MANCHESTER 1824

The University of Manchester

Slides L1-L6

Fundamentals of Databases

Alvaro A A Fernandes, SCS, UoM

The University of Manchester

L1

Introduction to Data Management

Fundamentals of Databases

Alvaro A A Fernandes, SCS, UoM

Readings

This lecture relies on your having read the assigned mandatory readings associated with this lecture:

Abraham Silberschatz, Henry F. Korth, and S. Sudarshan. Database System Concepts. 6th Edition. McGraw-Hill, 2006. pp. 1-35.

You can download the material using this link:

https://contentstore.cla.co.uk/secure/link?id=4151d4df-4e05-e611-80bd-0cc47a6bddeb

If you fail to study the material, you're likely to find the lecture harder to follow.

More information is available in the Learning Activities Manual.

In This Lecture

X

- We will learn that data is an enterprise asset and that DBMSs are crucial to manage it well.
- We will learn about the importance of adopting different levels of abstraction in designing and implementing databases.
- We will learn that data models lead to a distinction between schemas and instances that enables a logical view of the data.

Database Management Systems

- Database management systems (DBMSs) can be seen in the broader context of software engineering strategies.
- Engineering software is hard and costly.
- It is not cost-effective to develop every piece of software from scratch.
- It is not cost-effective to develop every piece of software in a bespoke way.
- We factor out certain functionalities into software products that act as service providers for applications.
  - We don't build bespoke operating systems (OSs) from scratch.
  - The services they provide are universally useful and comprise a stable set.

Database Management Systems

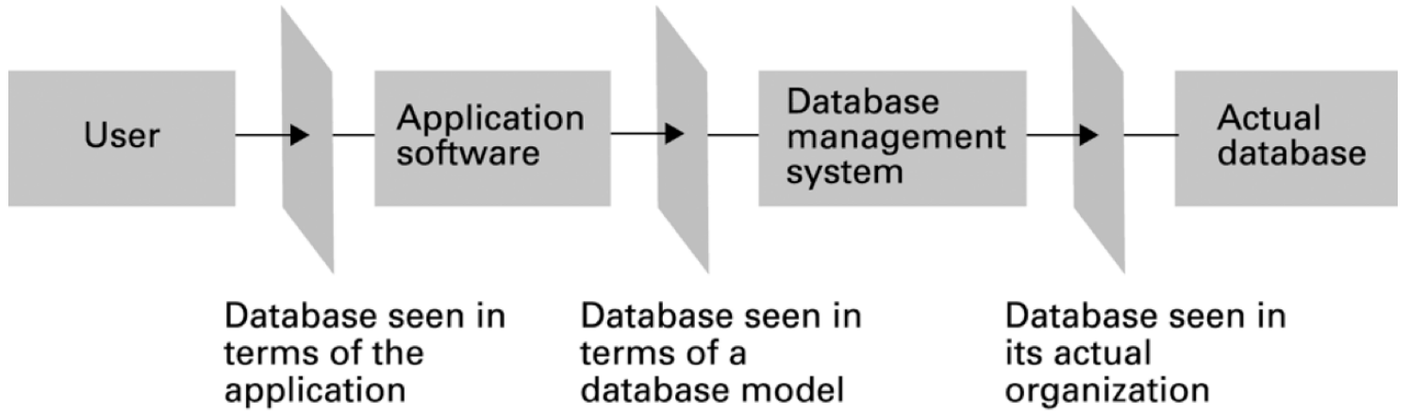
- DBMSs, like OSs, factor out functionalities as services that they provide for applications because it has been shown over decades that:
  - data-related services are universally useful,
  - there is a stable set of them.
- For that, DBMSs enforce
  - several principles (such as logical/physical independence)
  - using advanced techniques (such as transaction and recovery management)
  - that emerge to software engineers as a design methodology.

How do we view data?
How should we manage it?

- Data is an asset, as crucial to an organization as capital or human resources.

- A database management systems (DBMS) is a sophisticated tool for managing data intra- and inter-organizationally.

- It exposes services that allow people and applications to have a single, unified view of the organization through its data assets.

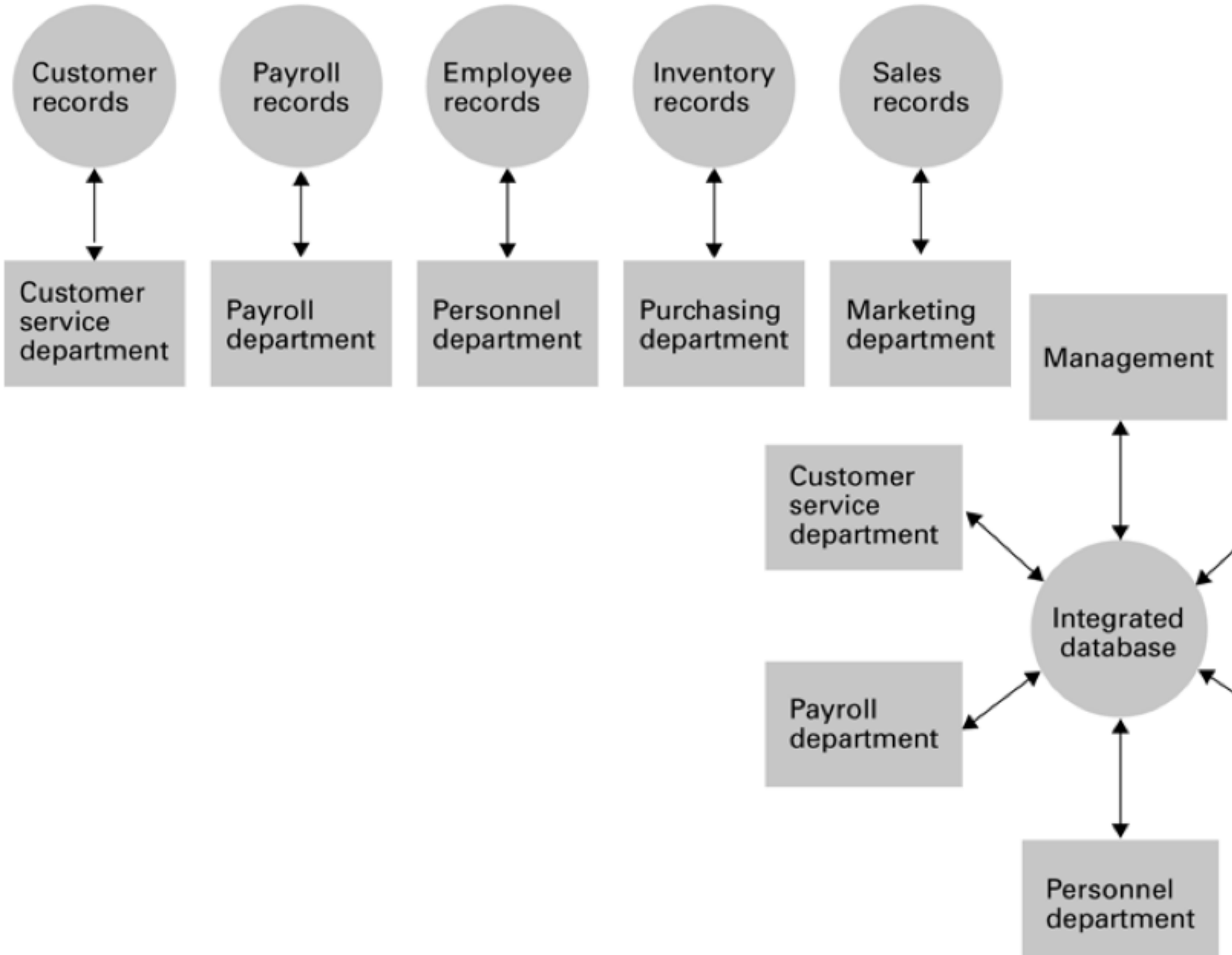


Where are DBMSs used?

- Inside organizations: to support front-end and back-end business processes

- Across organizations: to allow cooperation and coordination

Why DBMSs? Why not use files?



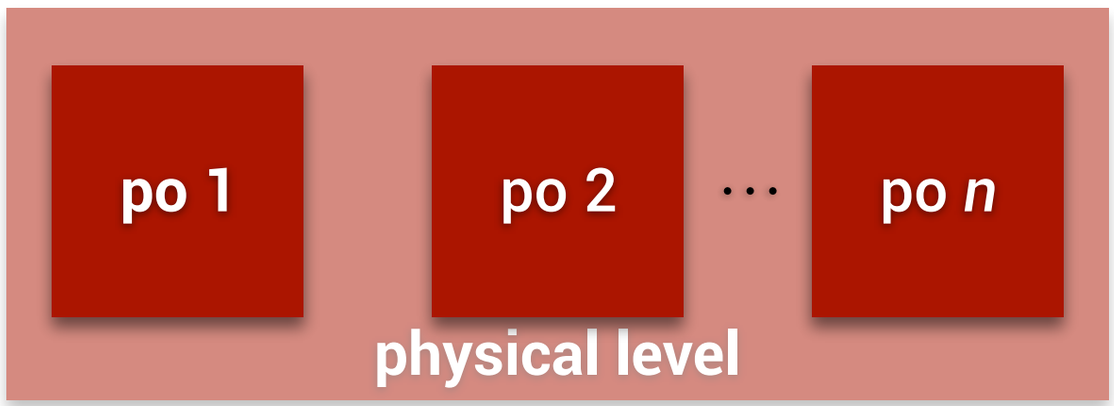
J. Glenn Brookshear, Computer Science: An Overview, 10th ed. Pearson, 2009, ISBN 978-0-321-54428-5

Why DBMSs? Why not use files?

- The applications that matter to an organization change faster than the data they use.
- It is better to loosely couple application and data.
- This also allows decoupling a logical view of data from storage technology and formats.
- A DBMS enables such decoupling, so applications can change as needed.
- A DBMS therefore lowers the cost of developing new applications.
- A DBMS offers a set of sophisticated services beyond storage, including:
  - efficient, scalable querying and updating;
  - concurrent, fault-tolerant transactions;
  - privacy, and role-based access and authorization control.

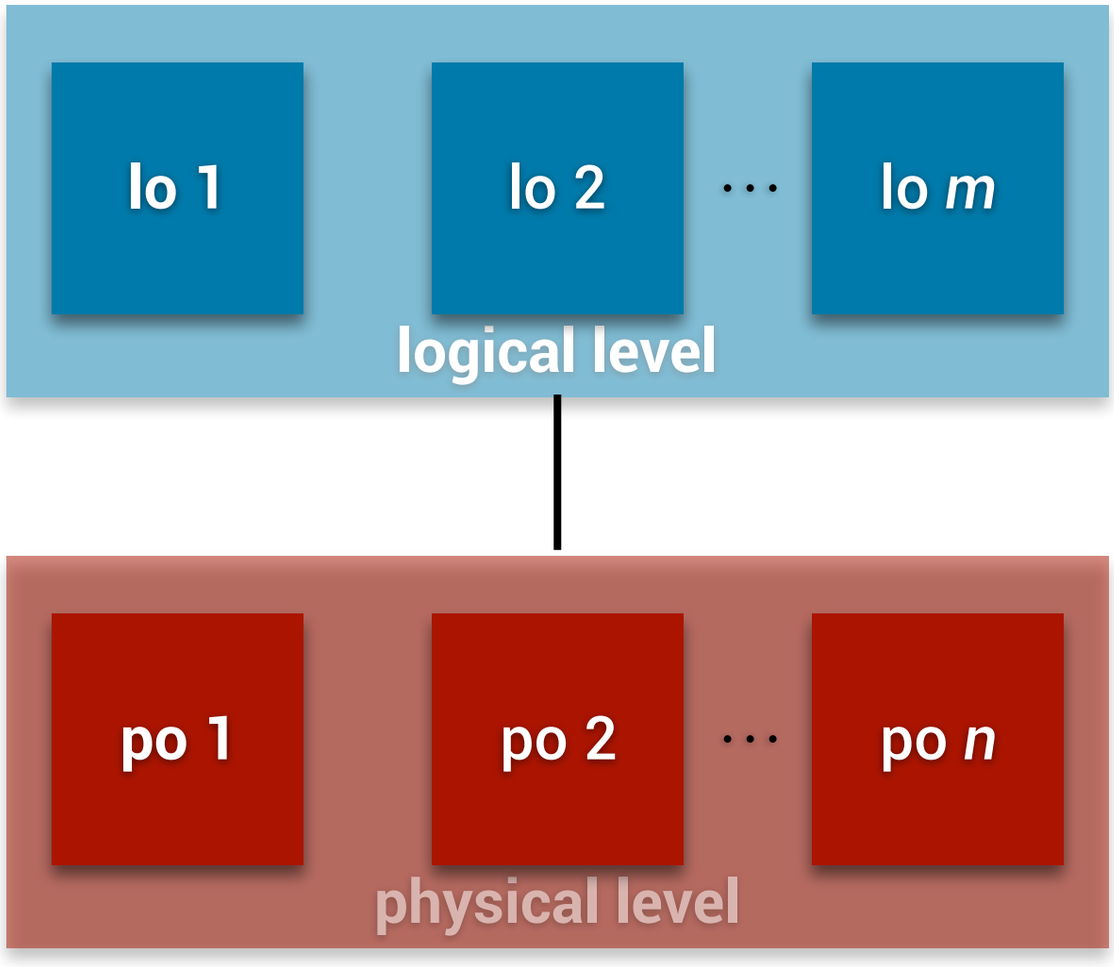
How do different levels of abstraction in DBMSs work?

- The physical level
  - defines formats, records, files, clustering, indexes, compression, replication, redundancy,
  - is expressed as a storage or **physical model**.



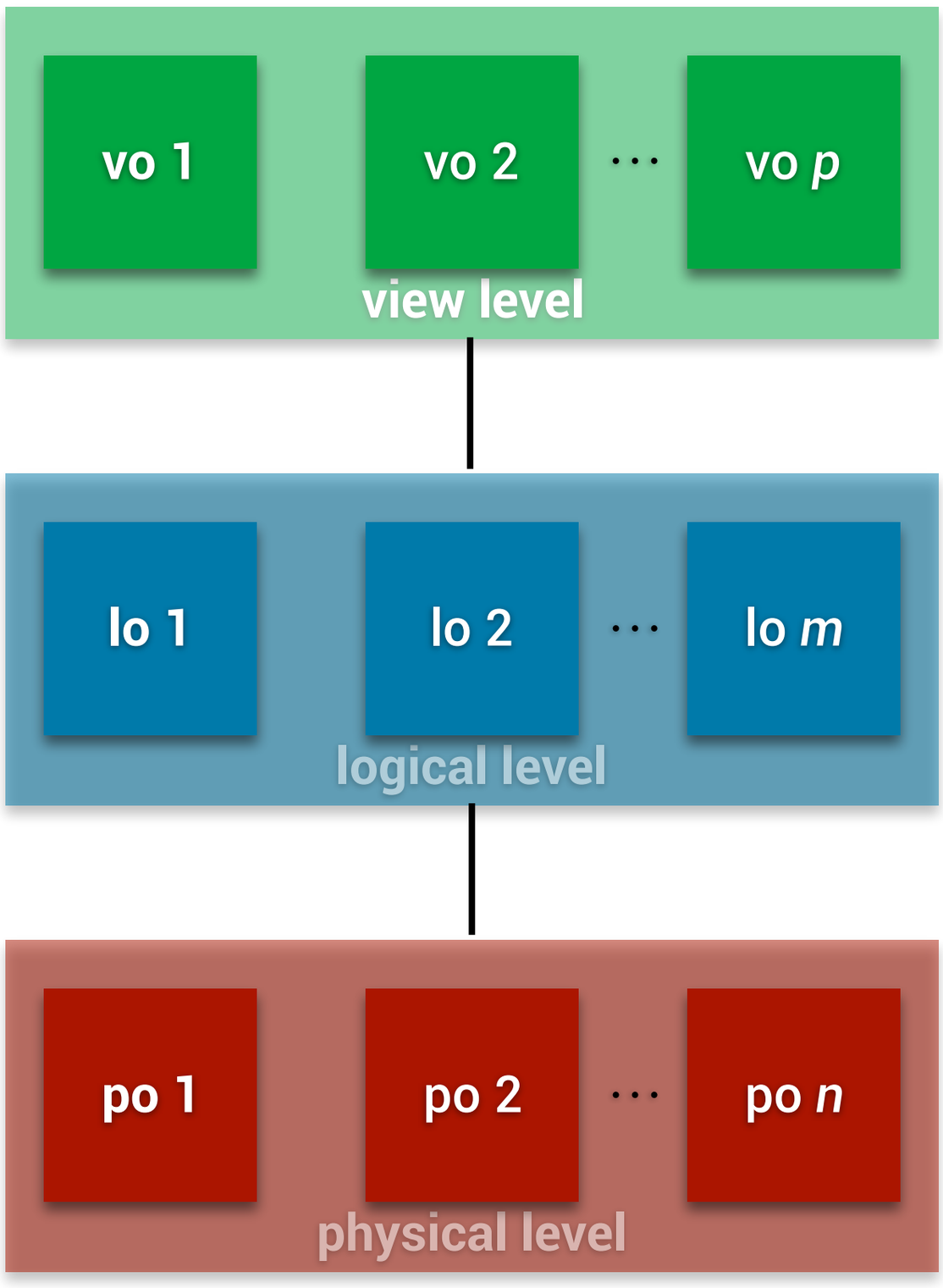
How do different levels of abstraction in DBMSs work?

- The logical level
  - defines the concepts that give the entire enterprise a specific character:
    - supplier, product, stock_unit, customer, order, payment, etc.,
    - employee, stores, banker, accounts_payable, accounts_receivable, etc.;
  - is expressed by a **logical data model**, possibly derived from a **conceptual model**.



How do different levels of abstraction in DBMSs work?

- The view level
  - defines what software applications work with and partitions functionality:
    - Human resource applications do not see balances in bank account.
    - Point-of-sale applications do not see employee salaries.
  - is expressed by a **database application model** (comprising queries, views, triggers, etc.).



What is a (logical) data model?

- A data model is a formal construct for describing data, their relationship, semantics and constraints.
- The relational model is the most widely-used:
  - rows of columns are organized in tables that may reference one another, and
  - if they do, then no dangling allowed.
- It comes with special database languages for interaction, e.g., a relational algebra.
- A DBMS supports one (or more) data models and associated languages.
- The Oracle DBMS, e.g., supports:
  - relational,
  - object-relational,
  - XML-based, and
  - document-based
  - using SQL, XQuery, etc.

What are schemas and instances?

- Each data model defines:
  - whether there is a distinct notion of a schema (e.g., recent data models don't),
  - what the schema is allowed to declare (e.g., which structures and which constraints).
- Then, instances must be valid w.r.t. the schema:
  - must have compatible structure, and
  - must satisfy the constraints.

- A schema defines the logical structure of the database, e.g.:

```
share(account,customer)
```

- An instance is the actual content of the database at a particular point in time, e.g.:

```
share(015467,'Mary Quaint')
share(015467,'Vivienne Eastwood)
```

- Schemas are similar to types and variables.
- Instances are similar to variable values (i.e., they change over time).

What different types of database language there are?

- There are different kinds of database language:
  - DDL: Data Definition Language
  - DML: Data Manipulation Language
  - (D)QL: Query Language
- Again, the terminology is not crisp:
  - Sometimes, there is no special DDL.
  - Often, QL is not distinguished from DML.

- The DDL updates the data dictionary.
- It is a declaration language: it defines the schema (i.e., types and constraints).
- The DML updates the database.
- It is a state-transition language, it changes values (e.g., rows and columns).
- The QL is a pure expression language: evaluation returns values without changing the database instance.

Why are different types of database language needed?

- The main advantage of this approach is enforcing **separation of concerns**, e.g.
  - DDL is enterprise-wide, for all applications.
  - DML is for reflecting changes caused by events in the life of the enterprise (e.g., a new account).
  - QL is for retrieving information (e.g., the current average balance).
  - The lack of assignment makes the QL easier to rewrite for optimization purposes.

- In practice, the distinction is not often enforced.
- SQL ⊇ DDL ∪ DML ∪ QL
- SQL has become extremely expressive with modern extensions.

Why are different types of database language needed?

- The main advantage of this approach is enforcing **separation of concerns**, e.g.
  - DDL is enterprise-wide, for all applications.
  - DML is for reflecting changes caused by events in the life of the enterprise (e.g., a new account).
  - QL is for retrieving information (e.g., the current average balance).
  - The lack of assignment makes the QL easier to rewrite for optimization purposes.
- In practice, the distinction is not often enforced:
  - SQL ⊇ DDL ∪ DML ∪ QL
  - SQL has become extremely expressive with modern extensions.

In the Next Lecture

- We'll begin to explore the steps in **database design**, which will take us all of eight lectures to complete.
- The first step is to learn about the approach to conceptual modelling known as **entity-relationship modelling**.

MANCHESTER 1824
The University of Manchester

L2

Designing Databases: The Entity-Relationship Approach

Fundamentals of Databases

Alvaro A A Fernandes, SCS, UoM

Readings

This lecture relies on your having read the assigned mandatory readings associated with this lecture:

Ramez Elmasri and Shamkant Navathe. Fundamentals of Database Systems. New International Edition. Pearson, 2013. pp. 201-245.

You can download the material using this link:

https://contentstore.cla.co.uk/secure/link?id=73ff189a-8463-e611-80c6-005056af4099

If you fail to study the material, you're likely to find the lecture harder to follow.

More information is available in the Learning Activities Manual.

In the Previous Lecture

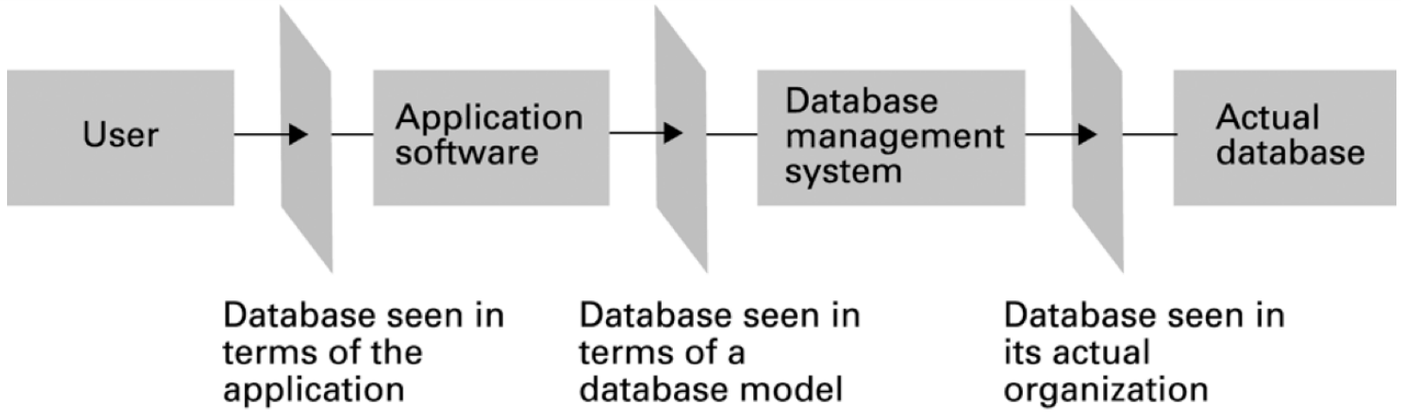
- We learned that data is an enterprise asset and that DBMSs are crucial to manage it well.
- We learned about the importance of adopting different levels of abstraction in designing and implementing databases.
- We learned that data models lead to a distinction between schemas and instances that enables a logical view of the data.

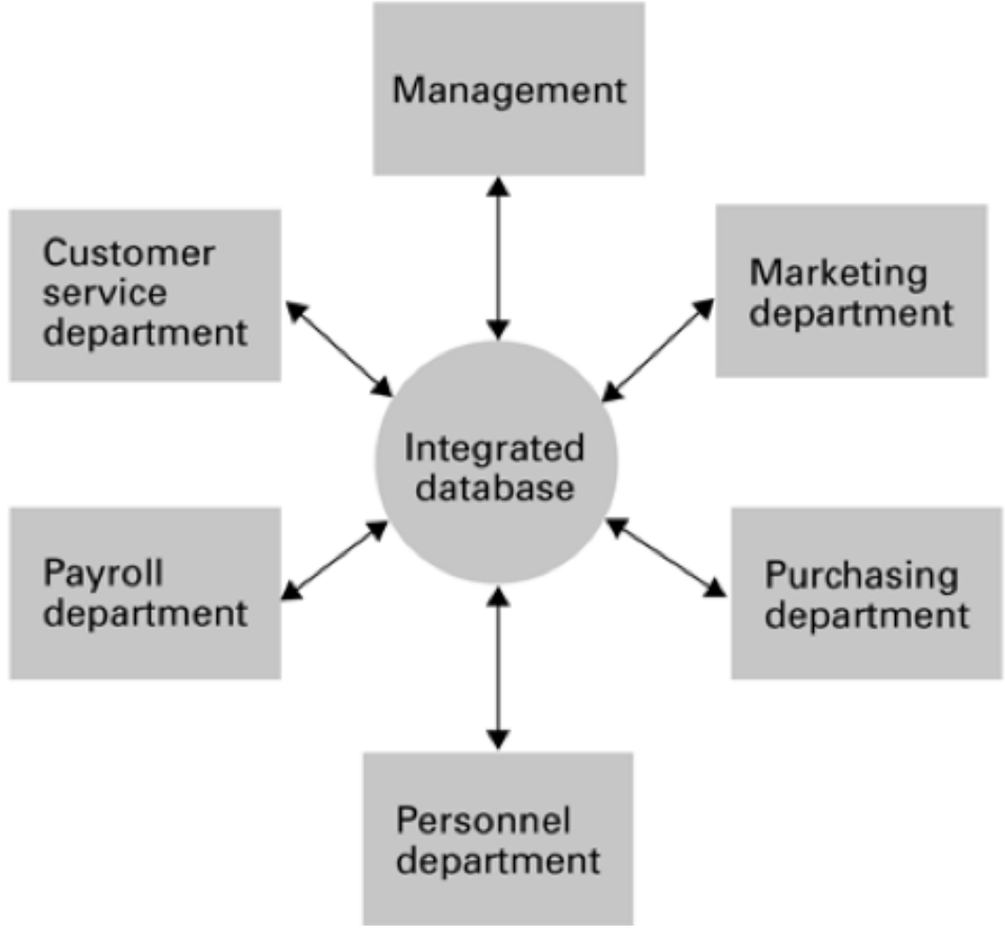
In This Lecture

- What are database applications?
- Why conceptual modelling?
- What are the phases in the design of database applications?
- What is entity-relationship (ER) modelling?

