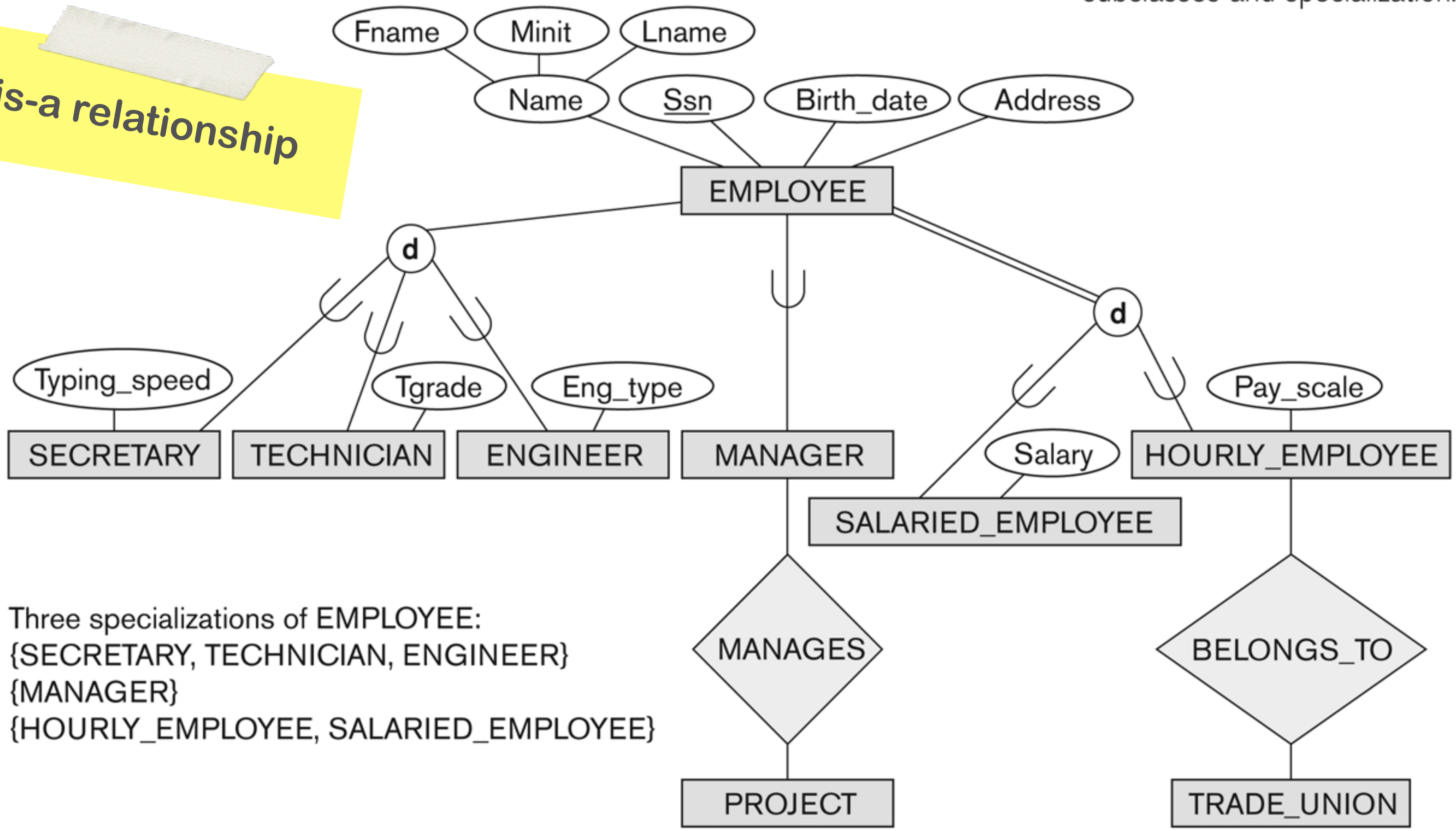
Let’s briefly review important EER inheritance concepts

Figure 4.1
EER diagram notation to represent subclasses and specialization.

