Databases

03 Relational Databases and Normalization

Dr. Lucas Iacono

Graz University of Technology, Austria

Computer Science and Biomedical Engineering
Institute of Human-Centred AI

Course calendar

- Part 1: Data Modeling and relational databases
  - (1) Mon. 20.10.2025: Overview of databases and data storage systems + Group Project
  - (2) Mon. 27.10.2025: Data modeling + Exercise 1
  - **(3) Mon. 03.11.2025: Relational databases and normalization (+ bit of SQL for Ex. 1)**
  - (4) Mon. 10.11.2025: Query languages (SQL) + Exercise 2 presentation

Last week

- **Database Design**
- **Entity relationship modeling (ER) and diagrams**
  - 1) Entities, 2) relationships, 3) attributes
  - Multiplicities (e.g., M-M) → Modified Chen Notation (incoming connection)

Last week

- Database Design
- Entity relationship modeling (ER) and diagrams
  - 1) Entities, 2) relationships, 3) attributes
  - Multiplicities (e.g., M-M) --> Modified Chen Notation (incoming connection)

- Group project concept
  - Any questions so far?
  - Form groups of 5 in TeachCenter! (so also merge smaller groups)

This week: Agenda

- Relational Data Model
- Database Normalization

+ a bit of SQL for Exercise 1

DB Design Lifecycle Phases

#1 Requirements engineering

Collect and analyze data and application requirements

Specification documents

DB Design Lifecycle Phases

#1 Requirements engineering

Collect and analyze data and application requirements

Specification documents

#2 Conceptual Design (last lecture, exercise 1 + group project)

Model data semantics and structure, independent of logical data model

ER model / diagram

DB Design Lifecycle Phases

#1 Requirements engineering

Collect and analyze data and application requirements

Specification documents

#2 Conceptual Design (last lecture, exercise 1 + group project)

Model data semantics and structure, independent of logical data model

ER model / diagram

#3 Logical Design (this lecture, exercise 1 + group project)

Model data with **implementation primitives** of **concrete data model**

e.g., relational schema + integrity constraints, views, permissions, etc

Relational Data Model

Recap: Relational databases (1970/80s – now)

- Organize a body of data into simple tables of related information
- Easy to access, merge and change data

Recap: Relational databases (1970/80s – now)

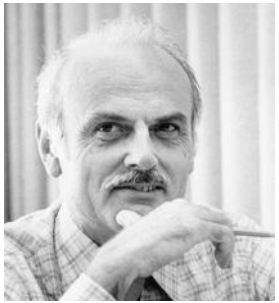
- Organize a body of data into simple tables of related information
- Easy to access, merge and change data

- Independence of **data scheme** and physical storage
- SQL (structured query language)

Recap: Relational databases (1970/80s – now)

- Organize a body of data into simple tables of related information
- Easy to access, merge and change data

- Independence of data scheme and physical storage
- SQL (structured query language)

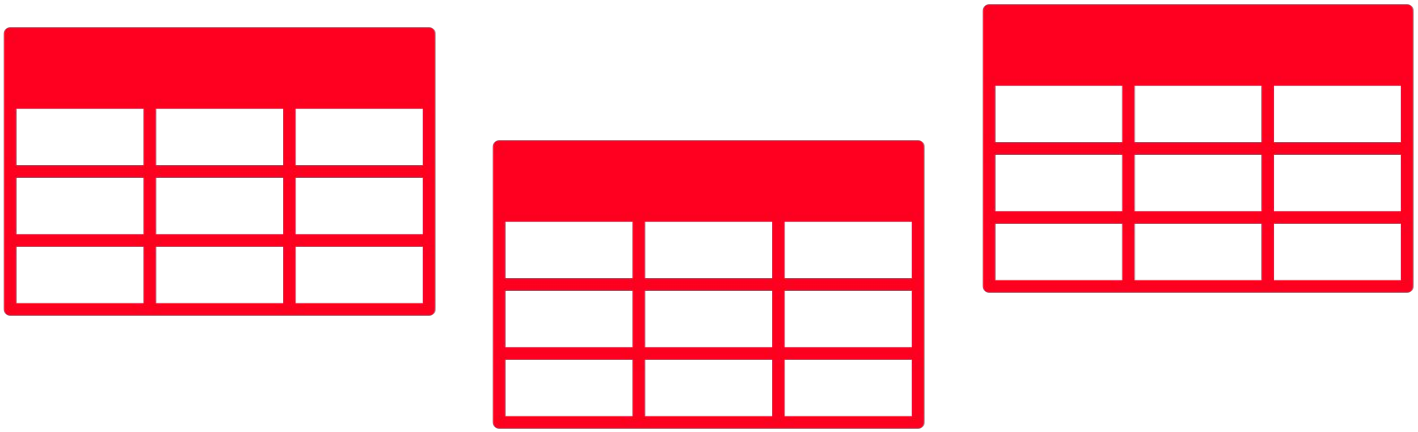


Edgar F. “Ted” Codd @ IBM Research (Turing Award ‘81)

[E. F. Codd: A Relational Model of Data for Large Shared Data Banks. Comm. ACM 13(6), **1970**]

Relational model

- represent data as a **collection of tables**



Relation

- represents a **table**

(precisely a **part of a table** but we use it as table)

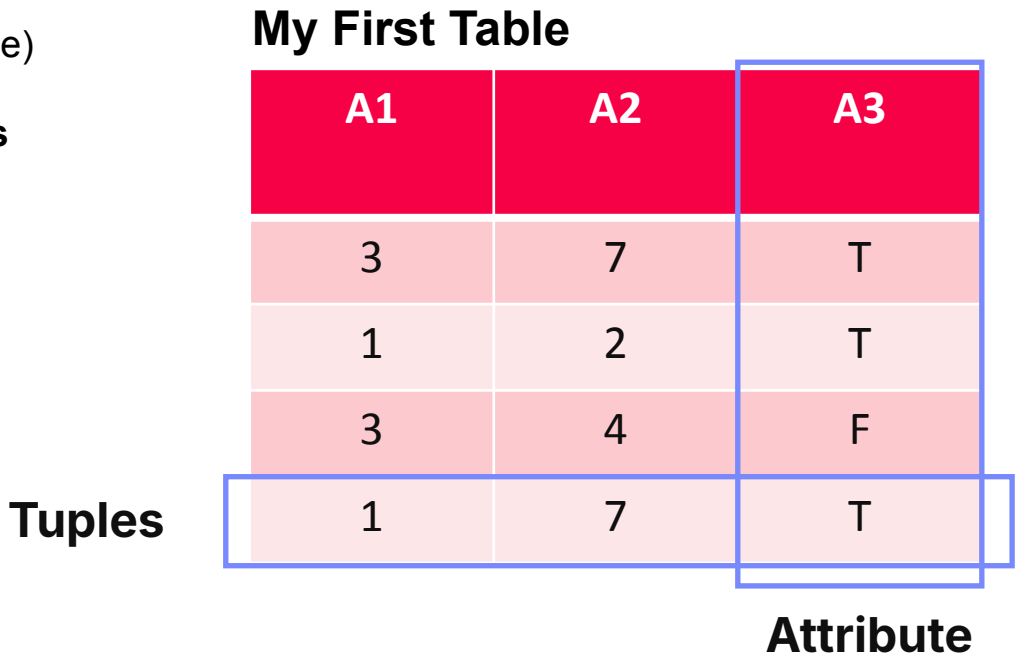
A1	A2	A3
3	7	T
1	2	T
3	4	F
1	7	T

Relation

- represents a table

(precisely a part of a table but we use it as table)

- **attributes** represent the **columns headers**
- **tuples** represent the **rows**



Relation

- represents a **table**

(precisely a part of a table but we use it as table)

- **attributes** represent the **columns headers**
- **tuples** represent the **rows**

- **Relation schema RS:**
  Set of k attributes **{A_1,…,A_k}**
- Attribute **A_j** is based on **value domain D_j**

My First Table

A1	A2	A3
3	7	T
1	2	T
3	4	F
1	7	T

Tuples

Attribute

Domain

- Domain **D (value domain)** represents a **data type**
- Also defines the **set of values allowed for an attribute**

