Data Handling: Databases

Prof. Dr. Michael Grossniklaus

Universität
Konstanz

Department of Computer and Information Science
Database and Information Systems Group
https://dbis.uni-konstanz.de/ → Student Corner → Teaching

Spring Semester 2022

Part I

Database Design

1. The Entity-Relationship Model

This section's goal

After completing this chapter, you should be able to

- explain the **three phases of database design,**
  Why are multiple phases useful?
- evaluate the significance of the **Entity-Relationship Model** (ER model) for DB design,
- enumerate the basic constructs of the ER model,
- develop **ER diagrams** (schemas in the ER model) for a given application,
- **translate** ER models into equivalent (as far as possible) relational models.

Database design (1)

Definition: Database Design

Database Design is the process of developing a **database schema** for a given application.
DB design is a subtask of the overall software engineering effort.

- Overall goal of DBMS usage: Efficiently develop programs to support given real-world tasks.
- These programs need to **store data persistently.**
- To develop these programs, apply proven methods of **software engineering**—specialized to support **data-intensive** computations.

Database design (2)

The specification of programs and data is intertwined:

- The schema should contain the data needed by the programs.
- Programs are often easy to develop once the structure of the data to be manipulated has been specified.

Data, however, is an **independent resource:**

- Typically, additional programs will be developed later based on the collected data.
- Also, **ad-hoc queries** will be posed against the DB.

Database design (3)

During DB design, a **formal model of the relevant aspects of the real world** ("mini world", "domain of discourse") must be built.

> *Once the DB is up and running, questions will be posed against the DB. Knowledge of these questions beforehand is important input to the DB design process and helps to identify the relevant parts of the real world.*

In some sense, the real world is the **measure of correctness** for the DB schema: the possible DB states should correspond to the states of the real world.

Database design (4)

DB design is not easy for a variety of reasons

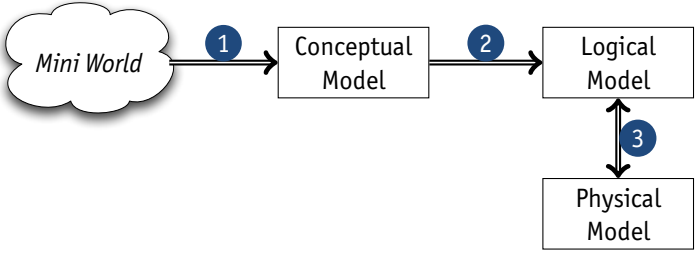
- **Expertise:** The designer must become an expert for the application domain or needs to talk to such experts. Depending on the domain, these experts might *not* be used to give a complete and formal account of their field.
- **Flexibility:** The real world typically permits exceptions and corner cases. Does the DB schema need to reflect these?
- **Size:** Database schemas may become huge (in terms of number of relations, attributes, constraints).

Due to this complexity, DB design is a **multi-step** process.

Database design (5)

Definition: Three Phases of DB Design

1. **Conceptual Database Design.** Produces the initial model of the mini world in a **conceptual data model** (*e.g.*, in the ER model).
2. **Logical Database Design.** Transforms the conceptual schema into the data model supported by the DBMS (*e.g.*, the relational model).
3. **Physical Database Design.** Design indexes, table distribution, buffer size, *etc.*, to maximize performance of the final system.



Database design (6)

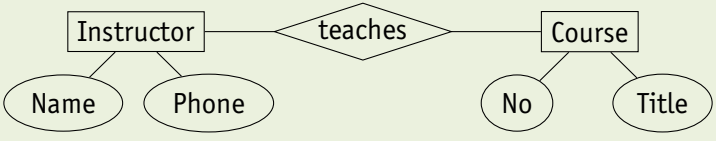
Why multiple design phases?

- Partition the problem, attack one sub-problem after the other.
  For example, during conceptual design, there is no need to worry about performance aspects or limitations of the specific SQL dialect of the RDBMSs.
- DBMS features do not influence the conceptual design and only partially influence the logical design.
  Thus, the conceptual design work is not invalidated, if a different DBMS is used later on.

Example (1)

ER schemas are typically shown in graphical notation:

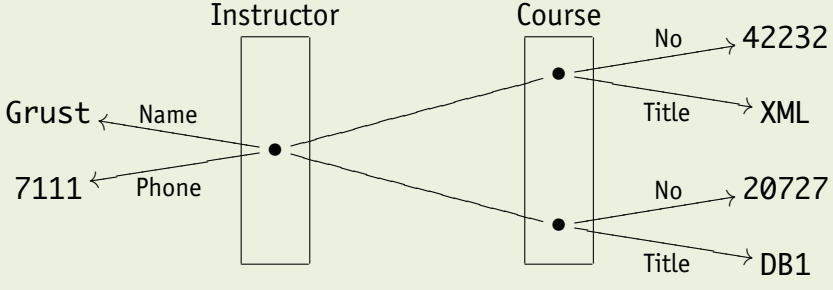
"Classical" ER graphical notation



- This mini world talks about **instructors** and **courses.**
- Instructors **teach** courses.
- Instructors have a **name** and a **phone number.**
- Courses are **enumerated** and have **titles.**

Example (2)

A possible state of this mini world



The ER model

The **Entity-Relationship Model** is often referred to as a **semantic data model,** because it more closely resembles real world scenarios than, *e.g.*, the relational model.

- In the ER model, we model the concept of "Instructors."
  In the relational model we deal with names and phone numbers.
- In the ER model, there is a distinction between **entities** (objects) and **relationships** between such entities.
  In the relational model, both concepts are represented by relations.

Basic ER model concepts (1)

Entities

- An **object** in the mini world about which information is to be stored. Examples: *persons, books, courses*.
  Note: entities do not have to correspond to objects of physical existence. Entities may also represent conceptual objects like, e.g., vacations.
- The mini world that has to be modeled can contain only a **finite number of objects.**
- It must be possible to distinguish entities from each other, *i.e.*, objects must have some **identity.**
  Examples: entity book identified by ISBN number, entity vacations identified by travel agency booking number.

Basic ER model concepts (2)

Attribute

- A **property** or feature of an entity (or relationship, see below).
  Example: the title of this course entity is "Databases."
- The **value** of an attribute is an element of a data type like string, integer, date. These values have a **printable representation** (which entities have not).

Basic ER model concepts (3)

Relationship

- A **relation**—not in the strict relational model sense—**between pairs of entities** (a binary relationship).
  Example: Grust (an entity) teaches (a relationship) the course "Foundations of Databases" (an entity).
- "Relationship" is often used as an abbreviation for "Relationship type" (see below).

Basic ER model concepts (4)

- **Entity Type**
  - A set of **similar entities** (similar with respect to the information which has to be stored about them), *i.e.*, entities which have the same attributes.

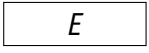
  Example: All faculty members of an institute.
- **Relationship Type**
  - A set of **similar relationships,** *i.e.*, relationships which have the same attributes.

  Example: X teaches course Y.

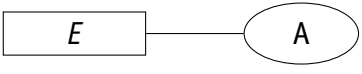
ER diagrams (1)

In the literature, you will find a vast variety of graphical notations for ER diagrams. Here, we use the "classical" boxes-and-diamonds symbols.

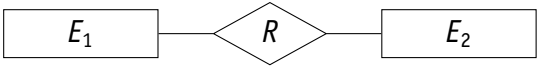
- **Entity type** E



- **Attribute** A **of entity type** E

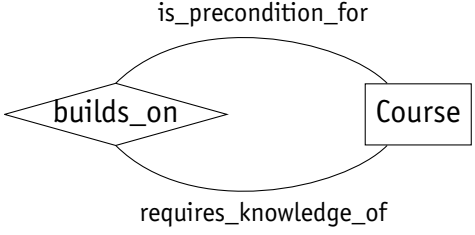


- **Relationship** R **between entity types** E_1 **and** E_2



ER diagrams (2)

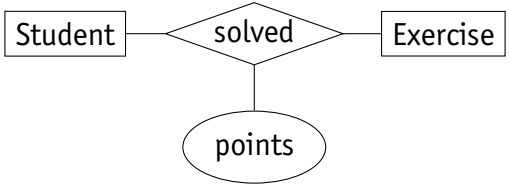
- Relationships may also exists between **entities of the same type** (sometimes misleadingly called “recursive relationships”)



- In this case, **role names** have to be attached to the connecting edges.
  In the ER model, role names may be attached to any kind of relationship for documentation purposes.

ER diagrams (3)

Relationships may have attributes, too



- This models the fact that a number of *points* is stored for every pair of a student *X* and an exercise *Y* such that *X* submitted a solution for *Y*.

Graphical syntax

1. An ER diagram contains
   - **boxes, diamonds, ovals,** plus **interconnecting lines.**
2. Boxes, diamonds, and ovals are each **labelled** by a string.
   - Box labels are unique in the entire diagram.
   - Oval labels are unique for a single box or diamond.
   - Diamond labels are unique for a pair of connected boxes.
3. Interconnecting lines are only allowed between
   - box—diamond, box—oval, diamond—oval.
4. A diamond has exactly two connecting lines to boxes. There may be any number of connections to ovals.[1]
5. An oval has exactly one connecting line.

[1] But see *N*-ary relationships below.

ER example

✎ Modeling a mini world: define an ER diagram

- Information about researchers in the database field is to be stored.
- For each researcher, his/her last name, first name, e-mail address, and homepage URI are relevant.
- Researchers are affiliated with universities and assume a certain position (*e.g.*, professor, lecturer).
- Relevant university information are the name, homepage URI, and country.
- Researchers publish articles (of a given title) in journals.

ER schemas: Formalizing the semantics (1)

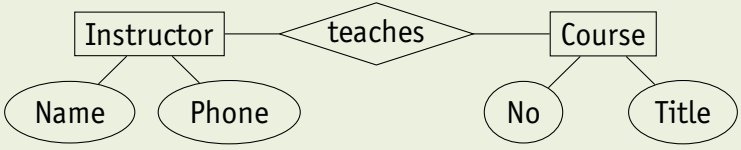
Definition: Database State (ER)

A **DB state** I interprets the symbols in the ER schema by defining

- a finite set $I(E)$ for every entity type E,
- a mapping $I(A)\colon I(E) \to val(D)$ for every attribute A of an entity type E, where D is the data type of A and $val(D)$ is the domain of D,
- a relation $I(R) \subseteq I(E_1) \times I(E_2)$ for every relationship R between entity types E_1 and E_2,
- a mapping $I(A)\colon I(R) \to val(D)$ for every attribute A of relationship R (D is the data type of A).

ER schemas: Formalizing the semantics (2)

Example state corresponding to ER schema

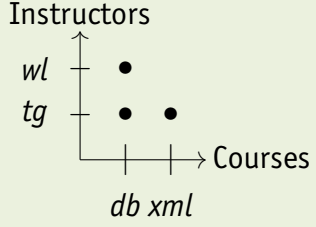


- $I(\text{Instructor}) = \{tg, wl\}$
- $I(\text{Name}) = f_1$ with $f_1(tg) = \texttt{'Grust'}, f_1(wl) = \texttt{'Lex'}$
- $I(\text{Phone}) = f_2$ with $f_2(tg) = 7111, f_1(wl) = 7132$
- $I(\text{Course}) = \{db, xml\}$
- $I(\text{No}) = g_1$ with $g_1(db) = 20727, g_1(xml) = 42232$
- $I(\text{Title}) = g_2$ with $g_2(db) = \texttt{'DB1'}, g_2(xml) = \texttt{'XML'}$

ER schemas: Formalizing the semantics (3)

Example state corresponding to ER schema (cont'd)

- $I(\text{teaches}) = \{(tg, db), (tg, xml), (wl, db)\}$

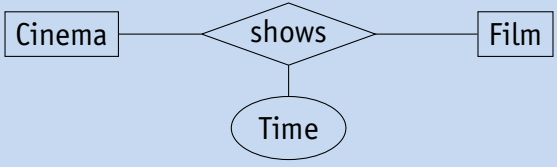


- **Note:** A relationship is interpreted by a set of **entity pairs** and thus is unique for two given entities.

ER schemas: Formalizing the semantics (4)

✎ Consequences of the ER semantics

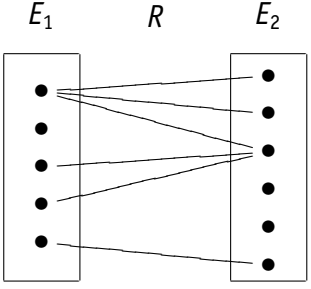
Consider the ER diagram below



- Suppose a cinema shows the same film twice a day (at 3pm and 6pm). Can this information be stored in the given schema?

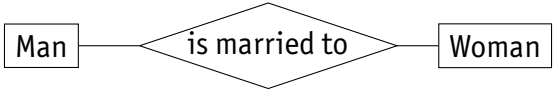
Cardinalities (1)

General ER relationship



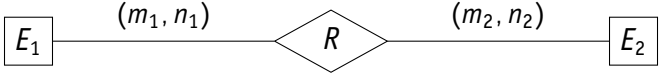
Cardinalities (2)

- In general, there is **no restriction** on how often an entity participates in an relationship R.
  - An entity can be connected to one entity of the other type, to more than one, or it can have no R-partner at all.
- However, specific **application semantics** dictate to how many E_2 entities an E_1 entity can be related:



Cardinalities (3)

The ER model introduces the **(min, max) notation** to specify an **interval** of possible participations in a relationship:



- An entity of type E_1 may be related to at least m_1 and at most n_1 entities of type E_2.
- Likewise, m_2 is the minimum number and n_2 is the maximum number of E_1 entities to which an E_2 entity is related.