is-a relationship



Basic Constraints

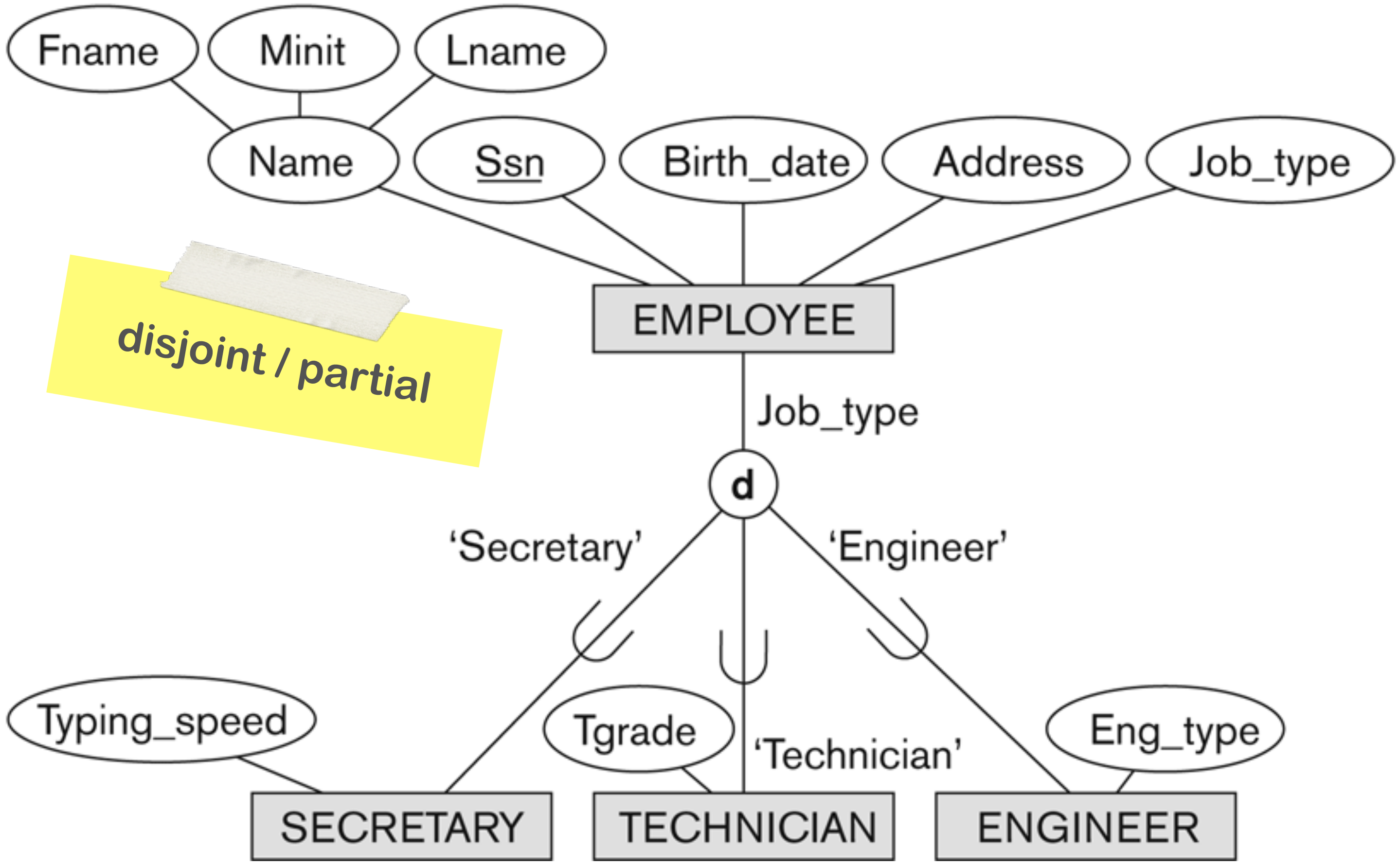
Partial / Disjoint:
- Single line / “d” in circle
- Each entity can be an instance of 0-1 subclasses

Complete / Disjoint:
- Double line / “d” in circle
- Each entity is an instance of exactly 1 subclass

Partial / Overlapping:
- Single line / “o” in circle
- Each entity can be an instance of 0-many subclasses

Complete / Overlapping:
- Double line / “o” in circle
- Each entity can be an instance of 1-many subclasses

Figure 4.4
EER diagram notation for an attribute-defined specialization on Job_type.