Data types (in PostgreSQL)

- **Numeric types**
  - **Integer**
    - 4 bytes (-2,147,483,648 to +2,147,483,647) --> for whole numbers
    - smallint, bigint are also possible if needed
  - **Serial**
    - 4 bytes --> Integer with auto-increment --> good for keys like Ids
  - **Real**
    - 4 bytes --> real values with 6 decimal digits precision
  - **Double precision**
    - 8 bytes --> real values with 15 decimal digits precision
  - **Numeric*(precision, scale)***
    - 24.345 has precision of 5 (24.345) and scale of 3 (345) --> for real values
    - If no parameter are provided, then size is variable --> Numeric

Important! No quotes needed to store any numeric type

Data types (in PostgreSQL)

- **Character types (strings)**
  - **Char*(n)***
    - For storing strings of max length of *n* (e.g., Char(5) allows `hello`)
    - Longer strings will result in an error
    - Shorter strings will be padded with with spaces to reach *n*
    - Char without parameter means Char*(1)*
  - **Varchar*(n)***
    - Same as Char*(n)* but …
    - Shorter strings will not be padded with spaces (they are just shorter)
    - Varchar without parameter means a string of any size
  - **Text**
    - String of any length
    - So same as Varchar without parameter (at least in PostgresSQL)

  Important: Single quotes needed to store any character type

Data types (in PostgreSQL)

- **Date/time types**
  - **Date**
    - **Input is flexible, e.g.,** 1999-01-08 **or** 1999-Jan-08 **is possible [yyyy-MM-dd]**
  - **Time**
    - **For storing times, e.g.,** 04:05:06 **[hh:mm:ss]**
    - **Possible to specify timezone as offset in hours to UTC, e.g.,** 04-05-06+02
    - **Needs then to be called Time with time zone**
  - **Timestamp**
    - **Concatenation of date and time [yyyy-MM-dd hh:mm:ss]**
    - **E.g.,** 1999-01-08 04:05:06
    - **Also possible with timezone, e.g.,** 1999-01-08 04:05:06+02
    - **Needs then to be called Timestamp with time zone or Timestamptz**
    - **to_timestamp() function if you have unix timestamps (seconds since 1.1.1970)**

Important: Single quotes needed to store any date/time type

Data types (in PostgreSQL)

- Other types
  - Boolean
    - For storing binary states
    - true / false or 1 / 0 is possible
  - NULL
    - This is not a type but used to specify missing values for all types
    - E.g., to indicate a missing address for a customer
    - Cannot be set if the attribute has the constraint NOT NULL
    - For comparisons:
      - WHERE x = NULL does not work
      - WHERE x IS NULL does work
  - XML / JSON
    - To store XML or JSON strings

Full list: https://www.postgresql.org/docs/current/datatype.html

Domain

- Domain **D (value domain)** represents **a data type**
- Also defines the **set of values allowed for an attribute**

Examples:

Age

- can be represented with **Integer**
- 1,2,3,....,100

Domain

- Domain D (value domain) represents a data type
- Also defines the set of values allowed for an attribute

Examples:

Age
- can be represented with **Integer**
- 1,2,3,....,100

Name
- can be represented as a **String** (array of characters)
- 'Jane Doe', 'John Doe'

Domain

- Domain **D (value domain)** represents a **data type**
- Also defines the **set of values allowed** for an attribute

Examples:

Age
- can be represented with **Integer**
- 1,2,3,....,100

Name
- can be represented as a **String** (array of characters)
- 'Jane Doe', 'John Doe'

Date of Birth
- can be represented as a **String** or **Date**
- Can have **constraints** (**Date of Birth cannot be beyond the current date**)
- '21-08-1995', '02-07-2041'

Domain

- Domain **D (value domain)** represents a **data type**
- Also defines the **set of values allowed** for an attribute

A1 INT	A2 INT	A3 BOOL
3	7	T
1	2	T
3	4	F
1	7	T

Examples:

Age
- can be represented with **Integer**
- 1,2,3,....,100

Name
- can be represented as a **String** (array of characters)
- 'Jane Doe', 'John Doe'

Date of Birth
- can be represented as a **String** or **Date**
- Can have **constraints** (**Date of Birth cannot be beyond the current date**)
- '21-08-1995', '02-07-2041'

Cardinality

- **Cardinality** of a relation: number of tuples in the relation

	A1 INT	A2 INT	A3 BOOL
	3	7	T
	1	2	T
	3	4	F
Tuple	1	7	T

Cardinality

- **Cardinality** of a relation: **number of tuples** in the relation

A1 INT	A2 INT	A3 BOOL
3	7	T
1	2	T
3	4	F
1	7	T

cardinality: **4**

Tuple

Rank

- Rank of a relation: **number of attributes**

Attribute

A1 INT	A2 INT	A3 BOOL
3	7	T
1	2	T
3	4	F
1	7	T

Rank

- Rank of a relation: number of attributes

Attribute

Rank: **3**

A1 INT	A2 INT	A3 BOOL
3	7	T
1	2	T
3	4	F
1	7	T

Duplicates and Ordering

- We work with **bag of tuples** (which allows **duplicated tuples**)
- **Ordering** of tuples and attributes is **irrelevant**

A1 INT	A2 INT	A3 BOOL
3	7	T
1	2	T
3	4	F
1	2	T

Example UniversityDB

Professor

PID	Title	Firstname	Lastname
1	Univ.-Prof. Dipl.-Inf. Dr.	Stefanie	Lindstaedt
3	Assoc.Prof. Dipl.-Ing. Dr.techn.	Viktoria	Pammer-Schindler
7	Dr	Lucas	Iacono

Courses

CID	Title	ECTS
INF.01017UF	Data Management (VO)	3
INF.02018UF	Data Management (KU)	1
706.010	Databases	3
706.520	Data Integration and Large-Scale Analysis	5

Primary Keys

- **Minimal set of attributes** X that **uniquely identifies tuples** in a relation R
- E.g., PID=1

1	Univ.-Prof. Dipl.-Inf. Dr.	Stefanie	Lindstaedt

Primary Keys

- **Minimal set of attributes** X that **uniquely identifies tuples** in a relation R
- E.g., PID=1

1	Univ.-Prof. Dipl.-Inf. Dr.	Stefanie	Lindstaedt

- **#1 Unique:** No other tuple with same value
- **#2 Defined:** Not null
- **#3 Minimal:** No attribute can be removed wrt #1

Primary Key Examples

- **Example Airports**
  - Airport Name
  - IATA (International Air Transport Association) code
  - ICAO (International Civil Aviation Organization) code
  - Latitude
  - Longitude
  - Altitude

Graz Airport
GRZ
LOWG
46.9911
15.4396
1115 [ft]

Primary Key Examples

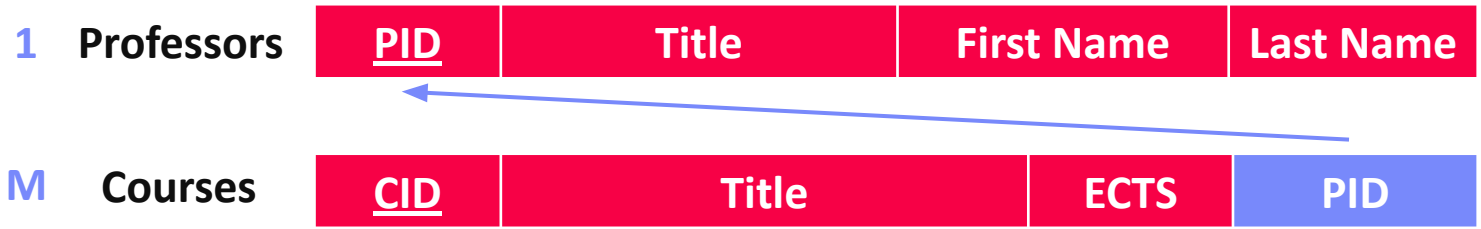
- **Example Airports**
  - Airport Name
  - IATA (International Air Transport Association) code
  - ICAO (International Civil Aviation Organization) code
  - Latitude
  - Longitude
  - Altitude