Cardinalities (4)

Σ Formal Semantics

- For every DB state I and every entity $e_1 \in I(E_1)$:

$$m_1 \leqslant \left|\{e_2 \in I(E_2) \mid (e_1, e_2) \in I(R)\}\right| \leqslant n_1$$

- For every DB state I and every entity $e_2 \in I(E_2)$:

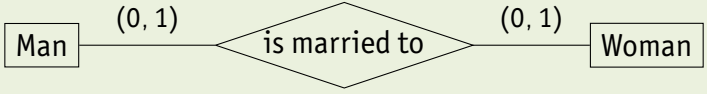
$$m_2 \leqslant \left|\{e_1 \in I(E_1) \mid (e_1, e_2) \in I(R)\}\right| \leqslant n_2$$

Extensions

- "$*$" may be used as maximum, if there is **no limit.**
- $(0, *)$ means no restriction at all (most general relationship).

Cardinalities (5)

Marriage



"A man can be married to at most one woman and vice versa."

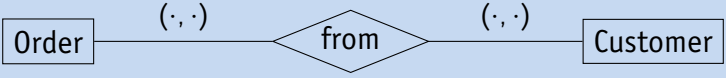
Airport Locations

"An airport lies in exactly one country. A country may have arbitrarily many airports (and maybe none at all)."

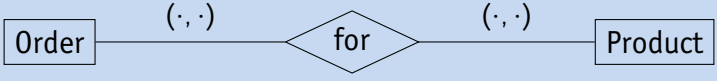
Cardinalities (6)

✎ Derive cardinalities from verbal specifications

"*Besides normal customers, the database may contain customers who have not yet ordered anything.*"



"*An order can contain several products.*"



Selecting cardinalities

- Sometimes a sketch of a valid database state may help in selecting the right cardinalities, *e.g.*, for the state sketched on slide 205, a viable cardinality for E_1—R may be $(0, 3)$.
- Application knowledge might lead to **weaker restrictions**, in this example $(0, 5)$ or $(0, *)$.
  A cardinality (a, b) *is* ***weaker*** *than* (c, d)*, iff* $a \leqslant c$ *and* $b \geqslant d$*,*
  i.e., iff $[a, b] \supseteq [c, d]$.
- In real applications, the cardinalities $(0, 1)$, $(1, 1)$, and $(0, *)$ are the most common and especially **easy to enforce in the relational model.**

Common cases (1)

- Normally, the **minimum cardinality** will be 0 or 1, and the **maximum cardinality** will be 1 or $*$.
  - Thus, only the $(0, 1)$, $(1, 1)$, $(0, *)$, $(1, *)$ cardinalities are common in practice.
- To understand a relationship, one must know the cardinality specifications on **both sides.**
- The **maximum cardinalities** on each side are used to distinguish between **many-to-many**, **one-to-many** / **many-to-one**, and **one-to-one** relationships.

Common cases (2)

Many-to-many relationships:

- Both maximum cardinalities are $*$ (the minimum cardinalities are 0 or 1)

Many-to-many relationship

- This is the most general/least restrictive case of a relationship.
- When translated into the relational model, the representation of many-to-many relationships requires an extra table.

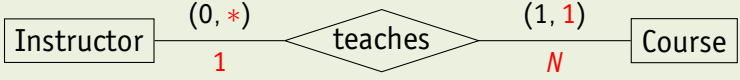
Common cases (3)

One-to-many relationships:

- Maximum cardinality 1 on the "many" side and $*$ on the "one" side[2]

One-to-many relationship

"One instructor teaches many courses, but each course is run by exactly one instructor."



- One-to-many relationships do *not* require an extra table in an equivalent representation in the relational model.

[2] Notice the antisymmetry!

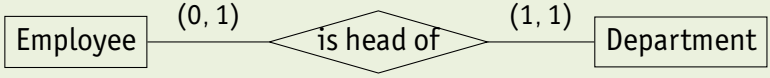
Common cases (4)

One-to-one relationships:

- Maximum cardinality 1 on both sides

One-to-one relationship

"Each department has exactly one department head, some employees are the head of one department."

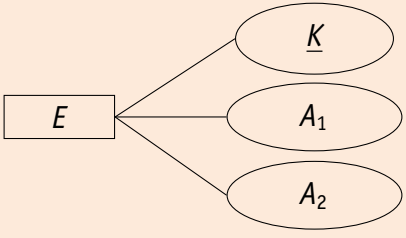


- Note how **mandatory** or **optional** participation in a relationship determines the **minimum cardinalities.**

Keys (1)

Definition: Key (ER)

A **key** K of an entity type E[3] is an attribute of E which **uniquely identifies** the entities of this type. No two different entities share the same value for the key attribute, *i.e.*, $I(K)$ is **injective** for all DB states I. Composite keys are allowed.



[3] Only *entity* types can have key attributes.

Keys (2)

- **Importance of keys**
  - The ER model does **not require** the declaration of a key for each entity type E since entities already have a distinguished "object identity", namely the elements of $I(E)$.

  *But see **weak entities** introduced shortly.*
- The **translation of ER schemas into relational schemas** requires the declaration of ER keys, though.

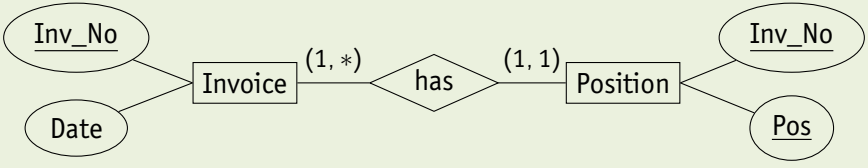
- If there is no natural key, add **artificial identifiers** (*e.g.*, integers, remember attributes EMPNO, DEPTNO from above) which then represent the elements of $I(E)$.

Weak Entities (1)

In many schemas, some entities describe a kind of **detail** that **cannot exist** without a **master** (or owner) **entity.**
In such a case,

1. there is a **relationship with cardinality** (1, 1) **on the detail entity side,** and in addition
2. the **key of the master is inherited and becomes part of the key of the detail entity.**

Example



Weak Entities (2)

- Without a specific ER construct for this case, we would require the following additional constraint:
  - If two entities are in “has” relationship,
  - then their attribute “Inv_No” are required to have identical values.

  For example, invoice #12 cannot have position 2 in invoice #42 as detail.
- Such constraints occur, if **an entity does not have a key by itself, but it is only unique in the context of some other (master) entity.**

Weak Entities (3)

Note: In weak entities, keys are **always composite.**

- Examples
  - A classroom is identified by a building *and* a room number.
  - A section in a book is identified by a chapter *and* a section title.
  - A web page URI is composed of a web server DNS address *and* a path on that server.

There is also an **existence dependency.**

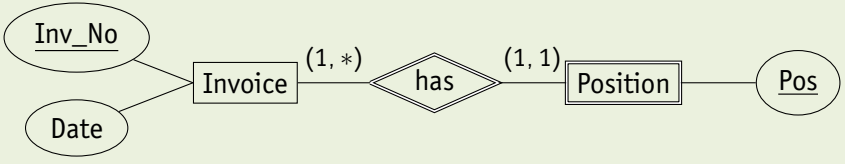
- If the building is pulled down, the classrooms automatically disappear.
- If the web server is shut down, all URIs on that server cease to function.

Weak Entities (4)

In the ER model, such scenarios are modeled via **weak entities**[4].

In ER diagrams, weak entities and their identifying relationships are indicated by **double lines**

Weak Entities



For the weak entity, the **inherited part of the key is not shown.**

[4] Non-weak entities are also called **strong entities.**

Weak Entities (5)

- Usage of weak entities is required only if the key of one entity **contains** the key of a related entity.
  A weak entity adds further key attributes to the master key.
- Weak entities are normal entities except that their key is constructed in a special way.

Weak entities can take part in other relationships

- Weak entities may themselves be masters of other weak entities (building an entire hierarchy of dependencies).

Weak Entities (6)

✎ Modeling with weak entities

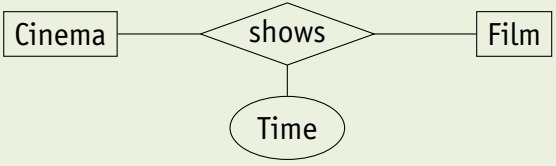
Model a set of online quizzes (multiple choice tests).

- Each quiz is identified by a title, each question within a quiz is numbered, and each possible answer to a given question is referenced by a letter.
  For each question and answer, the associated text is stored. Answers are classified into correct and incorrect ones.
- What is the complete key for each of the occurring entity types?

Relationships connecting more than two entities

Remember our example from above

Cinema timetables



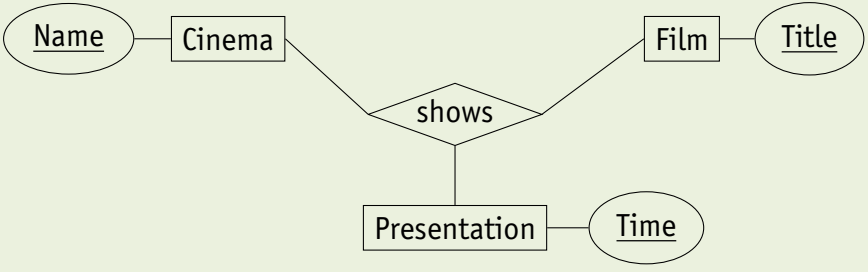
If a cinema shows the same movie more than once, we are in trouble. There are basically two options.

1. Allow the use of n-ary relationships, connecting $n \geqslant 2$ entity types.
2. Disallowing those, introduce additional entity types.

N-ary relationships

We allow relationships that "connect" more than 2 entity types. In the example, a *ternary* relationship will do

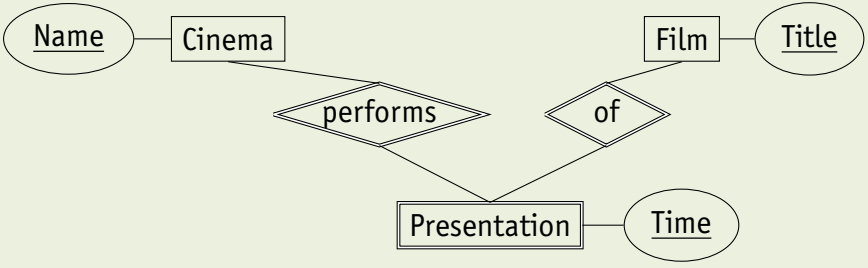
Cinema timetables with ternary relationship



Association Entities

Use an **association entity:** turn relationship into weak entity with multiple owners. Inherits keys of all owners.

Cinema timetables with association entity



Extended ER modeling

The basic ER model discussed so far has been extended in a variety of directions. Most notably, **generalization/specialization** hierarchies have been introduced.

Generalization ... denotes the process to "factor out" common characteristics (*e.g.*, attributes, constraints) of several entity types into a more general, common **supertype entity**.

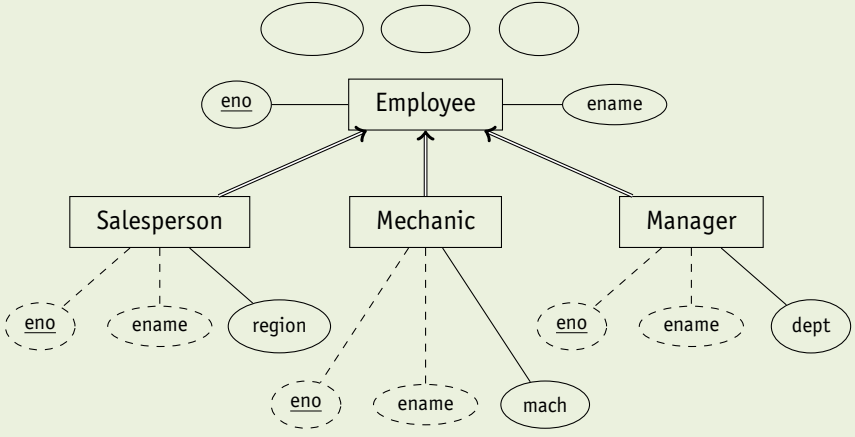
Specialization ... denotes the opposite process: deriving new, specialized **entity subtypes** from a given type, possibly adding specific characteristics (such as, *e.g.*, more attributes or more restrictive constraints).

Inheritance ... means the fact that (no matter how we ended up with the hierarchy) entity subtypes fulfill all restrictions (presence of attributes, their domains, constraints) that are defined for the supertype. They can, however, strengthen them.

Generalization vs. Specialization

Generalization: when we start from the subtypes (going up ↑).
Specialization: when we start from the supertypes (going down ↓).

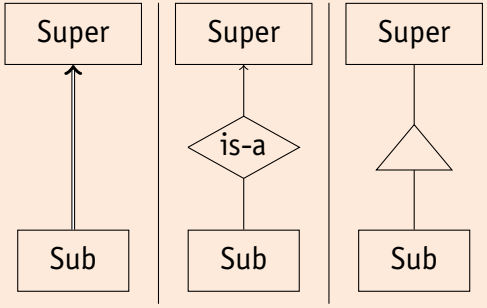
Generalization vs. specialization: different types of employees



Notation

... again, a number of different notations are in use. Some of them are given below.