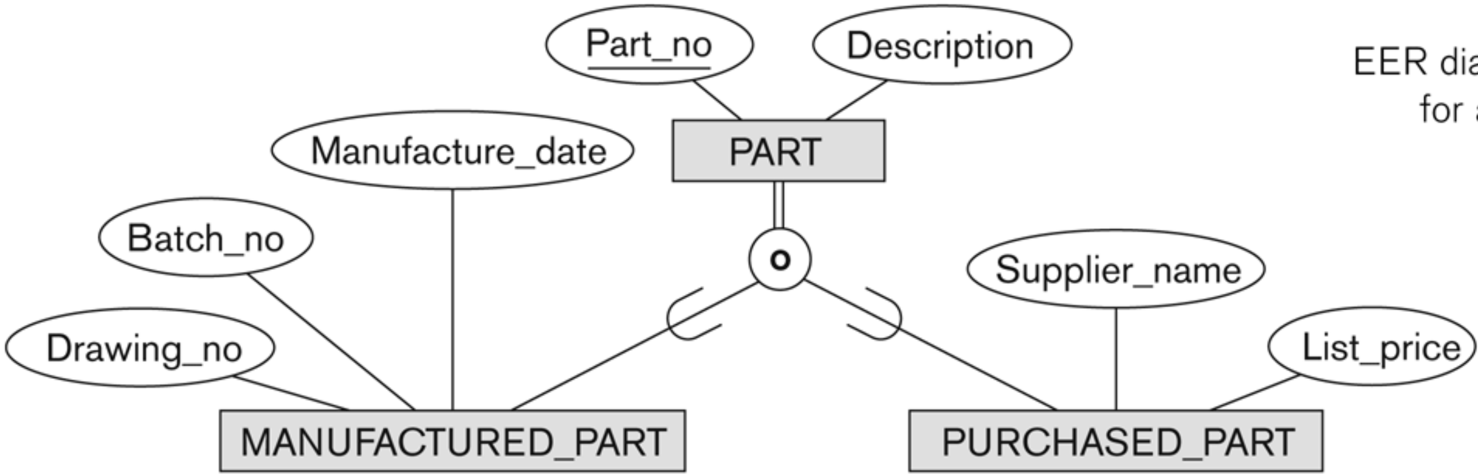


Figure 4.5
EER diagram notation for an overlapping (nondisjoint) specialization.

overlapping / total

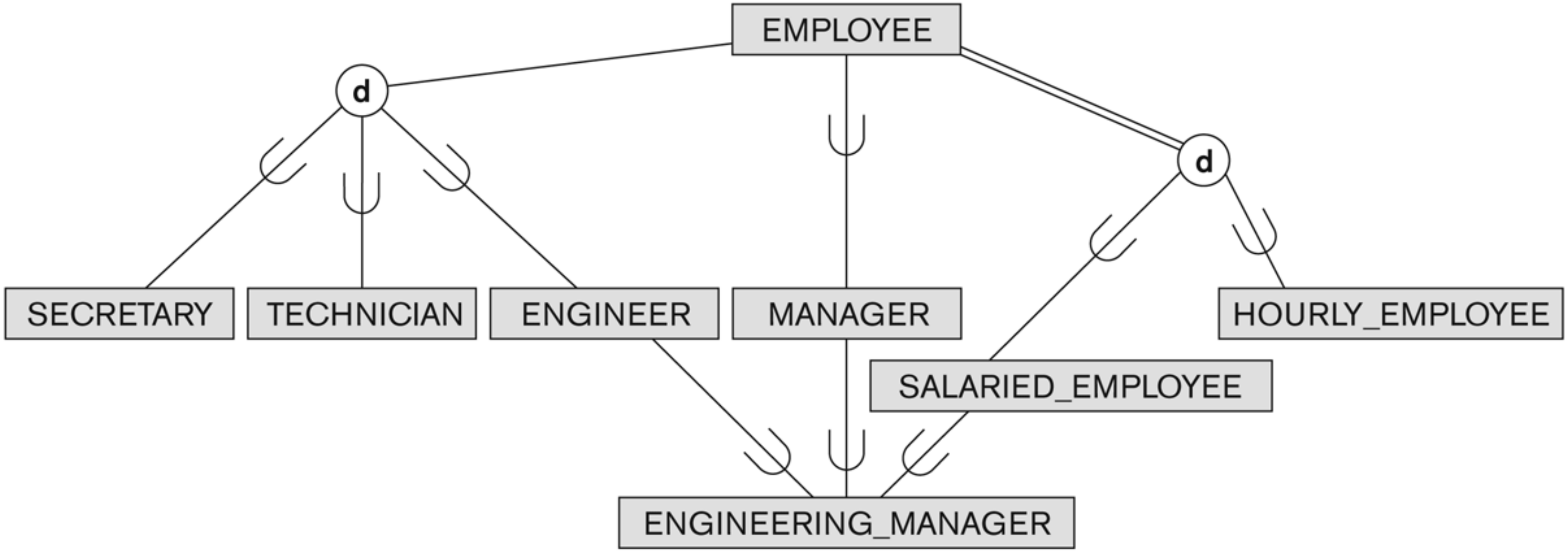


Figure 4.6
A specialization lattice with shared subclass ENGINEERING_MANAGER.

Union Types

All of the subclasses so far had a **single superclass**

We may need to model a single inheritance relationship with **more than one superclass**

Such a subclass is called a **category** or **UNION TYPE**

Example:

The owner of a registered vehicle is either a person, a bank, or a company

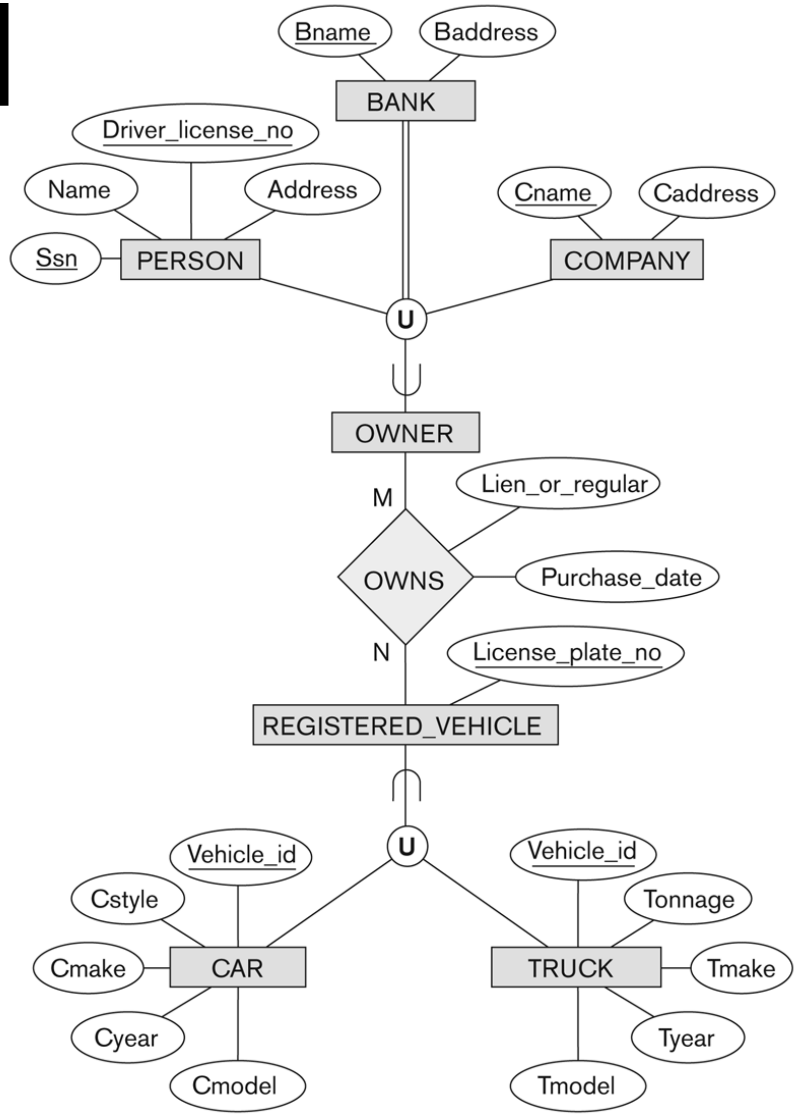


Figure 4.8
Two categories (union types): OWNER and REGISTERED_VEHICLE.