Graz Airport
GRZ
LOWG
46.9911
15.4396
1115 [ft]

- **Q: Primary Keys**
  - IATA **or** ICAO, **or**
  - [Surrogate key - AID]

Foreign Key

- Reference to a **primary key in another relation**



Example UniversityDB - SQL

Professors

PID	Title	Firstname	Lastname
1	Univ.-Prof. Dipl.-Inf. Dr.	Stefanie	Lindstaedt
3	Dr	Lucas	Iacono
7	Dipl. Ing.	Adrian	Spataru

```
CREATE TABLE Professors (
    PID INTEGER PRIMARY KEY,
    Title VARCHAR(128),
    Firstname VARCHAR(128),
    Lastname VARCHAR(128)
);
```

```
--> alternative for composite key:
...
PRIMARY KEY(A1, A2)
...
```

```
--> SERIAL keyword for
auto-increment
```

Example UniversityDB - SQL

Courses

CID	Title	ECTS	PID
INF.01017UF	Data Management (VO)	3	3
INF.02018UF	Data Management (KU)	1	3
706.010	Databases	3	3

```
CREATE TABLE Courses (
    CID VARCHAR(24) PRIMARY KEY,
    Title VARCHAR(256),
    ECTS INTEGER,
    PID INTEGER
        REFERENCES Professors(PID)
);
```

```
--> alternative for composite key:
...
FOREIGN KEY(A1, A2)
  REFERENCES Professors(A1, A2)
...
```

Referential Integrity Constraints

PID	Title	Firstname	Lastname
1	Univ.-Prof. Dipl.-Inf. Dr.	Stefanie	Lindstaedt
3	Dr.	Lucas	Iacono
7	Dipl. Ing.	Adrian	Spataru

CID	Title	ECTS	PID
INF.01017UF	Data Management (VO)	3	3
INF.02018UF	Data Management (KU)	1	3
706.010	Databases	3	18

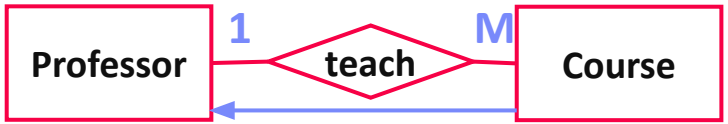
Referential Integrity Constraints

- **Foreign Keys:**
  - Reference to a **primary key** in **another relation**
  - **Referential integrity:** FK **need to reference existing** tuples or **NULL**

Referential Integrity Constraints

- **Foreign Keys:**
  - Reference to a **primary key** in **another relation**
  - **Referential integrity:** FK **need to reference existing** tuples or **NULL**

```
DELETE FROM Professors WHERE PID=18
```



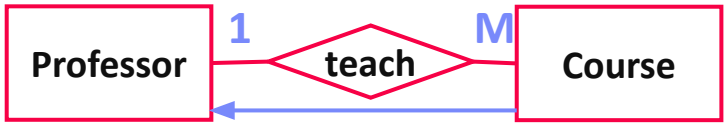
- **Enforcing Referential Integrity**
  - **#1 Error** (default)

Referential Integrity Constraints

- **Foreign Keys:**
  - Reference to a **primary key** in **another relation**
  - **Referential integrity:** FK **need to reference existing** tuples or **NULL**

- **Enforcing Referential Integrity**
  - **#1 Error** (default)
  - **#2 Propagation** on request
    - E.g., for existential dependence

```
DELETE FROM Professors WHERE PID=18
```

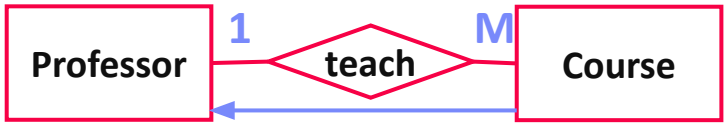



```
CREATE TABLE Courses (...
   PID INTEGER REFERENCES Professors(PID)
      ON DELETE CASCADE);
```

Referential Integrity Constraints

- **Foreign Keys:**
  - Reference to a **primary key** in **another relation**
  - **Referential integrity:** FK **need to reference existing** tuples or **NULL**

```
DELETE FROM Professors WHERE PID=18
```



- **Enforcing Referential Integrity**
  - **#1 Error** (default)
  - **#2 Propagation** on request
    - E.g., for existential dependence

```
CREATE TABLE Courses (...
   PID INTEGER REFERENCES Professors(PID)
      ON DELETE CASCADE);
```

  - **#2 Set NULL** on request
    - E.g., for independent entities

```
CREATE TABLE Courses (...
   PID INTEGER REFERENCES Professors(PID)
      ON DELETE SET NULL);
```

Referential Integrity Constraints

PID	Title	Firstname	Lastname
1	Univ.-Prof. Dipl.-Inf. Dr.	Stefanie	Lindstaedt
3	Dr.	Lucas	Iacono
7	Dipl. Ing.	Adrian	Spataru

CID	Title	ECTS	PID
INF.01017UF	Data Management (VO)	3	3
INF.02018UF	Data Management (KU)	1	3
706.010	Databases	3	NULL

Constraints

- **Domain/Semantic Constraints**
  - **Value constraints** of individual **attributes**
  - **CHECK:** Value ranges or enumerated valid values

```
CREATE TABLE Courses (
   CID INTEGER PRIMARY KEY,
   ECTS INTEGER
      CHECK (ECTS BETWEEN 1 AND 10)
      DEFAULT 3
);
```

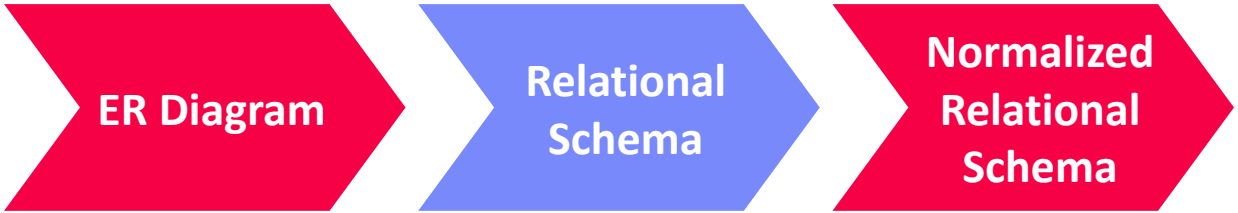
Constraints

- **Domain/Semantic Constraints**
  - **Value constraints** of individual **attributes**
  - **CHECK:** Value ranges or enumerated valid values

