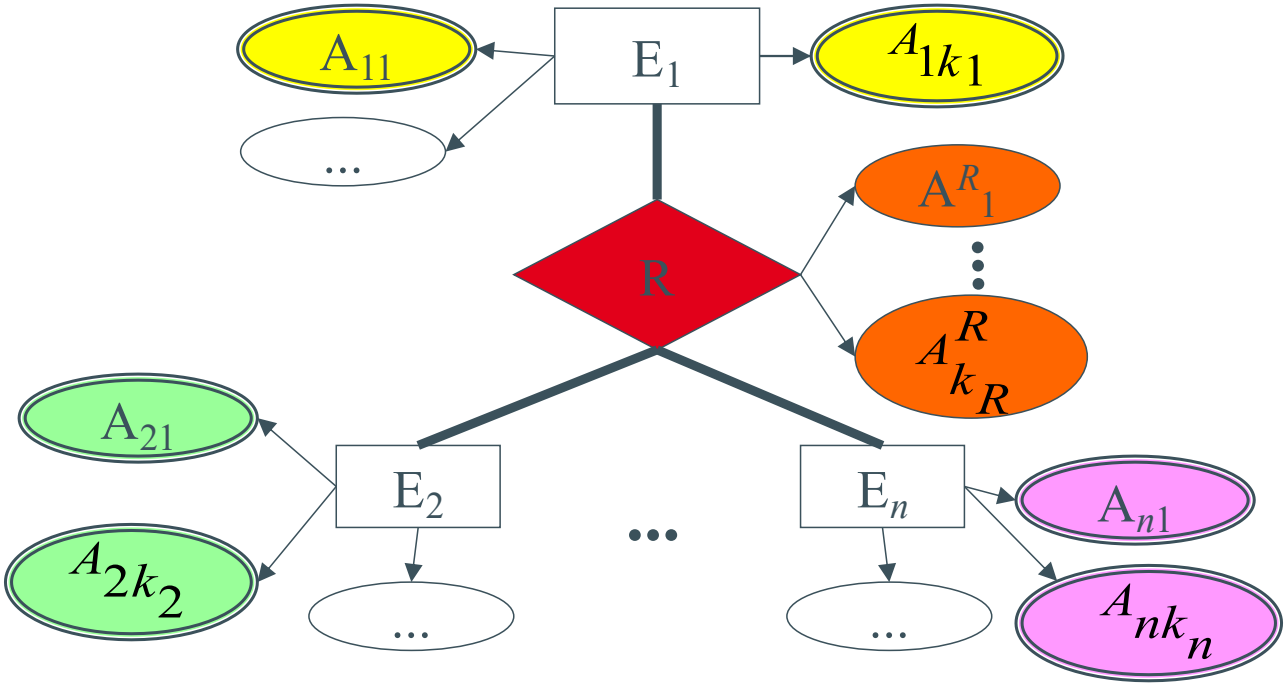
Students: {[StudID:integer, *Name*: *string, Semester*: *integer*]}

Lectures: {[LectureID:integer, *Title*: *string, Weekly hours*: *integer*]}

Professors: {[ProfID:integer, *Name*: *string, Type*: *string, Office*: *integer*]}

Assistants: {[AssID:integer, *Name*: *string, Area of expertise*: *string*]}

Relational representation of relationship types



$$R: \{[\underbrace{A_{11},\ldots,A_{1k_1}}_{\text{Keys of } E_1}, \underbrace{A_{21},\ldots,A_{2k_2}}_{\text{Keys of } E_2}, \ldots, \underbrace{A_{n1},\ldots,A_{nk_n}}_{\text{Keys of } E_k}, \underbrace{A_1^R,\ldots,A_{k_R}^R}_{\text{Attributes of } R}]\}$$