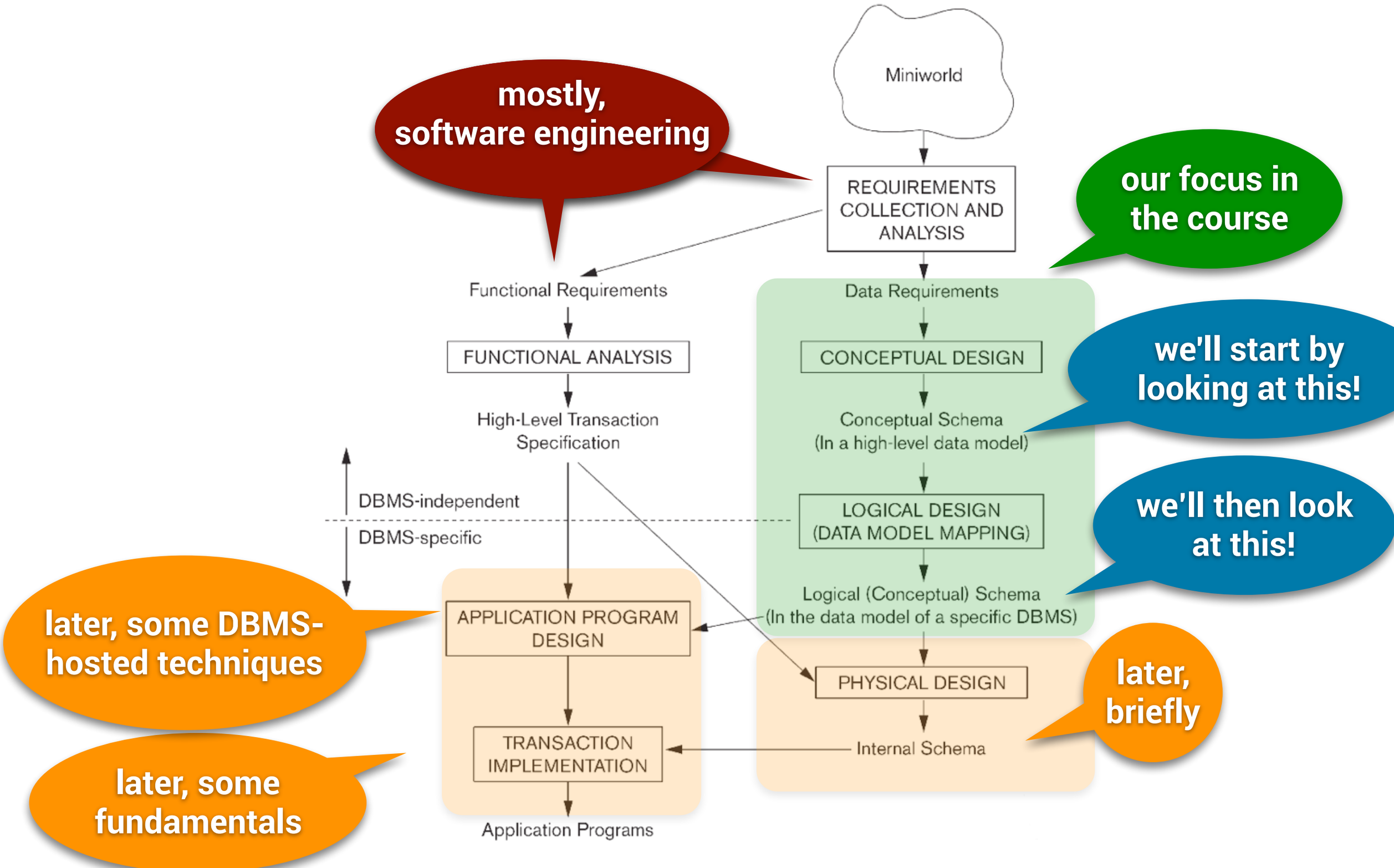
Conceptual Design of Database Applications

- **Database applications** comprise:
  - an integrated database
  - plus application programs that
  - implement queries and updates to that database





Database Application Design Phases

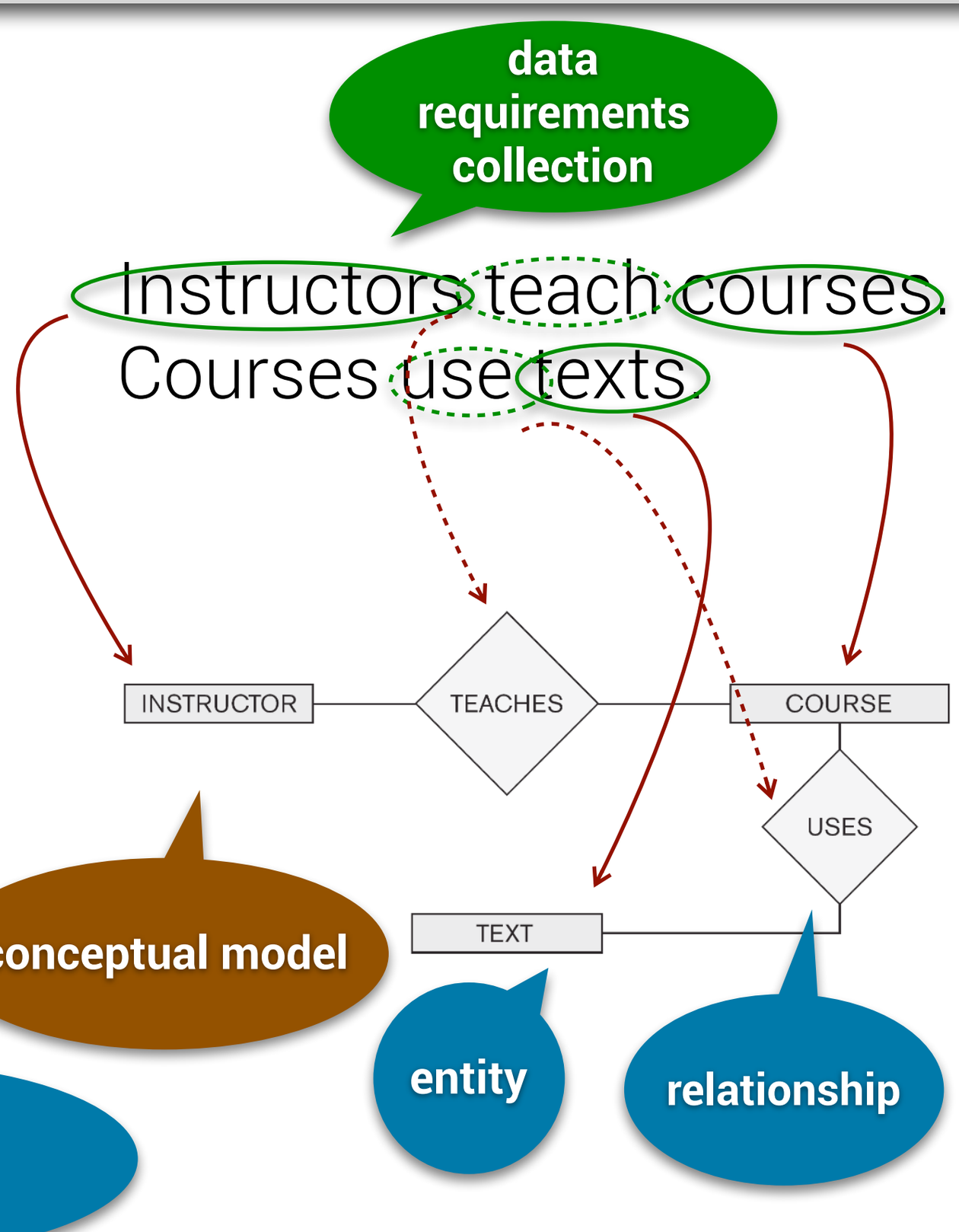


Conceptual Design of Database Applications

- **Entity-Relationship (ER) Modelling** is a popular, high-level conceptual data modelling approach

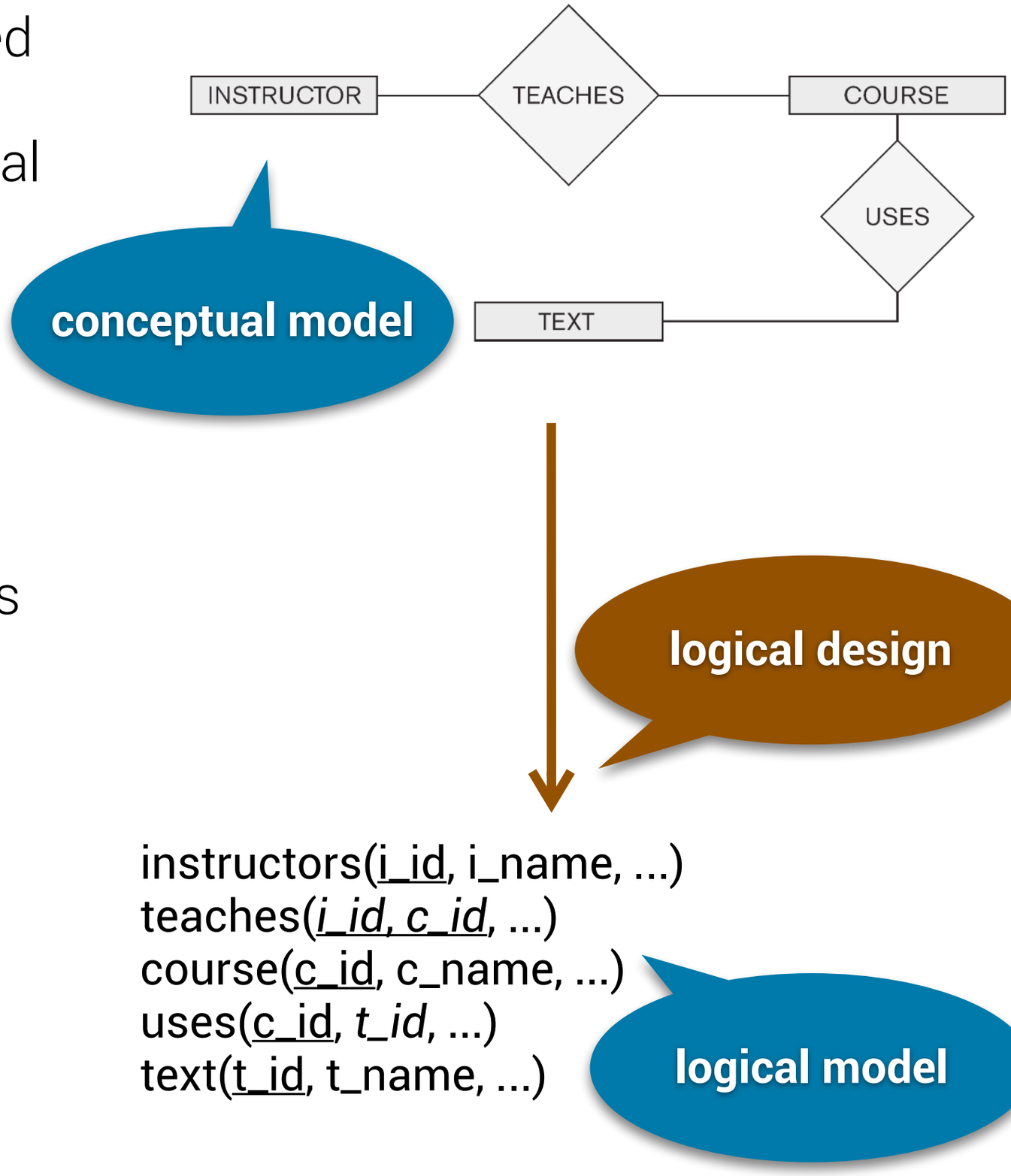
- Its formal simplicity and flexibility makes it a good common language to connect end users and database designers

- It focuses on data requirements, not overall business logic



From Conceptual to Logical Models

- A **conceptual model** (also called a **conceptual schema**) is the main outcome of the conceptual design phase.
- It is a rigorous, but high-level, description of the data requirements
- It includes detailed descriptions of the entity types, relationship types, and of important constraints on the latter
- It aims to make the mapping into a more rigorous logical model fairly systematic.



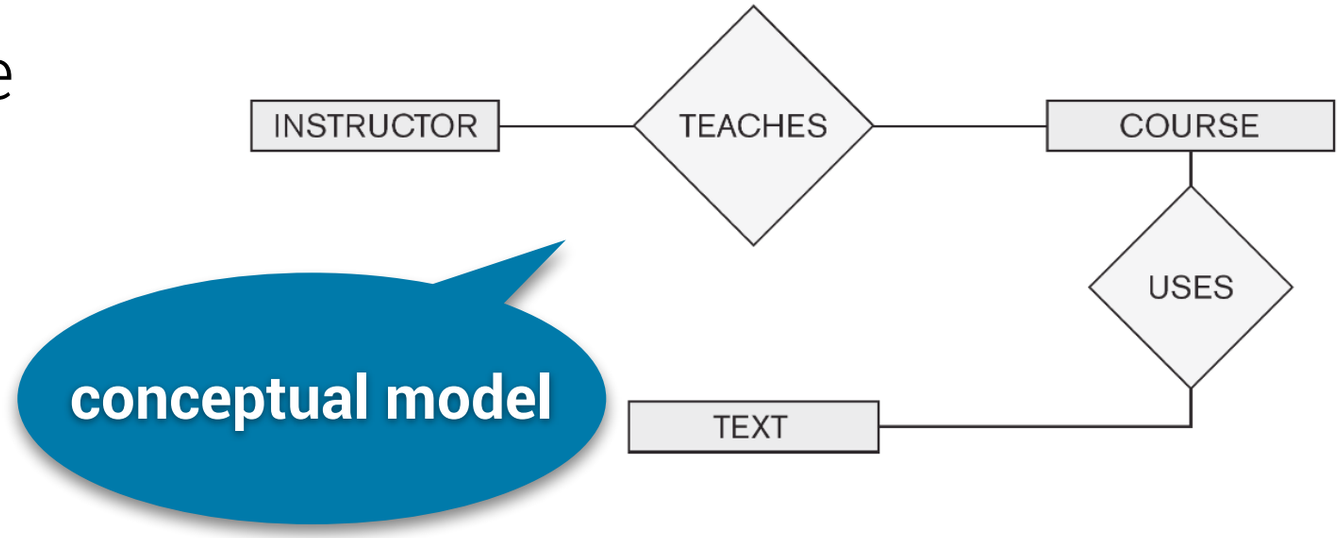
From Conceptual to Logical Models

- A conceptual model is suitable to being mapped into an (**implementation**) **logical model** or **schema**

- In our case, the conceptual model is an **ER model**

- By a **logical model** or **schema** we mean one (e.g., relational) that is directly supported by the DBMS we intend to implement the database application on

- In our case, the logical model is a **relational model**



ER Modelling: Basic Constructs

- The basic constructs in ER modelling capture the data requirements in the form of:
  - Entity types
  - Attribute types
  - Relationship types

similar to relations

similar to attributes

but less constrained

how are they captured then?

no direct counterpart in the relational model

ER Modelling: Attribute Types

- Attribute types can be:
  - **Composite** v. **simple** (or **atomic**)
  - **Single-valued** v. **multivalued**
  - **Stored** v. **derived**
  - **NULL-valued**
  - **Complex-valued** (i.e., arbitrarily-nested composite/ multivalued)
- Try not to confuse the notion of a '*composite*' attribute type with that of a '*complex*' one

ER Modelling: Keys

similar to relational model

- In ER modelling, a **key** (or **uniqueness**) **constraint** holds.

similar to relational model

- Entities must be uniquely identifiable in every possible entity set of an entity type.

similar to relational model

- Keys are attributes whose values are distinct for each individual entity in any entity set.

similar to relational model

- There can be more than one key in an entity type.

unlike the relational model

- An entity type that has no natural or assigned key is said to be **weak**.

ER Modelling: A Word of Caution on Keys

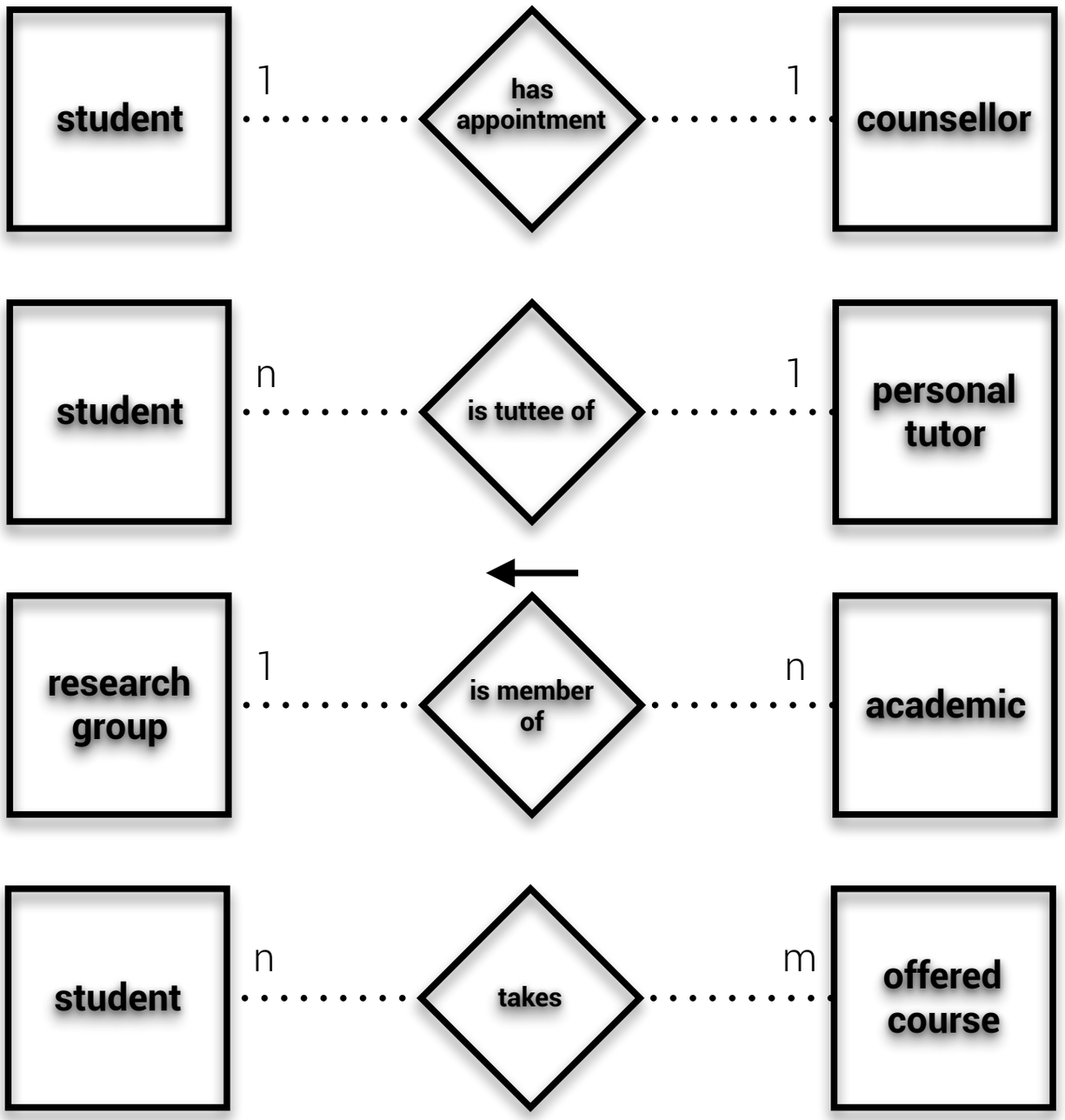
- Try not to confuse the **ER** notion of '**key**' with the **relational** ones of '**primary key**' and '**foreign key**'.

- The notion of key in ER modelling is related (but not identical) to the notion of a primary/foreign key in a relational database.

- It is best compared to the relational notion of a *candidate key*, which a designer may choose to make primary or not.

- Also, foreign keys model relationships implicitly in the relational model, but the 'R' in 'ER' tells you that *relationships are modelled explicitly in the ER model*.

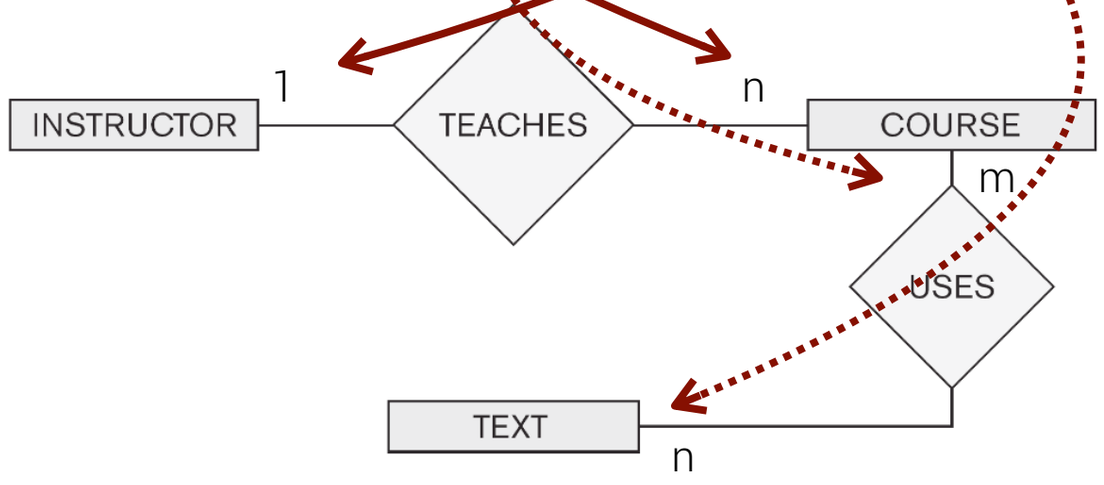
ER Modelling: Constraints on Relationship Types



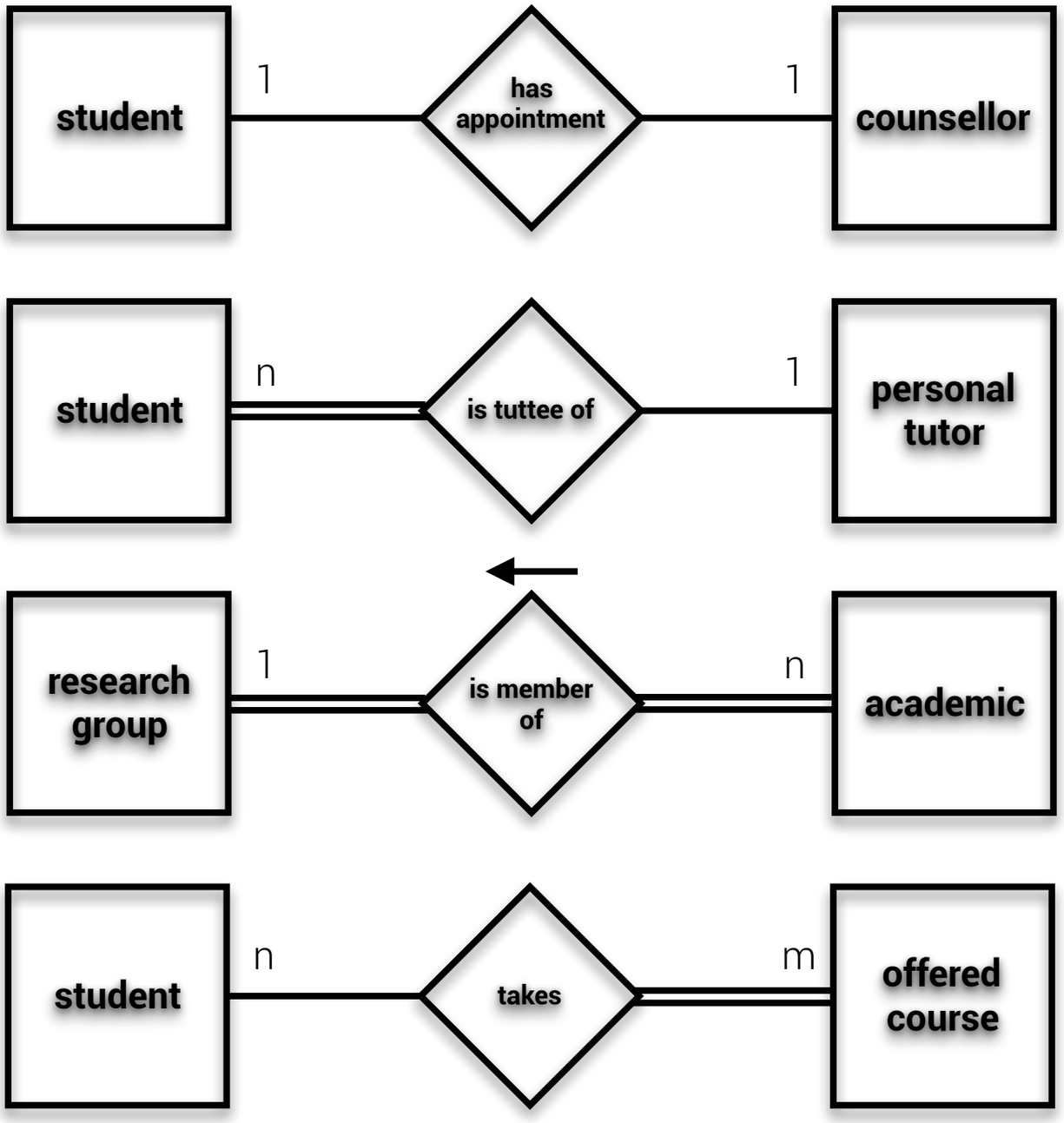
- A **cardinality ratio constraint** specifies the maximum number of relationship instances that an entity can participate in
- It can be **1:1**, **1:N**, **N:1**, **N:M** (or, equivalently, **M:N**)

How cardinality ratio constraints are captured

Instructors teach courses.
An instructor teaches many courses.
A course is taught by one instructor only.
Courses use texts.
A course uses many texts.
A text is used by many courses.



ER Modelling: Constraints on Relationship Types



- A **participation constraint** specifies whether the existence of the entity depends on its being related to another entity

double line

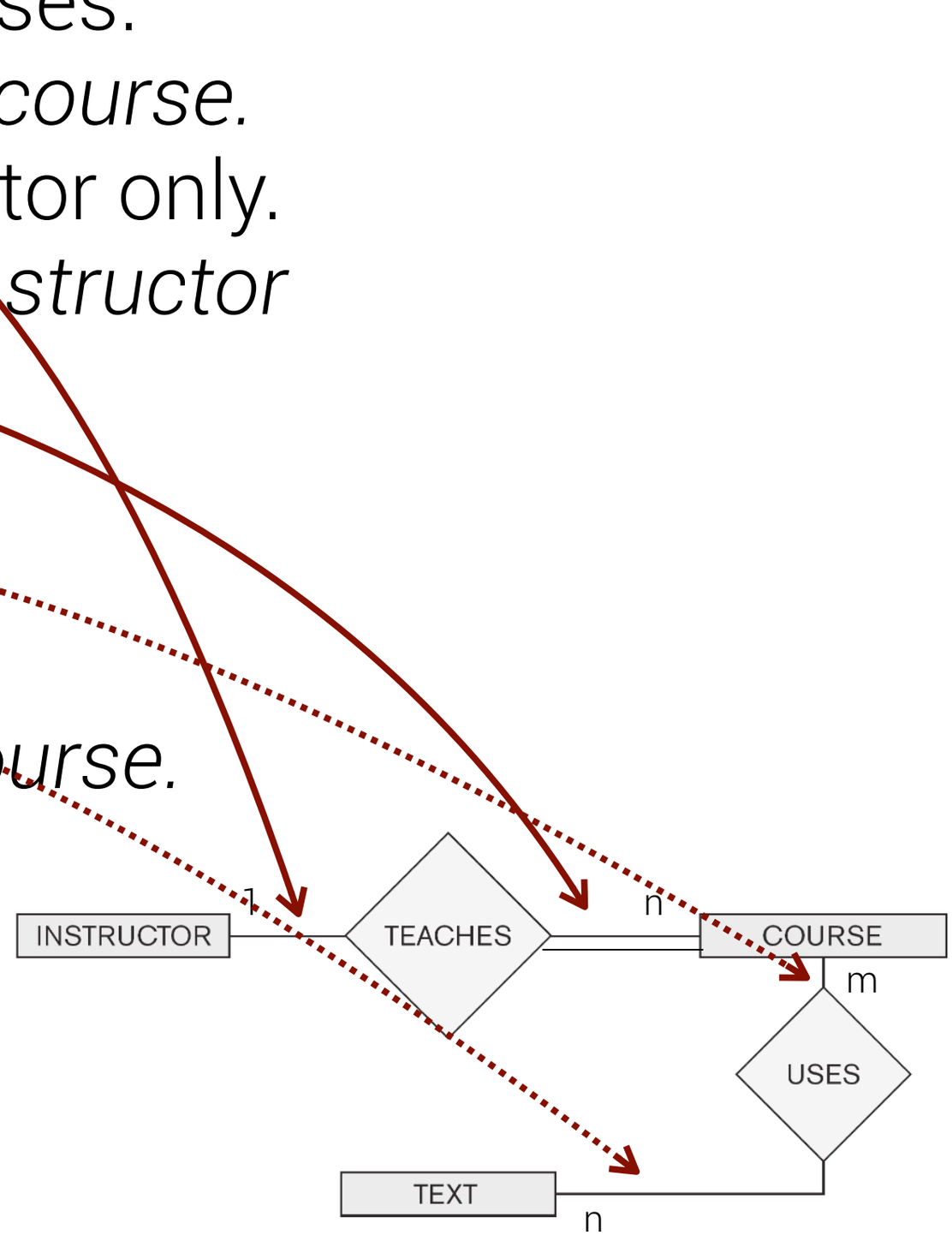
- If participation is **total**, then every entity in the total entity set must participate in some relationship instance

single line

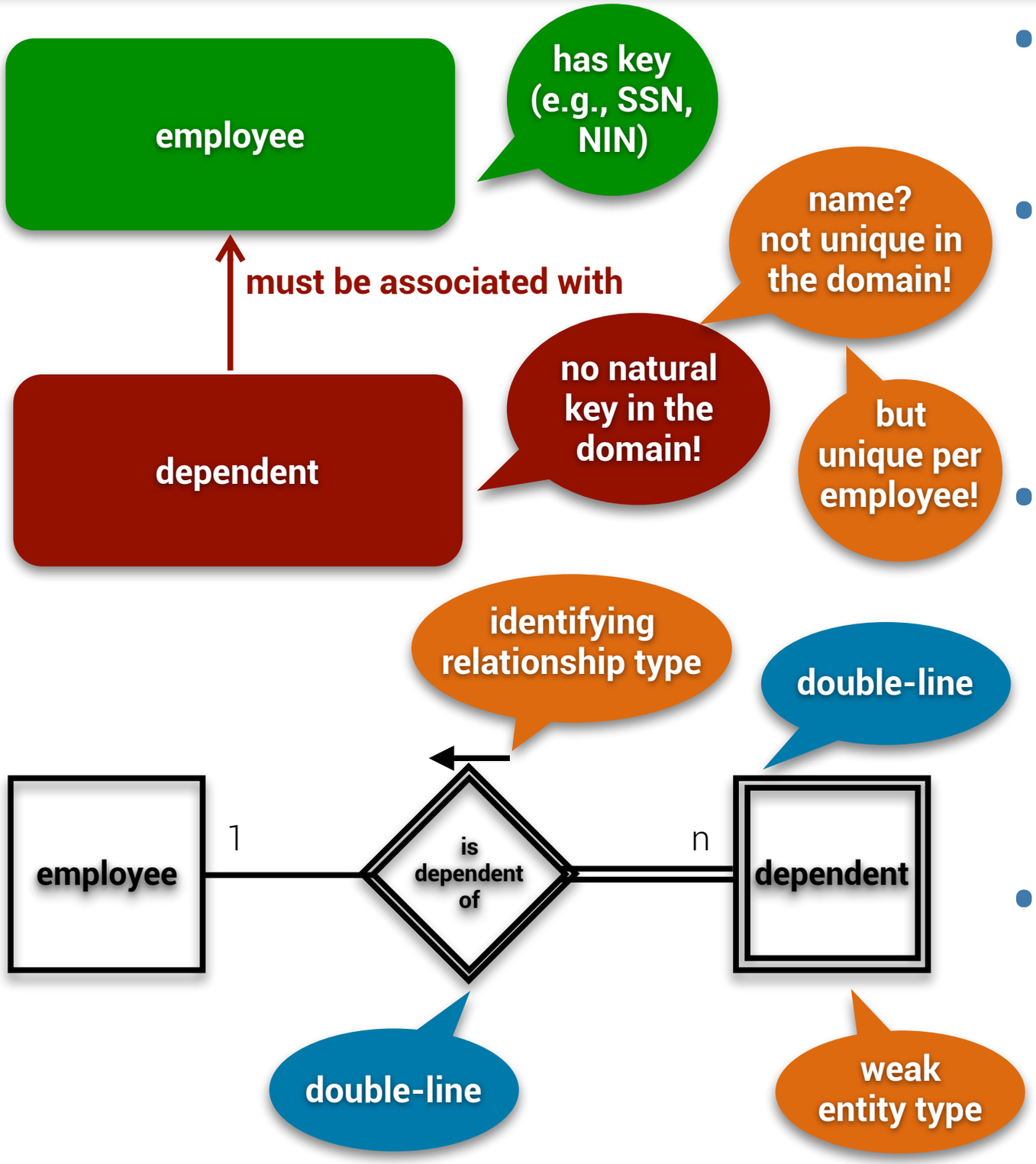
- If participation is **partial**, then some entities may not participate in any relationship instance

How participation constraints are captured

Instructors teach courses.
An instructor teaches many courses.
An instructor may not teach any course.
A course is taught by one instructor only.
A course must be taught by an instructor
Courses use texts.
A course uses many texts.
A course may not use any text.
A text is used by many courses.
A text may not be used by any course.