Graphical syntax for generalization hierarchies ("IS-A"-relationships)



Additional constraints

The following additional constraints can—sometimes—be specified with generalizations/specializations.[5]

disjoint ... If S_1 and S_2 are subtypes of the same supertype E, no E can be an S_1 and an S_2 at the same time.

No Animal can be a Fish and a Mamal.

covering ... If S_1 and S_2 are subtypes of the same supertype E, each E must either be an S_1 or an S_2 (or both).

All Persons are either Men or Women, there are no “others”.

derived ... E has an attribute A, whose value determines whether it is an S_1 or an S_2.

Employees have a distinguishing Job_title attribute.

[5] All of these are not restricted to *two* subtypes!

2. A Quick Introduction to the Relational Model

This section's goal

After completing this chapter, you should be able to

- explain **basic notions** of the **relational model**
  - table/relation, row/tuple, column/attribute, column value/attribute value,
- explain the meaning of **keys** and **foreign keys**,
- understand that there are better and worse relational database schemas.

The Relational Model

The **relational model** structures data in **table form**, *i.e.*, a relational DB is a set of named tables.

A relational database schema defines...

1. **Names of tables** in the database,
2. the **columns of each table**, *i.e.*, the **column name** and the **data types** of the column entries,
3. **integrity constraints**, *i.e.*, conditions that data entered into the tables is required to satisfy.

Example

- Assume a database maintaining information about a small real-world subset: a company's departments and employees.
- Two tables:
  - EMP: information about employees.
  - DEPT: information about departments.

Table DEPT

DEPT

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

Columns' types

The three **columns** of table DEPT have the following **data types**

- DEPTNO has data type NUMERIC(2), *i.e.*, the column can hold two-digit integer entries $-99 \ldots 99$.
  An integrity constraint can be used to exclude negative department numbers.
- DNAME has type VARCHAR(14), *i.e.*, the entries are character strings of variable length of up to 14 characters.
- LOC has type VARCHAR(13).

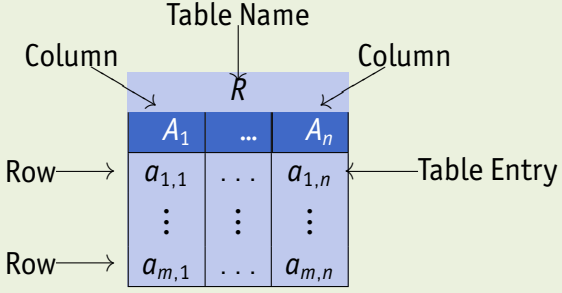
Table rows

- A **relational database state** (instance of a given schema) defines, for each table, **a set of rows.**
- In the current state shown on the slide above, table DEPT has four rows.
- The relational model **does not define any particular order of the rows** (*e.g.*, first row, second row).[6]
  Rows can be sorted for output, though.
- Each row specifies values for each column of the table.

[6] formally, a table is a relation, *i.e.*, a **set**.

Summary

A relational table



Towards a formalization

A more theoretically inclined person, or the typical database text book, would use the following equivalent terminology:

- Table $\equiv$ **relation**
- Row $\equiv$ **tuple**
- Column $\equiv$ **attribute**
- Data type of a column $\equiv$ **domain** of the attribute
- Table entry $\equiv$ **attribute value**

Definition: Relation

Formally, a table is a subset of the Cartesian product of the domains of the column data types, *i.e.*, a relation in the mathematical sense.

In an RDBMS, relations are always finite, may involve any number of columns, and we typically ignore the order of columns.

Keys

The column DEPTNO is declared as the **key** of table DEPT.

Relational Keys

A key always **uniquely identifies a single row** in its associated table.

A key is a constraint!

- Table DEPT, for example, already contains a row with key DEPTNO $= 10$.
- If one tries to add another row with the same value 10 for DEPTNO, the DBMS responds with an error.

Constraints

- Keys are an example of **constraints:** conditions that the table contents (DB state) must satisfy *in addition* to the basic structure prescribed by the columns.
- Constraints are declared as part of the DB schema.
- More than one key can be declared for a table.

More keys?

One could, for example, discuss whether DNAME should also be a key (in addition to DEPTNO already being a key). This would exclude the possibility that there can ever be two or more departments of the same name.

- Keys and other constraints are treated in more detail later.

Another example table (I)

Table EMP (data about employees) with the following columns:

- EMPNO: A unique number for every employee.
- ENAME: Employee name.
- JOB: Employee position (*e.g.*, ENGINEER)
- MGR: Direct supervisor of this employee.
- HIREDATE: Employee hire date.
- SAL: Employee salary.
- COMM: Commission (only for salespeople).
- DEPTNO: Department where this employee works.

Another example table (II)

Table EMP

EMP

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	20
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	20
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		20
7782	CLARK	MANAGER	7839	09-JUN-81	2450		20
7788	SCOTT	ANALYST	7566	09-DEC-82	3000		20
7839	KING	PRESIDENT		17-NOV-81	5000		20
7844	TURNER	SALESMAN	7698	09-SEP-81	1500	0	20
7876	ADAMS	CLERK	7788	12-JAN-83	1100		20
7900	JAMES	CLERK	7698	03-DEC-81	950		20
7902	FORD	ANALYST	7566	03-DEC-81	3000		20
7934	MILLER	CLERK	7782	23-JAN-82	1300		20

Foreign Keys

- **Physical pointers are unknown** to the relational model.
- However, the relational model provides a kind of "**logical pointer**".
  - For example, the column DEPTNO in table EMP **refers to** column DEPTNO in table DEPT.
- Since DEPTNO is declared as a key in table DEPT, a department number uniquely identifies a single row of DEPT.
- A value for DEPTNO (in table EMP) can be seen as a "logical address" of, or reference to, a row in table DEPT.

Foreign Keys are constraints, too!

- By including a department number in table EMP, each row in EMP "points to" a single row in table DEPT.
- It is important for the integrity of the database, that the department numbers in EMP in fact occur in DEPT.

"Dangling pointers"

If a row in table EMP contains a DEPTNO value of 70, the reference "dangles."

A DBMS, however, does *not* crash as would be the case with real physical pointers (memory addresses). In SQL, references are followed by comparing column values. Nevertheless, such a column entry is a kind of error and should be avoided.

This is the purpose of **foreign keys.**

Declaring Foreign Keys

The relational model (and thus, any RDBMS) permits to declare column DEPTNO as a **foreign key** that references table DEPT.

Definition: DB modifications in the presence of Foreign Keys

The DBMS will refuse

- an insertion into table EMP with a value for DEPTNO that **does not appear** in DEPT,
- a deletion of a row in DEPT that is **still referenced** by a row in EMP,[a]
- corresponding updates (changes) of DEPTNO values.

[a] It might be reasonable to recursively delete all employees of that department, too (**cascading delete**).

More Foreign Keys

- Table EMP also contains a second foreign key:
  Column MGR contains the employee number of the employee's direct supervisor.
- This shows that
  - it is possible that a foreign key refer to another row in the **same table** (or even the **same row**),
  - the foreign key column and referenced key may have **different names.**

Null values

- The relational model allows column entries to remain empty (*i.e.*, the column/row contains a **null value**).
- For example, in table EMP:
  1. only salespeople have a commission,
  2. the company president has no supervisor.
- When a table schema is declared, one can specify for each column whether null values are accepted or not (this is another example of a constraint specified for the table).
- Null values are treated specially in comparisons.

Designing “good” relational tables

Not every table makes a good relation...

Employees and departments revisited

EMPDEPT

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO	DNAME	LOC
7369	SMITH	CLERK	7902	17-DEC-80	800		20	RESEARCH	DALLAS
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30	SALES	CHICAGO
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	20	RESEARCH	DALLAS
7566	JONES	MANAGER	7839	02-APR-81	2975		20	RESEARCH	DALLAS
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	20	RESEARCH	DALLAS
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		20	RESEARCH	DALLAS
7782	CLARK	MANAGER	7839	09-JUN-81	2450		20	RESEARCH	DALLAS
7788	SCOTT	ANALYST	7566	09-DEC-82	3000		20	RESEARCH	DALLAS
7839	KING	PRESIDENT		17-NOV-81	5000		20	RESEARCH	DALLAS
7844	TURNER	SALESMAN	7698	09-SEP-81	1500	0	20	RESEARCH	DALLAS
7876	ADAMS	CLERK	7788	12-JAN-83	1100		20	RESEARCH	DALLAS
7900	JAMES	CLERK	7698	03-DEC-81	950		20	RESEARCH	DALLAS
7902	FORD	ANALYST	7566	03-DEC-81	3000		20	RESEARCH	DALLAS
7934	MILLER	CLERK	7782	23-JAN-82	1300		20	RESEARCH	DALLAS

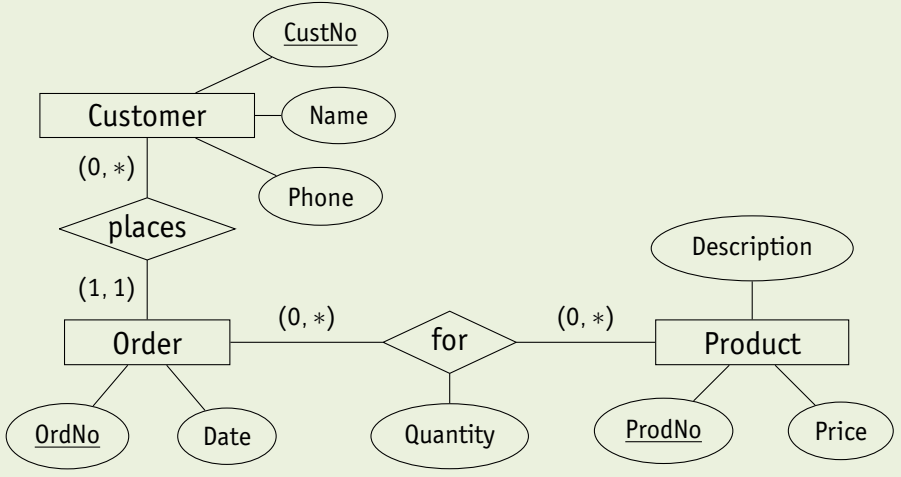
“Bad” relational tables

Some Observations of what might characterize “bad” design

- Avoid redundancy!
  - save space (maybe less important)
  - save overhead during updates (would blow up application code)
- Avoid columns, for which some rows have values while others don’t.
  - yields “null values” (complicates query predicates)
- Mix columns that describe properties of several real world concept into 1 relation.
  - complicates understanding of table “semantics”
- … more to come, see later.

3. Translation of ER diagrams into the relational model

ER diagram example



Step 1: Entities (1)

Transforming an ER entity *E*

1. Create a **table for each entity.**
   The table name is *E* (conventionally: *E* + `'s'`).
2. The **columns** of this table are the **attributes of the entity type.**
3. The **primary key of the table** is the **primary key of the entity type.**
   If E's key is composite, so will be the relational key. If E has no key, add an ***artificial key*** *to the table.*

Step 1: Entities (2)

Entity tables

Customers

CustNo	Name	Phone
10	Jones	624-9404
11	Smith	

Orders

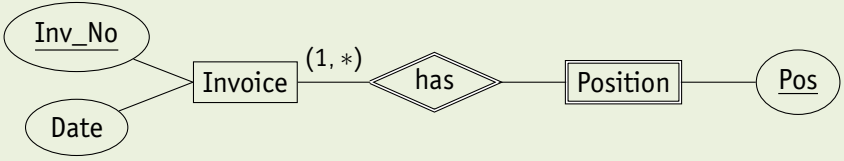
OrdNo	Date
200	2/15/04
201	2/16/04

Products

ProdNo	Description	Price
1	Apple	0.50
2	Kiwi	0.25
3	Orange	0.60

Step 1b: Weak entities

Weak entity example



When a **weak entity** is translated, the **key attributes of the owner entity** are added as a **key and foreign key**

$$\texttt{Position}\ (\underline{\texttt{Pos}},\ \underline{\texttt{Inv\_No}} \rightarrow \texttt{Invoice},\ \ldots)$$

- This automatically implements the relationship.
- In SQL, it makes sense to specify `ON DELETE CASCADE` for the foreign key: if an invoice is deleted, all its positions will be removed from the DB state, too.

Step 2: One-to-many relationships (1)

Transforming a relationship *R*

1. If *R* has **maximum cardinality 1 on one side**, *R* is **one-to-many**[7].
   Example: Customer–(0, ∗)–places–(1, 1)–Order, "one customer places many orders."
2. In this case, add **key of "one" side as a column to the "many" table** to implement *R*.
3. This column will be a **foreign key referencing a row in the table representing the related entity.**

[7] If *R* has maximum cardinality 1 on both sides, it is actually one-to-one, see below.

Step 2: One-to-many relationships (2)

Relationship tables

Orders (OrdNo, Date, CustNo → Customers)

Orders		
OrdNo	Date	CustNo
200	2/15/04	11
201	2/16/04	11

Customers		
CustNo	Name	Phone
10	Jones	624-9404
11	Smith	

Convention: use relationship and role to name foreign key column

Orders (OrdNo, Date, placed_by → Customers)

Step 2: One-to-many relationships (3)

Transforming a one-to-many relationship *R* (cont'd)

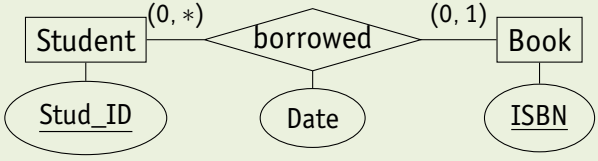
4. If the **minimum cardinality is 1** on the "many" side (see example), null values are **not allowed** in the foreign key column (column `placed_by` in example).