Relational representation of relationship types

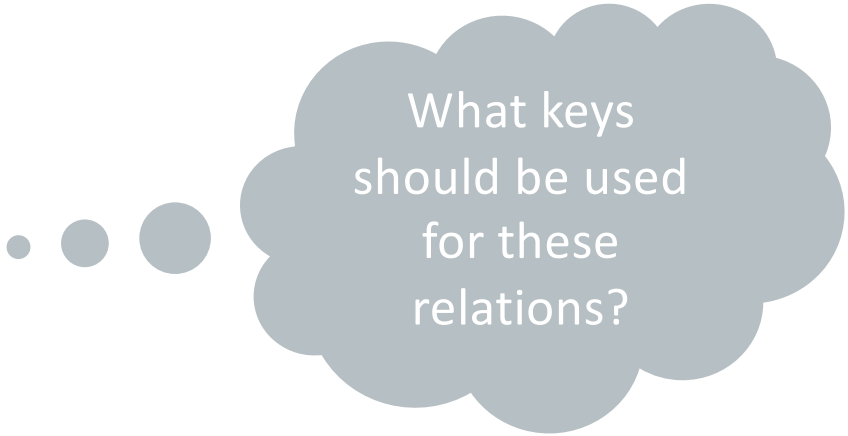
attend : {[StudID: integer, LectureID: integer]}

read : {[ProfID: integer, LectureID: integer]}

works for : {[AssID: integer, *ProfID: integer*]}

presuppose : {[predecessor: integer, successor: integer]}

examine : {[StudID: integer, LectureID: integer, ProfID: integer, Grade: decimal]}

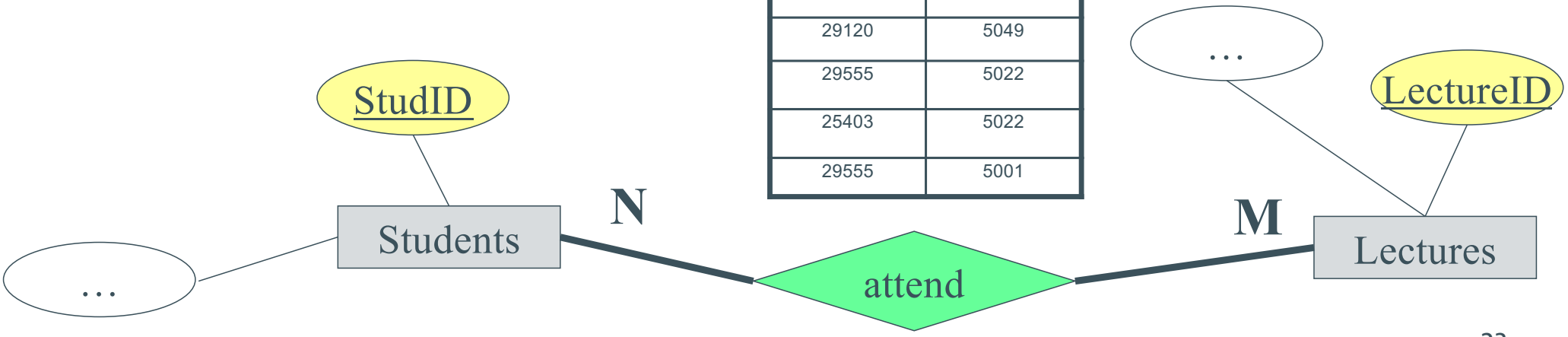


Example: Relation *attend*

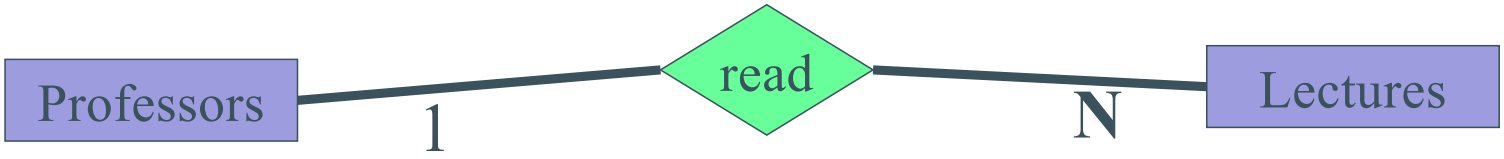
Students	
StudID	...
26120	...
27550	...
...	...

attend	
StudID	LectureID
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

Lectures	
LectureID	...
5001	...
4052	...
...	...