ER Modelling: Weak Entity Types



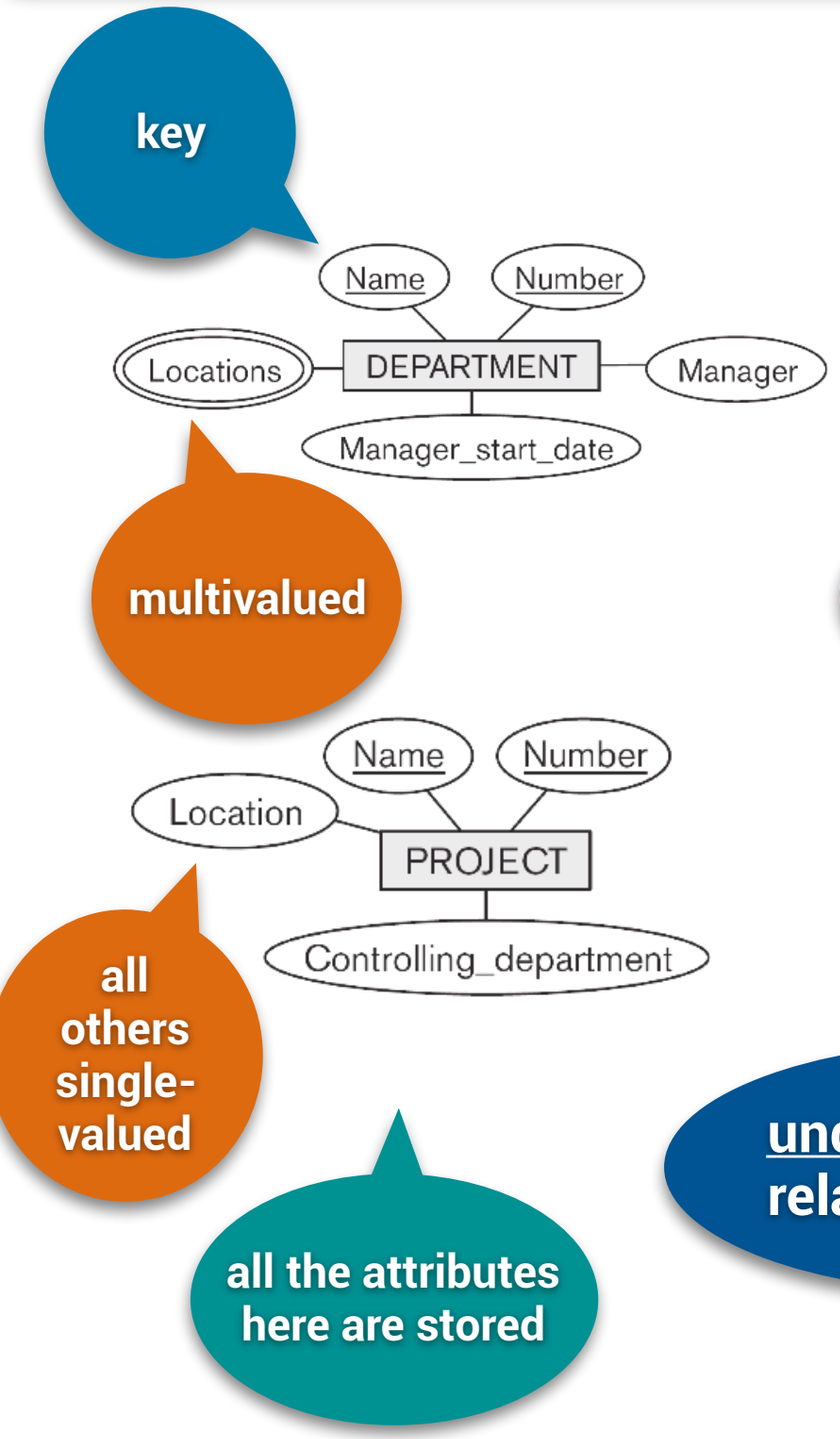
- A **weak entity type** does not have key attributes of their own
- It only has a **partial** (or **weak**) **key** that identifies each entity in the entity set in the context of another entity of another entity type
- It can only be properly identified (i.e., retain its conceptual status and have an associated entity set) by being related to specific entities of some other entity type, called the **owner**
- An **identifying relationship** is one that relates a weak entity type to its owner, and **always** has a *total* participation constraint

MANCHESTER 1824

The University of Manchester

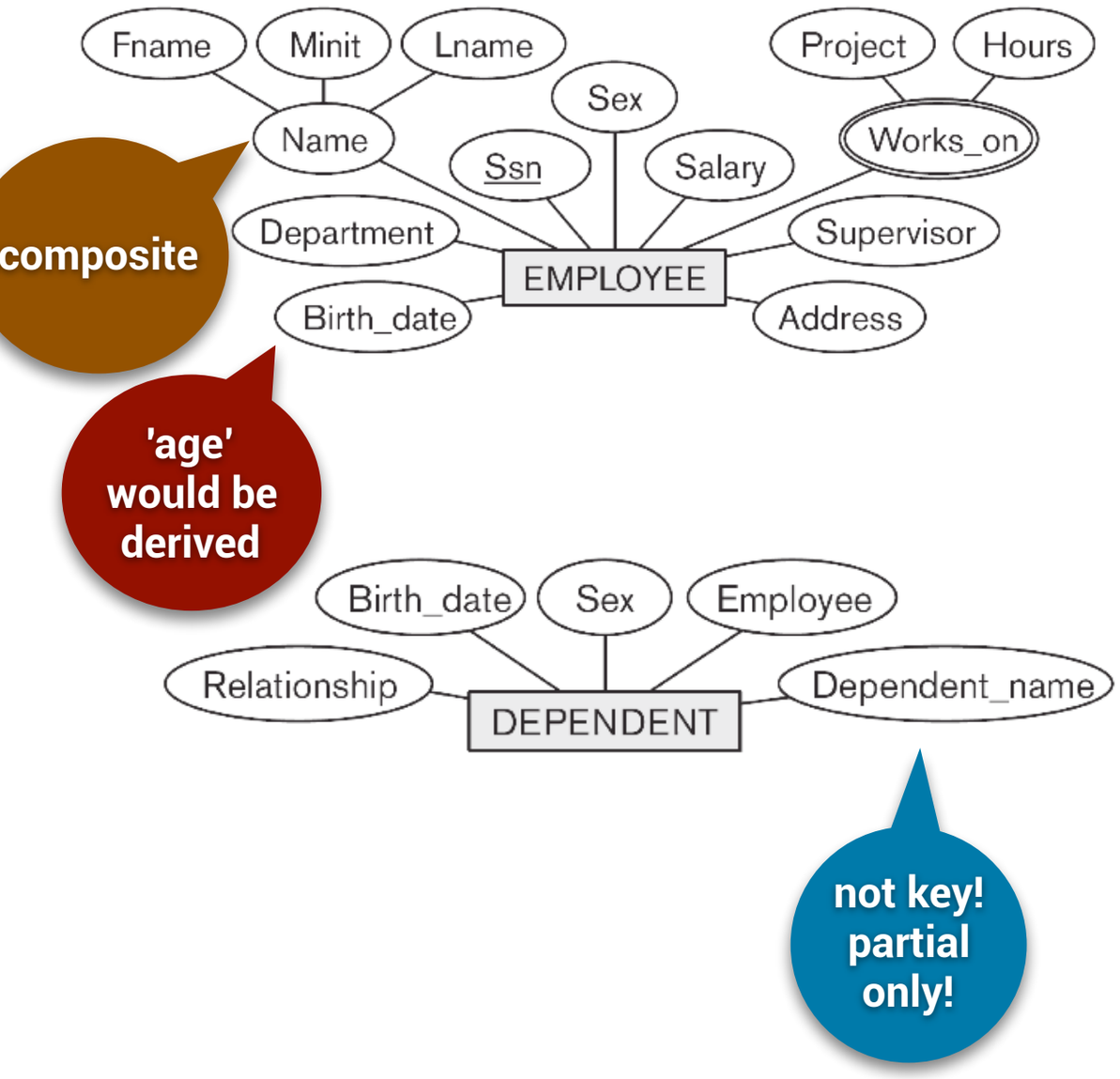
Deriving ER Model from Data Requirements Specification

Annotating a Requirements Specification



- Assume our miniworld encompasses the **employees**, **departments** and **projects** of a company.
- The company is organized into departments, which have a *name*, a *number* and a *location* (possibly more than one).
- Each department <u>is managed by a</u> **manager**, who is an employee, and we wish to store the *date* in which the latter started managing the department.
- Each department <u>controls various</u> projects.

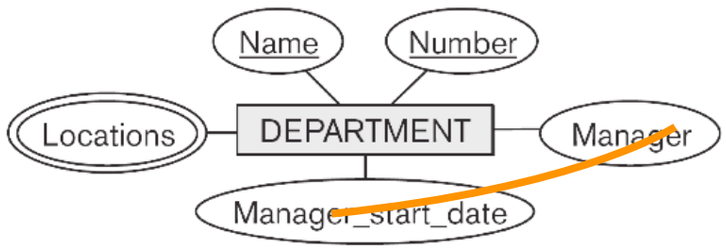
Annotating a Requirements Specification

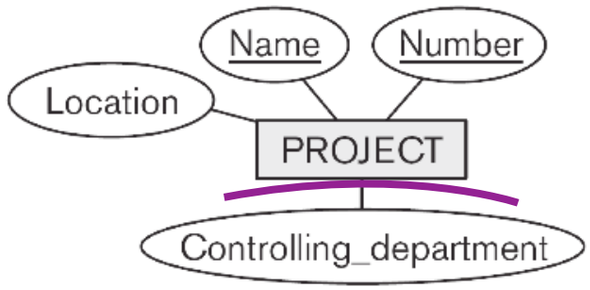


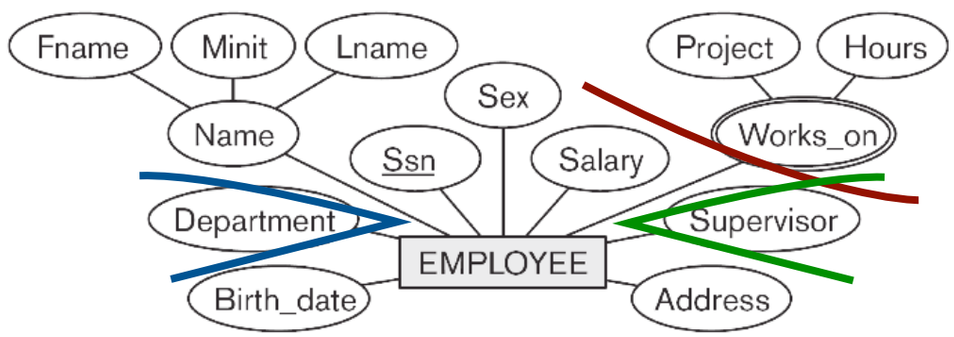
- An employee works for only one department but may work on more than one project (not necessarily controlled by the same department).
- We wish to store how many *hours* an employee works on each project and which employee supervises which employee.
- For each employee, we wish to store the employee's *name*, *social security number*, *address*, *salary*, *gender*, and *birth date*.
- We also wish to store information about the **dependents** of each employee, viz., their *name*, *gender*, *birth date*, and *relationship to the employee*.

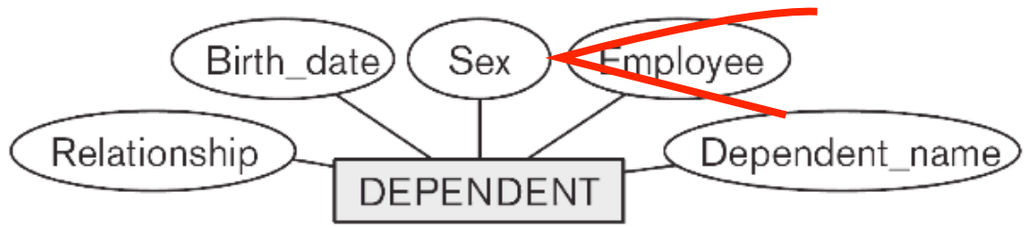
ER Modelling: From Implicit to Explicit Relationships

- To make relationships explicit we:
  - Change attributes that represent relationships into relationship types
  - Determine the cardinality ratio and participation constraints of each relationship type



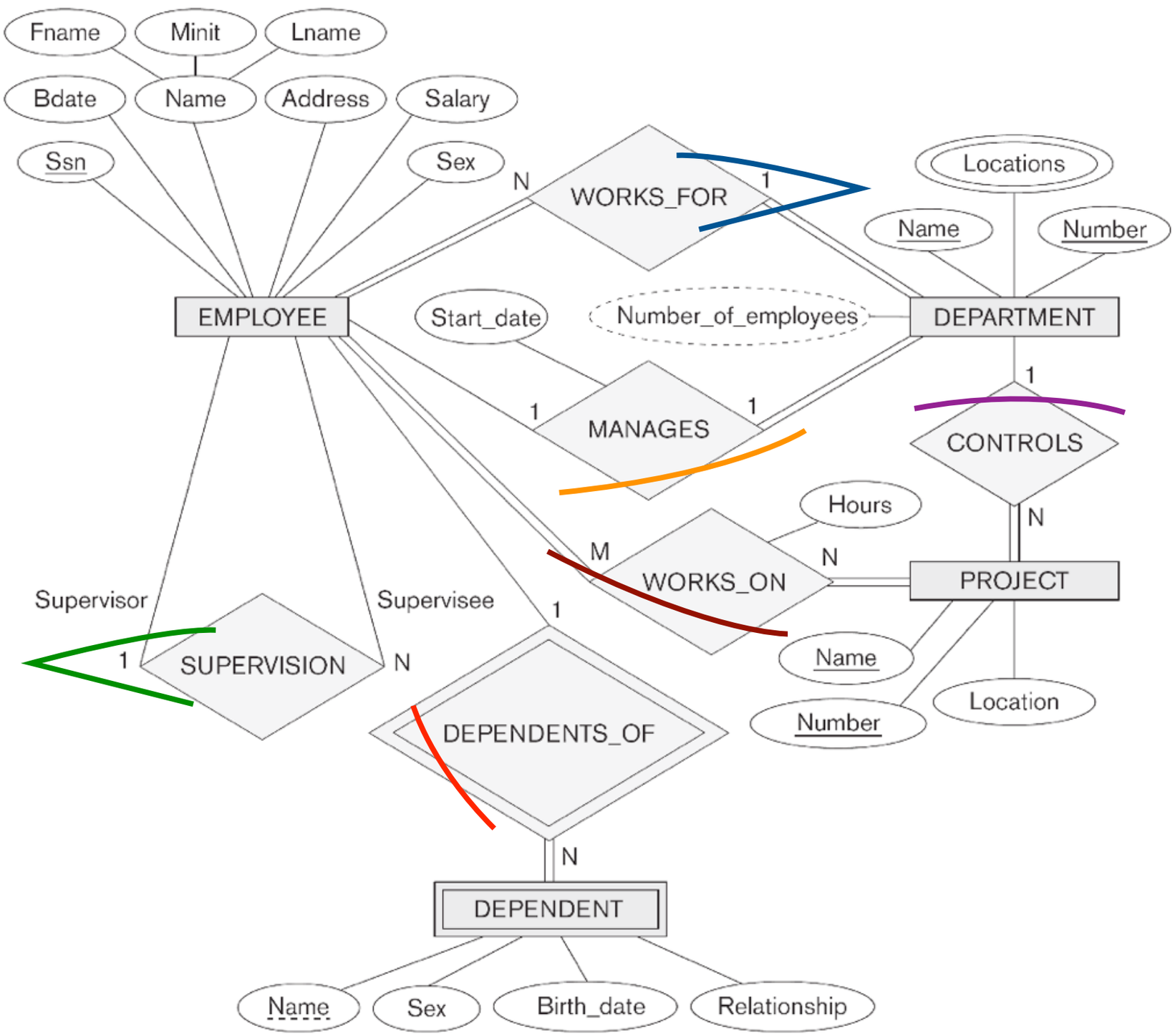






Conceptual Design: COMPANY Database

X



manages = dept.manager + dept.manager_start_date

works_for = emp.department

works_on = emp.works_on

controls = proj.controlling_department

dependents_of = dep.employee

supervision = emp.supervisor

In The Next Lecture

- (Study the notation, read the mandatory readings!)
- We'll continue our exploration of conceptual modelling.
- We'll see that how the ER conceptual model can be enhanced to capture more application semantics.

The University of Manchester

L3

Designing Databases: The Enhanced Entity-Relationship Approach

Fundamentals of Databases

Alvaro A A Fernandes, SCS, UoM

Readings

This lecture relies on your having read the assigned mandatory readings associated with this lecture:

Ramez Elmasri and Shamkant Navathe. Fundamentals of Database Systems. New International Edition. Pearson, 2013. pp. 246-286.

You can read the material online using this link:

lib.myilibrary.com/Open.aspx?id=527132

This is a shared resource. Don't hog it. When you're done, logout and close the browser to free it for others.

If you fail to study the material, you're likely to find the lecture harder to follow.

More information is available in the Learning Activities Manual.

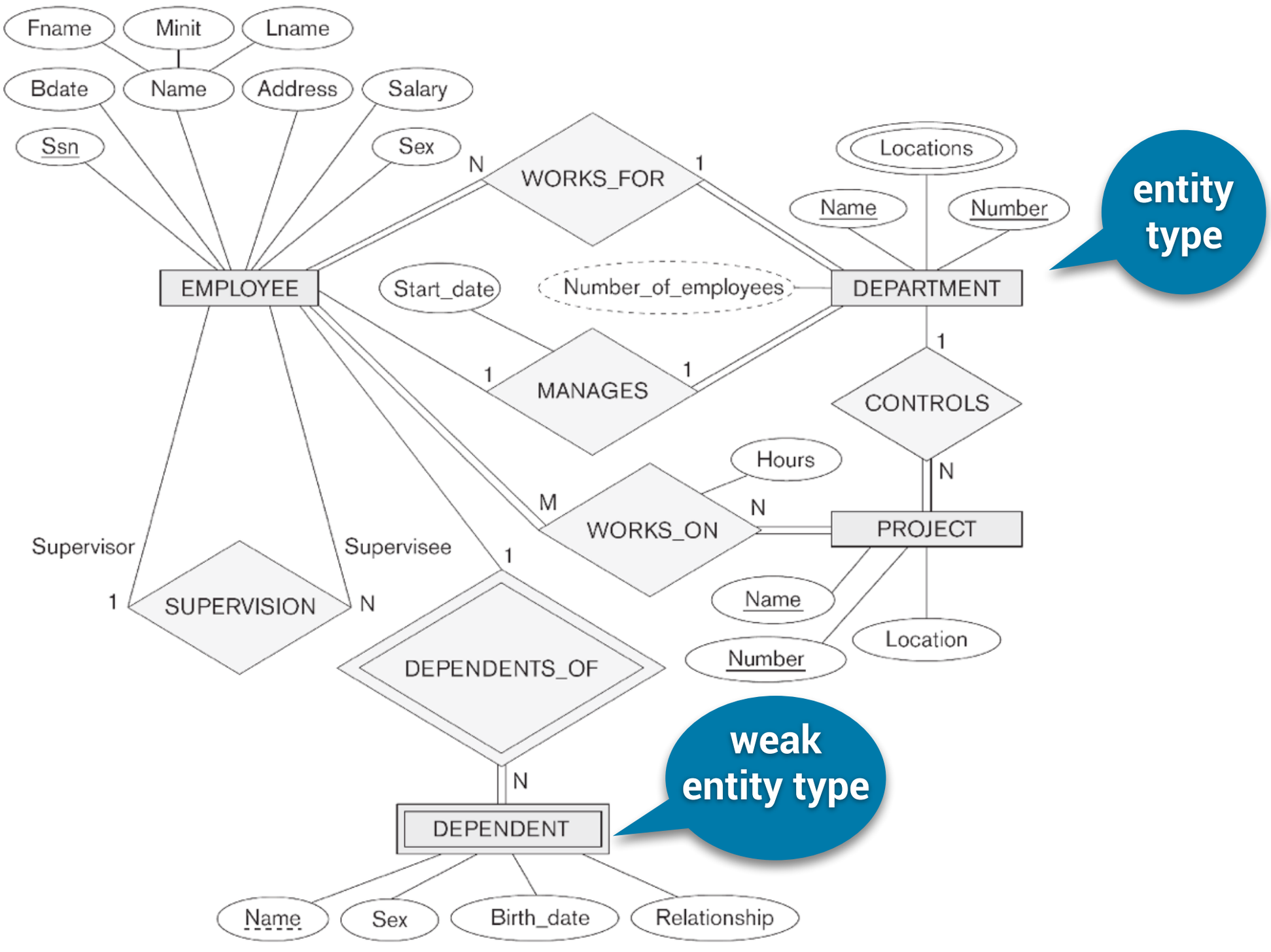
In Previous Lectures

- We learned that data is an enterprise asset and that DBMSs are crucial to manage it well.
- We learned the importance of adopting different levels of abstraction in designing and implementing databases.
- We learned how data models lead to a distinction between schemas and instances that enables a logical view of the data.
- We learned of the practical benefits of performing conceptual modelling before logical design and why the ER approach serves well this purpose.

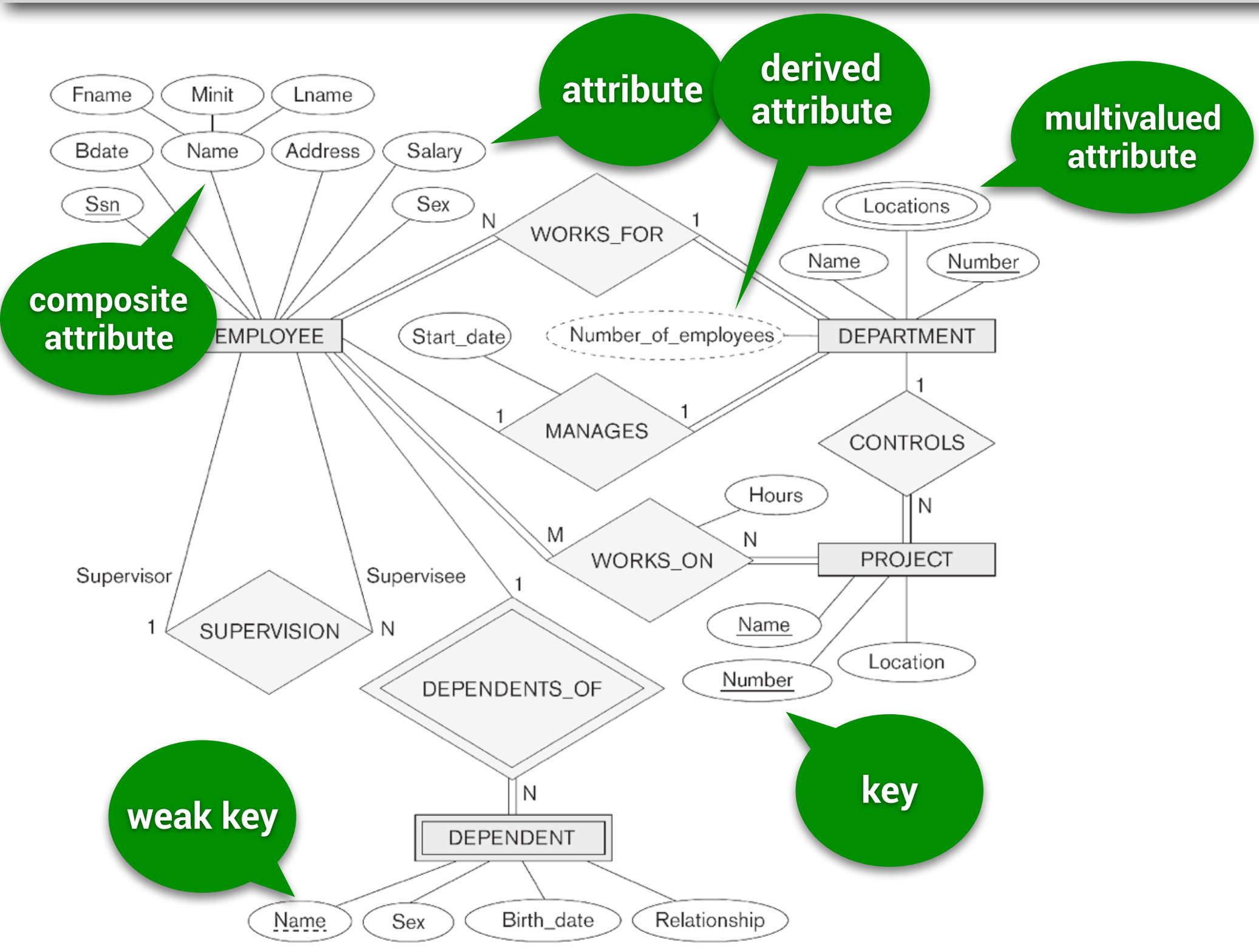
In This Lecture

- What is the enhanced entity-relationship model (EER)?
- What are specialization/ generalization (S/G) processes in EER modelling?
- What is EER-inheritance and what does it apply to?
- What constraints apply to S/G relationships?
- How do S/G hierarchies and lattices arise?
- What are union types?
- What design guidelines are useful in EER modelling?