   If the **minimum cardinality is 0**, null values **are allowed** in the forign key column.
   The foreign key is null for those entities that do not participate in R at all.

Step 2: Relationship attributes

To transform **one-to-many relationship attribute(s)**, *e.g.*

ER diagram example



- **store the relationship attribute(s) together with the reference to the related entity**, *e.g.*

Corresponding entity table

Books (ISBN, …, borrowed_by → Students, Date)

Step 2: One-to-many relationships

Translation variant for one-to-many relationships

One-to-many relationships *R* **with cardinality (0,1)** can be translated into a table of their own

`borrowed_by (ISBN→Books, Stud_ID→Students, Date)`

- Extra table holds key values of related entities plus relationship attributes.
- Key attributes on side of the (0,1)-cardinality become key of this relation.
  "Each book can be borrowed only once at the same time."

This does not model *R* correctly if the cardinality is (1,1)

Why?

Step 3: Many-to-many relationships (1)

Transforming a many-to-many relationship *R*:

1. If *R* has maximum cardinality $*$ on **both sides**, *R* is **many-to-many.**
   Example: Order–$(1, *)$*–for–*$(0, *)$*–Product, "an order contains many products, a product may be part of many orders."*
2. *R* becomes its **own table.**
3. The columns of this table are the **keys of both participating entity types.**
4. These columns act as **foreign keys** referencing the entities and, at the same time, together form a **composite key** for the extra table.

Step 3: Many-to-many relationships (2)

Transforming a many-to-many-relationship *R* (cont'd)

5. Relationship attributes are added as columns to the table representing *R*.

Example table

```
for (OrdNo→Orders, ProdNo→Products, Quantity)
```

Composite key?

Is it really necessary that both entity keys (here: OrdNo, ProdNo) form a composite key for the relationship table?

Step 3: Many-to-many relationships (3)

Example tables

for (OrdNo → Orders, ProdNo → Products, Quantity)

for		
OrdNo	ProdNo	Quantity
200	1	1
200	2	1
201	1	5

Orders		
OrdNo	Date	CustNo
200	2/15/04	11
201	2/16/04	11

Products		
ProdNo	Description	Price
1	Apple	0.50
2	Kiwi	0.25
3	Orange	0.60

Step 3: Many-to-many relationships (4)

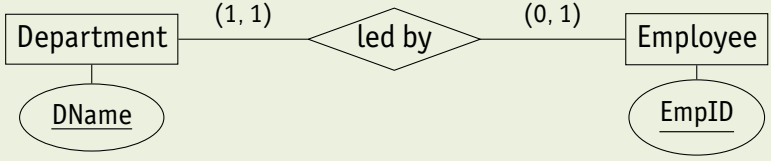
Transforming a many-to-many relationship R (cont'd)

- **Note:** Minimum cardinalities other than 0 for R **cannot be enforced** by the relational model per se, *i.e.*, in terms of key constraints.
  Order–$(0, *)$–*for*–$(1, *)$–*Product* $\Leftrightarrow$ *"Every product occurs in at least one order."*
- If this is important to guarantee database consistency (valid DB state), this constraint needs to be **checked by the application program** or a **general RDBMS constraint mechanism.**

Step 4: One-to-one relationships (1)

Transforming a one-to-one relationship *R*

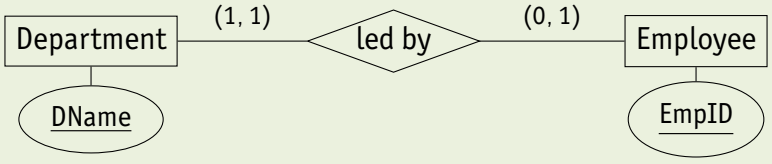
ER diagram example



1. If *R* has maximum cardinality 1 on **both sides**, *R* is **one-to-one.**
2. We can essentially transform as if *R* were one-to-many, but additional key constraints are generated.

Step 4: One-to-one relationships (2)

To which entity table do we add the `led_by` attribute?



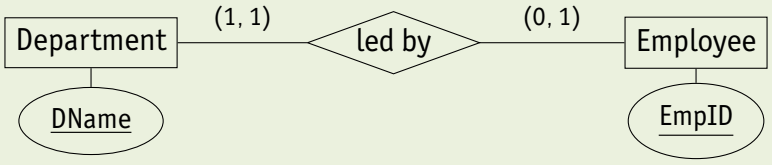
- Since we have Department–(1, 1)–led_by (*"every department is led by exactly one employee"*), it makes sense to host the relationship in the Department table

  `Department (DName, …, led_by → Employee)`

- We may declare the foreign key `led_by` as `NOT NULL`. (This is not possible if `led_by` is hosted in the Employee table.)

Step 4: One-to-one relationships (3)

ER diagram and corresponding entity table



Department (DName, …, led_by → Employee)

- **Note:** led_by now also is a key for the Department table.

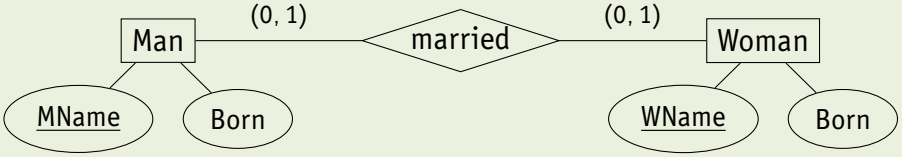
✎ led_by is a key for table Department

Why is this the case in the example?

- This key constraint enforces the maximum cardinality of 1 (on Employee).

Step 4: One-to-one relationships (4)

ER diagram example



Two variants

1. Any of the two (*not both!*) entity tables may host the `married` foreign key (null values allowed).
2. Translate the relationship into a table of its own

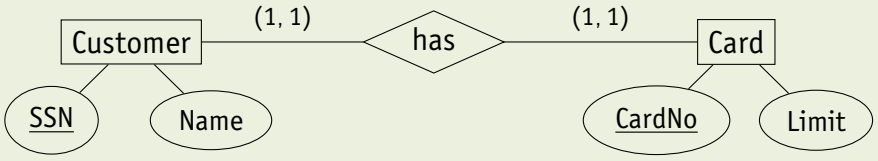
   `married (MName` $\rightarrow$ `Man, WName` $\rightarrow$ `Woman)`

✎ A one-to-one relationship in an extra table

What would be the correct key(s) for table `married`?

Step 4: One-to-one relationships (5)

ER diagram example



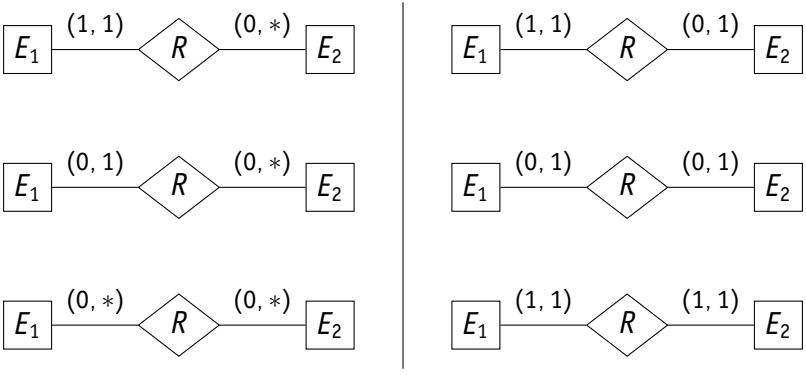
In order to **enforce the minimum cardinality 1 on both sides, the entity tables need to be merged**

`CustomerCard (SSN, Name, CardNo, CreditLimit)`

- No NULL values are allowed.
- Both, SSN and CardNo, are candidate keys of this table. One is selected as primary key, the other is an alternative key.

Limitations (1)

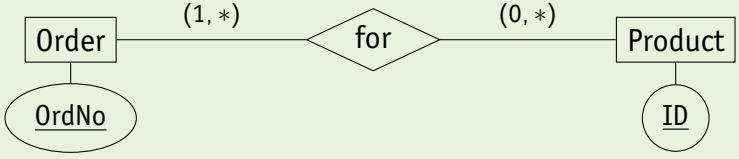
The following ER relationship cardinalities can be faithfully represented in the relational model.



For all other cardinalities, the constraint mechanisms of the relational model will not suffice.

Limitations (2)

Example



Relational (foreign) key constraints *cannot* enforce the minimum cardinality 1 ("Every purchase order includes at least one product item.")

⇒ Apply the translation method for $(0, *)$ cardinalities and enforce the constraint in the application (or via a separate constraint mechanism in the RDBMS, *e.g.*, a **trigger** whenever a tuple in table Order is inserted).

A note on ER attributes

Some ER variants permit the use of non-atomic attributes. For example, attributes may sometimes be

- composed of several components (like a record/struct in a programming language),
  … *address* = *(street, no, zip, city, state)*
- array- or set-valued,
  … *given_names* = *name**
- or even combinations thereof.

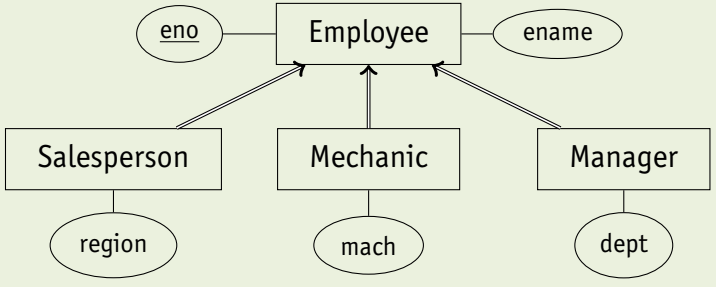
In such cases, due to the *first normal form* (1NF) restriction in an RDBMS, ER attributes need to be mapped in a non-trivial manner, *e.g.*

- onto several attributes in case of record-composition,
- onto separate tables in case of multi-valued attributes.

Mapping extended ER models to relational schemas

The relational model provides no direct representation for generalizations.[8] Thus, we need to use auxiliary constructions.

Generalization hierarchy



[8] But: see extensions in SQL-3

Relational modeling of generalizations (1)

Option 1: Each (super, sub) entity type is mapped to a separate relation.

Separate relations

```
Employee     (eno, ename)
Salesperson  (eno, region)    [eno → Employee]
Mechanic     (eno, machine)   [eno → Employee]
Manager      (eno, dept)      [eno → Employee]
```

Notice

- Each table contains only those attributes that are specific for this type (no inherited attributes).
- The key is inherited down the hierarchy (also as a foreign key).
- Access to inherited attributes within a subtype requires joins.
- No redundant storage (except for the keys) is introduced.

Relational modeling of generalizations (2)

Option 2: Use only one relation for all types.

One single relation

```
Employees  (eno, ename, region, machine, dept)
```

Notice

- Lots of NULL values are introduced.
- A discriminating attribute (such as Job_title, in our example) might be introduced to support testing the kind of entity at hand.
- No joins are necessary to access all attributes.
- Hardly viable in case of multiple inheritance (redundancy)!

Relational modeling of generalizations (3)

Option 3: Use independent relations for subtypes.

Independent subtype relations

```
Salesperson  (eno, ename, region)
Mechanic     (eno, ename, machine)
Manager      (eno, ename, dept)
```

Notice

- Concept of generalization is lost.
- Redundancy in schema (constraints)!
- Redundancy in tuples in case of non-disjoint specializations!
- Not possible with non-covering specializations (or supertype relation needed in addition).
- No joins are necessary to access all attributes.

Relational modeling of generalizations (4)

More options … and how to select?

- Parts of the options shown can be combined.
- In a larger generalization hierarchy, choices are not completely independent.
- Guidelines for the selection may be obtained by considering, *e.g.*
  - ease of use (how difficult to express are typical queries?),
  - additional integrity constraints (which constraints can be expressed at all/easily?),
  - the rest of the schema (which subtypes are involved in which relationships?),
  - performance criteria (how efficient are the most frequent operations?).

Summary: Data Modeling

- **Entity-Relationship Models**
  - Entities (“objects”) and relationships between them
  - Both described by attributes
  - Relationships characterized by their cardinality
  - Quite a few variants, most extensions include generalization/specialization
- **Relational (Database) Model**
  - Everything represented as relations
  - Each with key and, possibly, foreign keys as “links” between them
  - Comes with *formal* quality criteria (Normal Forms, *see next section*)
  - Rather implementation-oriented (compared to ER)

4. Relational Normal Forms

This section's goal

After completing this chapter, you should be able to

- work with **functional dependencies** (FDs),
  define them, detect them in DB schemas, decide implication, determine keys
- explain insert, update, and delete **anomalies,**
- explain **BCNF,** test a given relation for BCNF, and transform a relation into BCNF,
- detect and correct violations of **4NF,**
- detect **normal form violations** on the level of ER,
- decide if and how to **denormalize** a DB schema, and
- explain the link to Entity-Relationship modeling.

Overall Goal

We want to make the (relational) DBMS guarantee *all* our (relevant *integrity constraints*)!

Unfortunately, though, the RDBMS is pretty ignorant w.r.t. *general* constraints: all it cares about are the "*model-inherent*" constraints: **key**s and **foreign key**s.