Short EER Quiz

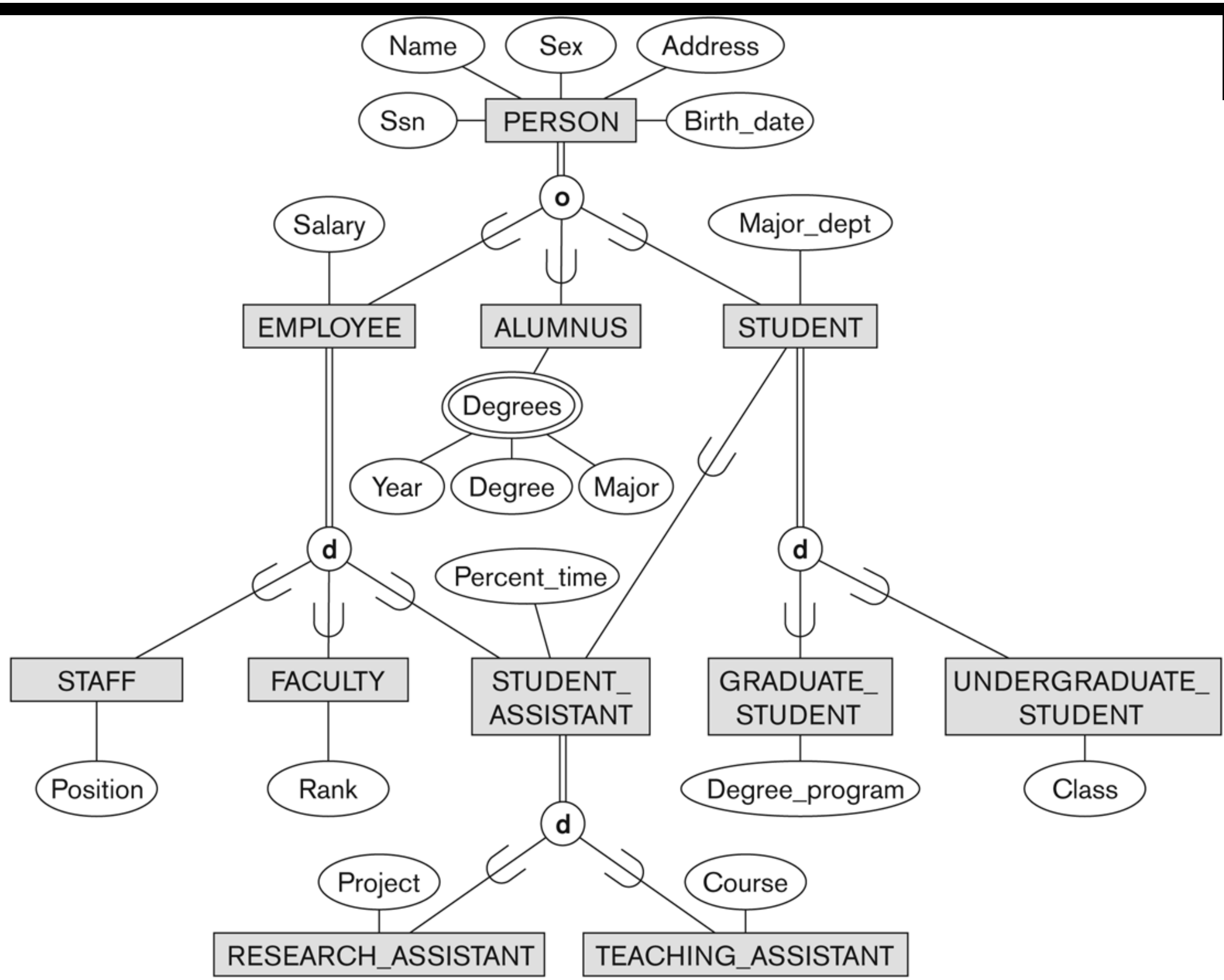


Figure 4.7
A specialization lattice with multiple inheritance for a UNIVERSITY database.

Key Takeaways

- **Basic Notation of Generalization/Specialization in EER**
  - IS-A Relationships
  - Attribute Inheritance
- **Types of Generalization/Specializations**
  - Total / partial
  - Disjoint / overlapping
- **Basics of union types and multiple inheritance**

CHALMERS UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG

Relational Model

LECTURE 4

Dr. Philipp Leitner

philipp.leitner@chalmers.se

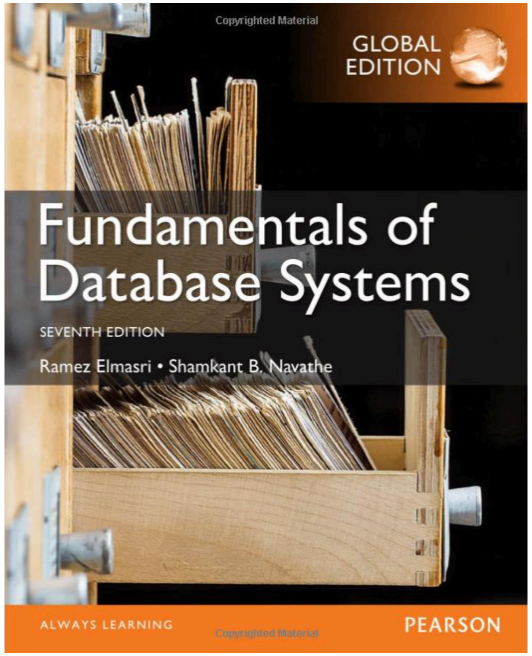
@xLeitix

LECTURE 4

Covers …

Chapter 5

Please read this up until next lecture!