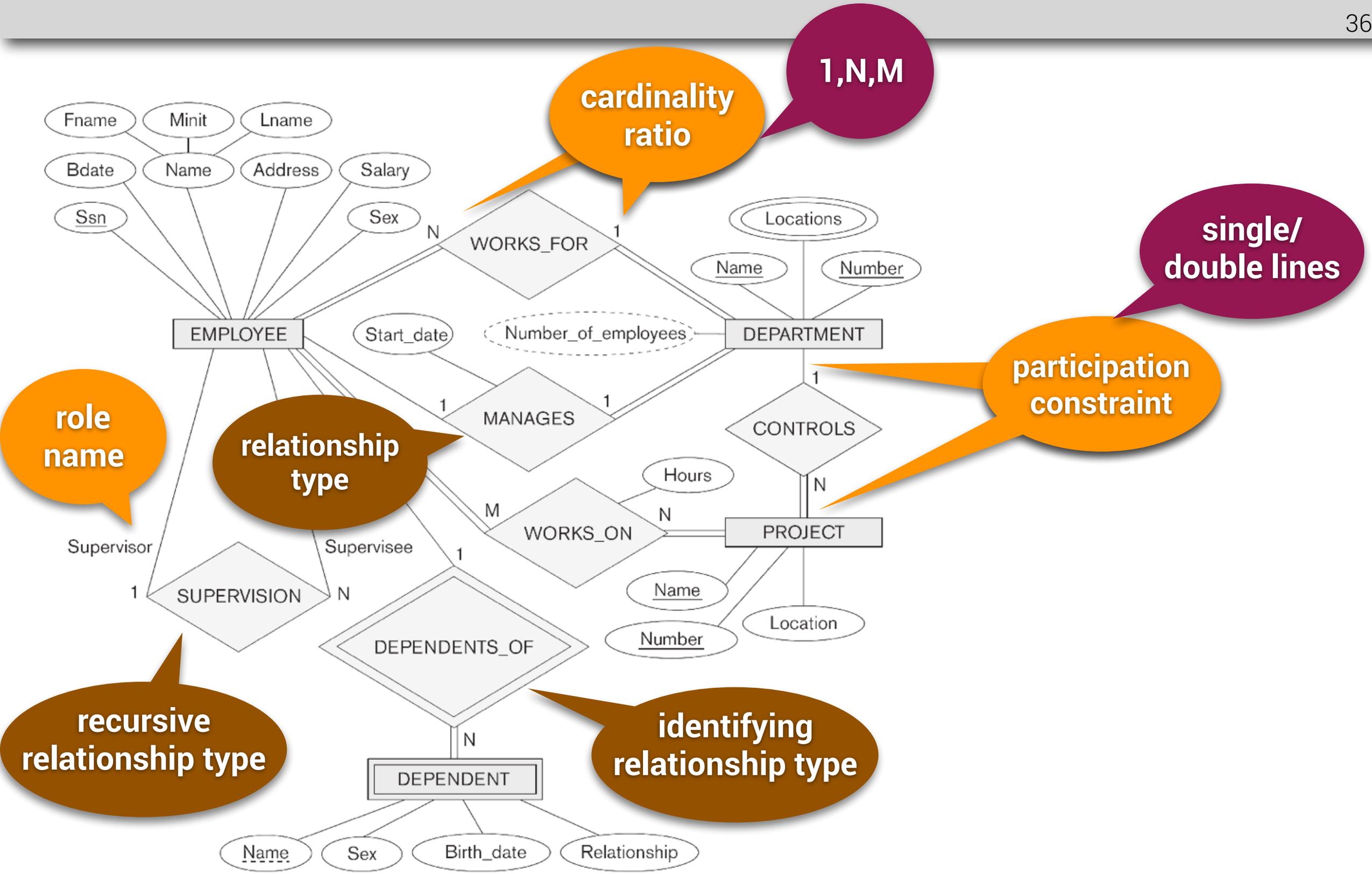
Review: ER Constructs and Notation



Review: ER Constructs and Notation



Review: ER Constructs and Notation



Specialization and Generalization

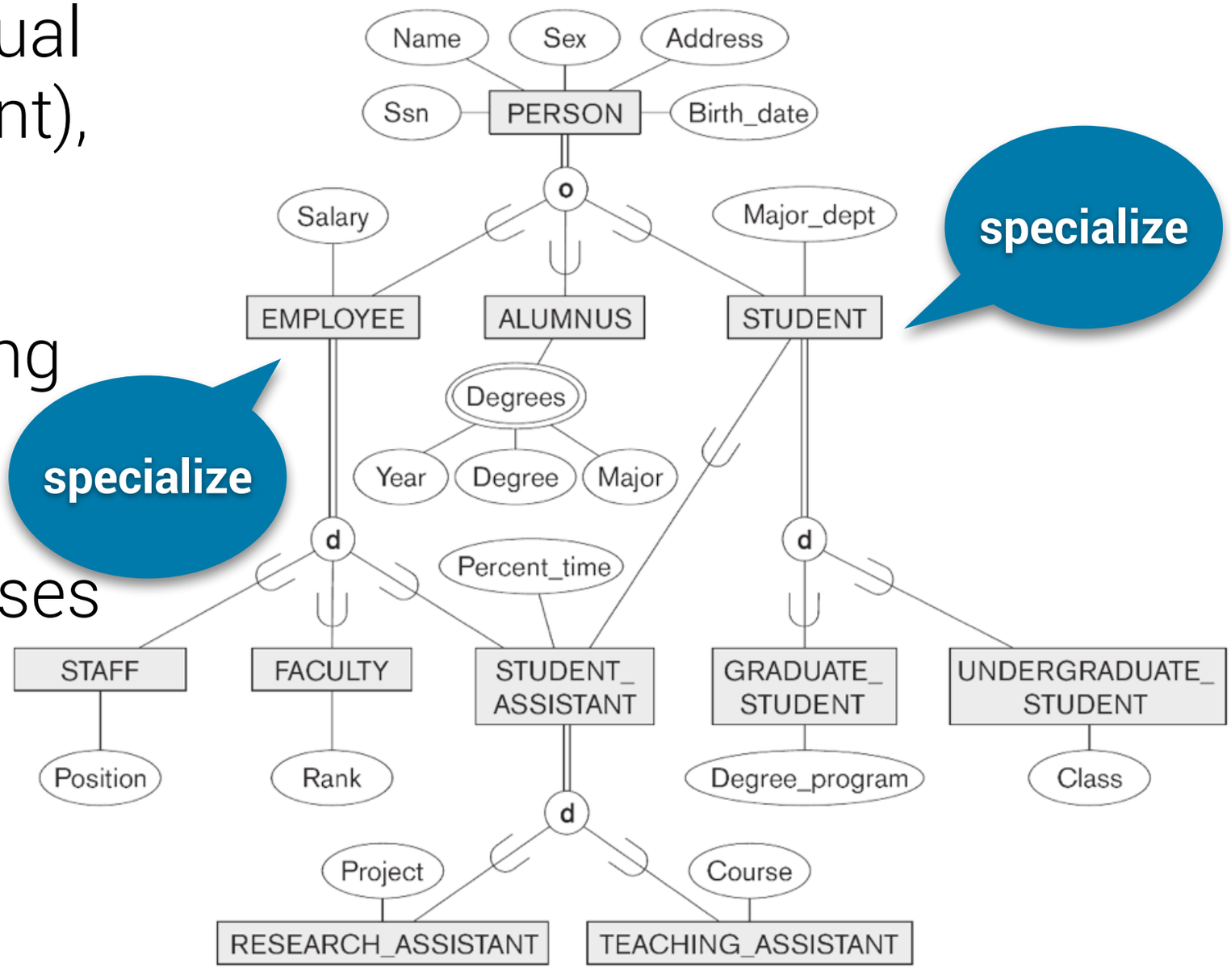
- In conceptual modelling, it is common for the designer to identify a set-subset relationship between entities.
- Some of these are useful in capturing relevant semantics.
- By 'relevant' here, we mean something that allows the final DBMS (and applications that use it) to be
  - more accurate in how they capture reality, and
  - (often) more efficient in doing so.

An Intuitive Example

- Assume you have a STUDENT entity.
- You then realize that some students have different attributes that apply to them.
- For example, a PhD student may have an account number to be used when s/he is paid for teaching assistant hours, but an UG student doesn't.
- An UG student accumulates credits, but a PhD doesn't.
- This suggests that it is useful (more accurate, and possibly more efficient) to specialize STUDENT into PhD_STUDENT and UG_STUDENT.
- This is top-down (i.e., from a set to its subsets) conceptual analysis.

Refining ER Schemas with Specialization/Generalization

- In top-down conceptual analysis (or refinement), we:
  - start with an existing entity type
  - identify its subclasses of interests
  - then specialize

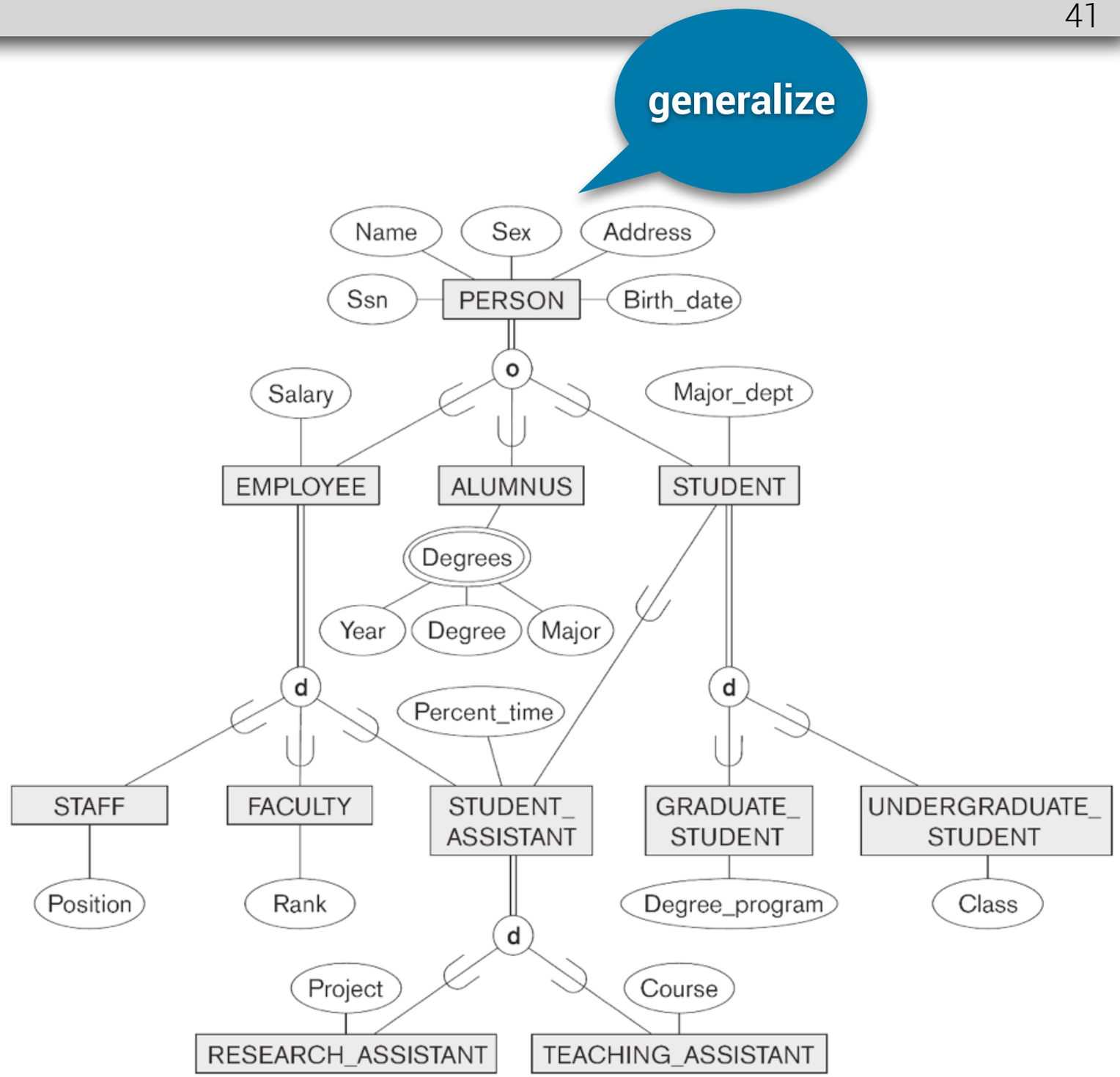


Specialization v. Generalization

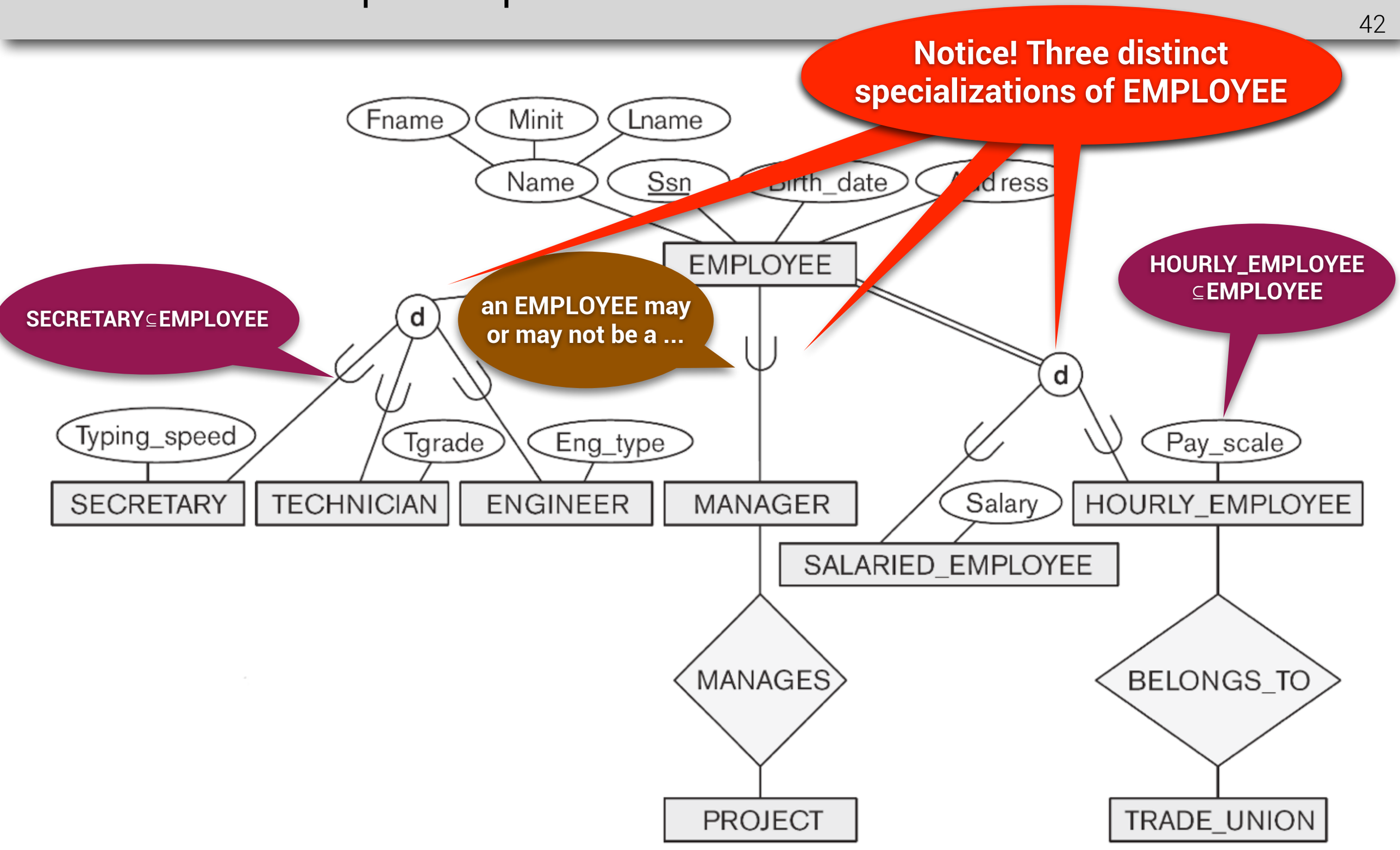
- The previous example is of a specialization (i.e., we start with a set and define one or more subsets of it).
- In generalization, we start with one or more sets and define a superset of it.
- For example, one might have started with entities such as BORROWER, MORTGAGE_HOLDER, ACCOUNT_HOLDER and then decide to generalize these into CUSTOMER.
- This is bottom-up (i.e., from several sets to their superset) conceptual synthesis.

Refining ER Schemas with Specialization/Generalization

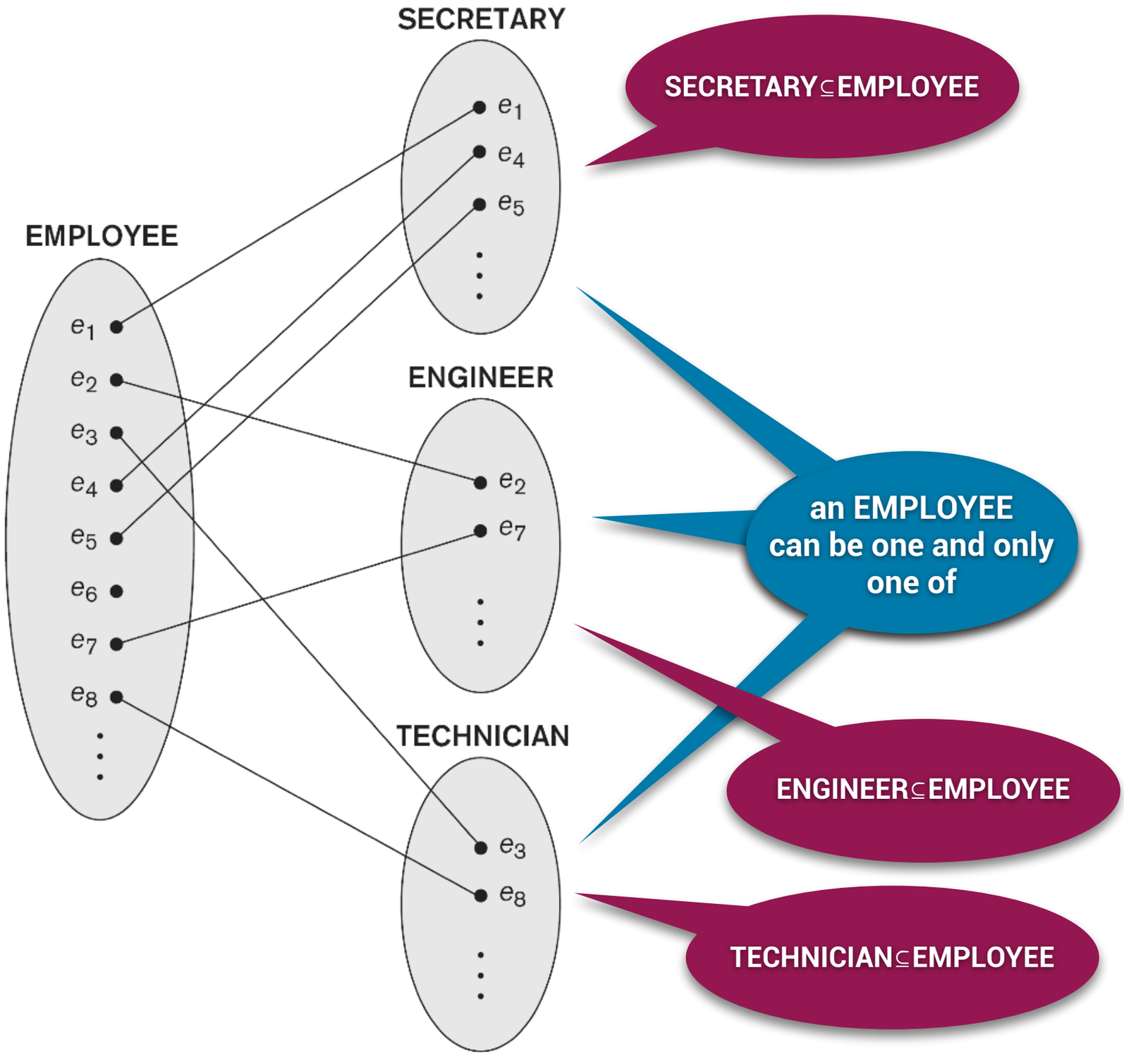
- In bottom-up conceptual synthesis, we:
  - start with with a set of existing entity types
  - identify their superclasses of interest
  - then generalize



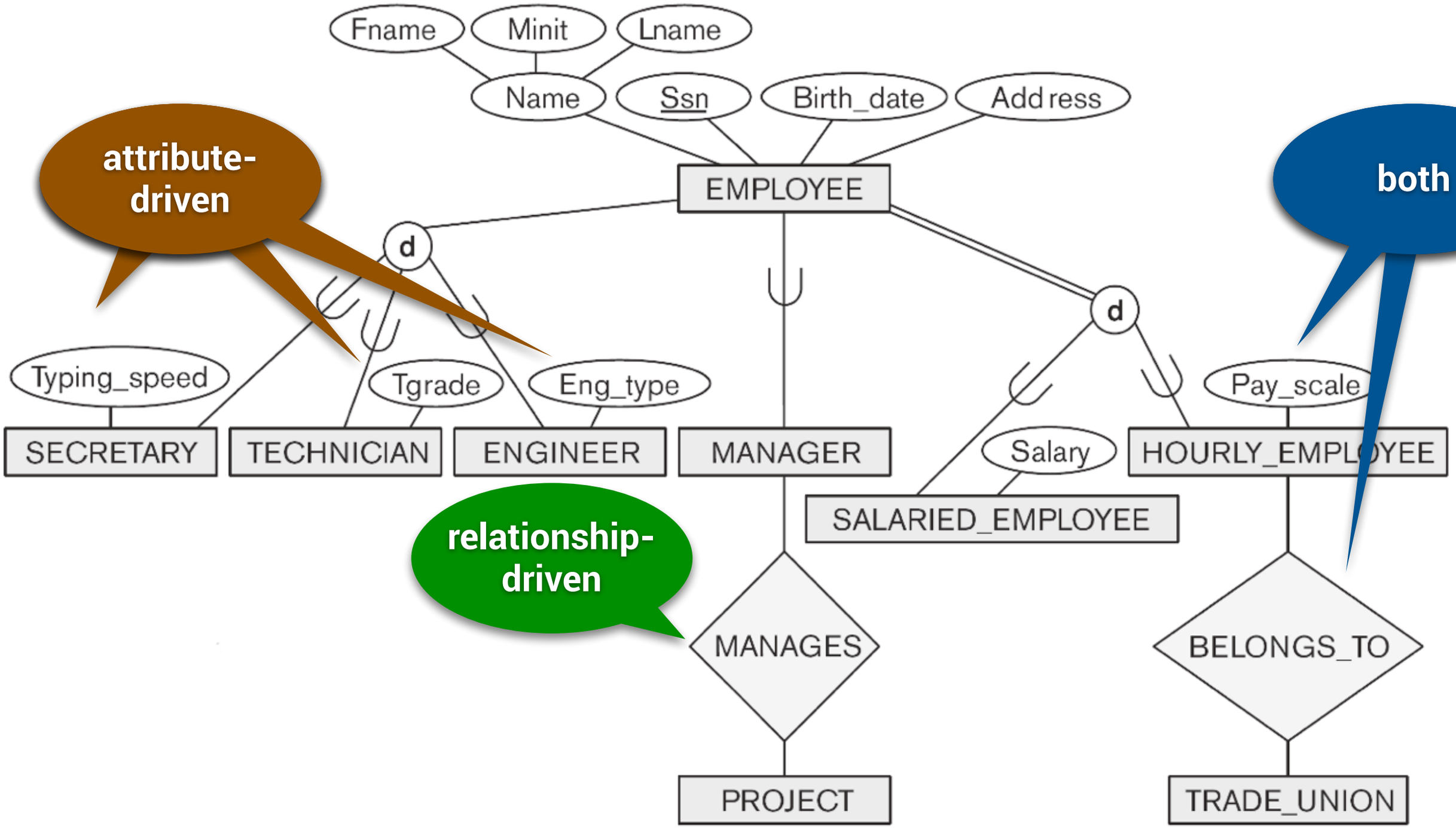
EER: Example Specializations



EER: Example Instances of a Specialization



EER: Attribute/Relationship-Driven Identification



EER: Subclass Characterization

- An entity subtype may be:
  - **Attribute-defined**
  - **Predicate- (or condition-) defined**
  - **User-defined**

Assuming that both salary or residency are stored attributes, then the following subtypes are **attribute-defined**:

Lower_Tax_Employee ⊂ Employee

≡

{e in Employee |
e.salary < 50K and
e.residency = 'UK'}

Upper_Tax_Employee ⊂ Employee

≡

{e in Employee |
not e in Lower_Tax_Employee}

EER: Subclass Characterization

- An entity subtype may be:
  - **Attribute-defined**
  - **Predicate- (or condition-) defined**
  - **User-defined**

Assuming that residency is stored but EU_member is a Boolean-valued function, then the following subtypes are **predicate-defined**:

Home_Student ⊂ Student ≡
{s in Student |
s.residency = country and
EU_member(country)}

Overseas_Student ⊂ Student ≡
{s in Student |
not s in Home_Student}

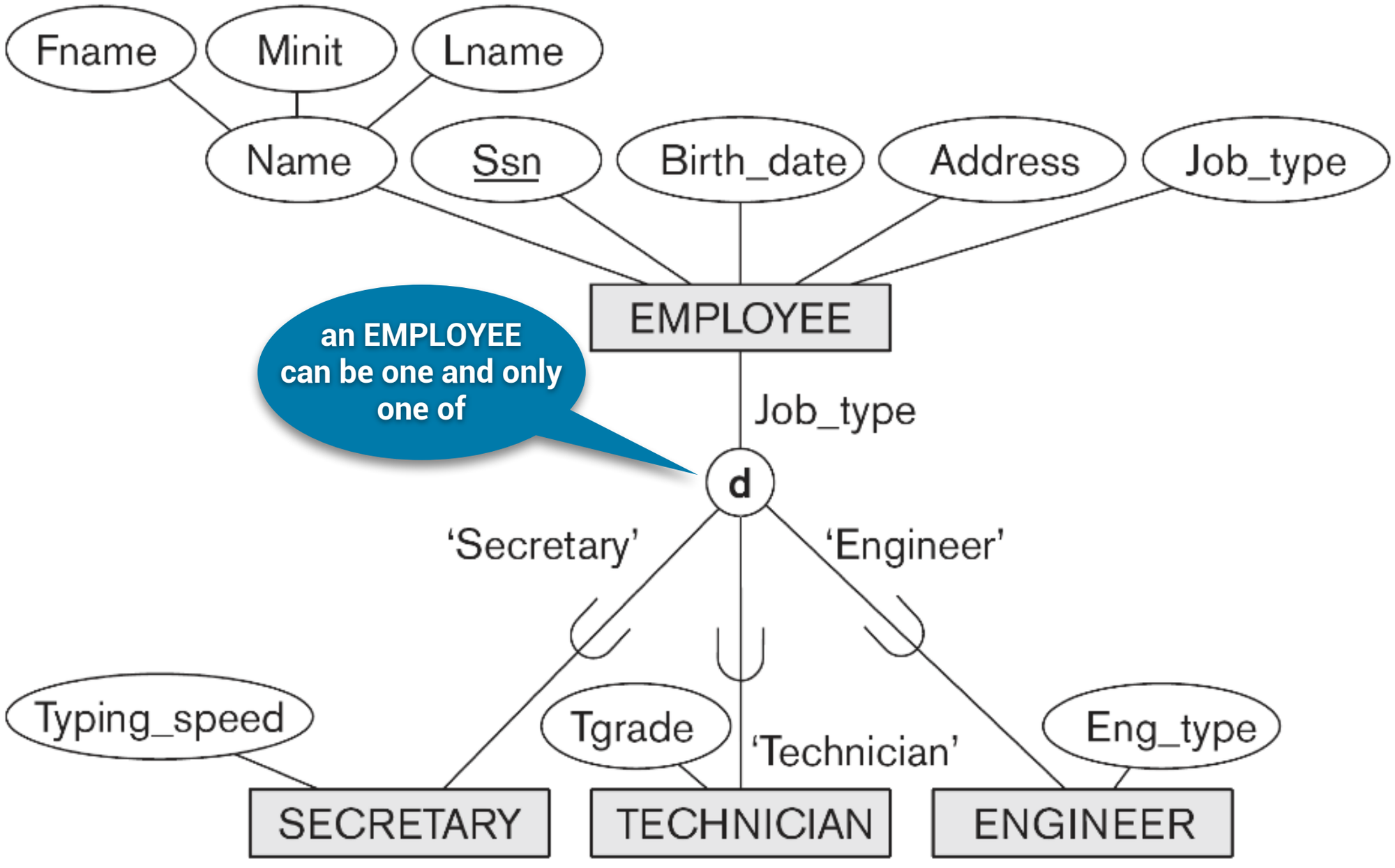
EER: Subclass Characterization

- An entity subtype may be:
  - **Attribute-defined**
  - **Predicate- (or condition-) defined**
  - **User-defined**