So, we aim at expressing all constraints (or the most important ones, at least) in the form of those key and foreign key constraints ...

Relational database design theory

- **Relational database design theory** is mainly based on a class of constraints called **Functional Dependencies (FDs).** FDs are a **generalization of keys.**
- This theory defines when a relation is in a particular **normal form** (*e.g.*, in Third Normal Form or 3NF) for a given set of FDs.
- It is usually a sign of **bad DB design**, if a schema contains relations that **violate the normal form** requirements.
  There are exceptions and tradeoffs, however.

Redundancy as a source of trouble

If a normal form is violated, some data are stored **redundantly,** and information about different concepts is **intermixed.**

Redundant data storage

COURSES

CRN	TITLE	INAME	PHONE
22268	Databases I	Grust	7111
42232	Functional Programming	Grust	7111
31822	Graph Theory	Klotz	2418

- The *phone number* for each instructor is stored multiple times (once for every course the instructor is teaching). — **Bad!**

Normal Forms

- Today, the **Third Normal Form (3NF)** is considered the standard relational normal form used in practice (and education).
- **Boyce-Codd Normal Form (BCNF)** is a bit more restrictive, easier to define, and will be a better match for our intuition of good DB design.
  In rare circumstances, a relation might not have an equivalent BCNF form while preserving all its FDs (while this can always be guaranteed for 3NF).
- In a nutshell, BCNF requires that **all FDs are already enforced (represented) by keys.**

Normalization

- **Normalization algorithms** can construct "good" relational schemas from a set of attributes and a set of FDs alone.
- In practice, relations are derived from ER models and normalization is used as an additional check (for BCNF) only.
- When an ER model is well designed, the resulting derived relational tables will **automatically be in BCNF** (even 4NF).
- Awareness of normal forms can help to detect design errors already in the ER design phase.

First Normal Form (1NF)

Definition: First Normal Form (1NF)

First Normal Form (1NF) requires that all table entries (attribute values) are **atomic** (*i.e.*, of a base type, *not* lists, sets, records, relations, ...).

- The relational model is already defined this way. All further normal forms assume that tables are in 1NF.
- SQL-3 (1999*ff*) "Object-Relational" DBMSs allow violation of that constraint.
- Some research has been investigating consequences of allowing table entries to be non-atomic (structured). Such systems are commonly referred to as **NF^2 DBMS** ($NF^2 = NFNF =$ Non-First Normal Form).

Non First Normal Form

NF^2 relation ($\neg$ 1NF) aka. *Nested* Relation:

COURSES

CRN	TITLE	TAUGHT_BY	STUDENTS	
			FNAME	LNAME
22332	Databases I	Grust	John	Smith
			Ann	Miller
31822	Graph Theory	Klotz	FNAME	LNAME
			Ann	Miller

N.B. A discussion of 1NF does not really belong here (1NF is not based on functional dependencies at all).

“Structured” attribute values

It is **not** a violation of 1NF, if

1. a table entry contains values of a domain that exhibits an internal structure (*e.g.*, all CRN attended by a student collected in a `CHAR(100)` attribute separated by spaces),
2. related information is represented by several columns for a single row (*e.g.*, columns `DEGREE1`, `DEGREE2`, `DEGREE3` to represent the degrees a person has obtained).

✎ These are signs of bad design anyway.

Why?

Functional Dependencies (FDs)

An example of a **functional dependency (FD)** in our latest example table is

$$\texttt{INAME} \rightarrow \texttt{PHONE} \quad .$$

Intuition
Whenever two rows of a relation agree in the instructor name INAME, they **must** also agree in the PHONE column values.

Example

COURSES

CRN	TITLE	INAME	PHONE
22268	Databases I	Grust	7111
42232	Functional Programming	Grust	7111
31822	Graph Theory	Klotz	2418

The meaning of FDs

- The real-world reason for the validity of INAME $\rightarrow$ PHONE is that the contact phone number **only depends on the instructor**, not on other course data.
- This FD is read as
  "INAME *{functionally, uniquely} determines* PHONE."
- Also: "INAME *is a determinant for* PHONE."
  Saying that A is a **determinant** for B is actually stronger than the FD $A \rightarrow B$ since A must be minimal and $A \neq B$ (see below).
- A determinant is like a **partial key:** it uniquely determines *some* attributes, but not *all* in general.

FDs and keys

- A **key** uniquely determines **all** attributes of its relation. In the example, the FDs CRN $\rightarrow$ TITLE, CRN $\rightarrow$ INAME, CRN $\rightarrow$ PHONE hold, too.
  There will never be two distinct rows with the same CRN*, so the FD condition is trivially satisfied.*
- The FD INAME $\rightarrow$ TITLE is **not** satisfied, by the sample table (and likely in reality). At least two rows agreeing on INAME disagree on TITLE

Example

COURSES

CRN	TITLE	INAME	PHONE
22268	Databases I	Grust	7111
42232	Functional Programming	Grust	7111
31822	Graph Theory	Klotz	2418

Notation for FDs

- In general, an FD takes the form

$$\{A_1, \ldots, A_n\} \rightarrow \{B_1, \ldots, B_m\} \quad .$$

- In discussing FDs, typically the focus is on a **single relation** R. All A_i, B_j are attributes of the same relation R. In some cases, we may make the context explicit by writing

$$\{A_1, \ldots, A_n\} \underset{R}{\longrightarrow} \{B_1, \ldots, B_m\} \quad .$$

- In most cases, we omit the (curly) set braces and simply write

$$A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m \quad .$$

Formally

Definition: Functional Dependency

The **functional dependency (FD)**

$$A_1, \ldots A_n \rightarrow B_1, \ldots B_m$$

holds for a relation R, if and only if for every legal database state I, and for all tuples $t, u \in I(R)$:

$$\begin{aligned} & t.A_1 = u.A_1 \wedge \cdots \wedge t.A_n = u.A_n \\ \Rightarrow \quad & t.B_1 = u.B_1 \wedge \cdots \wedge t.B_m = u.B_m \quad . \end{aligned}$$

Sometimes, we will also talk about FDs holding in particular DB states I, but it should be understood that FDs—as constraints—are used to restrict *all possible* (legal) states.

Canonical form of FDs

- An FD with m attributes on the right-hand side (rhs)

$$A_1, \ldots A_n \rightarrow B_1, \ldots B_m$$

  is **equivalent** to the m FDs:

$$\begin{array}{lcl} A_1, \ldots, A_n & \rightarrow & B_1 \\ \vdots & & \vdots \\ A_1, \ldots, A_n & \rightarrow & B_m \end{array}$$

✎ Prove this!

- Thus, in the following discussion it suffices to consider FDs with a single attribute on the rhs.

FDs are constraints

For the DB design process, the only interesting FDs are those that **hold for all possible database states.**

- FDs are **constraints** (like keys).

Non-interesting FDs

COURSES

CRN	TITLE	INAME	PHONE
22268	Databases I	Grust	7111
42232	Functional Programming	Grust	7111
31822	Graph Theory	Klotz	2418

In this example state, the FD TITLE $\rightarrow$ CRN "holds". But this is probably not true in general (it is a task of DB design to verify whether this is mere coincidence).

FDs and database design

- There are tools for analyzing example database states for possible FDs, and then asking the DB designer whether these FDs hold in general.
- If an FD (or any other constraint) does not hold in the example state, it certainly cannot hold in general.
- Database design based on normal forms requires the DB designer to **collect all FDs that generally hold in the mini-world.** This is a design task that cannot be automated.
  *Actually, it is sufficient to collect a representative subset of FDs which **implies** all valid FDs (see below).*

FDs vs. keys (1)

FDs are a generalization of keys

$$A_1, \ldots, A_n \text{ is a key of relation } R(\underline{A_1, \ldots, A_n}, B_1, \ldots, B_m)$$
$$\Downarrow$$
$$A_1, \ldots, A_n \rightarrow B_1, \ldots B_m \text{ holds.}$$

Given the FDs for a relation R, one can **derive a key for** R**,** *i.e.*, a set of attributes $A_1, \ldots, A_n$ that functionally determines the other attributes of R (see below).

FDs and keys (2)

Conversely, FDs can be explained with keys. FDs are "keys" for some columns, *e.g.*, the FD INAME $\rightarrow$ PHONE means that INAME is key of[9]

$$\pi_{\texttt{INAME},\texttt{PHONE}}(\texttt{COURSES}) \quad \equiv$$

INAME	PHONE
Grust	7111
Klotz	2418

An FD $A \rightarrow B$ holds in the original relation R (here: COURSES*) if the projection result $\pi_{A,B}(R)$ contains no rows with duplicate A values.*

FDs are *partial keys*. It is the **goal of DB normalization to turn all FDs into real keys[10].**

[9] "$\pi_A(R)$" is the relational algebra notation for a projection (`SELECT DISTINCT` A `FROM` R)

FDs describe functions

- Any relation *R* describes a partial **function** (given in the form of a lookup table):
  - function input are the values of the key attributes of *R*,
  - function output are the values of the remaining attributes.
- In the example table COURSES, **another function** is represented in the very same table: this function maps instructor names to phone numbers.
  This "embedded" function is partial, too: its domain is the active domain of the INAME *attribute.*

Computable functions

FDs indicate the presence of a function, but sometimes we even know **how** to compute the indicated function.

- Example: BIRTHDAY and AGE are represented in the same table. Then, of course, BIRTHDAY $\rightarrow$ AGE holds.
- We even know how to compute AGE given BIRTHDAY (and the current date). Clearly, AGE is redundant.
- In such cases, database normalization theory will point out the potential redundancy, but will not be able to actually remove it.

Example (1)

The following table holds information about "DB textbook classics" and their authors:

Textbooks

BOOKS

AUTHOR	NO	TITLE	PUBLISHER	ISBN
Elmasri	1	Fund. of DBS	Addison-W.	0805317554
Navathe	2	Fund. of DBS	Addison-W.	0805317554
Silberschatz	1	DBS Concepts	Mc-Graw H.	0471365084
Korth	2	DBS Concepts	Mc-Graw H.	0471365084
Sudarshan	3	DBS Concepts	Mc-Graw H.	0471365084

A book may have multiple authors, one author per row. Attribute NO is used to track authorship sequence (which may not be alphabetical).

... what might be useful/correct FDs in this scenario?

Example (2)

- The ISBN uniquely identifies a single book. Thus, the following FD holds

$$\texttt{ISBN} \rightarrow \texttt{TITLE}, \texttt{PUBLISHER} \ .$$

  — Equivalently, we could use two FDs

$$\begin{aligned} &\texttt{ISBN} \rightarrow \texttt{TITLE} \\ &\texttt{ISBN} \rightarrow \texttt{PUBLISHER} \ . \end{aligned}$$

- Since a book may have several authors, the following FD does **not** hold

$$\texttt{ISBN} \rightarrow \texttt{AUTHOR}$$

Example (3)

- One author can write many books, thus the following FD may **not** be assumed, even though it happens to hold in the given example DB state:

$$\texttt{AUTHOR} \rightarrow \texttt{TITLE}$$  (not true in general)

- It is possible that there are books with the same title but different authors and different publishers. So, TITLE determines no other attribute.

Example (4)

- For every book (we now know: books are identified by ISBN), there can only be one first (second, third, ...) author:

$$\texttt{ISBN}, \texttt{NO} \rightarrow \texttt{AUTHOR} \quad .$$

- At first glance, the author of any given book is also uniquely assigned a position in the authorship sequence:

$$\texttt{ISBN}, \texttt{AUTHOR} \rightarrow \texttt{NO}$$  (questionable)

  However, this is violated by an author list like *Smith & Smith*.

Example (5)

- We might also be tempted to assume the FD

$$\texttt{PUBLISHER}, \texttt{TITLE}, \texttt{NO} \rightarrow \texttt{AUTHOR}$$  (questionable)

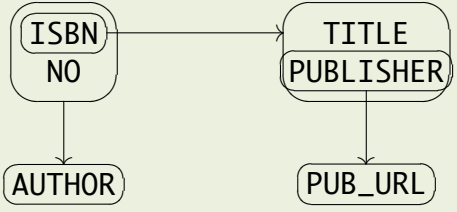
  If, in a new book edition, the authorship sequence changes, we are in trouble.

- During DB design, only unquestionable conditions should be used as FDs.
  DB normalization alters the **table structure** depending on the specified FDs. If it turns out later that an FD indeed does not hold, it is *not* sufficient to simply remove a constraint (*e.g.*, a key) from the schema with an `ALTER TABLE` SQL statement. Instead, tables will be created or deleted, data will be copied, and application programs may have to be changed.

Example (6)

A set of FDs can be visualized as a *hypergraph*[11] (PUB_URL has been added to make the example more interesting).

FD hypergraph



[11] In a hypergraph, edges connect **sets of nodes.**

FD quiz

✎ Homework grades DB

1. Which FDs should hold for this table in general?
2. Identify an FD which holds in this state but not in general!

HOMEWORK_RESULTS

STUD_ID	FIRST	LAST	EX_NO	POINTS	MAX_POINTS
100	Andrew	Smith	1	9	10
101	Dave	Jones	1	8	10
102	Maria	Brown	1	10	10
101	Dave	Jones	2	11	12
102	Maria	Brown	2	10	12

Implication of FDs

Notice the existence of FDs we're not really interested in, *e.g.*,

- CRN $\rightarrow$ PHONE is a consequence of known facts, since we already knew CRN $\rightarrow$ INAME and INAME $\rightarrow$ PHONE.
  Whenever $A \rightarrow B$ and $B \rightarrow C$ hold, $A \rightarrow C$ is automatically satisfied.
  (If f, g are functions, then $g \circ f$ is a function, too.)
- PHONE $\rightarrow$ PHONE holds, but this is not very interesting.
  FDs of the form $A \rightarrow A$ always hold (for every DB state); they're called "trivial" FDs.