What we will be covering

Basics of Relational Modelling

Database Constraints

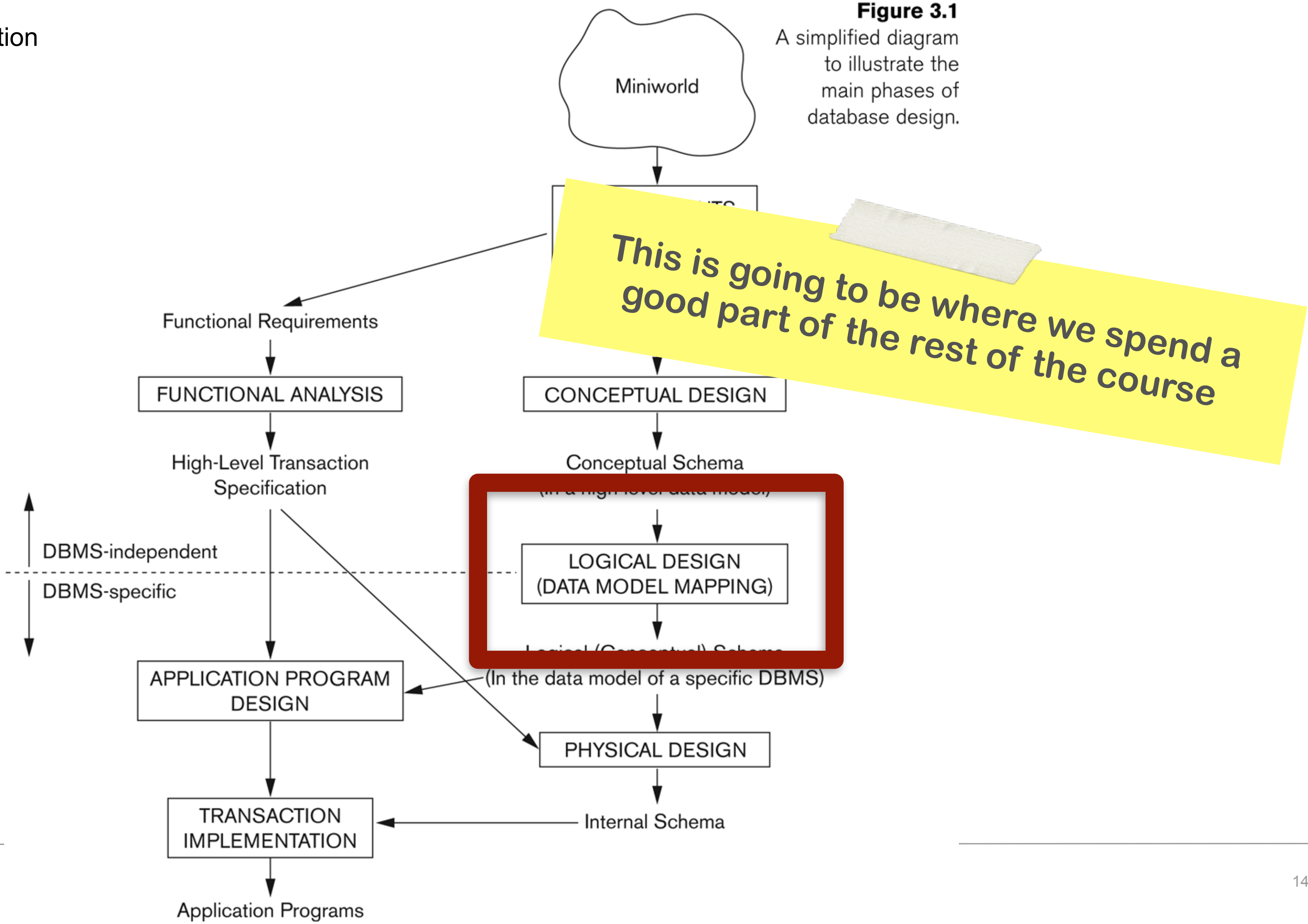


Figure 3.1
A simplified diagram to illustrate the main phases of database design.

Formal Relational Model

Relational databases (i.e., SQL) are based on a formal model

Simple

Powerful

—> Relational Model

Based on research from the 70-ties, remains relevant until today

Formal Relational Model

In the next three lectures we will introduce the **formal** model and a simple notation we use to discuss the model

Basically the underlying mathematics

The actual implementation (**SQL**) will follow afterwards

SQL and RM are **similar but not identical**

An initial terminology overview

What are …

(Mathematical) **Sets?**

(Mathematical) **Tuples?**

An initial terminology overview

The core of the Relational Model are **Relations**

Basically a table of values (cp. an *entity type*)

Each column has a header indicating the meaning of the values in the column. We call these the **attributes** of the relation.

The number of attributes of a relation is called the **degree** or **arity** of the relation.

Relations contain a **set of rows**

Mathematical set - no duplicates, and order does not matter

Each row in a relation (table) is a **tuple**, a set of values.

Each tuple has exactly as many values as the degree of the relation - but some may be **NULL**. The order in a tuple matters!

Relation Name

Attributes

STUDENT

Tuples

Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	NULL	19	3.21
Chung-cha Kim	381-62-1245	375-4409	125 Kirby Road	NULL	18	2.89
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	749-1253	25	3.53
Rohan Panchal	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	NULL	19	3.25

Figure 5.1
The attributes and tuples of a relation STUDENT.

On ordering

Ordering of tuples in a relation:

- The tuples are **not** considered to be ordered
- It’s confusing because we often list them in a tabular form

Ordering of attributes in a relation schema (and of values within each tuple):

- Attributes and values **are** ordered

Order matters

Attributes

Relation Name

STUDENT

Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	NULL	19	3.21
Chung-cha Kim	381-62-1245	375-4409	125 Kirby Road	NULL	18	2.89
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	749-1253	25	3.53
Rohan Panchal	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	NULL	19	3.25

Tuples

Order irrelevant

Figure 5.1
The attributes and tuples of a relation STUDENT.

This is considered *identical* to previous relation state!

Figure 5.2
The relation STUDENT from Figure 5.1 with a different order of tuples.