Refinement of a derived relational schema



Read is a 1:N-Relation

Initial draft

Lectures : {[*LectureID, Title, SWS*]}

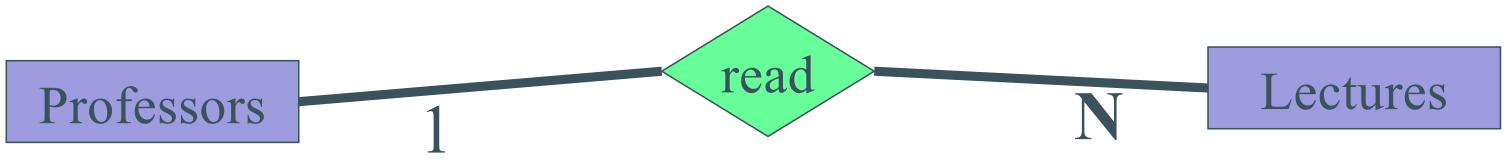
Professors : {[*ProfID, Name, Rang, Raum*]}

read: {[*LectureID, ProfID*]}

Example of relations *Professors and Lectures*

Professors			
ProfID	Name	Type	Office
2125	Sokrates	C4	226
2126	Russel	C4	232
2127	Copernicus	C3	310
2133	Popper	C3	52
2134	Augustinus	C3	309
2136	Curie	C4	36
2137	Kant	C4	7

Lectures			
LectureID	Title	Weekly hours	readBy
5001	Foundations	4	2137
5041	Ethics	4	2125
5043	Epistemology	3	2126
5049	Socratic Methods	2	2125
4052	Logic	4	2125
5052	Philosophy of Science	3	2126
5216	Bioethics	2	2126
5259	Vienna Circle	2	2133
5022	Belief and Knowledge	2	2134
4630	Kant: 3 Critics	4	2137



Refinement of a derived relational schema

1:N-Relation

Initial draft

Lectures : {[*LectureID, Title, SWS*]}
Professors : {[*PersNr, Name, Type, Office*]}
read: {[*LectureID, ProfID*]}

Refinement by foreign key (combine *Professors* and *read*)

Lectures : {[*LectureID, Title, SWS, readBy*]} with *readBy* being a foreign key to *Professors*
Professors : {[*ProfID Name, Type, Office*]}

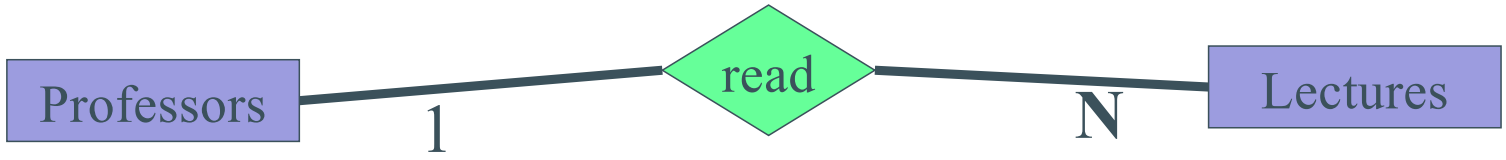
More general rule:

Relations with the same keys can be combined with foreign keys. But only those!

Caution: It does not work the other way

Professors				
ProfID	Name	Type	Office	reads
2125	Socrates	C4	226	5041
2125	Socrates	C4	226	5049
2125	Socrates	C4	226	4052
...	...	...	...	...
2134	Augustinus	C3	309	5022
2136	Curie	C4	36	??

Lectures		
LectureID	Title	Weekly hours
5001	Foundations	4
5041	Ethics	4
5043	Epistemology	3
5049	Socratic Methods	2
4052	Logic	4
5052	Philosophy of Science	3
5216	Bioethics	2
5259	Vienna Circle	2
5022	Belief and Knowledge	2
4630	Kant: 3 Critics	4



Anomalies

Professors				
ProfID	Name	Type	Office	reads
2125	Socrates	C4	226	5041
2125	Socrates	C4	226	5049
2125	Socrates	C4	226	4052
...	...	...	...	...
2134	Augustinus	C3	309	5022
2136	Curie	C4	36	??