Definition: Implication of FDs

A set of FDs $\{\alpha_1 \rightarrow \beta_1, \ldots, \alpha_n \rightarrow \beta_n\}$ **implies** an FD $\alpha \rightarrow \beta$, if and only if every DB state that satisfies $\alpha_i \rightarrow \beta_i$, $1 \leqslant i \leqslant n$, also satisfies $\alpha \rightarrow \beta$.

Computing implied FDs

The DB designer is normally not interested in all FDs that hold, but only in a **representative FD set** that implies all other FDs.

The following rules allow us to compute all implied FDs $\mathcal{F}^+$ (known as the "transitive closure of $\mathcal{F}$") from a given starting set of FDs $\mathcal{F}$

Σ Armstrong Axioms

- **Reflexivity**
  If $\beta \subseteq \alpha$, then $\alpha \to \beta$.
- **Augmentation**
  If $\alpha \to \beta$, then $\alpha \cup \gamma \to \beta \cup \gamma$.
- **Transitivity**
  If $\alpha \to \beta$ and $\beta \to \gamma$, then $\alpha \to \gamma$.

N.B. These are not really *axioms*, since they follow from the definition of FDs.

Checking whether some FD is implied

The Armstrong axioms can be used to check whether $\alpha \to \beta \in \mathcal{F}^+$ for a given set of FDs $\mathcal{F}$.

However, a simpler (*i.e.*, more efficient) way is to compute the **cover** α^+ of α and then to check if $\beta \subseteq \alpha^+$.

Definition: Cover

The **cover** α^+ of a set of attributes α is the set of all attributes B that are uniquely determined by the attributes α (with respect to a given FD set $\mathcal{F}$)

$$\alpha^+ := \{B \mid \mathcal{F} \text{ implies } \alpha \to B\}$$

If necessary, write $\alpha^+_{\mathcal{F}}$ to make the dependence on $\mathcal{F}$ explicit.

Corollary

A set of FDs $\mathcal{F}$ implies $\alpha \to \beta$, if and only if $\beta \subseteq \alpha^+_{\mathcal{F}}$.

Example (1)

Compute $\{\texttt{ISBN}\}^+$

Consider the following FDs:

$$\begin{aligned} \texttt{ISBN} &\rightarrow \texttt{TITLE}, \texttt{PUBLISHER} \\ \texttt{ISBN}, \texttt{NO} &\rightarrow \texttt{AUTHOR} \\ \texttt{PUBLISHER} &\rightarrow \texttt{PUB\_URL} \end{aligned}$$

1. We start with $x = \{\texttt{ISBN}\}$.
2. The FD $\texttt{ISBN} \rightarrow \texttt{TITLE}, \texttt{PUBLISHER}$ has a lhs which is completely contained in the current attribute set x.
3. Extend x by the FD's rhs attribute set, giving $x = \{\texttt{ISBN}, \texttt{TITLE}, \texttt{PUBLISHER}\}$.

Example (2)

Compute $\{\texttt{ISBN}\}^+$ (cont'd)

4. Now the FD PUBLISHER $\rightarrow$ PUB_URL becomes applicable.
5. Add rhs attribute set of the FD to the current attribute set x, giving $x = \{\texttt{ISBN}, \texttt{TITLE}, \texttt{PUBLISHER}, \texttt{PUB\_URL}\}$.
6. The FD ISBN, NO $\rightarrow$ AUTHOR is still not applicable (attribute NO $\notin x$).
7. No further way to extend set x, the algorithm returns
$$\{\texttt{ISBN}\}^+ = \{\texttt{ISBN}, \texttt{TITLE}, \texttt{PUBLISHER}, \texttt{PUB\_URL}\}$$
8. We may now conclude, *e.g.*, ISBN $\rightarrow$ PUB_URL.

Determining keys (1)

Given a set of FDs and the set of all attributes $\mathcal{A}$ of a relation R, one can determine all possible keys of R:

$$\alpha \subseteq \mathcal{A} \text{ is key of } R \iff \alpha^+ = \mathcal{A} \ .$$

Remember: Normally, we are interested in **minimal keys** only.

A superset of a key is a key again ($\{\texttt{ISBN}, \texttt{NO}\}$ is key for relation BOOKS and so is $\{\texttt{ISBN}, \texttt{NO}, \texttt{TITLE}\}$). We thus usually require that every $A \in \alpha$ is vital, *i.e.*, $(\alpha - \{A\})^+ \neq \mathcal{A}$.

Determining keys (2)

1. Start iterative key construction with the empty attribute set ($x = \emptyset$).
2. To avoid non-minimal keys, make sure the set x never contains a rhs B of FD $\alpha \rightarrow B$ when x already contains the lhs, *i.e.*, $\alpha \subseteq x$.
   W.l.o.g., we assume the rhs of all FDs to consist of a single attribute only.
3. As long as x does not yet determine all attributes of R (*i.e.*, $x^+ \neq \mathcal{A}$), choose any of the remaining attributes $X \in \mathcal{A} - x^+$.
4. Now the process splits (need to follow both alternatives):
   a) Simply add X to x.
   b) For each FD $\alpha \rightarrow X$, add α to x.

   The search for the key branches. We need to backtrack to such a branch if
   — we run into a dead-end (determine a non-minimal key), or
   — we have found a key and want to look for an alternative minimal key.

Key quiz

✎ Determine keys for a relation

RESULTS

STUD_ID	EX_NO	POINTS	MAX_POINTS
100	1	9	10
101	1	8	10
102	1	10	10
101	2	11	12
102	2	10	12

$$\mathcal{F} = \left\{ \begin{array}{rcl} \texttt{STUD\_ID}, \texttt{EX\_NO} & \rightarrow & \texttt{POINTS} \\ \texttt{EX\_NO} & \rightarrow & \texttt{MAX\_POINTS} \end{array} \right\}$$

1. Does $\mathcal{F}$ imply $\texttt{STUD\_ID}, \texttt{EX\_NO} \rightarrow \texttt{MAX\_POINTS}$?
2. Determine a key for relation RESULTS.

Determinants

Definition: Determinant (Non-trivial FD)

The attribute set $A_1, \ldots, A_n$ is called a **determinant** for attribute set $B_1, \ldots, B_m$, if and only if

1. the FD $A_1, \ldots, A_n \rightarrow B_1, \ldots B_m$ holds, and
2. the lhs is minimal, *i.e.*, whenever any A_i is removed then $A_1, \ldots, A_{i-1}, A_{i+1}, A_n \rightarrow B_1, \ldots B_m$ does not hold, and
3. the lhs and rhs are distinct, *i.e.*, $\{A_1, \ldots, A_n\} \neq \{B_1, \ldots, B_m\}$.

Consequences of bad DB design

- It is normally a severe sign of **bad DB design**, if a table contains an FD (encodes a partial function) that is **not implied by a key** (*e.g.*, INAME $\rightarrow$ PHONE).
- Among other problems, this **leads to the redundant storage of certain facts** (here: phone numbers).

Redundant data storage

COURSES

CRN	TITLE	INAME	PHONE
22268	Databases I	Grust	7111
42232	Functional Programming	Grust	7111
31822	Graph Theory	Klotz	2418

Problems with redundancy

- If a schema permits redundant data, additional **constraints** need to be specified to ensure that the redundant copies indeed agree.
- In the example, this constraint is precisely the FD $\texttt{INAME} \rightarrow \texttt{PHONE}$.
- **General FDs**, however, are *not* standard constraints of the relational model and thus **unsupported by RDBMS.**
- The solution is to transform FDs into **key constraints**. This is what **DB normalization** tries to do.

Update anomalies

- When a **single mini-world fact** needs to be changed (*e.g.*, a change in phone number for `Grust`), **multiple tuples** must be updated. This complicates application programs.
- Redundant copies potentially get out of sync and it is impossible/hard to identify the correct information.
- Application programs unexpectedly receive multiple answers when a single answer was expected.

`SELECT INTO` breaks due to unexpected result

```
SELECT PHONE
  INTO :phone
FROM   COURSES
WHERE  INAME = 'Grust'
```

Insertion anomalies

- The phone number of a new instructor cannot be inserted into the database until it is known what course(s) she/he will teach.
- Since CRN is a key, it cannot be set to NULL.

- Such insertion anomalies arise when **unrelated concepts** (here: courses and instructors' phone numbers) are **stored together in a single table.**
  Remember: a relational table should encode a *single* (partial) function.

Deletion anomalies

- When the last course of an instructor is deleted, his/her phone number is lost. *Even if* CRN *could be set to* NULL*: it would be strange that all courses held by an instructor may be deleted but the last.*
- Again: such deletion anomalies arise when **unrelated concepts** (here: courses and instructors' phone numbers) are **stored together in a single table.**

Boyce-Codd Normal Form

A relation R is in **Boyce-Codd Normal Form (BCNF)**, if and only if all its FDs are already implied by its key constraints.

Boyce-Codd Normal Form

For relation R in BCNF and any FD $A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m$ of R one of the following statements hold:

1. the FD is trivial, *i.e.*, $\{B_1, \ldots, B_m\} \subseteq \{A_1, \ldots, A_n\}$, or
2. the FD follows from a key, because $\{A_1, \ldots, A_n\}$ or a subset of it is already a key of R.

- BCNF $\Longleftrightarrow$ Every determinant is a possible key of R.
- If a relation R is in BCNF, ensuring its key constraints automatically satisfies all of R's FDs.
- The **three anomalies** (update/insertion/deletion) **do not occur.**

BCNF examples (1)

BCNF counter example

The relation

$$\texttt{COURSES}\ (\underline{\texttt{CRN}},\ \texttt{TITLE},\ \texttt{INAME},\ \texttt{PHONE})$$

with the FDs

$$\begin{aligned} \texttt{CRN} &\rightarrow \texttt{TITLE}, \texttt{INAME}, \texttt{PHONE} \\ \texttt{INAME} &\rightarrow \texttt{PHONE} \end{aligned}$$

is **not in BCNF** because the FD $\texttt{INAME} \rightarrow \texttt{PHONE}$ is not implied by a key:

- INAME is not a key of the entire relation,
- the FD is not trivial.

However, without the attribute PHONE (and the second FD) the relation COURSES ($\underline{\texttt{CRN}}$, TITLE, INAME) is in BCNF.

BCNF examples (2)

BCNF example

Each course meets once per week in a dedicated room:

$$\texttt{CLASS (CRN, TITLE, WEEKDAY, TIME, ROOM)}$$

The relation thus satisfies the following FDs (plus implied ones):

$$\texttt{CRN} \rightarrow \texttt{TITLE, WEEKDAY, TIME, ROOM}$$
$$\texttt{WEEKDAY, TIME, ROOM} \rightarrow \texttt{CRN}$$

The keys of CLASS are

- CRN, and
- WEEKDAY, TIME, ROOM.

Both FDs are implied by keys (in this case, their lhs even coincide with the keys), thus CLASS **is in BCNF.**

BCNF examples (3)

BCNF example

Consider the relation PRODUCT (NO, NAME, PRICE) and the following FDs:

$$\text{NO} \rightarrow \text{NAME} \qquad \text{PRICE}, \text{NAME} \rightarrow \text{NAME}$$
$$\text{NO} \rightarrow \text{PRICE} \qquad \text{NO}, \text{PRICE} \rightarrow \text{NAME}$$

This relation **is in BCNF:**

- The two left FDs indicate that NO is a key. Both FDs are thus implied by a key.
- The third FD is trivial (and may be ignored).
- The lhs of the last FD contains a key and is thus also implied by a key.

BCNF quiz

✎ Check for BCNF!

1. Is RESULTS (STUD_ID, EX_NO, POINTS, MAX_POINTS) with the following FDs in BCNF?

$$\begin{aligned} \texttt{STUD\_ID}, \texttt{EX\_NO} &\rightarrow \texttt{POINTS} \\ \texttt{EX\_NO} &\rightarrow \texttt{MAX\_POINTS} \end{aligned}$$

2. Is INVOICE (INV_NO, DATE, AMOUNT, CUST_NO, CUST_NAME) with the following FDs in BCNF?

$$\begin{aligned} \texttt{INV\_NO} &\rightarrow \texttt{DATE}, \texttt{AMOUNT}, \texttt{CUST\_NO} \\ \texttt{INV\_NO}, \texttt{DATE} &\rightarrow \texttt{CUST\_NAME} \\ \texttt{CUST\_NO} &\rightarrow \texttt{CUST\_NAME} \\ \texttt{DATE}, \texttt{AMOUNT} &\rightarrow \texttt{DATE} \end{aligned}$$

Third Normal Form

- **Third Normal Form (3NF)** is slightly weaker than BCNF: if a relation is in BCNF, it is automatically in 3NF.
  The differences are small. For most practical applications, the two can be considered as equivalent.
- BCNF is a clear concept: we want no non-trivial FDs except those which are implied by a key constraint.
- In some rare scenarios, this **preservation of FDs** (see below) is lost when a relation is transformed into[12] BCNF. This never happens with 3NF.