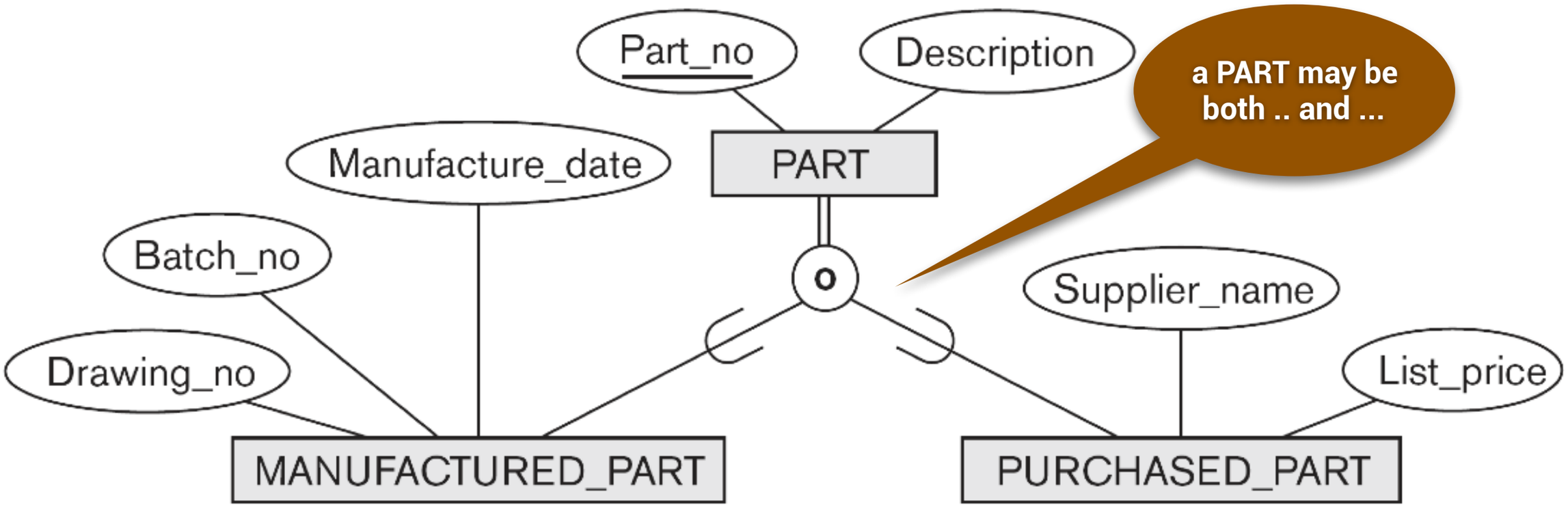
Assuming that prizeAwarded is a user input (as opposed to a programmed function), then the following subtype is **user-defined**:

PrizeRecipient ⊂ Student ≡
{s in Student |
prizeAwarded ←}

EER: Constraints on Subclass Relationships



EER: Constraints on Subclass Relationships



In The Next Lecture

X

- We'll start learning about the logical model of data known as the relational model.
- As a logical model, it is directly supported by existing DBMSs, and therefore is our target for implementation.

MANCHESTER 1824
The University of Manchester

L4

The Relational Model

Fundamentals of Databases

Alvaro A A Fernandes, SCS, UoM

Readings

This lecture relies on your having read the assigned mandatory readings associated with this lecture:

H. Garcia-Molina, J. D. Ullman, and J. Widom. Database Systems: The Complete Book. New International Edition, 2nd Edition. Pearson, 2014. pp. 13-62.

You can download the material using this link:

https://contentstore.cla.co.uk/secure/link?id=b892c15f-3068-e611-80c6-005056af4099

If you fail to study the material, you're likely to find the lecture harder to follow.

More information is available in the Learning Activities Manual.

In the Previous Lecture

- We learned about the conceptual model of data known as the entity-relationship model.
- We saw that it can be extended to capture more expressive modelling constructs such as specialization and generalization.

In This Lecture

- We'll start to have a deeper look at the relational model that underpins the most widely-deployed DBMSs today.
- We'll also begin to look at how SQL implements the relational languages by looking at its DDL and DML facilities.

What are the basic notions in data modelling?

- Given an **application domain**, e.g., a university, we want to
  - identify the **information needs** that arise and
  - **formally represent** them
  - as a means to guide the development of effective and efficient **data provision**.
- We want to do so in a **wide and comprehensive manner**, keeping data and applications separate.
- We need a **DBMS** to achieve this.

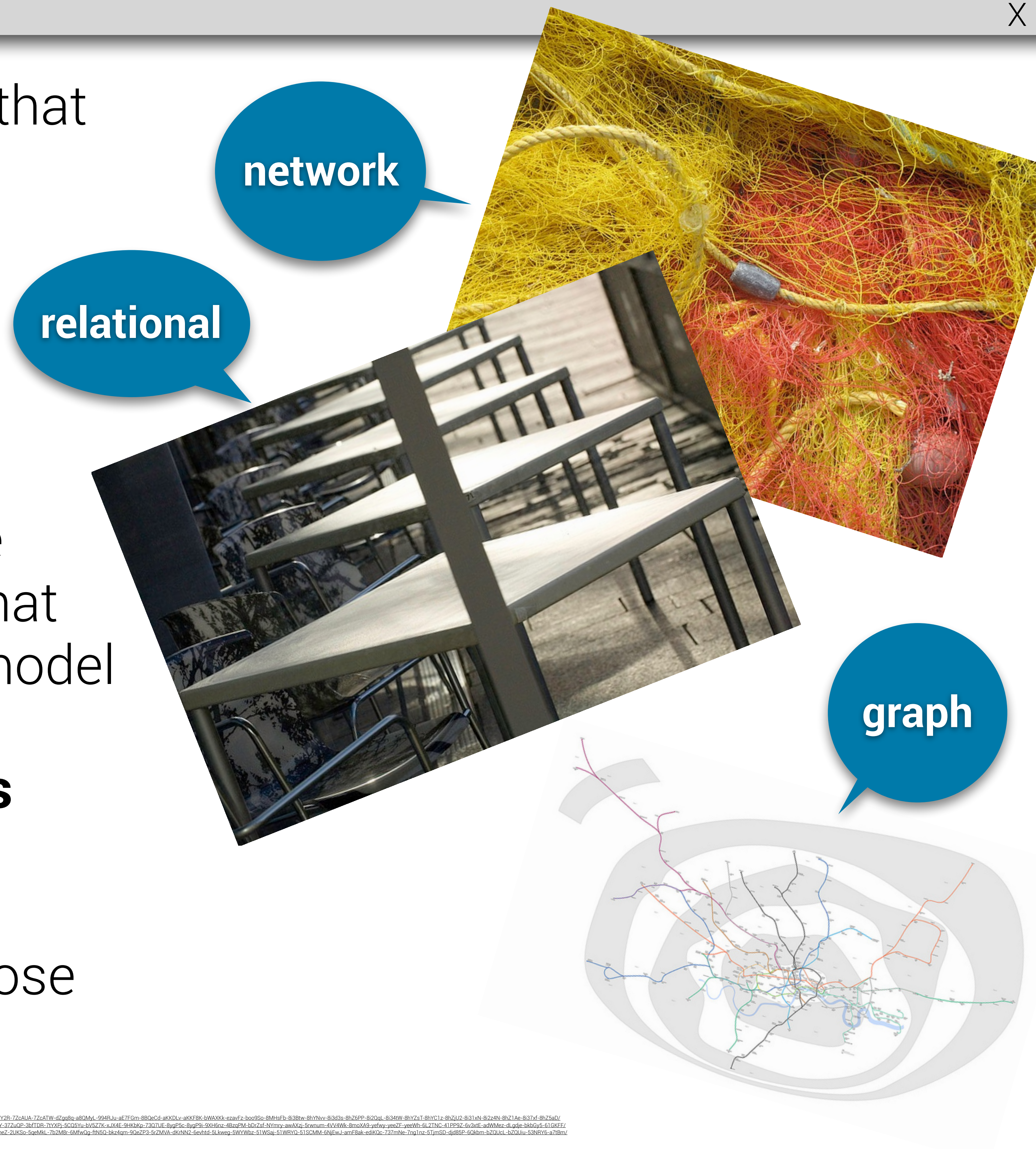
mancunion.com; barrymieny.deviantart.com; oracle.com

What are the basic notions in data modelling?

X

- We choose a **data model** that is implemented by some DBMS.

- Then, the DBMS will be capable of:
  - making the most of the **structural properties** that characterize the data model
  - enforcing its **semantics**
  - deriving **logical consequences** from those semantics



http://www.flickr.com/photos/infomatique/306917951/sizes/z/in/photolist-t831P-ruaV-4JQHjG-27j8Q-9y5wmt-4ZJwJQ-aA3Dco-4vFqea-bKeSf-drwqYE-9LT2KS-ezdGKQ-ezdJD9-acdwzg-ez8J7T-8i3qjd-8i3sdh-8hYnXn-8i3bRU-8hYDeX-5Kyue-25GgGW-7ha7o2-2adY2R-7ZcAUA-7ZcATW-dZgq8q-a8QMyL-994RJu-aE7FGm-8BQeCd-aKKDLv-aKKF8K-bWAXKk-ezavFz-boo9So-8MHsFb-8i3Btw-8hYNvv-8i3d3s-8hZ6PP-8i2QqL-8i34tW-8hYZsT-8hYC1z-8hZjU2-8i31xN-8i2z4N-8hZ1Ae-8i37xf-8hZ5aD/
http://www.flickr.com/photos/khianti/3980417845/sizes/z/in/photolist-74JFxT-74NhQq-oFa8-9mcPnq-2Jbut-3mdkFU-4VM2KP-61MBXY-xnotr-61HpfH-61Mz4Q-61HotB-61HqaK-61MCrY-61HrN6-61MB5G-61Mezm-61Mupu-2bFGb4-6EfoLW-gDkrY-7xZgBB-9ygBAY-37ZuQP-3bfTDR-7tYXPj-5CQ5Yu-bV5Z7K-xJX4E-9HKbKp-73Q7UE-8ygP5c-8ygP9i-9XH6nz-4BzqPM-bDrZsf-NYmry-awAXzj-5rwnum-4VV4Wk-8moXA9-yefwy-yeeZF-yeeWh-6L2TNC-41PP9Z-6v3xtE-adWMez-dLgdje-bkbGy5-61GKFF/
http://www.flickr.com/photos/inkiboo/8752689917/sizes/z/in/photolist-ekrPLR-5LpNkA-7hbFv-wb7AQ-5vqXPy-5vmDtK-5vmDgc-5vmDoe-5cxK9M-5cxKiF-5cC1Tj-bpJekW-76jkj-bmFo6-7oBWS-75KkK-abfQ1-6THey-6nvXvT-6nA66w-47eHf2-6bMsH8-4jQdm4-aWHheZ-2UKSo-5qeMkL-7b2M8r-6MfwQg-ftN5Q-bkz4qm-9QeZP3-5rZMVA-dKrNN2-6evhtd-5Lkweg-5WYWbz-51WSaj-51WRYQ-51SCMM-6NjEwJ-amF8ak-ediKQc-737mNe-7ng1nz-5TjmSD-djd85P-6Qkbm-bZQUcL-bZQUiu-53NRY6-a7tBm/

What are the basic notions in data modelling?

- For example, we need to store information about students, academics, departments, buildings, course units, etc.

- For each of those, we need, e.g., to know information about them

  - **students**: name, id, programme, courses taken, marks, etc.

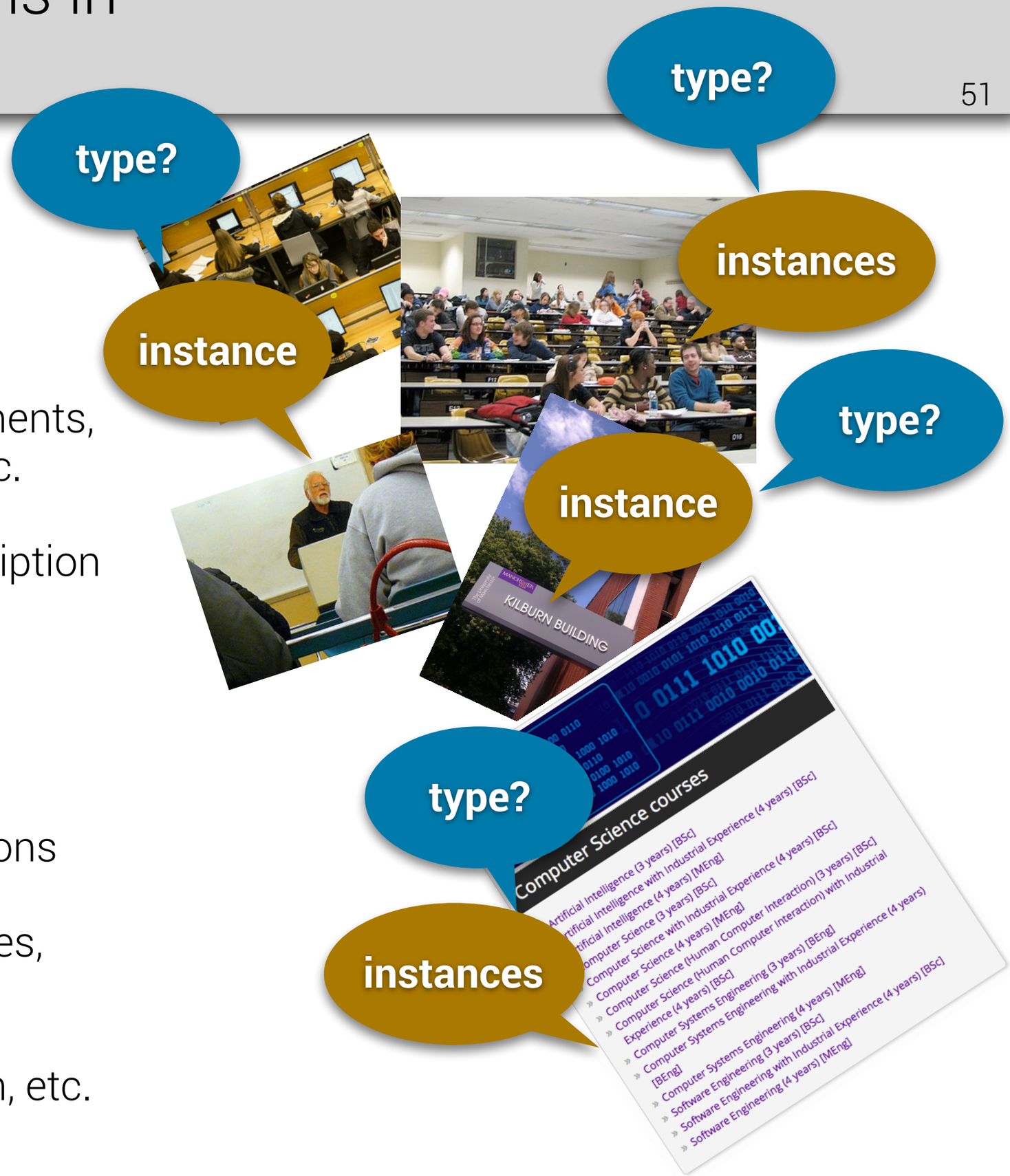
  - **instructors**: name, id, courses taught, etc.

  - **departments**: name, building, programmes offered, etc.

  - **course units**: name, credits, programme, students, lecturer, location, etc.

What are the basic notions in data modelling?

- **Abstractions** for types
  - concepts, entities, objects, **relations**, nodes, etc.
  - students, instructors, departments, course units, programmes, etc.
- **Properties** for value-based description
  - properties, **attributes**, etc.
  - name, id, credits, marks, etc.
- **Relationships** between abstractions
  - relationships, **references**, edges, etc.
  - courses taught, courses taken, etc.



What are the basic notions in data modelling?

- **Constraints** on what can be said, e.g., **referential** ones:
  - an instructor cannot teach a non-existent course

let

Jane_Eyre_Instructors

be the set of
all instructors that
teach some course unit that
is taken by
the student called Jane Eyre

- Valid **calculations or operations** on what is known, e.g., **set comprehensions**

even = {2x | x ∈ ℤ}
odd = {2x+1 | x ∈ ℤ}

The Relational Model

- **Single collection type**, i.e., **tables**, so a very **simple** model
- Leads to simple, but surprisingly expressive, **algebra** and **calculi**
- Effective: tables often match **how we think about data**
- Leads to **efficient implementations**: it underlies the most important commercial DBMSs today
- Basis of **SQL**: the most widely-used language for interacting with databases today

JaneEyreInstructors =

{ i.n |

teaches(i.n,c.n) ∧
takes(s.n,c.n) ∧
s.n = 'Jane Eyre' ∧
i ∈ instructor ∧
c ∈ course ∧
s ∈ student

}

a tuple relational calculus

equivalent!

a relational algebra

JaneEyreInstructors =

$\pi_{teaches.i.n}($
$\sigma_{takes.s.n='Jane\ Eyre'}($
teaches $\bowtie_{course}$ **takes))**

The Relational Model

- **Regular**, **rectangular tables** with **typed, atomic columns**
- Table definitions capture relation schemas/types, e.g., instructors
- Columns capture properties, e.g., instructor name
- Rows record individuals, e.g., 15151 (whose name is Mozart)
- Table states capture relation instances (i.e., type extensions)
- Formally, relation instances are subsets of the *n*-ary Cartesian product over attribute domains

domain integrity

homogeneous, type-conformant columns

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
[illegible]43	Singh	Finance	80000

(a) The *instructor* table

entity integrity

uniquely identifiable rows

The Relational Model

- Relationships are captured by references from a column in one table to a column in another

- For example, to which *department* an *instructor* belongs

- If we have operations on tables (e.g., selecting rows, projecting and aggregating column values, combining tables, etc.) we can answer a question such as

  - Is Computer Science broke?!!

values in the referring column must occur in the referred column

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

(a) The *instructor* table

referential integrity

dept_name	*building*	*budget*
Comp. Sci.	Taylor	100000
Biology	Watson	90000
Elec. Eng.	Taylor	85000
Music	Packard	80000
Finance	Painter	120000
History	Painter	50000
Physics	Watson	70000

(b) The *department* table

What is SQL?

- SQL is the most widely-used relational language
- It is based on a (declarative) relational calculus (i.e., set comprehensions)
- The most important SQL queries can be translated into an equivalent (procedural) algebraic expression for evaluation
- This approach of translating from a declarative expression to an equivalent procedural one
  - frees application designers from having to write efficient query execution plans
  - the DBMS is delegated the task of searching to find one such plan

What is SQL?

- SQL has an expressive QL subset centred on the well-known

  SELECT

  FROM

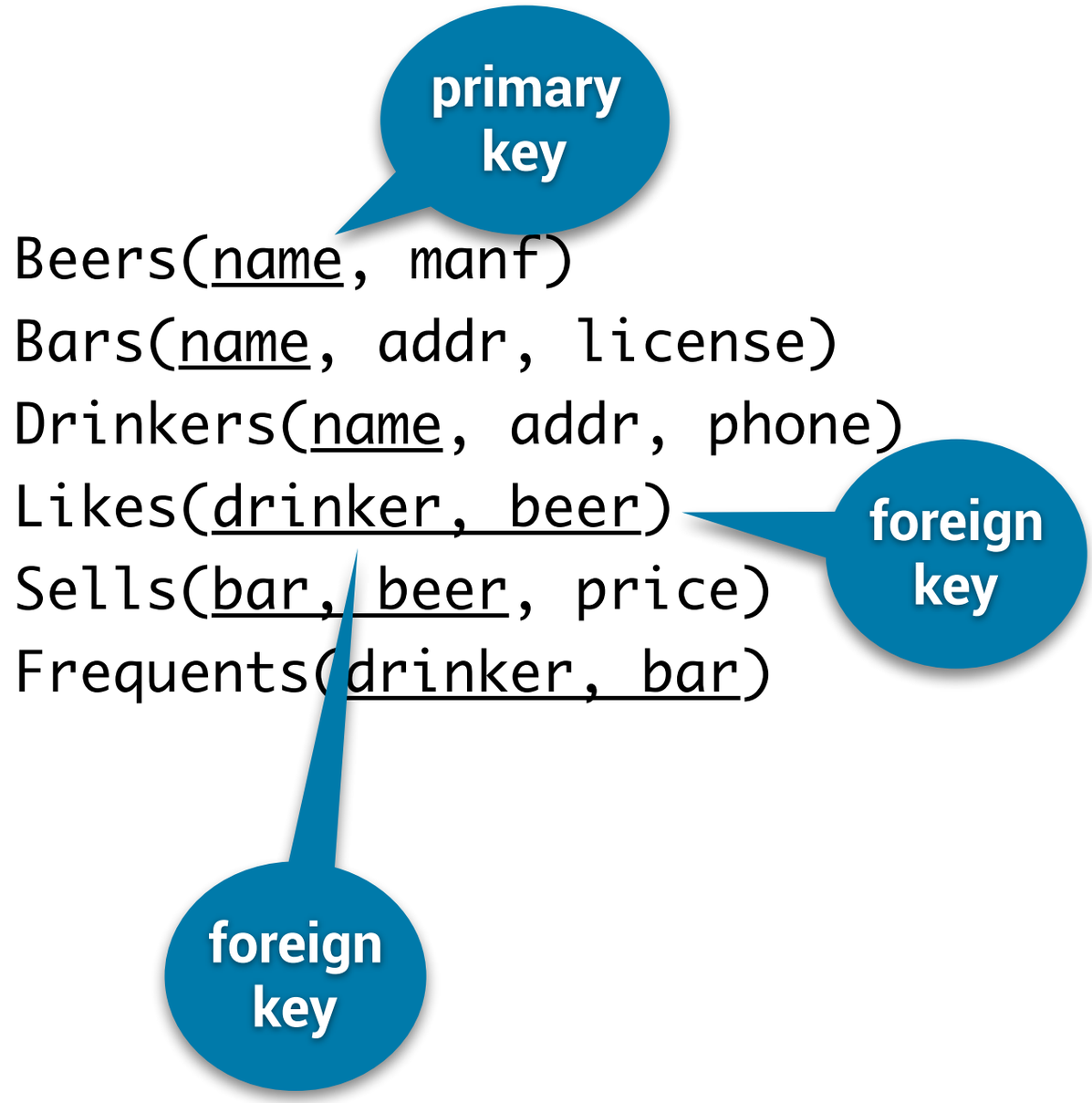
  WHERE

  GROUP BY idiom

- SQL has expressive DML capabilities:
  - Reactive triggers
  - Procedural constructs (e.g, loops, assignments, etc.)

BBD DB :: A Running Example

- Assume a database about beers, bars and drinkers.
- There must exist a choice of *a set of attribute values that uniquely identifies a tuple*: we call it the **primary key** of the relation.
- One relation may *use the primary key of another relation to model a relationship*: we call it a **foreign key**.



SQL as a DDL: Schemas

CREATE
ALTER
DROP

INDEX
VIEW
...

INTEGER
FLOAT
REAL
CHAR(n)
VARCHAR(n)
DATE
TIME
...

```
CREATE TABLE Bars (
    name    CHAR(20),
    manf    VARCHAR2(20),
    CONSTRAINT ba_pk PRIMARY KEY (name)
);
```

```
CREATE TABLE Beers (
    name    VARCHAR2(20),
    addr    VARCHAR2(20),
    license CHAR(3),
    CONSTRAINT be_pk PRIMARY KEY (name)
);
```

```
CREATE TABLE Sells (
    bar     CHAR(20),
    beer    VARCHAR2(20),
    price   REAL,
    CONSTRAINT s_pk PRIMARY KEY (bar, beer)
    CONSTRAINT s_fk_bar  FOREIGN KEY (bar)
            REFERENCES Bars(name),
    CONSTRAINT s_fk_beer FOREIGN KEY (beer)
            REFERENCES Beers(name)
);
```

SQL as a DDL: Constraints

specifies what the DBMS must do to rows in this table when events take place in the Bars or the Beers table

```
CREATE TABLE Sells (
    bar     CHAR(20),
    beer    VARCHAR2(20),
    price   REAL,
    CONSTRAINT s_pk
      PRIMARY KEY (bar, beer)
    CONSTRAINT s_fk_bar
      FOREIGN KEY (bar)
        REFERENCES Bars(name)
        ON DELETE SET NULL
        ON UPDATE CASCADE,
    CONSTRAINT s_fk_beer
      FOREIGN KEY (beer)
        REFERENCES Beers(name)
        ON DELETE SET NULL
        ON UPDATE CASCADE,
);
```

the events are either deletions or updates in the primary key

we can set the foreign key to NULL (e.g., on delete)

the default is to reject (i.e., prevent Bars from being changed)

we can CASCADE, i.e., update the foreign key here to the new value of the primary key in Bars

SQL as a DML: Insert Single


```
INSERT INTO Likes
VALUES ('Sally', 'Bud');
```

implicit order

```
INSERT INTO
Likes(beer, drinker)
  VALUES ('Bud','Sally' );
```

explicit order

```
CREATE TABLE Drinkers (
    name  CHAR(30) PRIMARY KEY,
    addr  CHAR(50) DEFAULT '123 Sesame St.',
    phone CHAR(16));
```

explicit default

implicit default

```
INSERT INTO Drinkers(name)
    VALUES ('Sally');
```

this input

```
('Sally', '123 Sesame St', NULL)
```

this result

SQL as a DML: Insert Set

d1 is made to point to Sally

d2 is made to point to all the other drinkers

```
INSERT INTO SallyPossibleBuddies
(SELECT d2.drinker
 FROM Frequents d1, Frequents d2
 WHERE d1.drinker = 'Sally' AND
       d2.drinker <> 'Sally' AND
       d1.bar = d2.bar
);
```

This ensures that Sally (d1) and the other drinker (d2) frequent the same bar.

SQL as a DML: Delete/Update


```
DELETE FROM Likes;
```

unconditional: deletes everything

delete from `Beers(name, manf)` all beers for which there is another beer by the same manufacturer.

only deletes the rows that satisfy the condition

```
DELETE FROM Beers b1
WHERE  EXISTS (SELECT name
                 FROM Beers b2
                WHERE b2.manf=b1.manf
                  AND b2.name<>b1.name);
```

```
UPDATE Drinkers
       SET phone = '555-1212'
     WHERE name = 'Fred';
```

updates one row (*name* **is PK)**

```
UPDATE Sells
       SET price = 4.00
     WHERE price > 4.00;
```

updates many rows (potentially)

In The Next Lecture

- We'll delve into the relational algebra.
- We'll look into how SQL can be mapped into the relational algebra.
- We'll explore how this declarative-to-procedural transformation is a key one in DBMSs.