STUDENT

Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	749-1253	25	3.53
Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	NULL	19	3.25
Rohan Panchal	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
Chung-cha Kim	381-62-1245	375-4409	125 Kirby Road	NULL	18	2.89
Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	NULL	19	3.21

Keys

Remember that relations are not supposed to have duplicate tuples

Each tuple has value (or set of values) that uniquely identifies it
- Called the **key**
- Avoids clashes

E.g., in the STUDENT relation:
- Key may be the Ssn (cp. Swedish *personnummer*)

If the data does not have a natural unique attribute, we often create artificial keys

Relational Schema

We call the **description** of a relation its **schema**

Denoted by `R(A1, A2, .....An)`

`R` is the name of the relation

The attributes of the relation are `A1, A2, ..., An`

Example:

`CUSTOMER (Cust-id, Cust-name, Address, Phone#)`

Each attribute has a **domain** (a set of valid values)

E.g., domain of `Cust-id` may be a 6 digit numbers

Tuple

A **tuple** is an ordered set of values

(enclosed in angled brackets `‘< … >’`)

Each value is derived from an appropriate domain

Example:

A row in **`CUSTOMER (Cust-id, Cust-name, Address, Phone#)`**

would be a 4-tuple and consist of four values, for example:

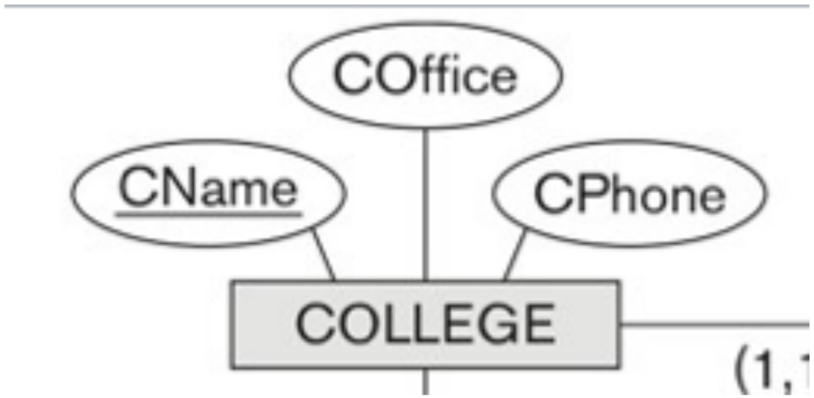
`<632895, "John Smith", "101 Main St. Atlanta, GA  30332", "(404) 894-2000">`

This is called a **4-tuple** as it has 4 values

Relations are set of such tuples

Example

Assume the following (fairly trivial) entity type:



Domain

Two definitions of a domain:

(1) the **logical definition**

E.g.,: `USA_phone_numbers` are the set of 10 digit phone numbers valid in the U.S.

(2) the **data-type** or **format**

E.g.,: `USA_phone_numbers` have a format: (ddd)ddd-dddd where each d is a decimal digit

In database design we primarily care about the format

The same domain can be the basis for multiple attributes

Example:

`Date` may be used to define two attributes named `"Invoice-date"` and `"Payment-date"`

Atomicity of values

All values are considered **atomic**

- They can't logically be divided further
- No composed values, as in ER

Example:

```
<1,"first key", 1982-11-25>
```

This should be seen as a value of the domain "date", not as a collection of integers and dashes

Null values

The domain of an attribute *may or may not* allow a special value, **NULL**, to indicate “no value”.

- This is a workaround for the constraint that every tuple needs a value for every attribute

NULL can mean different things:

- Unknown
- Unavailable
- Undefined

Figure 5.2
The relation STUDENT from Figure 5.1 with a different order of tuples.

STUDENT

Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	749-1253	25	3.53
Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	NULL	19	3.25
Rohan Panchal	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
Chung-cha Kim	381-62-1245	375-4409	125 Kirby Road	NULL	18	2.89
Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	NULL	19	3.21

For example:

- **NULL** for Barbara's office phone number may mean that we simply don't know it (unavailable)
- **NULL** for Chung-cha's office phone number may mean that the person has no office phone (undefined)
- **NULL** for Benjamin's office phone number may mean that we don't know whether the person has a phone (unknown)

Difference in semantics cannot be inferred from NULL values

State

The **state** of a relation is the collection of all current tuples and their values.

State can be **valid** (all constraints satisfied) or **invalid** (at least one tuple has at least one constraint violated)

Example for invalid state:

- Tuple has a value that is not in the domain of the corresponding attribute.
- There are other constraints as well - discussed later

Formal Definitions - Summary

Define a relation as:

`R(A1, A2, .........., An)`

`r(R)` ⊂ `dom (A1) X dom (A2) X … X dom(An)`

`r(R)`:

a specific state (or "value" or “population”) of R (set of tuples)

`r(R) = {t1, t2, …, tn}` where each `ti` is an n-tuple

`ti = <v1, v2, …, vn>` where each `vj` element-of `dom(Aj)`

Formal Definitions - Summary

Denoting attribute values - for a tuple `t`:

`t[1]` … value of first attribute in the list (remember - ordered!)