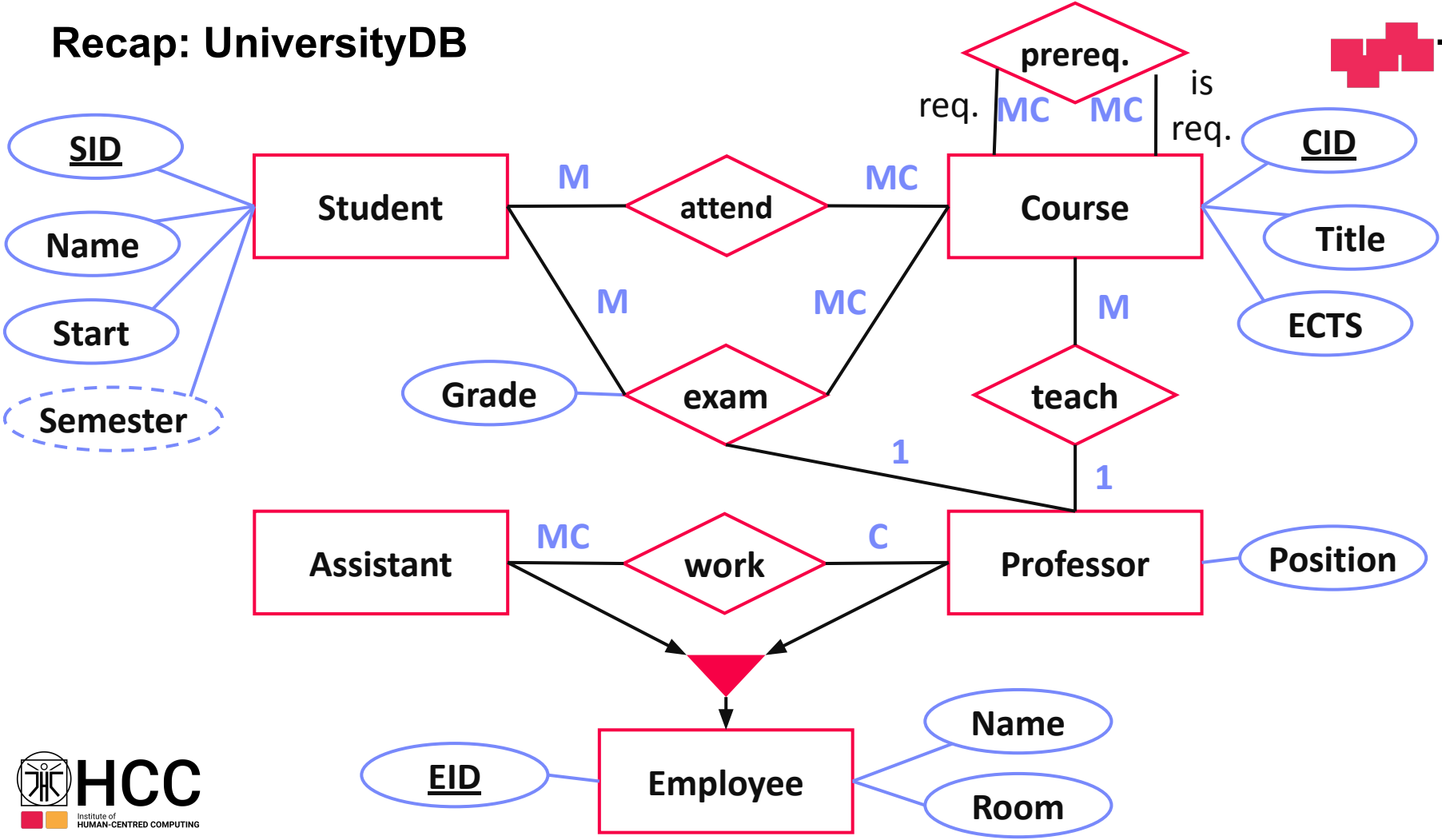
```
CREATE TABLE Courses (
   CID INTEGER PRIMARY KEY,
   ECTS INTEGER
      CHECK (ECTS BETWEEN 1 AND 10)
      DEFAULT 3
);
```

- **UNIQUE** Constraints
  - Enforce uniqueness of non-primary key attribute
- **NOT NULL** Constraints
  - Enforce known / existing values
- **DEFAULT** Constraints
  - Sets a default value when created

ER-Diagram to Relational Schema



Recap: UniversityDB

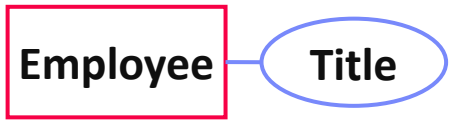


Step 1: Mapping Entity Types

- Each entity type **directly maps to a relation** --> create artificial key if needed, e.g.:

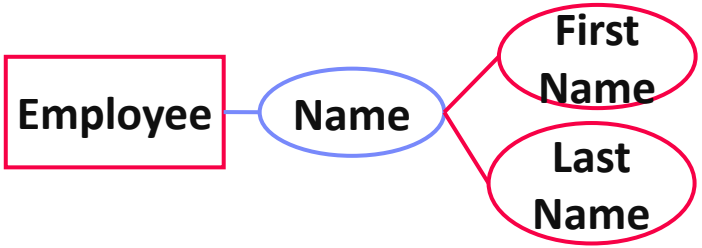
Student	Students(SID:**INTEGER**, Name:**VARCHAR**(128), Start:**DATE**)
Course	Course(CID:**VARCHAR**(24), Title:**VARCHAR**(256), ECTS:**INTEGER**)
Employee	Employee(EID:**INTEGER**, Name:**VARCHAR**(256), Room:**VARCHAR**(24))
Professor	Professor(EID:**INTEGER**, Title:**VARCHAR**(128))
Assistant	Assistant(EID:**INTEGER**)

Step 2: Mapping Attributes



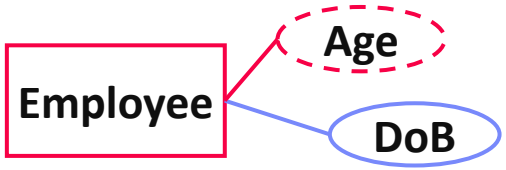
- **Atomic Attributes**
  - Direct mapping of ER attributes to **attributes of relation**
  - Choice of **data types and constraints**

Step 2: Mapping Attributes



- Composite Attributes
  - Split into atomic attributes

Step 2: Mapping Attributes



- Derived Attributes
  - Via views --> virtual table, which contains data that is based on other tables
    - Part of „Data Management“ course

Step 2: Mapping Attributes

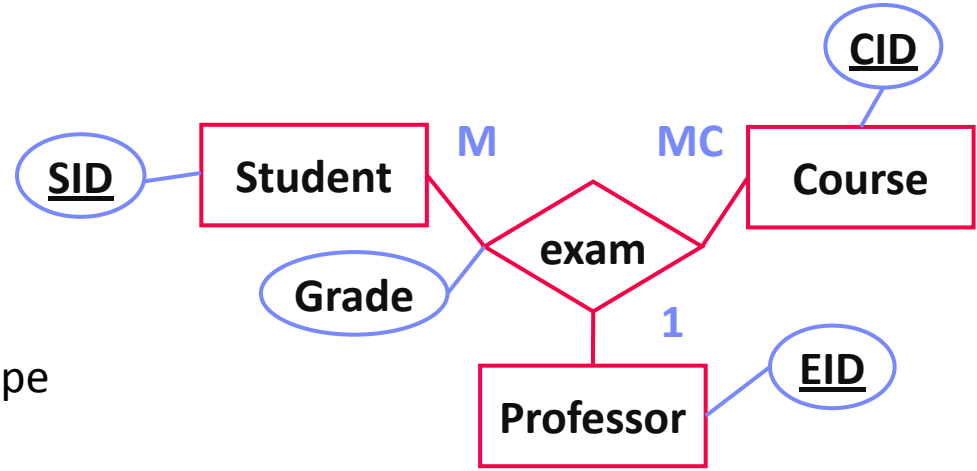
PID	Title	Firstname	Lastname
1	Univ.-Prof. Dipl.-Inf. Dr.	Stefanie	Lindstaedt
3	Dr.	Lucas	Iacono

NID	Number	PID
1	555-123	1
2	555-789	1
3	555-999	3

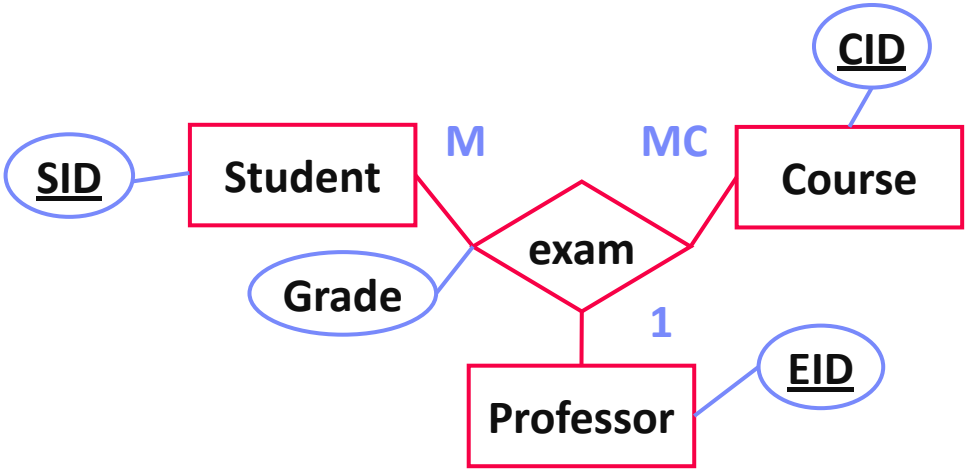
- **Multi-valued Attributes**
  - Relation with FK to **originating relation**
  - E.g., have a phone table, which has employee-id as foreign key