MANCHESTER 1824

The University of Manchester

L5

The Relational Languages

Fundamentals of Databases

Alvaro A A Fernandes, SCS, UoM

Readings

This lecture relies on your having read the assigned mandatory readings associated with this lecture:

Karen Morton et al. Pro Oracle SQL. 2nd Edition. Apress, 2013. pp. 1-24.

You can download the material using this link:

https://link.springer.com/chapter/10.1007/978-1-4302-6221-3_1

If you fail to study the material, you're likely to find the lecture harder to follow.

More information is available in the Learning Activities Manual.

In Previous Lectures

X

- We learned that data is an enterprise asset and that DBMSs are crucial to manage it well.
- We learned the importance of adopting different levels of abstraction in designing and implementing databases.
- We learned how data models lead to schemas and instances that enable a logical view of the data.
- We learned about the main components of DBMSs and the various architectures used to deploy them.
- We learned about conceptual modelling.
- We started learning about the relational approach to logical data modelling.
- We started learning SQL, focussing on its DDL and DML capabilities.

In This Lecture

X

- What is a relational algebra?
- What are the basic operations of relational algebra?
- How do we relate SQL to relational algebra?
- What are the main constructs and query types in SQL?

A Familiar Example: Elementary Algebra

operator semantics leads to evaluation

operators: +, -, *, /

2+2

operands: values from the Reals

4*(3/2)

e.g., 6

(4*3)/2

closed: inputs and outputs from the same set

operands: variables

x+1

operator semantics leads to equivalence laws

2/(4*x)

e.g., commutativity of +

x+y = y+x

e.g., associativity of *

(x*y)*z = x*(y*z)

A Relational Algebra

Which operator semantics? How to evaluate?

more later!

relations!

Which operands?

What do variables denote?

Which operators?

σ,π,⋈ρ,δ!
∪,∩,\!

Closed? Inputs and outputs one and the same type?

yes!

e.g., which commutative?

e.g., ⋈,...

e.g., which associative?

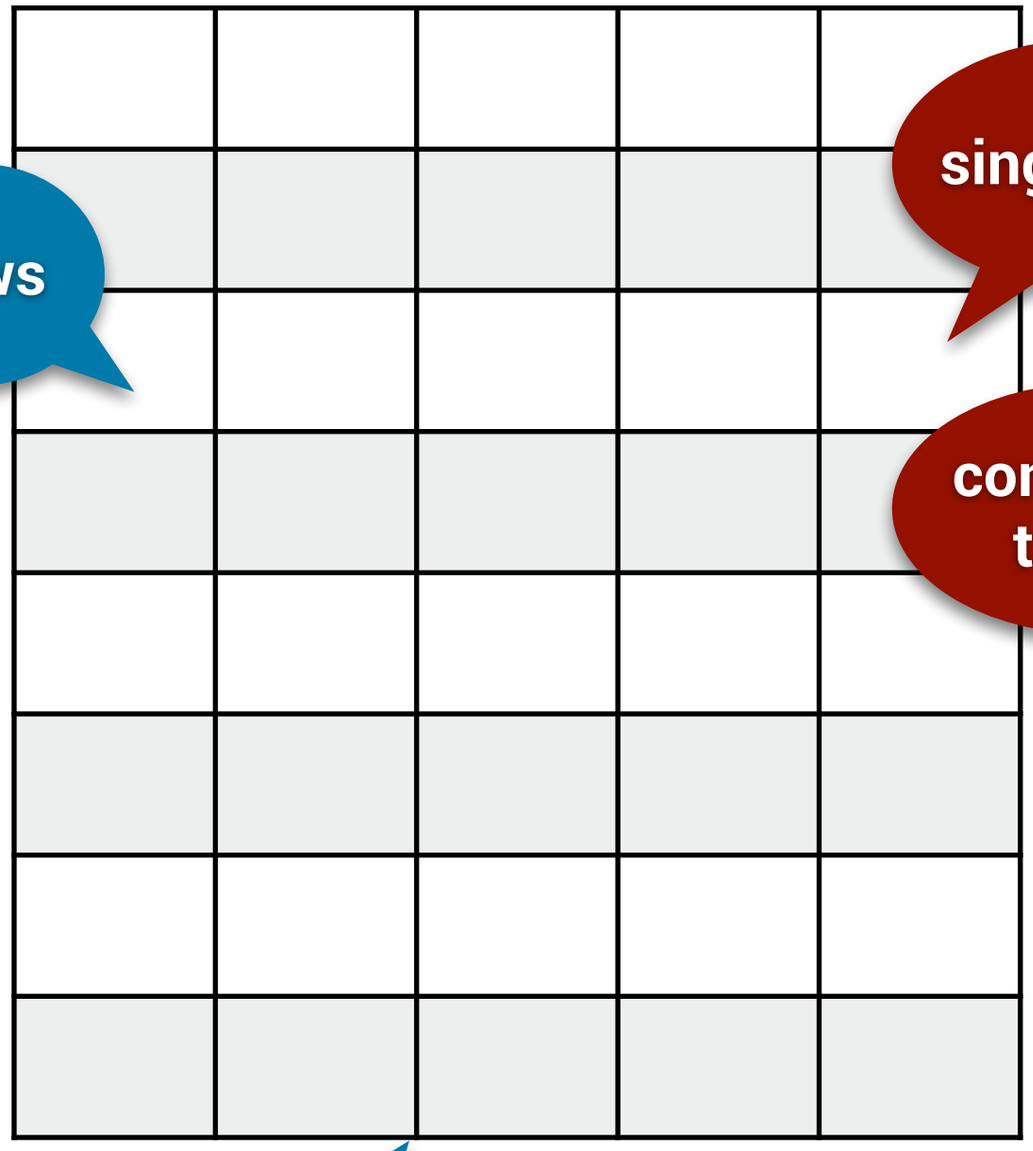
e.g., ⋈,...

Which equivalence laws does operator semantics lead to?

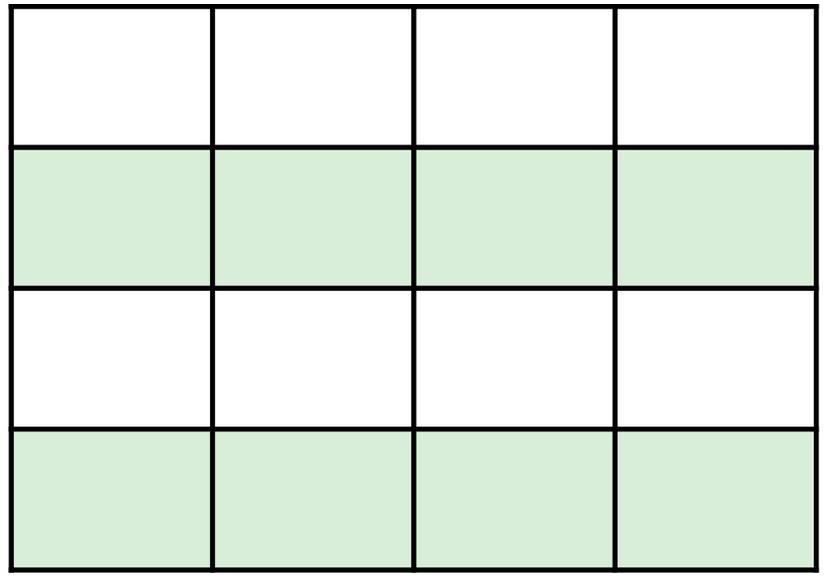
$\sigma_\theta(R \bowtie S) \equiv \sigma_\theta(R) \bowtie \sigma_\theta(S)$

How do we want to operate on relations?

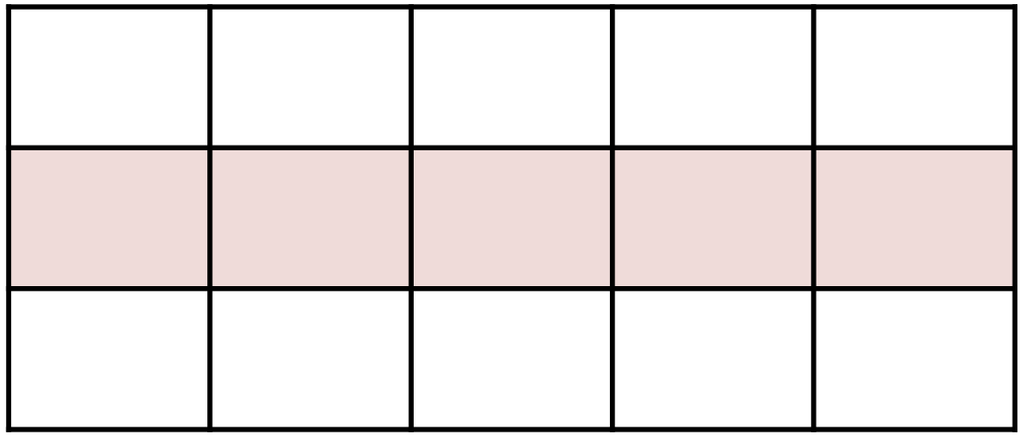
R(a:int, b:int, c:str, d:str, e:flt)



S(b:int, g:int, x:str, z:flt)



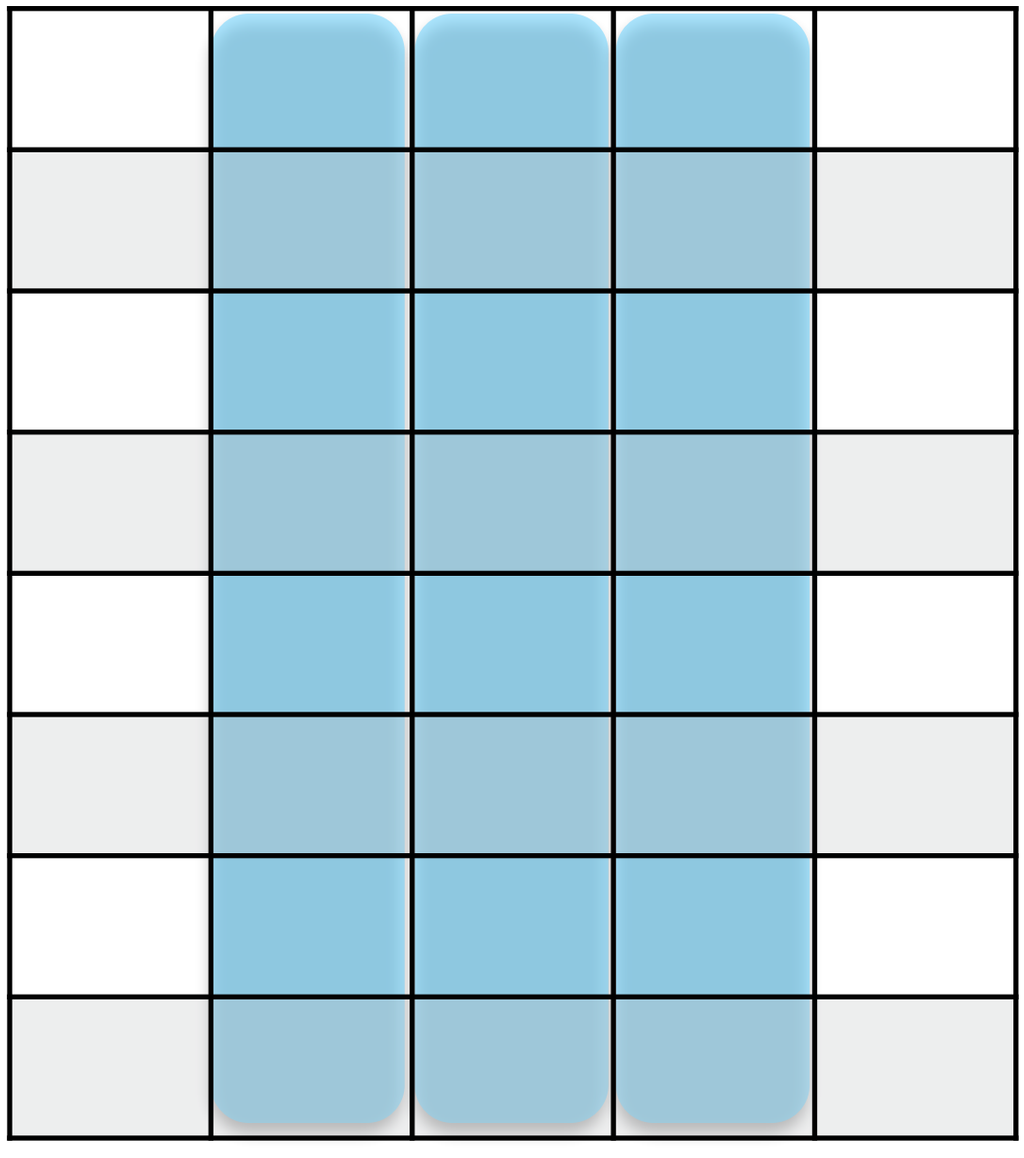
T(v:int, w:int, x:str, y:str, z:flt)



How do we want to operate on relations? The unary case

add/keep some columns

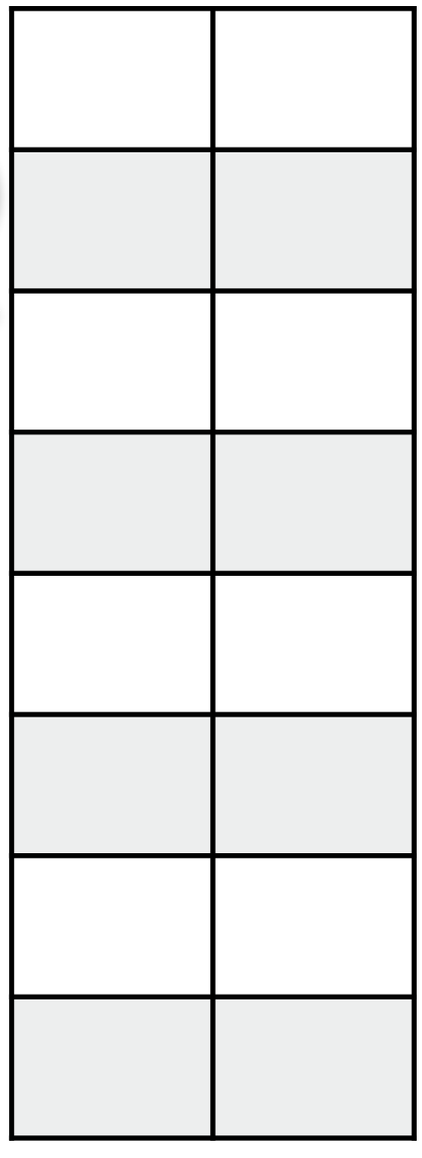
R(a:int, b:int, c:str, d:str, e:flt)



grow/ shrink left-to-right

projection: π

R'(a:int, e:flt)

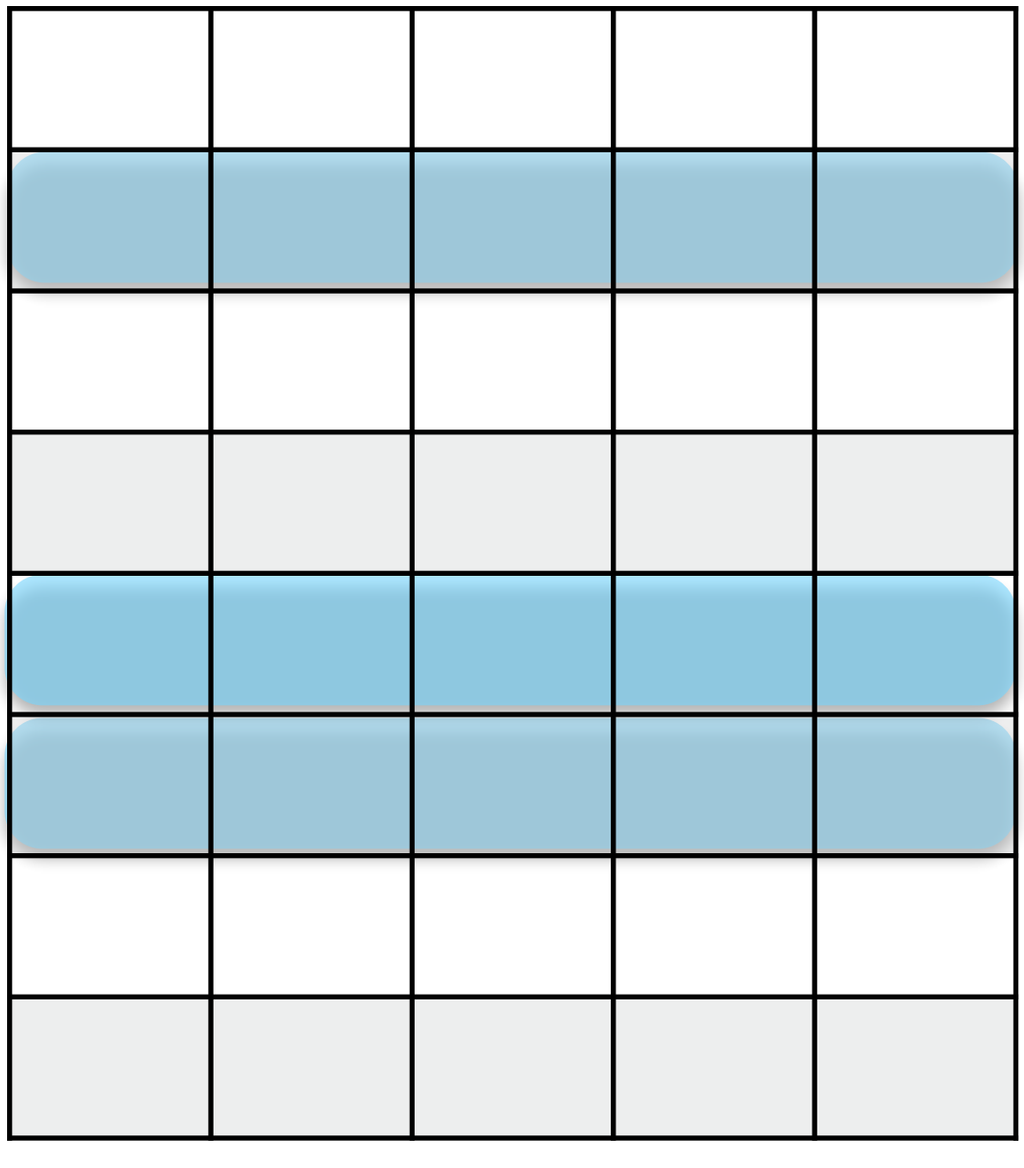


How do we want to operate on relations? The unary case

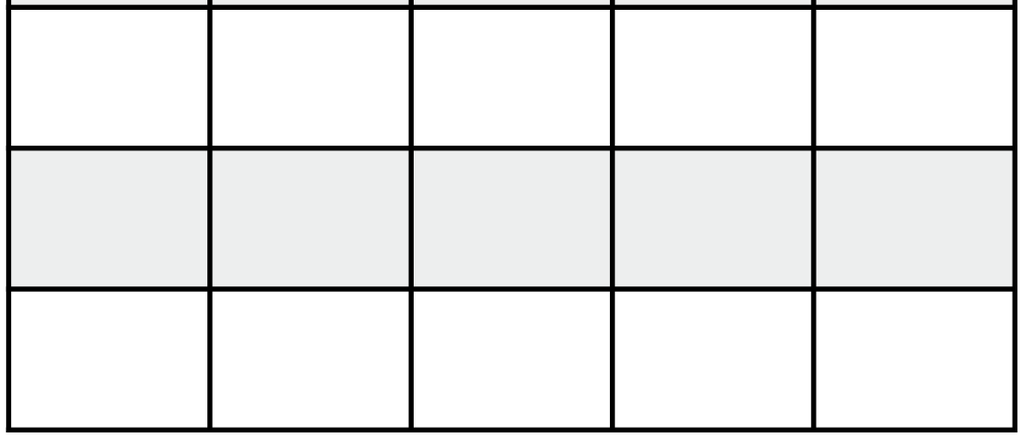
shrink top-to-bottom

keep only some rows

R(a:int, b:int, c:str, d:str, e:flt)



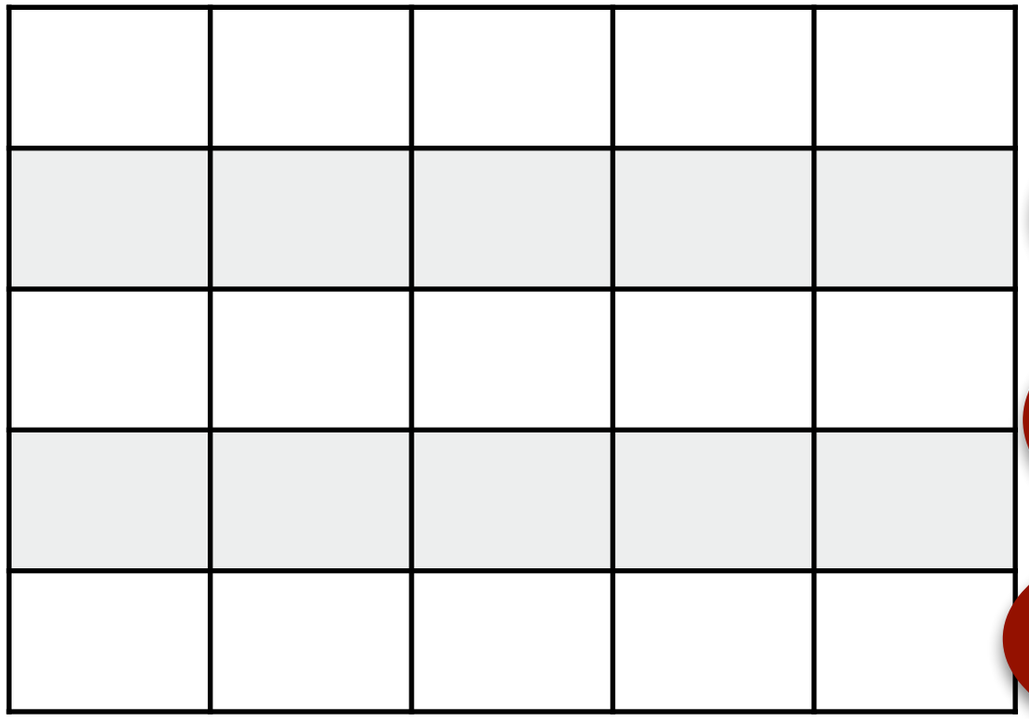
R′(a:int, b:int, c:str, d:str, e:flt)



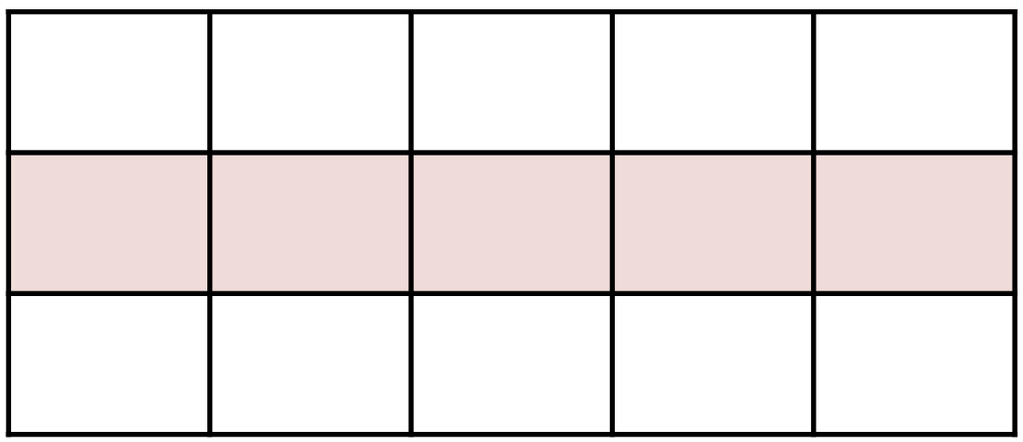
selection:σ

How do we want to operate on relations? The binary case

R(a:int, b:int, c:str, d:str, e:flt)



T(v:int, w:int, x:str, y:str, z:flt)



grow/shrink top-to-bottom

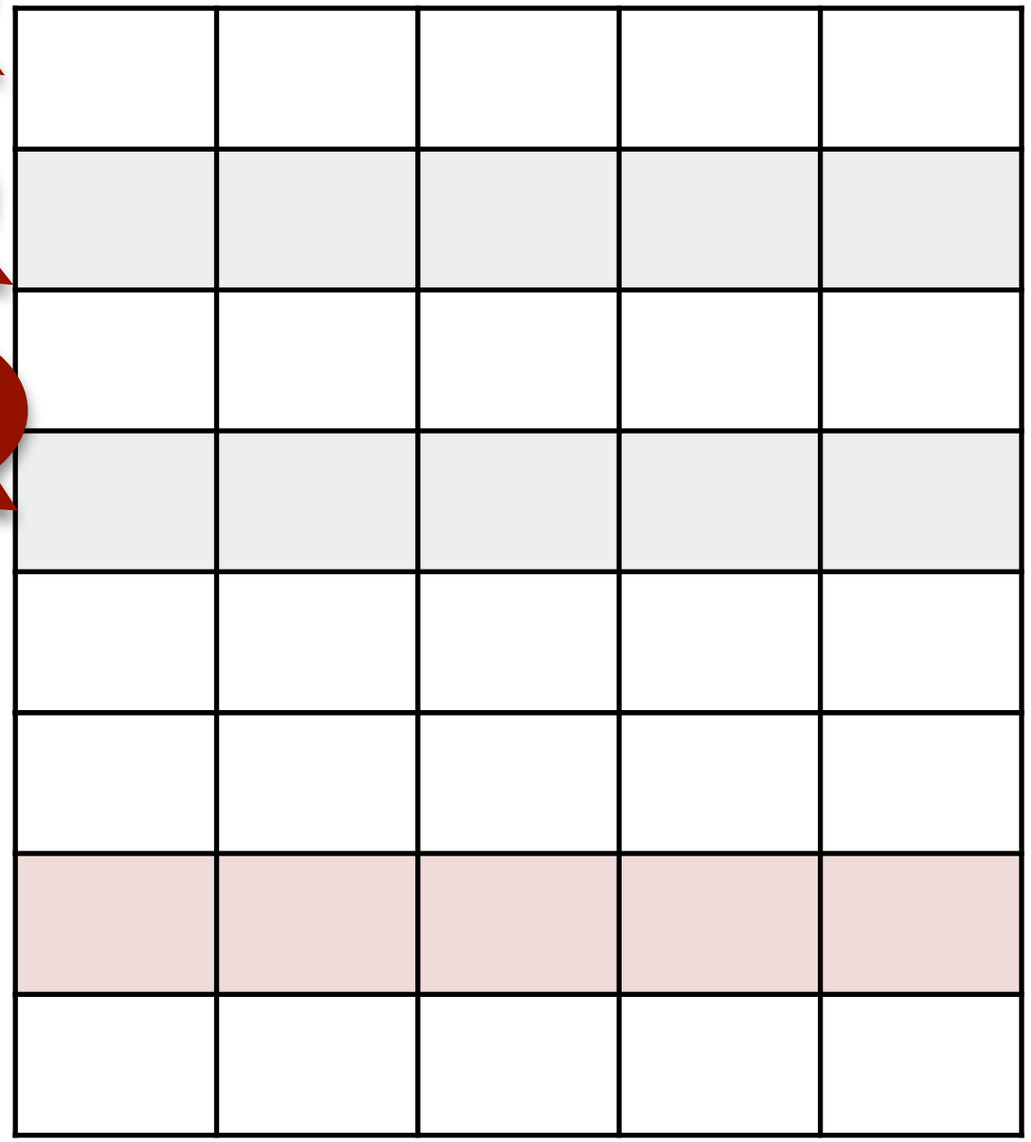
keep some rows and add some more

union: ∪

except: \

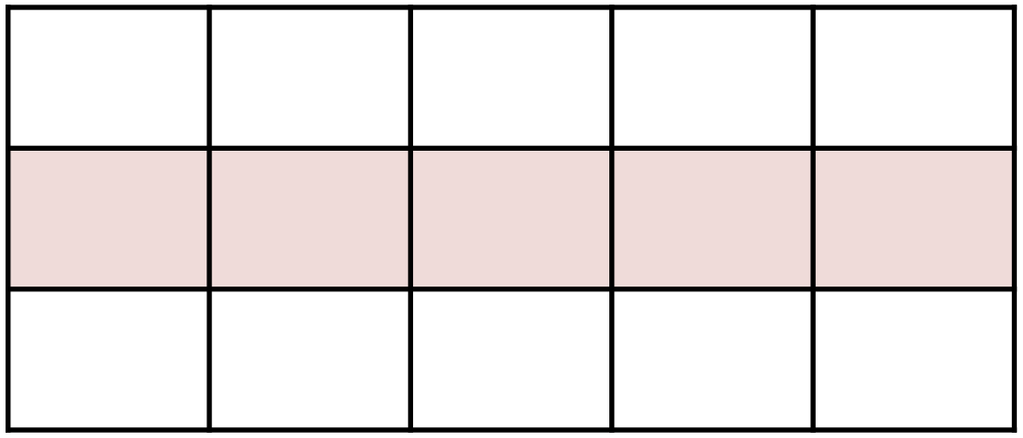
intersect: ∩

T'(v:int, w:int, x:str, y:str, z:flt)