`t[a]` … value of attribute a

`t.a` … same as `t[a]`

`t[X]`, with `X` a subset of attributes

… values of all attributes in `X`

Formal Definitions - Summary

Database state is collection of the state of all relations

`DB = {r1, r2, ..., rm}`

Database is in a valid state if all `ri` are valid

Constraints

We focus on schema constraints in the following

Constraints determine which values are permissible and which are not

Three main types:

1. **Inherent** or **implicit constraints**
   - Based on the relational model itself
   - E.g., relational model does not allow a list as a value for any attribute
2. **Schema-based** or **explicit constraints**
   - Expressed through the schema
   - E.g., domains, key constraints, others
3. **Application based** or **semantic constraints**
   - These are beyond the expressive power of the model and must be specified and enforced by the application programs
   - E.g., “the salary of an employee cannot go down”

Constraints

The **purpose** of constraints is to prevent the database from becoming invalid

Allows us to specify already during database design what kind of **invariants** we want to be true at all times

Prevents programmer mistakes, but (more importantly) helps us **avoid data corruption through transient failures**

Relational Integrity Constraints

Various types of (explicit schema-based) constraints:

- **Domain constraints** (value must fit domain, including whether **NULL** is allowed or not)
- **Key constraints** (keys can't be duplicated)
- **Entity integrity constraints** (primary keys can't be missing)
- **Referential integrity constraints** (links to foreign keys can't be broken)

We call those **relational integrity constraints**

More on keys

Superkey of `R`:

Set of attributes `SK` of `R` with the following condition

No two tuples in any valid relation state `r(R)` will have the same value for `SK`

Key of `R`:

A **minimal** superkey

A key is a superkey K such that removal of any attribute from K results in a set of attributes that is not a key anymore

Note that every Key is a also Superkey, but not vice versa

Example

`CAR(State, Reg#, SerialNo, Make, Model, Year)`

`CAR` has two keys:

`Key1 = {State, Reg#}`

`Key2 = {SerialNo}`

Both are also superkeys of `CAR`

`{SerialNo, Make}` would be a superkey but not a key

Candidate and primary keys

If a relation has several keys (candidate keys), one is chosen to be the **primary key**

Primary key attributes are <u>underlined</u>

Example:

Consider the **`CAR`** relation schema

`CAR(State, Reg#,` <u>**`SerialNo`**</u>**`, Make, Model, Year)`**

We choose <u>**`SerialNo`**</u> as the primary key

Primary key value is used to uniquely identify each tuple in a relation

Selecting primary keys

No hard rules which candidate key should be primary

But:

(1) We usually prefer **short** keys, e.g., ones that consist only of a single attribute

(2) We prefer keys that are also often used to identify real-world entities over ones that just happen to be unique

Every relation should have a primary key. If there are no candidates, we usually introduce an **artificial primary key**

In practical implementations we sometimes prefer artificial primary keys even if candidates would exist …

Entity integrity

Entity integrity rule says that

The primary key attributes PK cannot have `NULL` values in any tuple.

This is because primary key values are used to identify the individual tuples
If PK has several attributes, null is not allowed in any of these attributes

Note:

- Other attributes may also be constrained to disallow null values, even though they are not members of the primary key
- For primary keys, this is assumed and does not need to be specified

Entity integrity

Formally:

`t[PK] ≠ NULL` for any tuple `t` in `r(R)`

If `PK` has several attributes, NULL is not allowed in any of these attributes

Referential integrity

Example:

In the EMPLOYEE and DEPARTMENT relationships, the attribute that specifies which department an employee works in needs to refer to a department that actually exists.

A constraint **involving two relations**

The previous constraints involved only a single relation

Used to specify a **relationship among tuples** in two relations:

AKA referencing relation and the referenced relation

Foreign keys `FK`

Tuples in the referencing relation `R1` have attributes `FK` (the **foreign key**) that reference the primary key attributes `PK` of the referenced relation `R2`.

A tuple `t1` in `R1` is said to reference a tuple `t2` in `R2` if

`t1[FK] = t2[PK]`

Foreign keys `FK`

Note that (in 1:N or N:M relationships), **foreign keys are not keys themselves**!

E.g., multiple employees work in the same department, hence have the same value in their department FK

FKs may also be allowed to be NULL

If relationship is optional, e.g., employees may not be assigned to a department at all

Other types of constraints

The good thing about the constraints discussed so far is that we can **express them directly in the relational model**

And, similarly, later on directly in SQL

This is not true for **semantic integrity constraints**

E.g., *"the max. no. of hours per employee for all projects he or she works on is 56 hrs per week"*

Needs to be handled in the application that uses the database, or through **triggers** or **stored procedures**

Key Takeaways

Basic concepts of relational modelling

Relations, attributes, domains, tuples, sets

Formal notation (e.g., `t[Ssn]`)

Constraints and Keys

Keys, superkeys, primary keys, foreign keys