[12] "normalized into"

Formally: 3NF

Third Normal Form (3NF)

A relation R is in **Third Normal Form (3NF)**, if and only if every FD $A_1, \ldots, A_n \rightarrow B$[13] satisfies at least one of the following conditions:

1. the FD is trivial, *i.e.*, $B \in \{A_1, \ldots, A_n\}$,or
2. the FD follows from a key, because $\{A_1, \ldots, A_n\}$ or some subset of it is already a key of R, or
3. B is a **key attribute**, *i.e.*, element of a key of R.

- The only difference to BCNF is the additional clause 3.
- A **key attribute** is one appearing in some minimal key.[14]
- 3NF means that **every determinant of a non-key attribute is a key** (see clauses 2 and 3).

[13] Assume that FDs with multiple attributes on rhs have been expanded.

[14] Minimality requirement is important: otherwise all attributes were key attributes.

Second Normal Form

The **Second Normal Form (2NF)** is only interesting for historical reasons.

Definition: Second Normal Form (2NF)

A relation is in **Second Normal Form (2NF)** if and only if every non-key attribute depends on the **complete** key.

- If a relation is in 3NF or BCNF, it is automatically in 2NF.
- 2NF is "too weak", *e.g.*, the COURSES example actually satisfies 2NF (although it exhibits anomalies).

Partial functional dependency from the key

A relation violates 2NF, if and only if some non-key attribute B only *partially* depends on the key. Formally:

If the FD $A_1, \ldots, A_n \rightarrow B$ holds in R, such that

1. $A_1, \ldots, A_n$ is a **strict subset** of a minimal key, and
2. B is not contained in any minimal key (*i.e.*, is a non-key attribute),

then R is **not in 2NF**.

Relation not in 2NF

The homework results table

```
RESULTS (STUD_ID, EX_NO, POINTS, MAX_POINTS)
```

is **not in 2NF:** MAX_POINTS is already uniquely determined by EX_NO, *i.e.*, a proper part of a key.

Splitting relations

If a table R is not in some target normal form, we can **split** it into two tables (**decomposition**). The violating FD determines how to split.

Splitting "along an FD"

Remember: the FD INAME $\rightarrow$ PHONE is the reason why table

COURSES (<u>CRN</u>, TITLE, INAME, PHONE)

violates BCNF.

Split the table into

INSTRUCTORS (<u>CRN</u>, TITLE, INAME)

PHONEBOOK (<u>INAME</u>, PHONE)

If the FD $A_1, \ldots, A_n \rightarrow B_1, \ldots B_m$ violates BCNF, create a new relation $S(\underline{A_1}, \ldots, \underline{A_n}, B_1, \ldots, B_m)$ and remove $B_1, \ldots, B_m$ from the original relation R.

Lossless decomposition

Σ Decomposition Theorem

The split of relations is guaranteed to be **lossless**, if the intersection of the attributes of the new tables is a key of at least one of them.
Formally, let $sch(R) = \alpha \cup \beta$:

$$(\alpha \cap \beta \to \alpha \ \vee \ \alpha \cap \beta \to \beta) \implies \pi_\alpha(R) \bowtie \pi_\beta(R) \equiv R$$

N.B. This is a sufficient ("$\Rightarrow$"), but not a necessary ("$\not\Leftarrow$") condition!

"Lossy" decomposition

Original table (key A, B, C)

A	B	C
a_{11}	b_{11}	c_{11}
a_{11}	b_{11}	c_{12}
a_{11}	b_{12}	c_{11}

Decomposition

R_1

A	B
a_{11}	b_{11}
a_{11}	b_{12}

R_2

A	C
a_{11}	c_{11}
a_{11}	c_{12}

"Reconstruction" $R_1 \bowtie R_2$

A	B	C
a_{11}	b_{11}	c_{11}
a_{11}	b_{11}	c_{12}
a_{11}	b_{12}	c_{11}
a_{11}	b_{12}	c_{12}

What is actually “lost”?

Somewhat surprising nomenclature!

- As can be seen in the counter example, no *tuples* got lost...
- Quite on the contrary! **We obtained more tuples from the join.**
- What got lost, was the **information, which attribute values actually belonged together** in the original table!
  Less information might mean more tuples! (cf. entropy)

In case of a *lossless decomposition*, the join connects tuples depending on the attribute (values) in the intersection. If these values uniquely identify tuples in the other relation, we do not lose information.

Repeated decomposition

Lossless split condition satisfied

$$\{\texttt{CRN}, \texttt{TITLE}, \texttt{INAME}\} \cap \{\texttt{INAME}, \texttt{PHONE}\} = \{\texttt{INAME}\}$$

- All splits initiated by the above method for transforming relations into BCNF satisfy the condition of the decomposition theorem.
- It is **always possible** to transform a relation into BCNF by lossless splitting (if necessary, split repeatedly).

Full decomposition

However, not every lossless split is reasonable.

Example

STUDENTS		
SSN	FIRST_NAME	LAST_NAME
111-22-3333	John	Smith
123-45-6789	Maria	Brown

Splitting STUDENTS into STUD_FIRST (SSN, FIRST_NAME) and STUD_LAST (SSN, LAST_NAME) is lossless, but

- the split is *not* necessary to enforce a normal form,
- only requires costly joins in subsequent queries.

  N.B. *Some "unnecessary" splitting, or even full splitting down to* ***binary tables*** *on the* ***physical DB level****, however, may lead to high-performance DBMS implementations.*

Preserving FDs

- Besides losslessness, a property a good decomposition of a relation should guarantee is the **preservation of FDs.**
- The problem is that an FD can refer to attributes of a single relation only (at least within a DBMS that supports key constraints only, not general FDs).
- When decomposing a relation, there might be FDs that can no longer be expressed (in a single relation).

Definition: Dependency Preservation

For a relation R, let $\text{sch}(R)$ denote the set of attributes of R. When R is decomposed into $R_1, \ldots R_n$, such that $\bigcup_i \text{sch}(R_i) = \text{sch}(R)$, let $\mathcal{F}_i \subseteq \mathcal{F}^+$ denote the subset of FDs mentioning attributes in $\text{sch}(R_i)$.
The decomposition of R into $R_1, \ldots R_n$is **dependency preserving**, iff $\left(\bigcup_i(\mathcal{F}_i)\right)^+ = \mathcal{F}^+$.

FDs can get lost by decomposition

Example

Consider ADRESSES (STREET_ADDR, CITY, STATE, ZIP) with FDs

$$\text{STREET\_ADDR}, \text{CITY}, \text{STATE} \rightarrow \text{ZIP}$$
$$\text{ZIP} \rightarrow \text{STATE}$$

The second FD violates BCNF and would lead to a split into

- ADDRESSES1 (STREET_ADDR, CITY, ZIP) and
- ADDRESSES2 (ZIP, STATE).

But now the first FD can no longer be expressed.

- Most designers would not split the table (it's in 3NF but not in BCNF).
- If there are many addresses with the same ZIP code, there will be significant redundancy. If you split, queries will involve more joins.
- If this were a DB for a mailing company, a separate table of ZIP codes might be of interest on its own. Then the split looks feasible.

Minimal cover

For a given FD set $\mathcal{F}$, we compute an equivalent **minimal** (canonical) set $\mathcal{F}'$ with the same transitive closure, $(\mathcal{F}')^+ = \mathcal{F}^+$, by exhaustively applying:

Determine the Minimal Cover $\mathcal{F}'$

1. Canonicalize right-hand sides

   Replace every FD $\alpha \to B_1, \ldots, B_m$ by the corresponding set of FDs $\alpha \to B_i$, $1 \leqslant i \leqslant m$.

2. Minimize left-hand sides

   For each FD $A_1, \ldots, A_n \to B$ and each $i = 1, \ldots, n$, check if already $B \in \{A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_n\}^+_{\mathcal{F}}$ .If so, continue with $\mathcal{F} \leftarrow \mathcal{F}^-$, where $\mathcal{F}^- = (\mathcal{F} - \{A_1, \ldots, A_n \to B\}) \cup \{A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_n \to B\}$.

3. Remove implied FDs

   For each FD $\alpha \to B$, check if $B \in \alpha^+_{\mathcal{F}^-}$, where $\mathcal{F}^- = \mathcal{F} - \{\alpha \to B\}$.

   If so, continue with $\mathcal{F} \leftarrow \mathcal{F}^-$.

3NF Synthesis Algorithm

The following algorithm produces a lossless decomposition of a given relation into 3NF relations that preserves the FDs.

3NF Synthesis Algorithm

1. Compute a minimal cover of the given FDs $\mathcal{F}'$ (see above).
2. For each lhs α of an FD in $\mathcal{F}'$ create a relation with attributes $\mathcal{A} = \alpha \cup \{B \mid \alpha \to B \in \mathcal{F}'\}$.
3. If none of the relations constructed in step 2 contains a key of the original relation R, add one relation containing the attributes of a key of R.
4. For any two relations $R_{1,2}$ constructed in steps 2 and 3, if the schema of R_1 is contained in the schema of R_2, discard R_1.

Summary

- Tables should *not* contain FDs other than those implied by the keys (*i.e.*, all tables should be in **BCNF**).
  Such violating FDs indicate the combination of pieces of information which should be stored separately (presence of an embedded function). This leads to redundancy.
- A relation may be **normalized** into BCNF by splitting it.
  - We can *always* obtain a *lossless BCNF* decomposition (*not* necessarily FD *preserving*).
  - We can *always* obtain a *lossless and FD-preserving 3NF* decomposition.
- Sometimes it may make sense to avoid a split (and thus to violate BCNF). The DB designer has to carefully resolve such scenarios, incorporating application or domain knowledge.

Beyond BCNF & FDs

- The development of BCNF has been guided by a particular type of constraint: **Functional** Dependencies (FDs).
- The goal of normalization into BCNF/3NF is to
  - eliminate the redundant storage of data that follows from these constraints, and to
  - transform tables such that these constraints are automatically enforced by means of keys.
- However, there are **further types of constraints** which are also useful during DB design.

From Functional to Multi-Valued Dependencies

- The condition in the decomposition theorem is only
  - **sufficient** (it guarantees losslessness),
  - but **not necessary** (a decomposition might be lossless, even if the condition is not satisfied).
- **Multi-valued dependencies (MVDs)** are constraints that give a **necessary and sufficient** condition for lossless decomposition.
- MVDs lead to the **Fourth Normal Form (4NF).**

Fourth Normal Form

- 4NF intuitively...
  Whenever it is possible to split a table (i.e., the decomposition is lossless) and the split is not superfluous, then split!
- Still shorter...
  "Do not store unrelated information in the same relation."
- In practice, it is very rare that a relation violates 4NF but not BCNF.
  Such cases occur, however (e.g., when merging two binary relationships into a ternary relationship, see below).

Multi-Valued Dependencies (MVDs)

Multi-valued dependency

Suppose that every employee knows a set of programming languages as well as a set of DBMSs. Such knowledge typically is **independent.**

EMP_KNOWLEDGE

ENAME	PROG_LANG	DBMS
John Smith	C	Oracle
John Smith	C	DB2
John Smith	C++	Oracle
John Smith	C++	DB2
Maria Brown	Prolog	PostgreSQL
Maria Brown	Java	PostgreSQL

Multi-valued dependency (cont'd)

If the sets of known programming languages and DBMSs are indeed **independent** (in particular PROG_LANG $\not\rightarrow$ DBMS and DBMS $\not\rightarrow$ PROG_LANG), the table contains redundant data, and must be split:

EMP_LANG

ENAME	PROG_LANG
John Smith	C
John Smith	C++
Maria Brown	Prolog
Maria Brown	Java

EMP_DBMS

ENAME	DBMS
John Smith	Oracle
John Smith	DB2
Maria Brown	PostgreSQL

N.B. The original table is in BCNF (no non-trivial FDs)!

Observations

- Table EMP_KNOWLEDGE may only be decomposed, if the knowledge of DBMS and programming languages is **indeed independent.**
- Otherwise, decomposition leads to **loss of information.**
  If the semantics of EMP_KNOWLEDGE is that the employee knows the interface between the programming language and the DBMS (*i.e.*, the knowledge is *not* independent), we cannot deduce from the split tables whether John Smith indeed knows the C and C++ interfaces of both DBMS, Oracle and DB2.
- The **multi-valued dependency (MVD)**

$$\text{ENAME} \twoheadrightarrow \text{PROG\_LANG}$$

  indicates that the **set of values** in column PROG_LANG associated with every ENAME is **independent of all other columns.**
  The tables contains an embedded function from ENAME to **sets of** PROG_LANG.

Consequences of MVDs

Formally, ENAME $\twoheadrightarrow$ PROG_LANG holds, if, whenever two tuples agree on ENAME, one can exchange their PROG_LANG values and get two other tuples of the same table

- From the two table rows

ENAME	PROG_LANG	DBMS
John Smith	C	Oracle
John Smith	C++	DB2

and the MVD ENAME $\twoheadrightarrow$ PROG_LANG, we can conclude that the table must also contain the following rows.

ENAME	PROG_LANG	DBMS
John Smith	C++	Oracle
John Smith	C	DB2

- This expresses the **independence** of PROG_LANG for a given ENAME from the rest of the table columns.