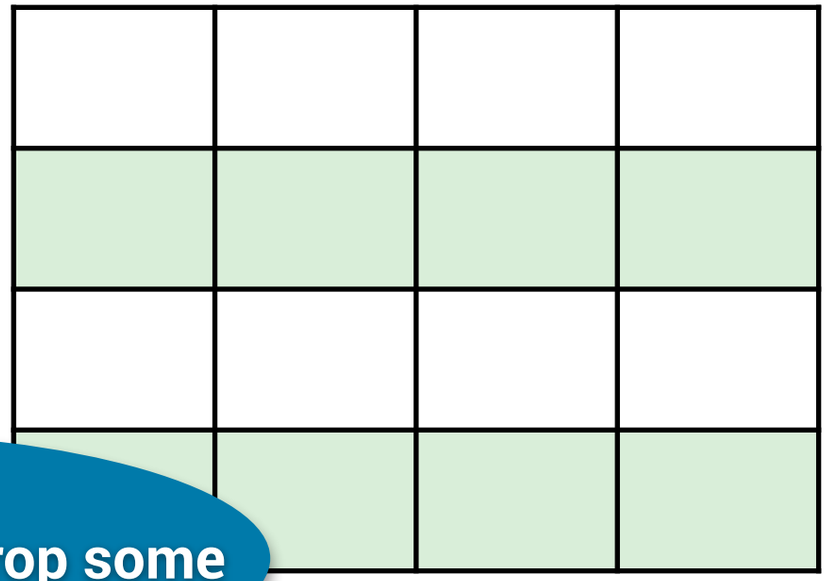


How do we want to operate on relations? The binary case

T(v:int, w:int, x:str, y:str, z:flt)



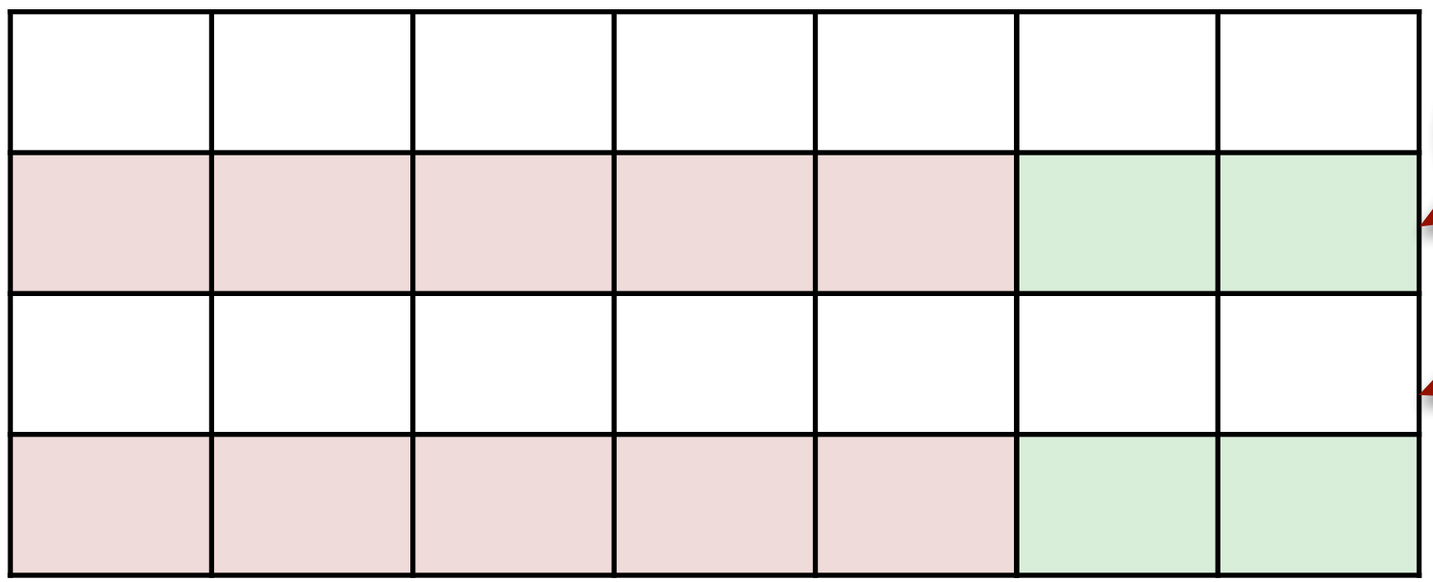
S(b:int, g:int, x:str, z:flt)



grow left-to-right

keep/add some columns and drop some columns

S'(v:int, w:int, x:str, y:str, z:flt, b:int, g:int)



product: ×

join: ⋈

Relational-Algebraic Operations: The (prefix) unary case

R(a:int, b:int, c:str, d:str, e:flt)

(<operator> **<parameters>** **(<operand>))**

π : projection
σ : selection
ρ : rename

π : attribute list
σ : predicate
ρ : attribute list

relation-denoting variable (or expression)

$\pi_{(a,e)}(R)$

$\sigma_{a>0}(R)$

$\rho_{a \to z}(R)$

$\rho_{R(z,b,c,d,e)}(R)$

Relational-Algebraic Operations: The (infix) binary case

T(v:int, w:int, x:str, y:str, z:flt)

S(b:int, g:int, x:str, z:flt)

((left-operand) <operator> <parameters> (right-operand))

relation-denoting variable (or expression)

∪ : union
\ : except
∩ : intersect
× : product
⋈: join

∪ : <none>
\ : <none>
∩ : <none>
× : <none>
⋈: predicate

relation-denoting variable (or expression)

$(T \cup S)$

$(T \setminus S)$

$(T \cap S)$

$(T \times S)$

$(T \bowtie_{T.z = S.z} S)$

Selection: Notation and Example

Sells		
bar	beer	price
Joe's	Bud	2.50
Joe's	Miller	2.75
Sue's	Bud	2.50
Sue's	Miller	3.00

$$JoeMenu := \sigma_{bar='Joe''s'}(Sells)$$

JoeMenu		
bar	beer	price
Joe's	Bud	2.50
Joe's	Miller	2.75

Projection: Notation and Example

drop!

keep!

keep!

Sells

bar	beer	price
Joe's	Bud	2.50
Joe's	Miller	2.75
Sue's	Bud	2.50
Sue's	Miller	3.00

Why only three tuples?!

$$BeerPrices := \pi_{beer,price}(Sells)$$

BeerPrices

beer	price
Bud	2.50
Miller	2.75
Miller	3.00

(Cartesian) Product: Notation and Example

R_1	
A	B
1	2
3	4

$R_3 := R_1 \times R_2$

R_2	
C	D
5	6
7	8
9	10

R_3			
A	B	C	D
1	2	5	6
1	2	7	8
1	2	9	10
3	4	5	6
3	4	7	8
3	4	9	10

T'1' R1 + T'5' in R2!

T'1' R1 + T'7' in R2!

T'1' R1 + T'9' in R2!

T'3' R1 + T'5' in R2!

⋮

Renaming: Notation and Example

before!

Bars	
name	addr
Joe's	Maple St.
Sue's	River Rd.

$Barz := \rho_{Bars(bar,addr)}\ (Bars)$

after!

Barz	
bar	addr
Joe's	Maple St.
Sue's	River Rd.

θ-Join: Notation and Example

one from Sells!

Sells		
bar	beer	price
Joe's	Bud	2.50
Joe's	Miller	2.75
Sue's	Bud	2.50
Sue's	Coors	3.00

Bars	
name	addr
Joe's	Maple St.
Sue's	River Rd.

one from Bars!

keep those concats for which θ is true!

$$\text{BarInfo} := \text{Sells} \bowtie_{\text{Sells.bar=Bars.name}} \text{Bars}$$

Is this table a bit odd?!

BarInfo				
bar	beer	price	name	addr
Joe's	Bud	2.50	Joe's	Maple St.
Joe's	Miller	2.75	Joe's	Maple St.
Sue's	Bud	2.50	Sue's	River Rd.
Sue's	Coors	3.00	Sue's	River Rd.

(Natural) Join: Notation and Example

X

Same name?

Sells		
bar	beer	price
Joe's	Bud	2.50
Joe's	Miller	2.75
Sue's	Bud	2.50
Sue's	Coors	3.00

Barz	
bar	addr
Joe's	Maple St.
Sue's	River Rd.

BarInfo := Sells ⋈ Barz

Better!

Keep only one of the columns!

BarInfo			
bar	beer	price	addr
Joe's	Bud	2.50	Maple St.
Joe's	Miller	2.75	Maple St.
Sue's	Bud	2.50	River Rd.
Sue's	Coors	3.00	River Rd.

Extended Projection: Notation and Example

X

R	
A	B
1	2
3	4
5	6
7	8

$$S := \pi_{A+B \rightarrow C, A, A \rightarrow A2}(R)$$

simple arithmetic

replication

A+B!

A

A again!

S		
C	A	A2
3	1	1
7	3	3
11	5	5
15	7	7

Complex Expression: Alternative Forms

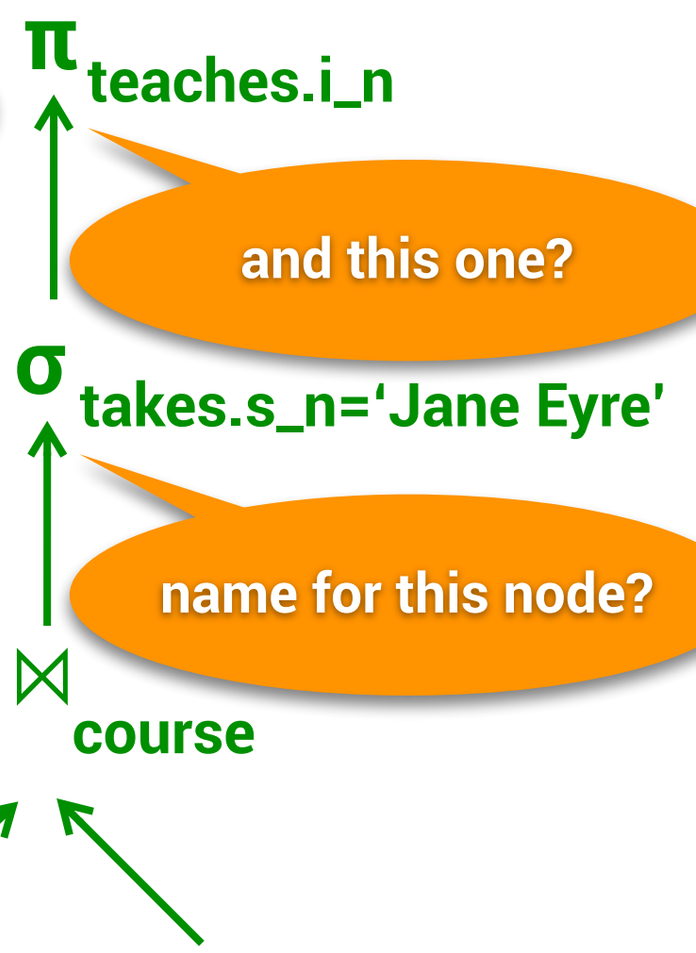
expression

linear, textual form

$\pi_{teaches.i\_n}($
$\sigma_{takes.s\_n='Jane\ Eyre'}($
teaches $\bowtie_{course}$ takes))

operator tree

dataflow form



named intermediate results

assignment sequence

Tmp1 := teaches $\bowtie_{course}$ Takes
Tmp2 := $\sigma_{takes.s\_n='Jane\ Eyre'}$(Tmp1)
JaneEyreInstructors := $\pi_{teaches.i\_n}$(Tmp2)

Direct Translation of Core SQL to RA

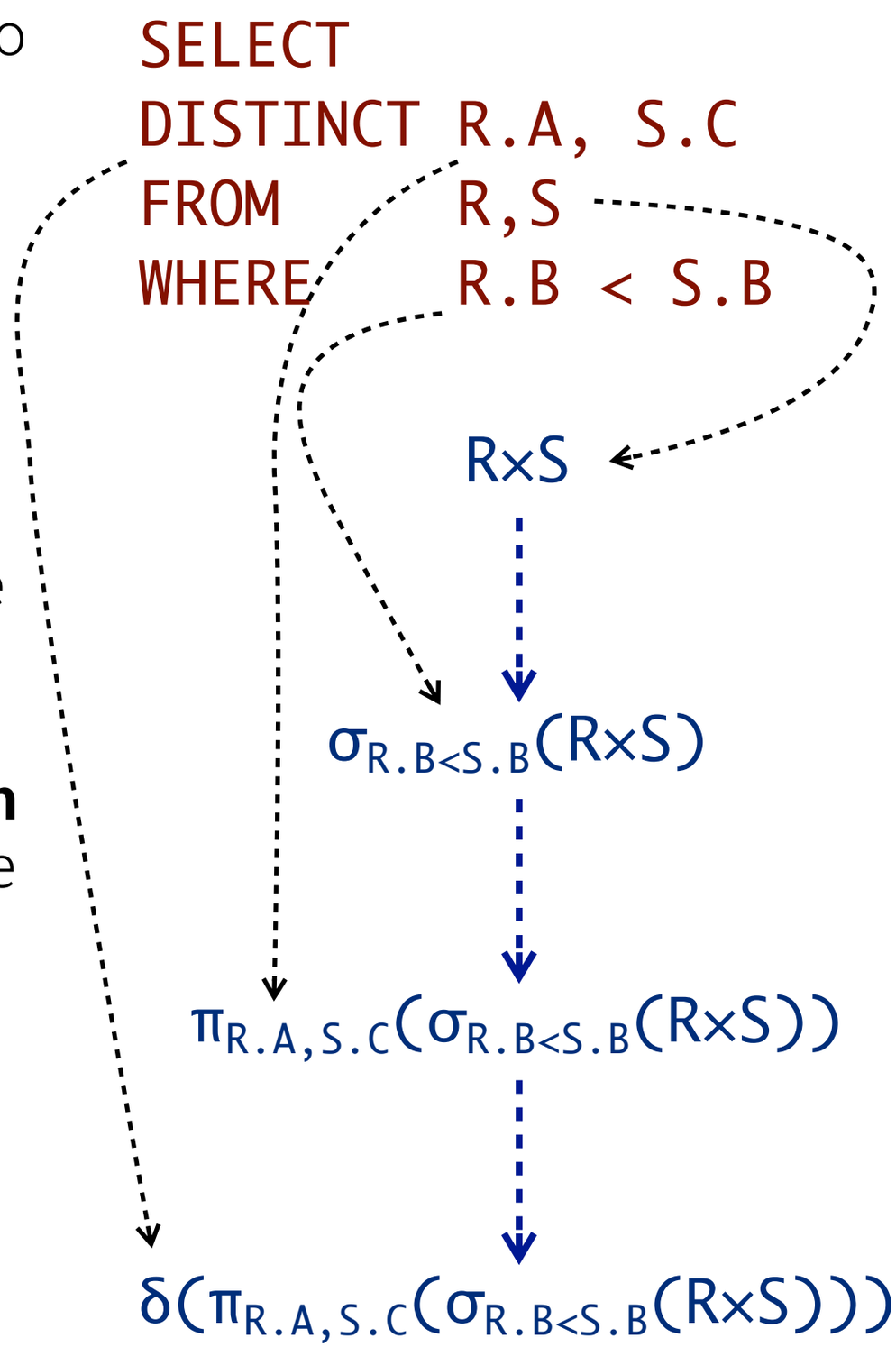
- The basic SQL query blocks map to RA as follows:

  1. Form the (cascaded binary) **product** of the relations in the `FROM` clause

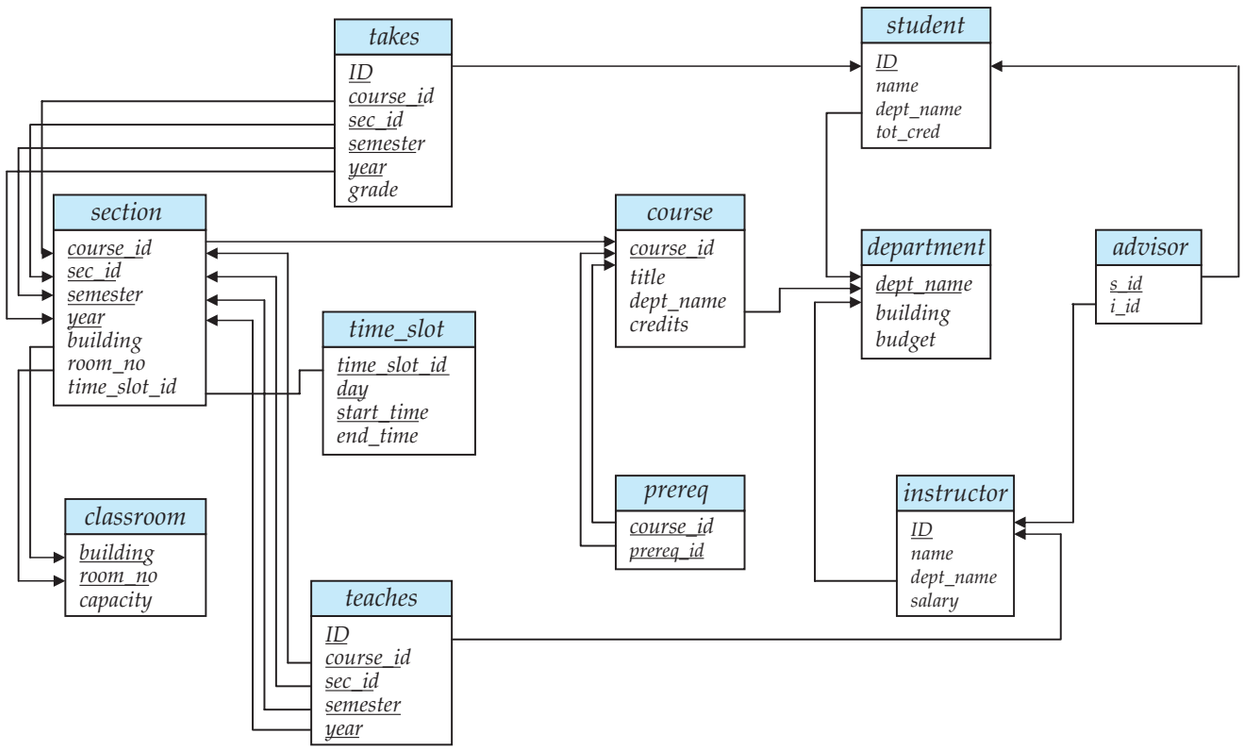
  2. To this result, apply a **selection** operator whose predicate is the one in the `WHERE` clause

  3. To this result, apply a **projection** whose attribute list is that in the `SELECT` clause

  4. If `DISTINCT` is requested, to this result, apply a **duplicate removal** operator



SQL: The SELECT Clause




```
SELECT name
FROM   instructor;
```

just this column!

```
SELECT DISTINCT name
FROM   instructor;
```

remove duplicates!

```
SELECT ALL name
FROM   instructor;
```

keep duplicates!

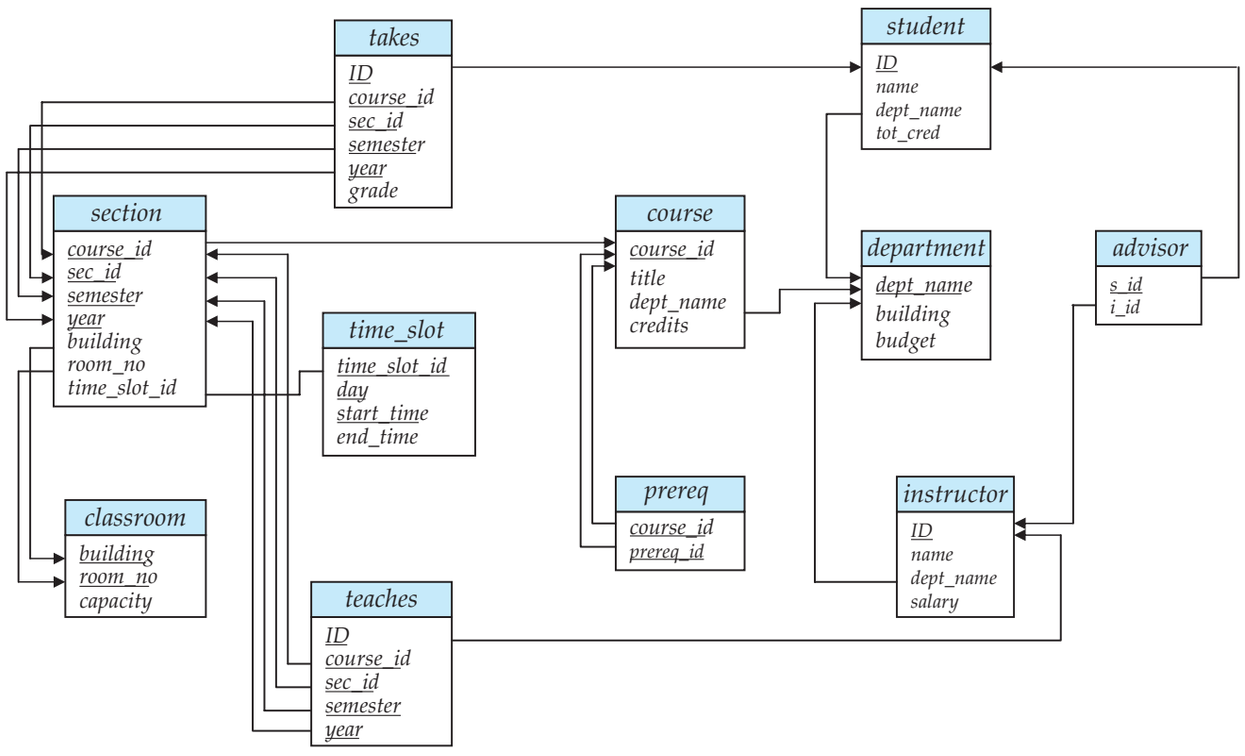
```
SELECT *
FROM   instructor;
```

all columns!

```
SELECT ID, name, salary/12
FROM   instructor;
```

derive this column!

SQL: The WHERE and FROM Clauses



complex predicates!

```
SELECT name
FROM   instructor
WHERE  dept_name = 'Comp. Sci.'
AND    salary > 80000;
```

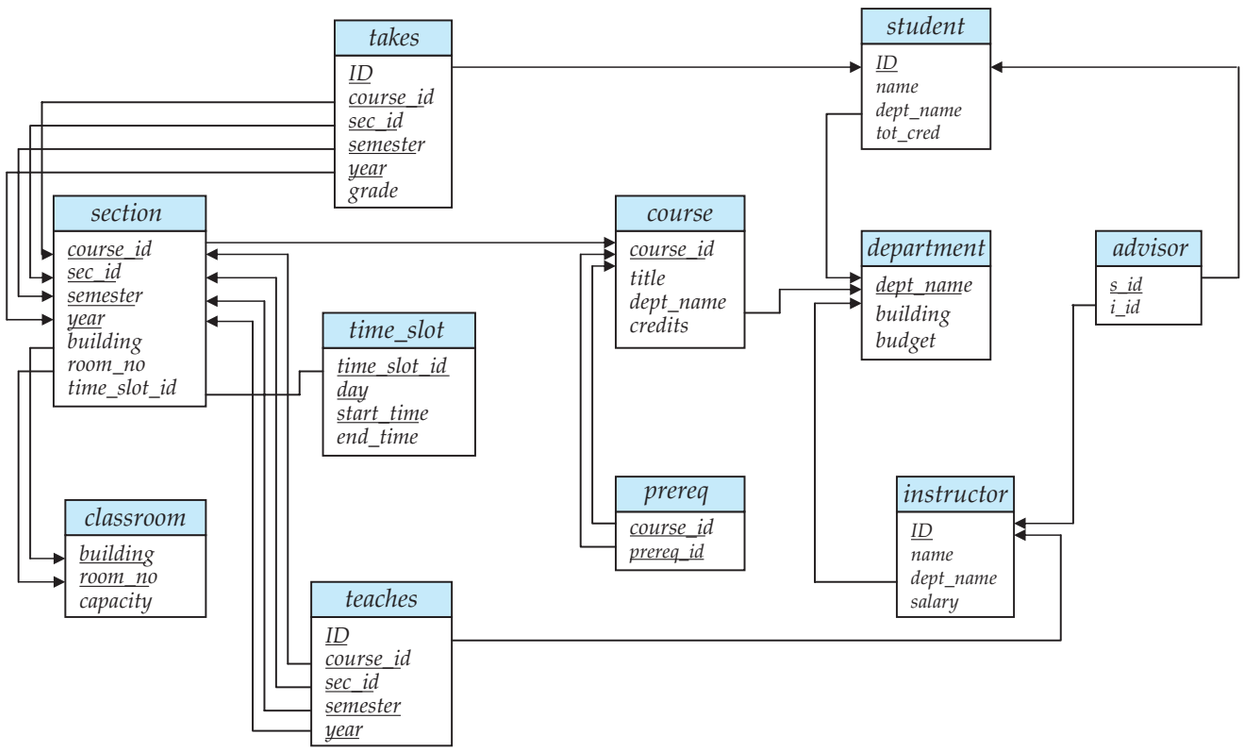
more than one? product!

```
SELECT *
FROM   instructor, teaches;
```

unless there is a predicate: join!

```
SELECT name
FROM   instructor, department
WHERE  instructor.dept_name =
       department.dept_name
AND    salary > budget;
```

SQL: Set Operations

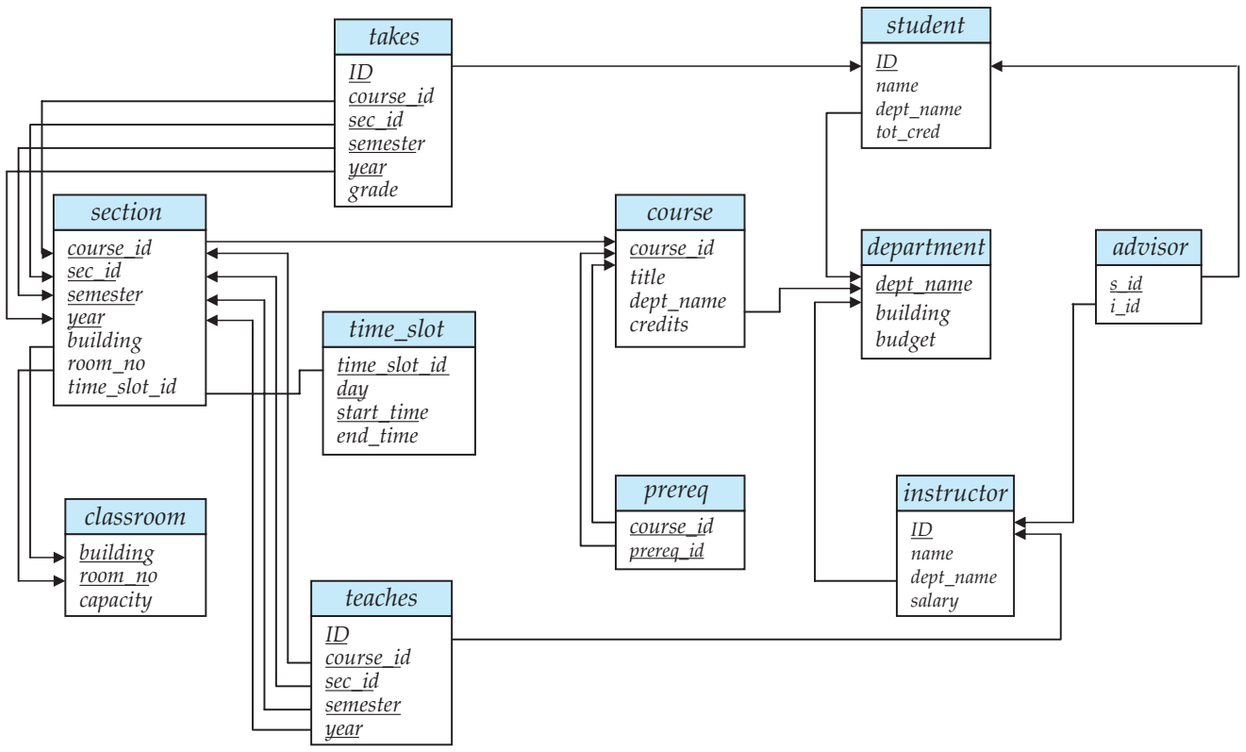



```
(SELECT course_id
 FROM   section
 WHERE  sem = 'Fall' and year = 2009)
UNION  ALL
(SELECT course_id
 FROM   section
 WHERE  sem = 'Spring' and year = 2010);

(SELECT course_id
 FROM   section
 WHERE  sem = 'Fall' and year = 2009)
EXCEPT ALL
(SELECT course_id
 FROM   section
 WHERE  sem = 'Spring' and year = 2010);

(SELECT course_id
 FROM   section
 WHERE  sem = 'Fall' and year = 2009)
INTERSECT ALL
(SELECT course_id
 FROM   section
 WHERE  sem = 'Spring' and year = 2010);
```