Step 3: Mapping Relationship Types

- **Generic Solution**
  - Map every M-M relationship type to a **relation**
  - **Compose primary key** of keys from involved entity types (**foreign keys**)
  - **Append attributes** of relationship type



Example

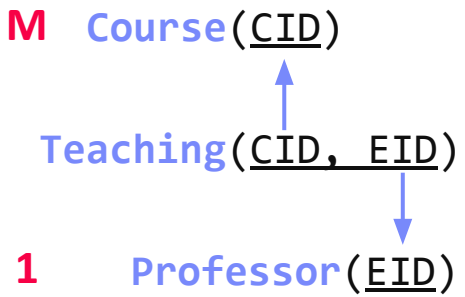


Exams (foreign keys: {SID}, {CID}, {EID}; primary key: {SID, CID, EID})

SID^{FK}	CID^{FK}	EID^{FK}	Grade
12345	706.010	7	1.0
12399	706.550	7	1.7
12399	706.010	7	1.3
12282	INF.01017UF	7	1.0

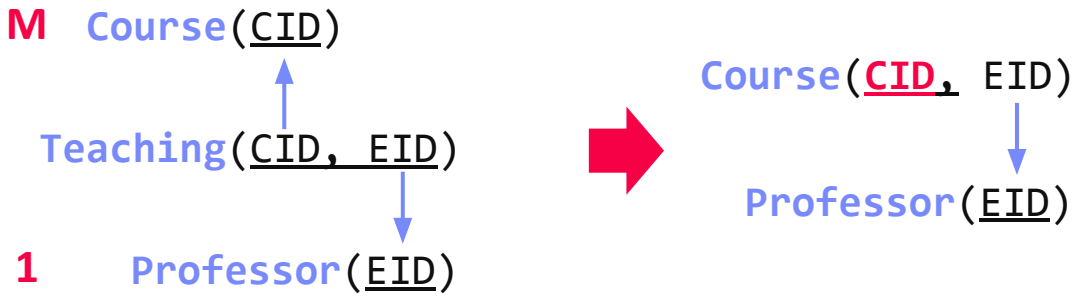
Step 4: Simplification

- **Issue: Unnecessary Relation per Relationship Type**
  → Simplify 1:1, 1:M, M:1 relationship types



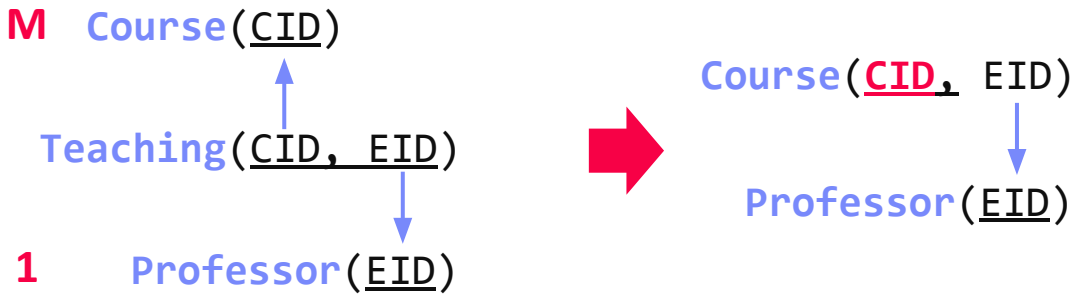
Step 4: Simplification

- **Issue: Unnecessary Relation per Relationship Type**
  → Simplify 1:1, 1:M, M:1 relationship types



Step 4: Simplification

- **Issue: Unnecessary Relation per Relationship Type**
  → Simplify 1:1, 1:M, M:1 relationship types



→ In case of M-M (multiple professors for one course), we would still need the Teaching relation

Step 4: Simplification

Rules

For **E1 – R – E2**

E.g., E1: Professor

E.g., E2: Course

E.g., R: teach

Cardinality	Implementation
1:1	One relation **E12**, PK from E1 or E2
C:1	One relation **E12**, PK from E2
1:M	Two relations **E1** + **E2**, E2 with FK to E1
M:M MC:MC	Three relations **E1**, **R**, **E2**; R with FKs to E1+E2, which is also the PK of R

Step 4: Simplification

Rules

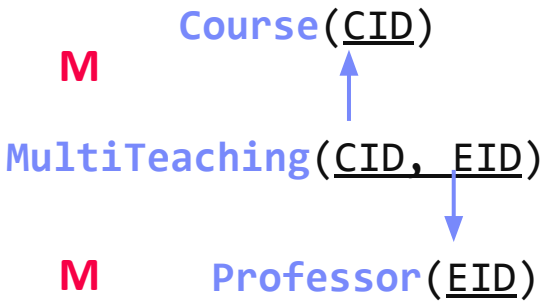
For **E1 – R – E2**

E.g., E1: Professor

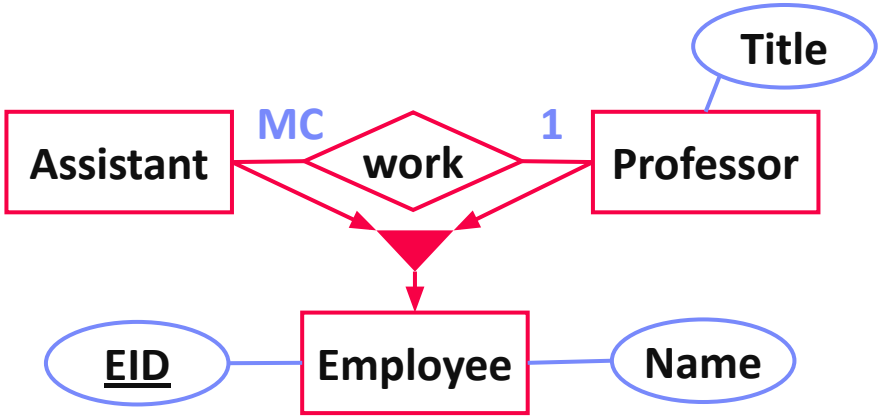
E.g., E2: Course

E.g., R: teach

Cardinality	Implementation
1:1	One relation **E12**, PK from E1 or E2
1:M	Two relations **E1** + **E2**, E2 with FK to E1
M:M MC:MC	Three relations **E1**, **R**, **E2**; R with FKs to E1+E2, which is also the PK of R