Formal definition

Definition: Multi-Valued Dependency

A **multi-valued dependency (MVD)**

$$A_1, \ldots, A_n \twoheadrightarrow B_1, \ldots, B_m$$

is satisfied in a DB state I, if and only if for all tuples t, u with $t.A_i = u.A_i$, $1 \leqslant i \leqslant n$, there are two further tuples t', u', such that

1. t' agrees with t except that $t'.B_j = u.B_j$, $1 \leqslant j \leqslant m$, and
2. u' agrees with u except that $u'.B_j = t.B_j$, $1 \leqslant j \leqslant m$.

(*i.e.*, the values of the B_j are swapped).

MVDs always come in pairs

If ENAME $\twoheadrightarrow$ PROG_LANG holds, then ENAME $\twoheadrightarrow$ DBMS is automatically satisfied.

- More general:

  For $R(A_1, \ldots, A_n, B_1, \ldots, B_m, C_1, \ldots, C_k)$ the MVD $A_1, \ldots, A_n \twoheadrightarrow B_1, \ldots, B_m$ is equivalent to the MVD $A_1 \ldots, A_n \twoheadrightarrow C_1, \ldots, C_k$: swapping the B_j values in two tuples is the same as swapping the values for all other columns (the A_i values are identical, so swapping them has no effect).

  That is: for any subsets A, B of attributes of some relation R

$$A \twoheadrightarrow B \quad \Longleftrightarrow \quad A \twoheadrightarrow \mathrm{sch}(R) - A - B.$$

Trivial MVDs

Definition: Trivial MVD

An MVD $A_1, \ldots, A_n \twoheadrightarrow B_1, \ldots, B_m$ for a relation R is **trivial**, if and only if

1. $\{B_1, \ldots, B_m\} \subseteq \{A_1, \ldots, A_n\}$, or
   For two tuples agreeing on the A_i (and thus also on the B_j) swapping has no effect.
2. $\{A_1, \ldots, A_n\} \cup \{B_1, \ldots, B_m\}$ are all columns of R.
   In this case, swapping tuple t, u gives the tuples u, t, i.e., the same tuples again.

FDs are a special case of MVDs

If the FD $A_1, \ldots A_n \to B_1, \ldots B_m$ holds, the corresponding MVD $A_1, \ldots, A_n \twoheadrightarrow B_1, \ldots, B_m$ is trivially satisfied.

The FD means that, if tuples t, u agree on the A_i, then they also agree on the B_j. In that case, swapping has no effect (yields t, u again).

FDs are a special case of MVDs

For all relations R and subsets A, B of their attributes:

$$A \to B \quad \Longrightarrow \quad A \twoheadrightarrow B \ .$$

Implications of FDs and MVDs

Just like an FD may be implied by (*i.e.*, automatically follow from) a given set of FDs, FDs and MVDs can also follow from a set of given FDs and MVDs.

Definition: Sound & complete set of rules for FD/MVD implication

- The three Armstrong Axioms for FDs.
- If $\alpha \overset{R}{\twoheadrightarrow} \beta$ then $\alpha \overset{R}{\twoheadrightarrow} \gamma$, where γ are the remaining columns of R.
- If $\alpha_1 \twoheadrightarrow \beta_1$ and $\alpha_2 \supseteq \beta_2$ then $\alpha_1 \cup \alpha_2 \twoheadrightarrow \beta_1 \cup \beta_2$.
- If $\alpha \twoheadrightarrow \beta$ and $\beta \twoheadrightarrow \gamma$ then $\alpha \twoheadrightarrow (\gamma - \beta)$.
- If $\alpha \rightarrow \beta$, then $\alpha \twoheadrightarrow \beta$.
- If $\alpha \twoheadrightarrow \beta$ and $\beta' \subseteq \beta$ and there is γ with $\gamma \cap \beta = \emptyset$ and $\gamma \rightarrow \beta'$, then $\alpha \rightarrow \beta'$.

Fourth Normal Form

The definition of 4NF is very similar to BCNF, but with a focus on implied MVDs (not FDs).

Definition: Fourth Normal Form (4NF)

A relation R is in **Fourth Normal Form (4NF)**, if and only if all its MVDs are already implied by its key constraints.

For a relation R in 4NF and any MVD $A_1, \ldots, A_n \twoheadrightarrow B_1, \ldots, B_m$ on R's attributes, one of the following statements holds:

1. the MVD is trivial or
2. the MVD is implied by a key, *i.e.*, $A_1, \ldots, A_n$ uniquely determine all other attributes.

4NF vs. BCNF

- Since every FD is also an MVD, 4NF is stronger than BCNF (*i.e.*, if a relation is in 4NF, it is automatically in BCNF).
- The relation

  EMP_KNOWLEDGE (ENAME, PROG_LANG, DBMS)

  is an example of a relation that is in BCNF, but not in 4NF.
  The relation has no non-trivial FDs (the key consists of all attributes).
- However, it is not very common that 4NF is violated, but BCNF is not.

(Binary) Decomposition Theorem

We can now formulate a *necessary and sufficient* ("$\Leftrightarrow$") condition for lossless decomposition (into 2 relations).

Σ Theorem: Lossless binary decomposition

Splitting a relation R into two parts R_1, R_2 is lossless, if and only if the common attributes multi-determine one (or the other) part:

$$R = \underbrace{\pi_{A_1,\ldots,A_n,B_1,\ldots,B_m}(R)}_{=R_1} \bowtie \underbrace{\pi_{A_1,\ldots,A_m,C_1,\ldots,C_k}(R)}_{=R_2}$$

$$\Updownarrow$$

$$A_1, \ldots, A_n \twoheadrightarrow B_1, \ldots, B_m \quad (\text{or } A_1, \ldots, A_n \twoheadrightarrow C_1, \ldots, C_k).$$

Obtaining relations in 4NF

- Any relation can be transformed into 4NF by splitting as shown above.
  To reach 4NF, it might be necessary to split multiple times.
- So the **essence of 4NF** is:
  - If a **decomposition** into two relations **is lossless** (*i.e.*, the original relation may always be reconstructed by a join),
  - and the two resulting relations do *not* have identical keys (then the split would be superfluous),
  - ▷ then this decomposition must be done.

Beyond 4NF

- There is also a **Fifth Normal Form (5NF)**, which is very rarely used in practice. We only sketch some issues here.[15]
- 4NF handles all cases where a decomposition into **two tables** is needed.
- However, it is theoretically possible that only a decomposition into **three or more tables is lossless**, but no decomposition into two tables is lossless.
  The required decomposition thus cannot be obtained by repeatedly splitting into two tables. Instead, one needs additional tables with overlapping join columns that only serve as a filter in the reconstructing join.

[15] See, *e.g.*, *Relational Database Theory*, Atzeni, DeAntonellis; Benjamin Cummings, 1993; ISBN 0-8053-0249-2.

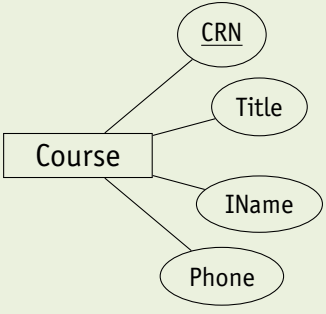
Normal Forms and ER design

- If a "good" ER schema is transformed into the relational model, the resulting tables will **satisfy all normal forms** (4NF, BCNF, 3NF, 2NF).
- A normal form violation detected in the generated relational schema indicates a flaw in the input ER schema. This needs to be corrected on the ER level.
- There is no independent normalization theory for the ER model.
- You can use "FD- (or MVD-) style" constraints among ER attributes to analyze ER schemas, though.
- Another (more traditional) way of looking at this is to ask yourself whether or not you "placed the ER attributes at the *right* constructs (entities, relationships)."

Examples (1)

Example: "FDs" in the ER model

The ER equivalent of the very first example in this chapter.



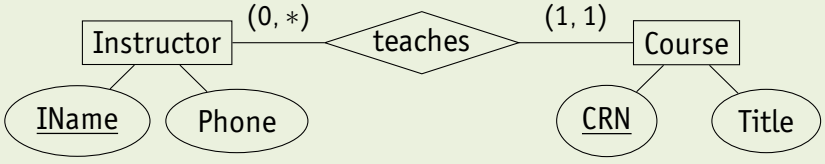
- Obviously, the FD IName $\rightarrow$ Phone leads to a violation of BCNF in the resulting table for entity Course.
- Also in the ER model, "FDs" between attributes of an entity should be implied by a key constraint.

Examples (2)

In the ER model, the solution to this problem is the "same" as in the relational model: we have to **split** the entity.[16]

Example (cont'd)

In this case, we discover that Instructor is an independent entity:

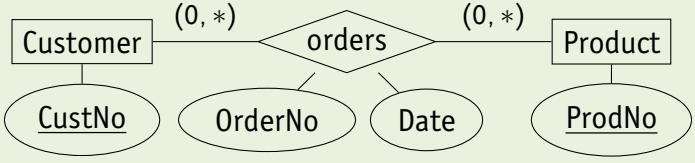


[16] You might as well say: correct the error of attaching attribute Phone to the wrong entity!

Examples (3)

"Functional dependencies" between **attributes of a relationship** always violate BCNF.

Example: BCNF violation on the ER level



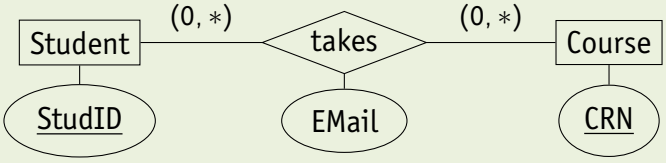
The FD OrderNo $\rightarrow$ Date violates BCNF.

The key of the table corresponding to the relationship "orders" consists of the attributes CustNo, ProdNo.

This indicates that the concept "Order" is an independent entity.

Examples (4)

Example: Wrong placement of attributes in ER



- The relationship is translated into

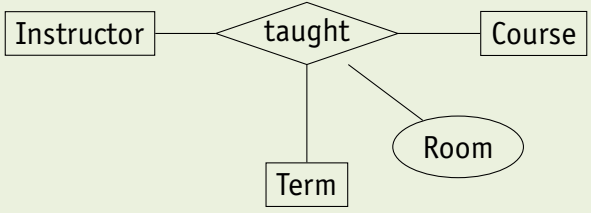
$$\texttt{TAKES (}\underline{\texttt{STUD\_ID, CRN}}\texttt{, EMAIL)}$$

- Then the FD $\texttt{STUD\_ID} \rightarrow \texttt{EMAIL}$ violates BCNF.
- Obviously, Email should rather be an attribute of Student.

Examples (5)

If an attribute of a ternary relationship depends only on two of the entities, this violates BCNF (actually, it even violates 2NF).

Example: Ternary relationships

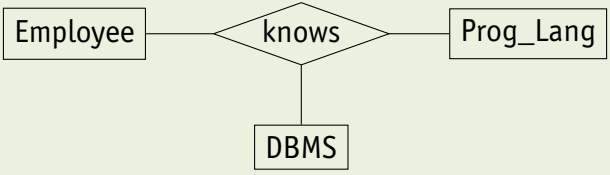


If every course is taught only once per term, then attribute Room depends only on Term and Course (but not Instructor).

Examples (6)

If **independent relationships** are combined, 4NF is violated.

Example: Independent relationships combined



If the knowledge of programming languages and DBMSs is independent, hence, use **two binary relationships** instead.

One between Employee and DBMS, one between Employee and Prog_Lang.

Normalization: Summary

- **Relational normalization** is about:
  - **Avoiding redundancy** and anomalies.
  - **Storing separate**, independent **facts** (functions) **separately.**
  - **Transforming general integrity constaints** into constraints that are supported by the DBMS: **keys**
- Relational normalization theory is mainly based on FDs, but there are other types of constraints (*e.g.*, MVDs, JDs).
- Whenever some **normal form is violated: split (decompose)** a relation "along" a violating dependency. Additional goals:
  - losslessness and
  - dependency preservation.
- Decent ER designs translate into relational schemas in a high normal form.

Denormalization

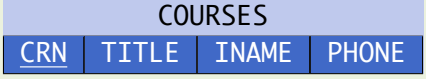
For **performance reasons**, it may be necessary to (later!) undo some effects of normalization, *e.g.*,

- store a less-normalized single table, instead of the two split high-normal-form counterparts,
- add redundant columns, ...

so as to avoid frequent/expensive join operations at query execution time.

Redundant data storage

If an application extensively accesses the phone number of instructors, performance-wise it may make sense to add column PHONE to table COURSES.



This **avoids the otherwise required joins** (on attribute INAME) between tables COURSES and PHONEBOOK.

Observations with the denormalized example schema

- Since there is still the separate table PHONEBOOK, **insertion and deletion anomalies are avoided.**
- But there will be **update anomalies** (changing a single phone number requires the update of many rows).
- The performance gain is thus paid for with
  - a more complicated application logic (*e.g.*, the need for triggers),
  - and the risk that a faulty application will turn the DB inconsistent.
- If the table COURSES and PHONEBOOK contain few tuples, it is probably a bad decision to violate BCNF here. Performance will be no problem anyway.

More denormalization options

Denormalization may not only be used to avoid joins.

- Complete **separate, redundant tables** may be created (increasing the potential for parallel DBMS operation, especially updates).
- Columns may be added which **aggregate** information in other columns/rows (*e.g.*, store the current BALANCE in a CUSTOMER table that also records PAYMENTs and INVOICEs).
  In this case, BCNF might not be violated, but the added information will be redundant nevertheless.