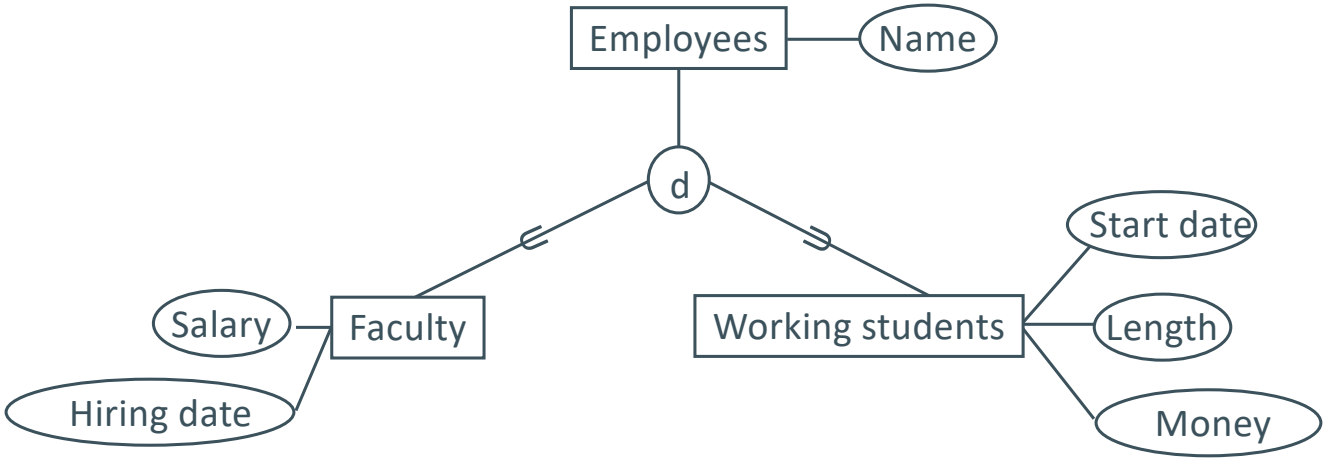
Lectures		
LectureID	Title	Weekly hours
5001	Foundations	4
5041	Ethics	4
5043	Epistemology	3
5049	Socratic Methods	2
4052	Logic	4
5052	Philosophy of Science	3
5216	Bioethics	2
5259	Vienna Circle	2
5022	Belief and Knowledge	2
4630	Kant: 3 Critics	4

Update anmaly: What happens if Socrates moves to another office?

Deletion anomaly: What happens, if Belief and Knowledge is cancelled?

Insertion anomaly: Curie could be new to the faculty and might not yet offer lectures

What about generalization?