Step 5: Mapping Specializations



- One relation per general and specialized entity

→ **Employee, Assistant, Professor**

Employee

EID	Name
7	Juan Fangio
5	Lewis Hamilton

Assistant

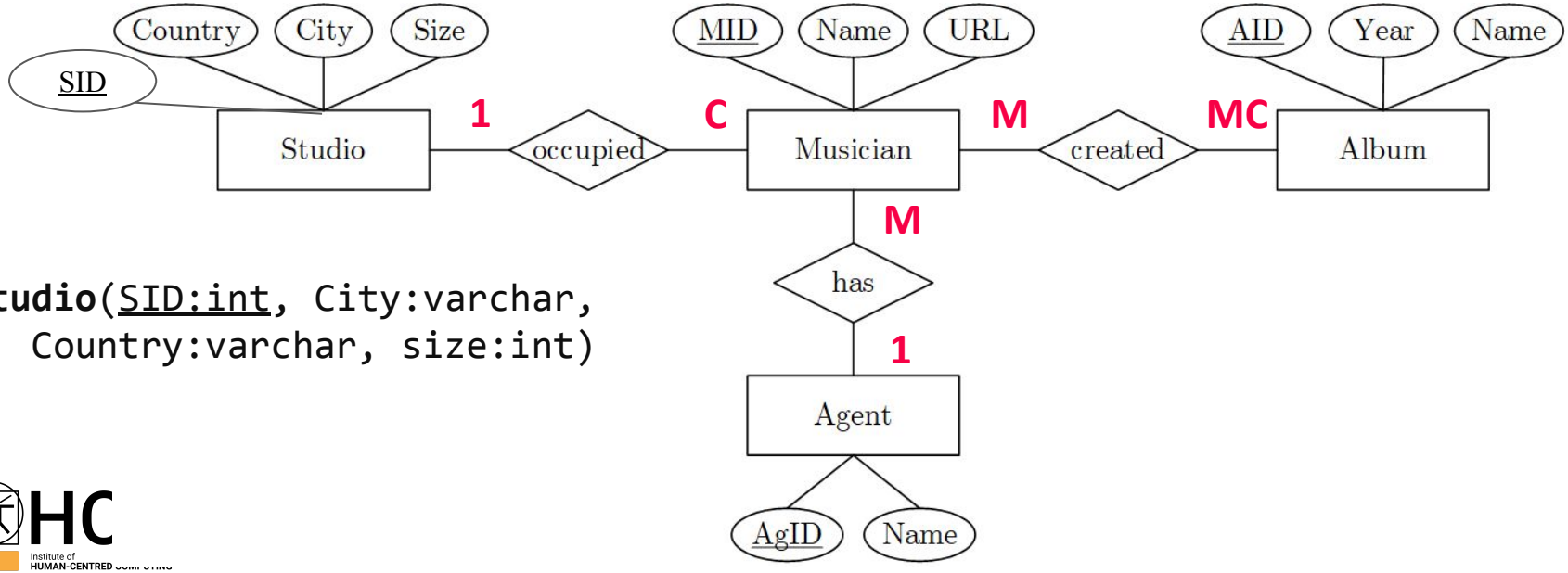
EID
5

Professor

EID	Title
7	Lecturer

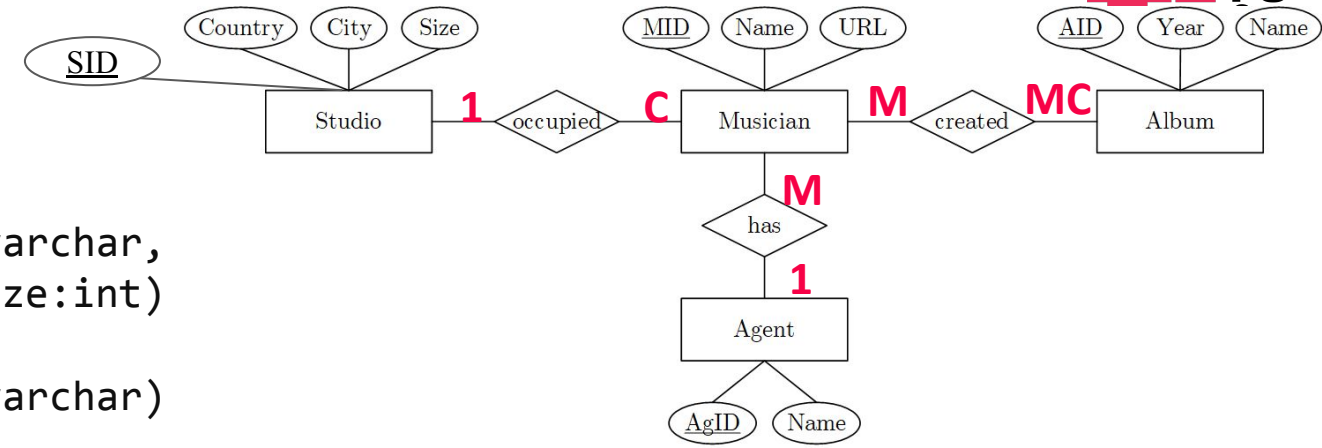
Example

- Task: Map the given ER diagram into a relational schema, including data types, primary keys, and foreign keys



Studio(<u>SID:int</u>, City:varchar,
Country:varchar, size:int)

Solution



Studio(SID:int, City:varchar,
Country:varchar, size:int)

Agent(AgID:int, Name:varchar)

Musician(MID:int, Name:varchar,
URL:varchar, SID^{FK}:int, $AgID^{FK}$:int)

Album(AID:int, Year:int, Name:varchar)

Created(MID^{FK}:int, AID^{FK}:int)

For exercise:

- **Use SQL Create Table(..) statements to specify your relational schema**
- **See slides for syntax**

Normalization

Poor Relational Schemas

ProfCourse (mixed entity types → redundancy)

EID	Name	CID	Title	ECTS
13	Spataru	INF.01014UF	Data Management (VO)	3
13	Spataru	INF.02018UF	Data Management (KU)	1
14	Kowald	706.010	Databases	3
7	Boehm	706.520	Data Integration and Large-Scale Analysis	5
7	Boehm	706.543	Architecture of Database Systems	5
7	Boehm	706.550	Architecture of Machine Learning Systems	5

EID	Name	CID	Title	ECTS
13	Spataru	INF.01014UF	Data Management (VO)	3
13	Spataru	INF.02018UF	Data Management (KU)	1
14	Kowald	706.010	Databases	3
7	Boehm	706.520	Data Integration and Large-Scale Analysis	5
7	Boehm	706.543	Architecture of Database Systems	5
7	Boehm	706.550	Architecture of Machine Learning Systems	5

- **Insert Anomaly:** How to insert a **new lecture or professor**? (only both is possible here)
- **Update Anomaly:** How to **update "Boehm"→ "Böhm"**? (multiple updates are needed)
- **Delete Anomaly:** What if we **delete all data-related lectures**? (also the professor is deleted)

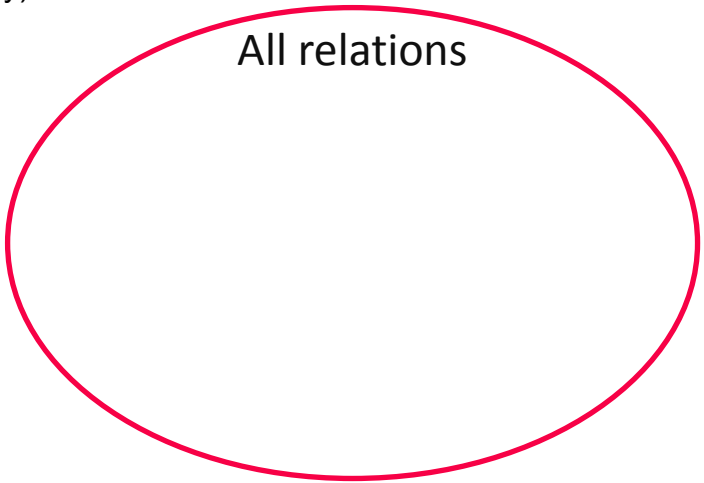
EID	Name	CID	Title	ECTS
13	Spataru	INF.01014UF	Data Management (VO)	3
13	Spataru	INF.02018UF	Data Management (KU)	1
14	Kowald	706.010	Databases	3
7	Boehm	706.520	Data Integration and Large-Scale Analysis	5
7	Boehm	706.543	Architecture of Database Systems	5
7	Boehm	706.550	Architecture of Machine Learning Systems	5

- **Insert Anomaly:** How to insert a new lecture or professor? (only both is possible here)
- **Update Anomaly:** How to update “Boehm” □ “Böhm”? (multiple updates are needed)
- **Delete Anomaly:** What if we delete all data-related lectures? (also the professor is deleted)

→ **Normalization:** **Find good schema to avoid redundancy, ensure consistency, and prevent information loss**

Overview Normalization

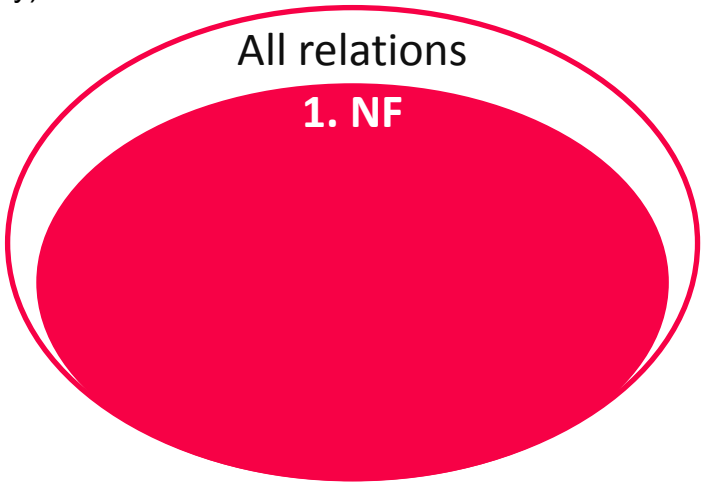
- **Normalization Process**
  - **Database design technique** of replacing a given collection of relations to get a progressively **simpler and more regular structure**
  - Approach of **improving the quality** (redundancy, inconsistencies)
  - Input: DB-Schema and functional dependencies (e.g., city→country)



Overview Normalization

- **Normalization Process**
  - **Database design technique** of replacing a given collection of relations to get a progressively **simpler and more regular structure**
  - Approach of **improving the quality** (redundancy, inconsistencies)
  - Input: DB-Schema and functional dependencies (e.g., city □ country)

- **1st Normal Form: no multi-valued attributes**

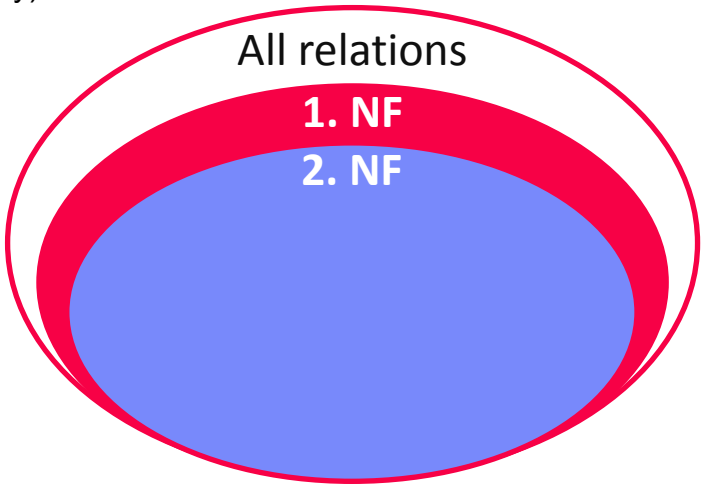


Overview Normalization

- **Normalization Process**
  - **Database design technique** of replacing a given collection of relations to get a progressively **simpler and more regular structure**
  - Approach of **improving the quality** (redundancy, inconsistencies)
  - Input: DB-Schema and functional dependencies (e.g., city □ country)

- **1st Normal Form: no multi-valued attributes**

- **2nd Normal Form: all non-key attributes**
  fully functional dependent on primary key



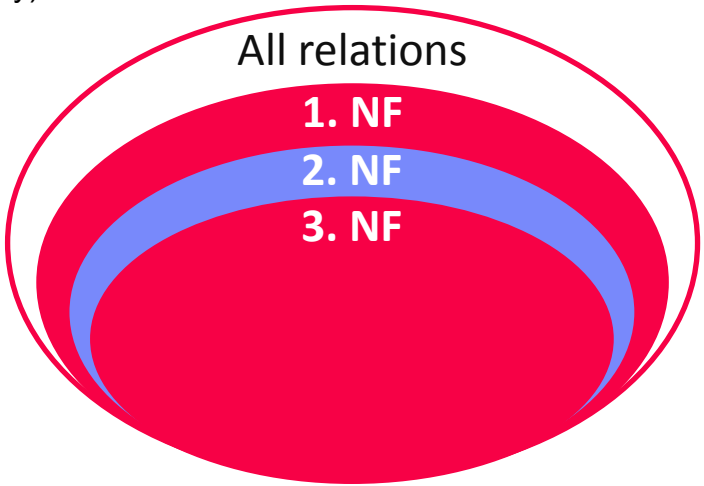
Overview Normalization

- **Normalization Process**
  - **Database design technique** of replacing a given collection of relations to get a progressively **simpler and more regular structure**
  - Approach of **improving the quality** (redundancy, inconsistencies)
  - Input: DB-Schema and functional dependencies (e.g., city □ country)

- **1st Normal Form: no multi-valued attributes**

- **2nd Normal Form: all non-key attributes fully functional dependent on primary key**

- **3rd Normal Form: no functional dependencies among non-key attributes**



Overview Normalization

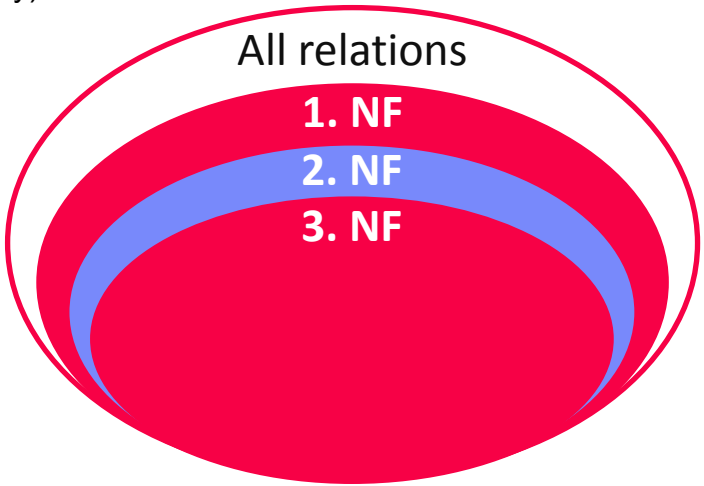
- **Normalization Process**
  - **Database design technique** of replacing a given collection of relations to get a progressively **simpler and more regular structure**
  - Approach of **improving the quality** (redundancy, inconsistencies)
  - Input: DB-Schema and functional dependencies (e.g., city □ country)

- **1st Normal Form: no multi-valued attributes**

- **2nd Normal Form: all non-key attributes fully functional dependent on primary key**

- **3rd Normal Form: no functional dependencies among non-key attributes**

→ **Also more available but 3rd NF is enough for us**



Unnormalized Relation

P#	PDesc	Qty	Project (Pr#, PrDesc, Mgr, Qty)			
203	CAM	30	12	Sorter	007	5
			73	Collator	086	7
206	COG	155	12	Sorter	007	33
			29	Punch	086	25
			36	Reader	111	16

Unnormalized Relation

P#	PDesc	Qty	Project (Pr#, PrDesc, Mgr, Qty)			
203	CAM	30	12	Sorter	007	5
			73	Collator	086	7
206	COG	155	12	Sorter	007	33
			29	Punch	086	25
			36	Reader	111	16

Issues

- Column **'Project'** is **not atomic, but set of tuples**
- **Redundancy** across projects appearing in multiple parts

Solution:

- Create multiple tables with PK-FK relationships

1st Normal Form

- **Definition and Approach**
  - Relation is in 1NF if all its **attributes are atomic (not multi-valued)**

  → **Split relations with 1:M, MC:MC and M:M relationships (without loosing information)**

Relation Part

P#	PDesc	Qty
203	CAM	30
206	COG	155

FK

Relation Project

P#	Pr#	PrDesc	Mgr	Qty
203	12	Sorter	007	5
203	73	Collator	086	7
206	12	Sorter	007	33
206	29	Punch	086	25
206	36	Reader	111	16

Functional Dependencies

FK

Relation Part

P#	PDesc	Qty
203	CAM	30
206	COG	155

Relation Project

P#	Pr#	PrDesc	Mgr	Qty
203	12	Sorter	007	5
203	73	Collator	086	7
206	12	Sorter	007	33
206	29	Punch	086	25
206	36	Reader	111	16

Functional Dependencies

FK

Relation Part

P#	PDesc	Qty
203	CAM	30
206	COG	155

Relation Project

P#	Pr#	PrDesc	Mgr	Qty
203	12	Sorter	007	5
203	73	Collator	086	7
206	12	Sorter	007	33
206	29	Punch	086	25
206	36	Reader	111	16

depend on Pr#

depends on (Pr#,P#)