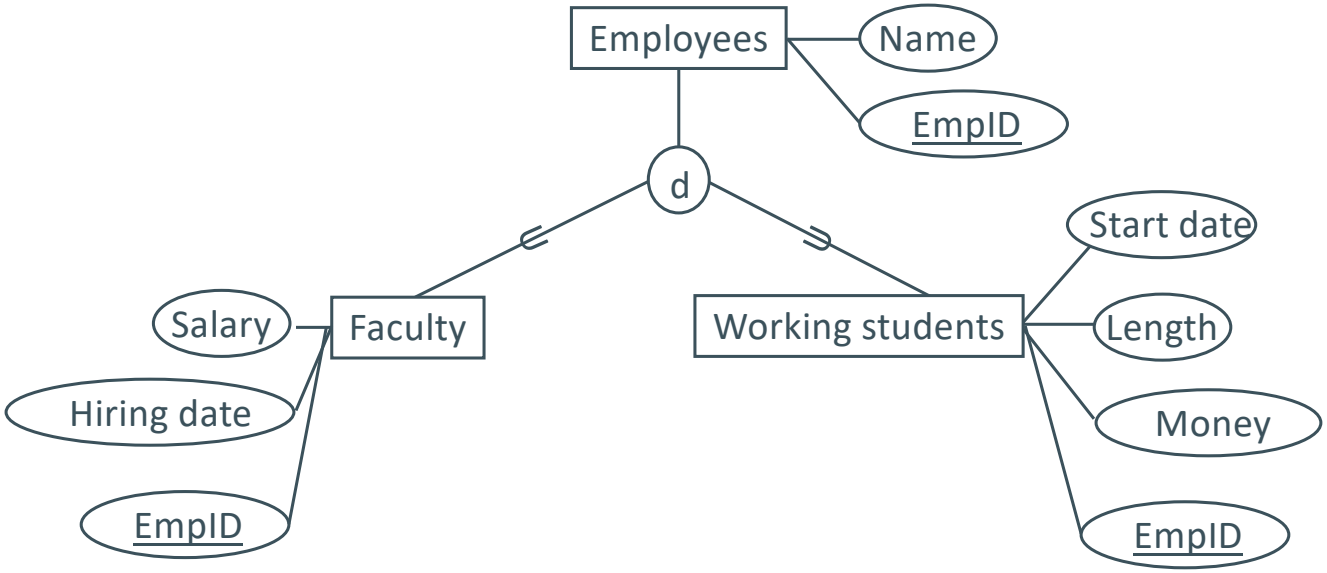
Employees: {[*Name*]}

Faculty: {[*Hiring date, Salary*]}

Working students:
{[*Start date, Length, Money*]}

Useful relational schema?

What about generalization?



Employees: {[*EmpID*, *Name*]}

Faculty: {[*EmpID*, *Hiring date*, *Salary*]}

Working students:
{[*EmpID*, *Start date*, *Length*, *Money*]}

For *Faculty* and *Working students*, *EmpID* is a foreign key to Employees

Agreement on notation

Let $\mathcal{R} = \{A, B, C, D\}$ be a relational schema.

Assume that r and t are tuples of a concrete relation adhering to schema $\mathcal{R}$.

Furthermore $a \subseteq \mathcal{R}$.

We agree on:

$r.a = t.a$ means that for all attributes a from a it holds that $r.a = t.a$,
where $tuple.attribute$ denotes the respective attribute component of a tuple

Functional dependencies

Schema

$\mathcal{R} = \{A, B, C, D\}$

Instance R

R			
A	B	C	D
a4	b2	c4	d3
a1	b1	c1	d1
a1	b1	c1	d2
a2	b2	c3	d2
a3	b2	c4	d3

Let $\alpha \subseteq \mathcal{R}$, $\beta \subseteq \mathcal{R}$

$\alpha \longrightarrow \beta$ if and only if
for all $r, s \in R$ it holds that
if $r.\alpha = s.\alpha$ then $r.\beta = s.\beta$

For R it holds that

$\{A\} \longrightarrow \{B\}$

$\{C, D\} \longrightarrow \{B\}$

Not: $\{B\} \longrightarrow \{C\}$

Notational convention: drop braces

$C,D \longrightarrow B$

and possibly commas

$CD \longrightarrow B$

Examples

Pedigree				
Child	Father	Mother	Grandpa	Grandma
Sophie	Alfons	Sabine	Lothar	Linde
Sophie	Alfons	Sabine	Hubert	Lisa
Niclas	Alfons	Sabine	Lothar	Linde
Niclas	Alfons	Sabine	Hubert	Lisa
...	...	...	Lothar	Martha
…	…	…	…	…

Child ⟶ Father, Mother

Child, Grandpa ⟶ Grandma

Child, Grandma ⟶ Grandpa

Summary and outlook

1. Introduction
   - Models and abstraction
   - Database systems
   - Applications and data independence
2. Entity-relationship model (ER model)
3. **Relational data model (RDM)**
   - From ER to RDM
4. **Relational design theory**
   - Relational algebra as a query language
   - Functional dependencies
   - Normal forms
5. **Structured Query Language (SQL)**
   - Data definition
   - Query specification
   - Data manipulation
6. **Query processing**
   - Architecture
   - Indexing
   - Query plans, optimization
7. **Transactions**
   - Transaction processing, schedules, locks
   - Recovery
8. **Other types of databases**
   - Vector databases
   - Data stream query languages