Functional Dependencies

FK

Relation Part

P#	PDesc	Qty
203	CAM	30
206	COG	155

Relation Project

P#	Pr#	PrDesc	Mgr	Qty
203	12	Sorter	007	5
203	73	Collator	086	7
206	12	Sorter	007	33
206	29	Punch	086	25
206	36	Reader	111	16

depend on Pr#

depends on (Pr#,P#)

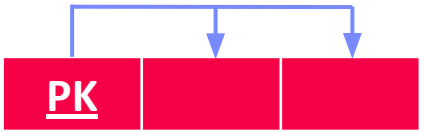
- **Issues**
  - **Insert anomaly** (e.g., no project without parts)
  - **Update anomaly** (e.g., redundant updated Mgr)
  - **Delete anomaly** (e.g., project deleted on last part)

→ **Domain knowledge is needed to check for dependencies**

2nd Norm

- Definition and Approach
  - Relation is in 2NF if it's **in 1NF** and every non-key attribute is **fully functional dependent** on primary key

→ Split relations with 1:M, MC:MC and M:M relationships (without loosing information)



2nd Norm

Relation Part

P#	PDesc	Qty
203	CAM	30
206	COG	155

Relation ProjectPart

FK FK

P#	Pr#	Qty
203	12	5
203	73	7
206	12	33
206	29	25
206	36	16

Relation Project

Pr#	PrDesc	Mgr
12	Sorter	007
73	Collator	086
29	Punch	086
36	Reader	111

3rd Normal Form

- **Definition and Approach**
  - Relation is in 3NF if it's **in 2NF** and there are **no dependencies among non-key attributes**

→ Split relations with 1:M, MC:MC and M:M relationships (without loosing information)

→Last example was already in 3rd normal form!

3rd Normal Form

Additional field: MgrName

Relation Project

Pr#	PrDesc	Mgr#	MgrName
12	Sorter	007	Edward
73	Collator	086	Cole
29	Punch	086	Cole
36	Reader	111	Smith

3rd Normal Form

Additional field: MgrName

NOT in 3NF

- **Pr# → PrDesc**
- **Pr# → Mgr#**
- **Mgr# → MgrName**

(and Mgr# is not the PK!)

Relation Project

Pr#	PrDesc	Mgr#	MgrName
12	Sorter	007	Edward
73	Collator	086	Cole
29	Punch	086	Cole
36	Reader	111	Smith

3rd Normal Form

Relation Project

Pr#	PrDesc	Mgr#
12	Sorter	007
73	Collator	086
29	Punch	086
36	Reader	111

FK

Relation Manager

Mgr#	MgrName
007	Edward
086	Cole
111	Smith

Final Example

- Task: Assume the functional dependency City→Country. Bring your schema into 3NF and explain why it is in 3NF

Studio(SID:int, City:varchar, Country:varchar, size:int)

Final Example

- Task: Assume the functional dependency City→Country. Bring your schema into 3NF and explain why it is in 3NF

  Studio(SID:int, City:varchar, Country:varchar, size:int)

- Solution:

  Studio(SID:int, City^{FK}:varchar, size:int)

  Cities(City:varchar, Country:varchar)

  - 1st Normal Form: no multi-valued attributes
  - 2nd Normal Form: all non-key attributes fully functional dependent on PK
  - 3rd Normal Form: no functional dependencies among non-key attributes

Exercise T1.2

Exercise T1.2

1.2 Mapping ER Model to Relational Model (10/25 points)

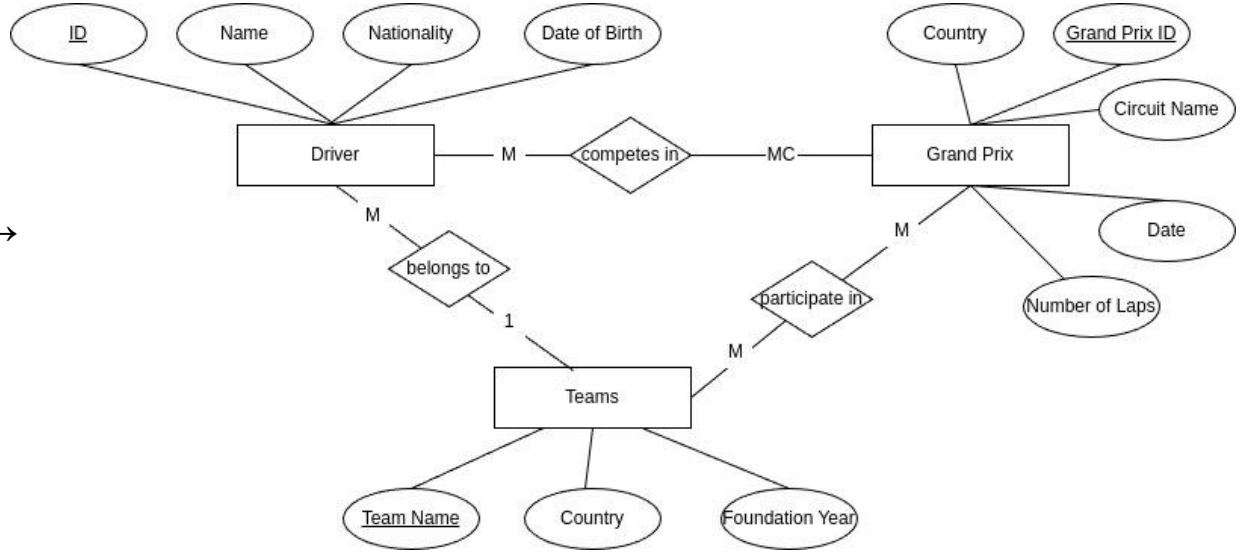
Create a relational schema for the ER diagram designed in **Task 1.1** and bring it into **third normal form**. This schema should include the **relations** and **typed attributes**, as well as all **primary and foreign keys**.

Please, create a SQL script (CreateSchema.sql) with CREATE TABLE statements.

Deliverable: CreateSchema.sql

Example similar to T1.2

From T1.1 →



Example similar to T1.2

```sql
-- Drop tables if they already exist
DROP TABLE IF EXISTS Drivers, Teams, GrandPrix, TeamGrandPrix, DriverGrandPrix CASCADE;

-- Table: Teams
CREATE TABLE Teams (
--     team_id SERIAL PRIMARY KEY,
    team_name VARCHAR(100) PRIMARY KEY,
    country VARCHAR(50) NOT NULL,
    foundation_year INT NOT NULL
);

-- Table: Drivers
CREATE TABLE Drivers (
    driver_id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    nationality VARCHAR(50) NOT NULL,
    date_of_birth DATE NOT NULL,
    team_name VARCHAR(100) NOT NULL REFERENCES Teams(team_name) -- Reference to Teams (1:M relationship)

);
```

Example similar to T1.2

```sql
-- Table: Grand Prix
CREATE TABLE GrandPrix (
    grand_prix_id SERIAL PRIMARY KEY,
    circuit_name VARCHAR(100) NOT NULL,
    country VARCHAR(50) NOT NULL,
    date DATE NOT NULL,
    number_of_laps INT NOT NULL
);

-- Table: Team Participation in Grand Prix (M:M Relationship)
CREATE TABLE TeamGrandPrix (
    team_name VARCHAR (100 )NOT NULL REFERENCES Teams(team_name),
    grand_prix_id INT NOT NULL REFERENCES GrandPrix(grand_prix_id)
);

-- Table: Driver Participation in Grand Prix (MC:M Relationship)
CREATE TABLE DriverGrandPrix (
    driver_id INT NOT NULL REFERENCES Drivers(driver_id),
    grand_prix_id INT NOT NULL REFERENCES GrandPrix(grand_prix_id)
);
```

Conclusions

- **Summary**
  - **Relational databases**
  - **Normalization**
  - **Exercise 1: T1.2**

Conclusions

- **Summary**
  - **Relational databases**
  - **Normalization**
  - **Exercise 1: T1.2**

- **Next weeks**
  - Mon. 10.11.2025: Query languages (SQL) + **Exercise 2 presentation**
  - Mon. 17.11.2025: **no lecture** (time for exercise 1)
  - **Exercise 1 submission: Tue. 18.11.2024**