SQL: Joins



commonest form of join!

```
SELECT instructor.name, teaches.course_id
FROM   instructor, teaches
WHERE  instructor.ID = teaches.ID;
```

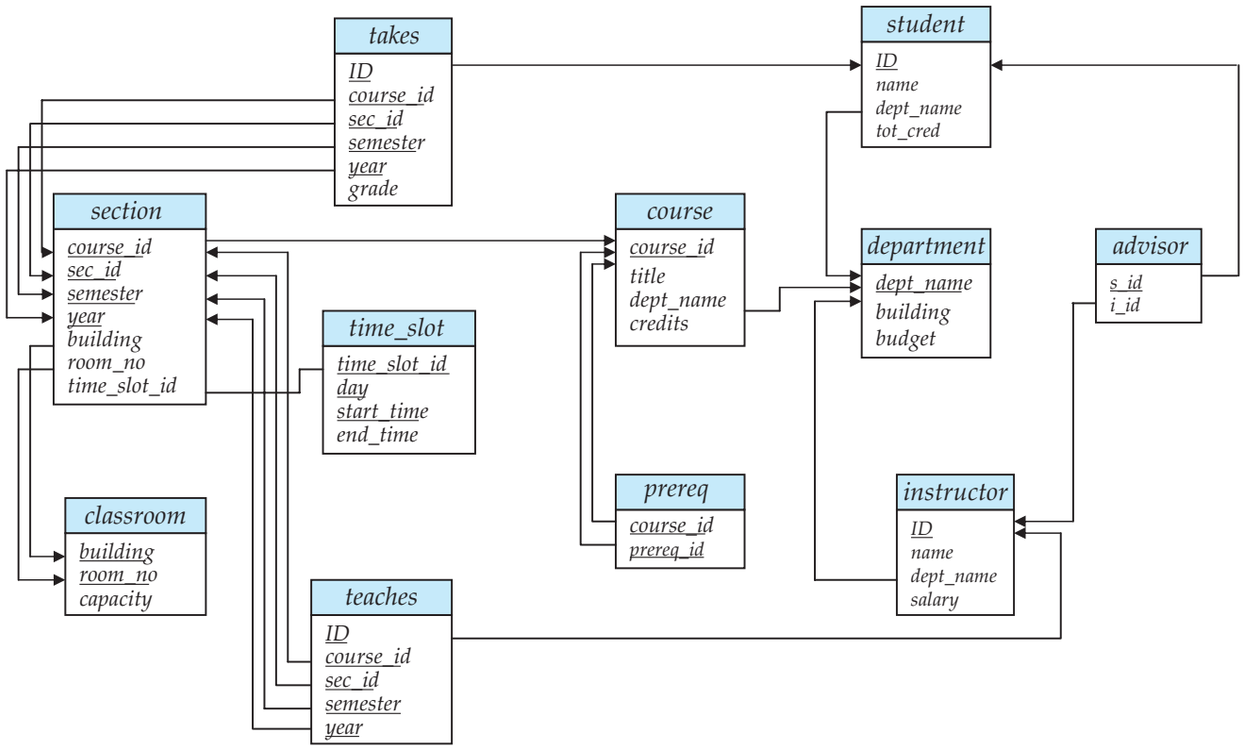
alternatively!

```
SELECT name, course_id
FROM   instructor JOIN teaches
USING  (ID);
```

if possible!

```
SELECT name, course_id
FROM   instructor NATURAL JOIN teaches;
```

SQL: Renaming



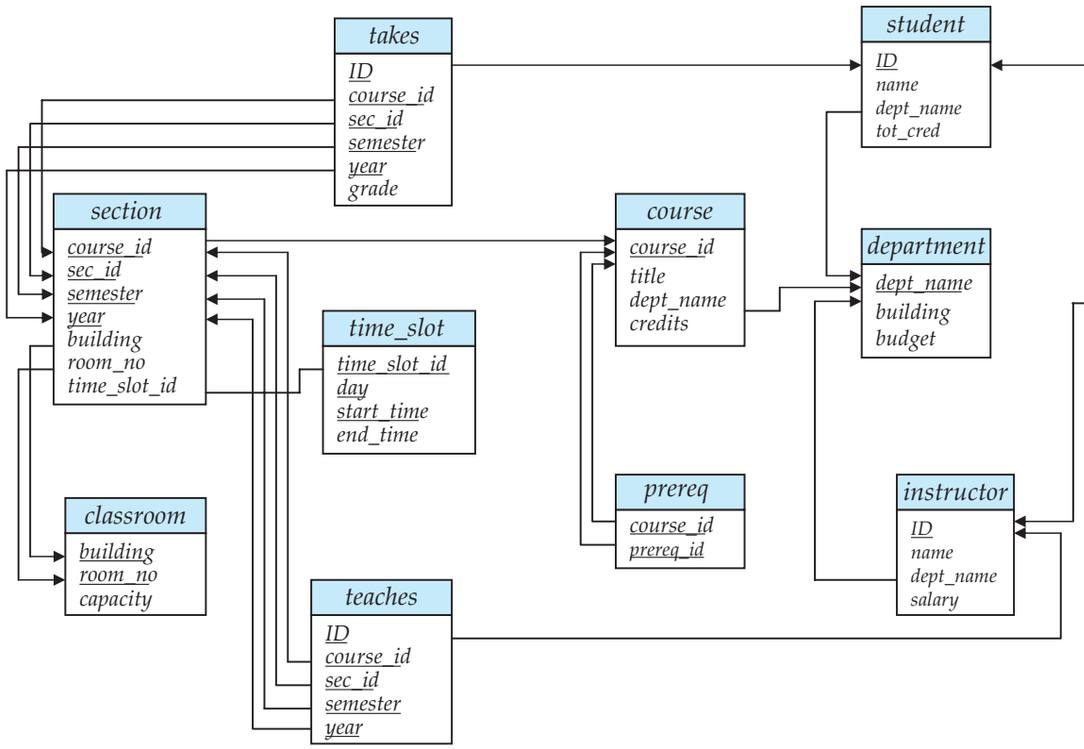

```
SELECT ID, name, salary/12 AS monthly_salary
FROM   instructor;
```

optional but recommended!

```
SELECT DISTINCT t.name
FROM   instructor AS t, instructor AS s
WHERE  t.salary > s.salary
AND    s.dept_name = 'Comp. Sci.';
```

must omit in Oracle!

SQL: More Predicates



string pattern matching

```
SELECT name
FROM   instructor
WHERE  name LIKE '%DAR%';
```

interval comparison

```
SELECT name
FROM   instructor
WHERE  salary BETWEEN 80000 AND 90000;
```

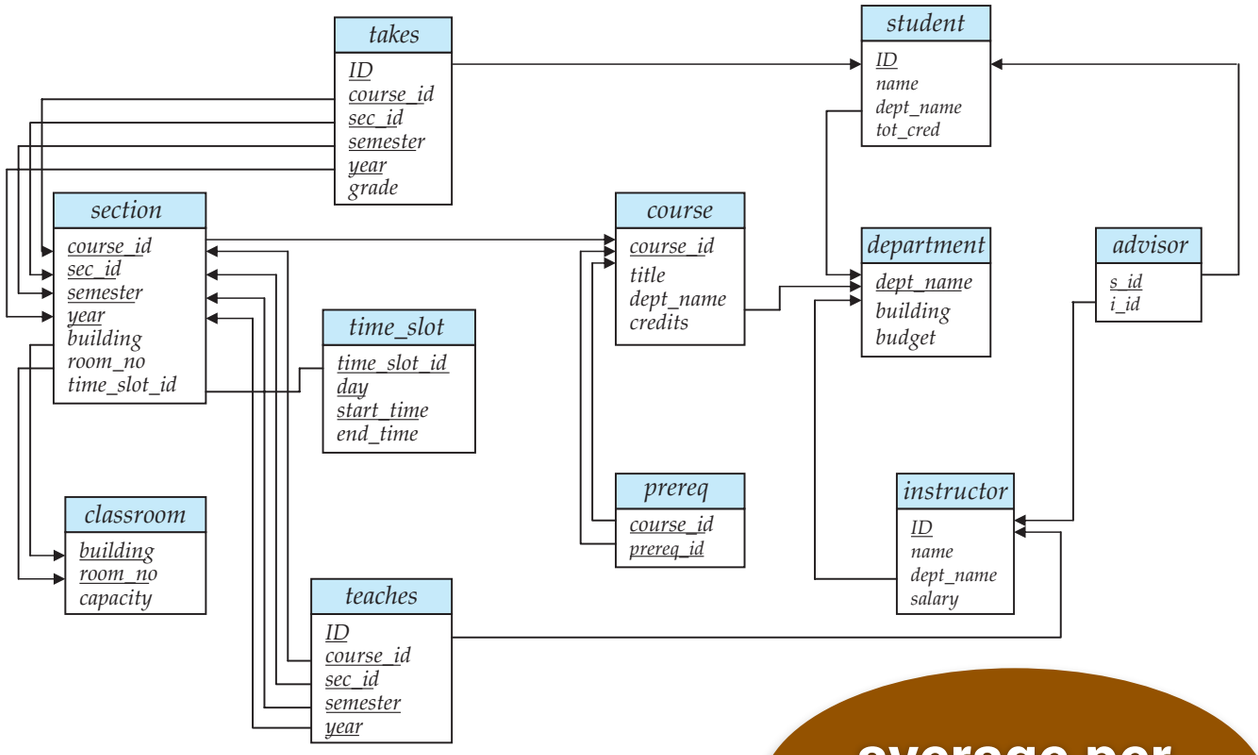
tuple

```
SELECT name, course_id
FROM   instructor i, teaches t
WHERE  (i.ID, t.dept_name) = (t.ID, 'Biology');
```

testing for NULL

```
SELECT name
FROM   instructor
WHERE  salary IS NULL;
```

SQL: Ordering and Aggregation



simplest form!

```
SELECT AVG(salary)
FROM   instructor;
```

remove duplicates!

```
SELECT COUNT (DISTINCT name)
FROM   instructor;
```

count all!

```
SELECT COUNT(*)
FROM   instructor;
```

average per department

```
SELECT   dept_name, AVG(salary)
FROM     instructor
GROUP BY dept_name;
```

but not all!

```
SELECT   dept_name, AVG(salary)
FROM     instructor
GROUP BY dept_name
HAVING   AVG(salary) > 42000;
```

In The Next Lecture

X

- We'll begin our exploration of conceptual modelling.
- We'll see that conceptual modelling is aimed at providing an intuitive approach to identifying information needs.
- We'll observe that a good conceptual model facilitates the generation of well-formed logical models.
- In particular, we'll learn about the formalism for conceptual modelling known as Entity-Relationship (ER) Modelling.

The University of Manchester

L6

Designing Databases: Mapping a Conceptual into a Logical Design

Fundamentals of Databases

Alvaro A A Fernandes, SCS, UoM

Readings

This lecture relies on your having read the assigned mandatory readings associated with this lecture:

Ramez Elmasri and Shamkant Navathe. Fundamentals of Database Systems. New International Edition. Pearson, 2013. pp. 287-308.

You can read the material online using this link:

lib.myilibrary.com/Open.aspx?id=527132

This is a shared resource. Don't hog it. When you're done, logout and close the browser to free it for others.

If you fail to study the material, you're likely to find the lecture harder to follow.

More information is available in the Learning Activities Manual.

In Previous Lectures

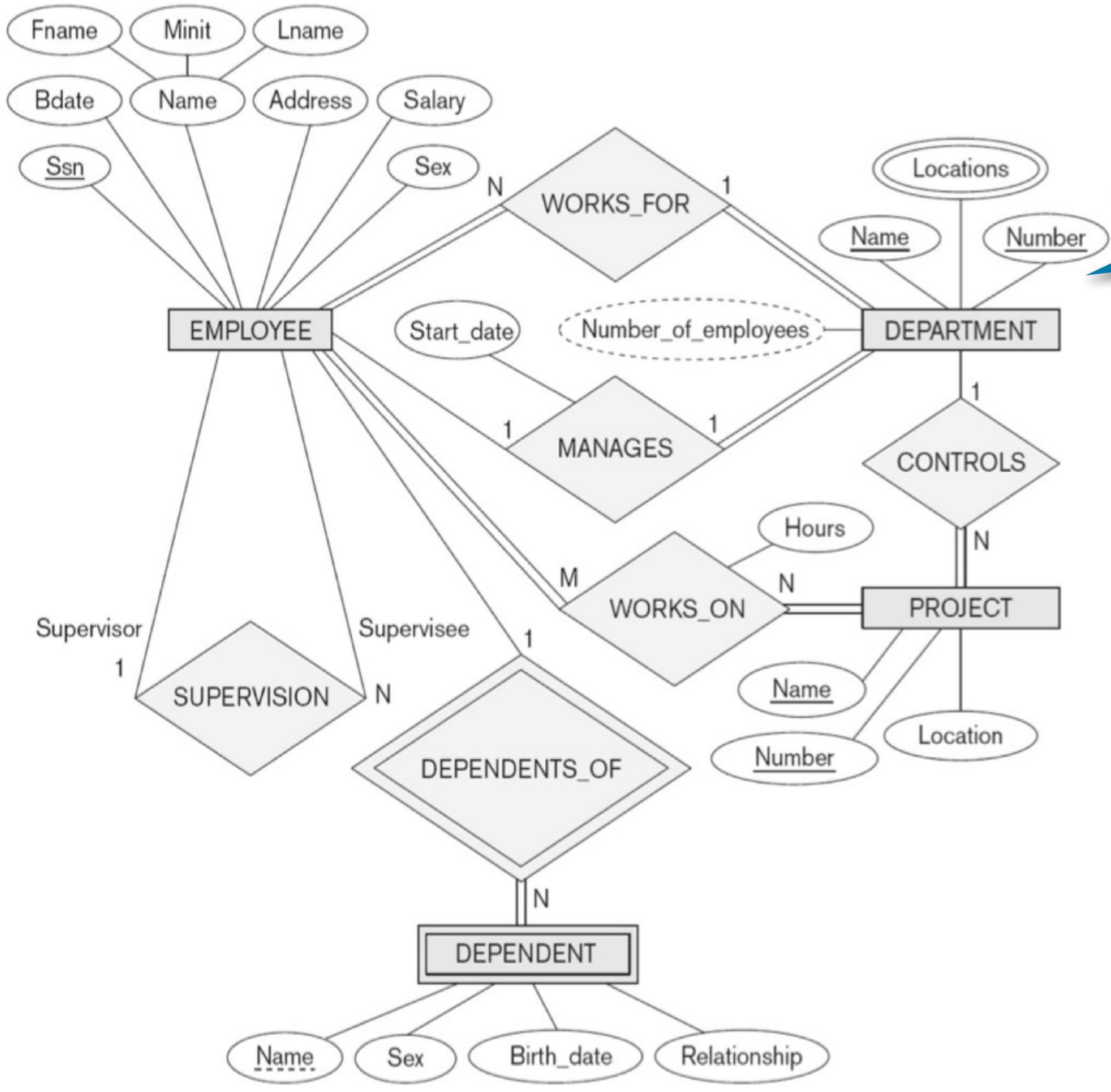
X

- We learned how data models lead to a distinction between schemas and instances that enables a logical view of the data.
- We learned about the relational approach to logical data modelling.
- We learned about the relational algebra and SQL, both its DDL and DML capabilities and its querying constructs.
- We learned of the practical benefits of performing conceptual modelling before logical design and why the EER approach serves well this purpose.

In This Lecture

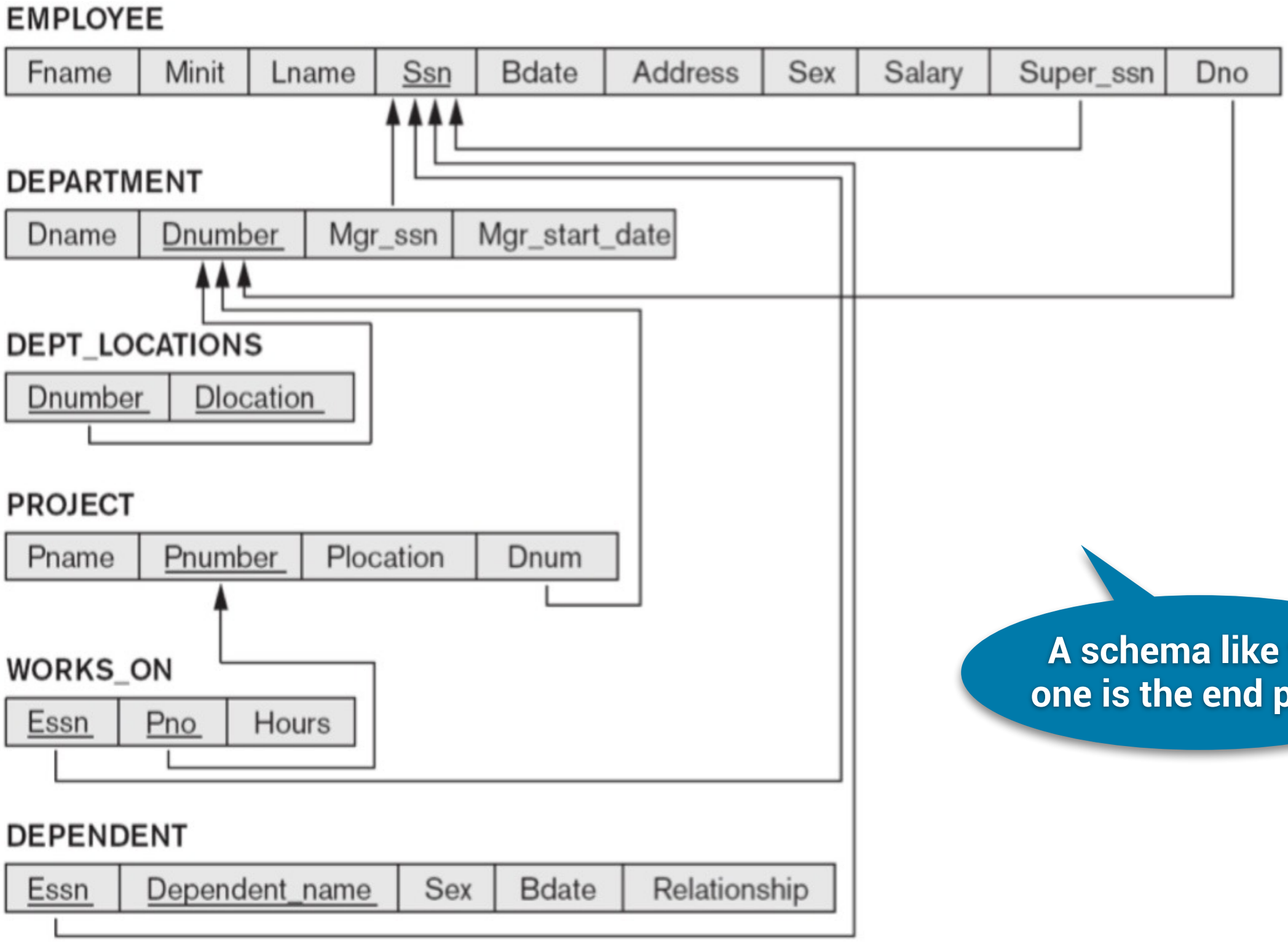
- We'll learn how a conceptual model can be mapped into a logical model that is capable of being implemented in a relational DBMS.

An Example ER Diagram for a COMPANY DB



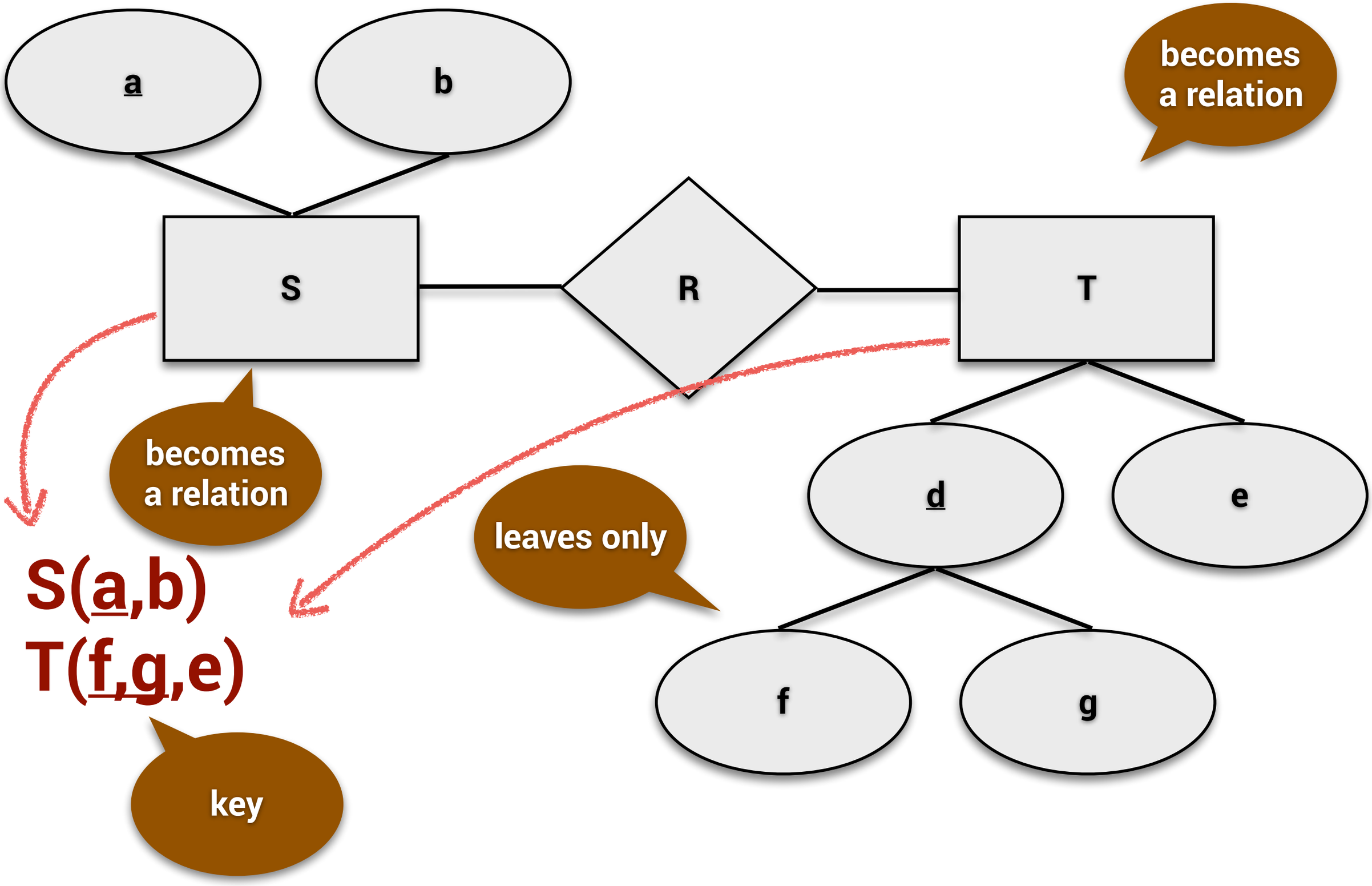
A model like this one is the starting point.

An Example Relational Schema for a COMPANY DB

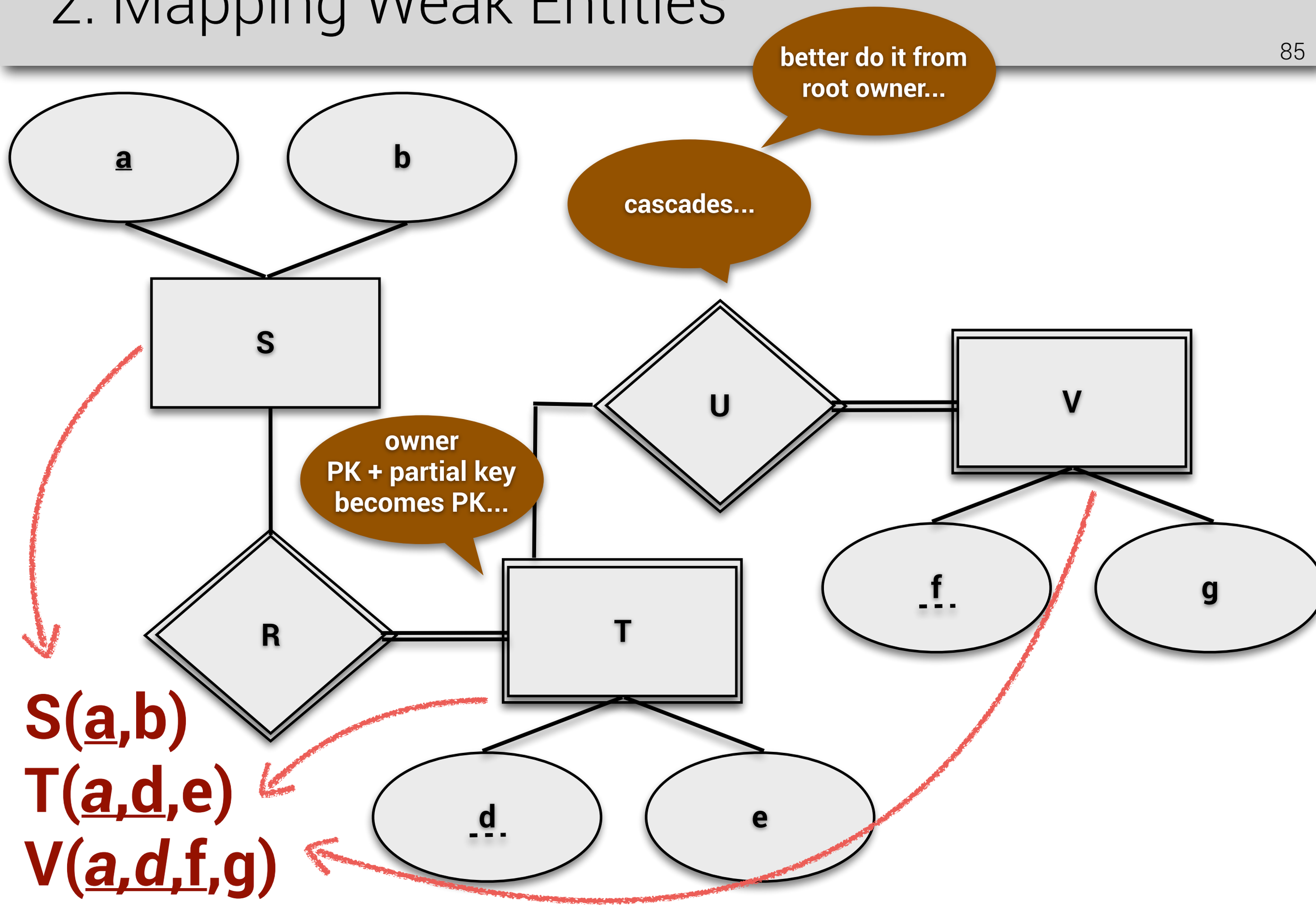


A schema like this one is the end point.

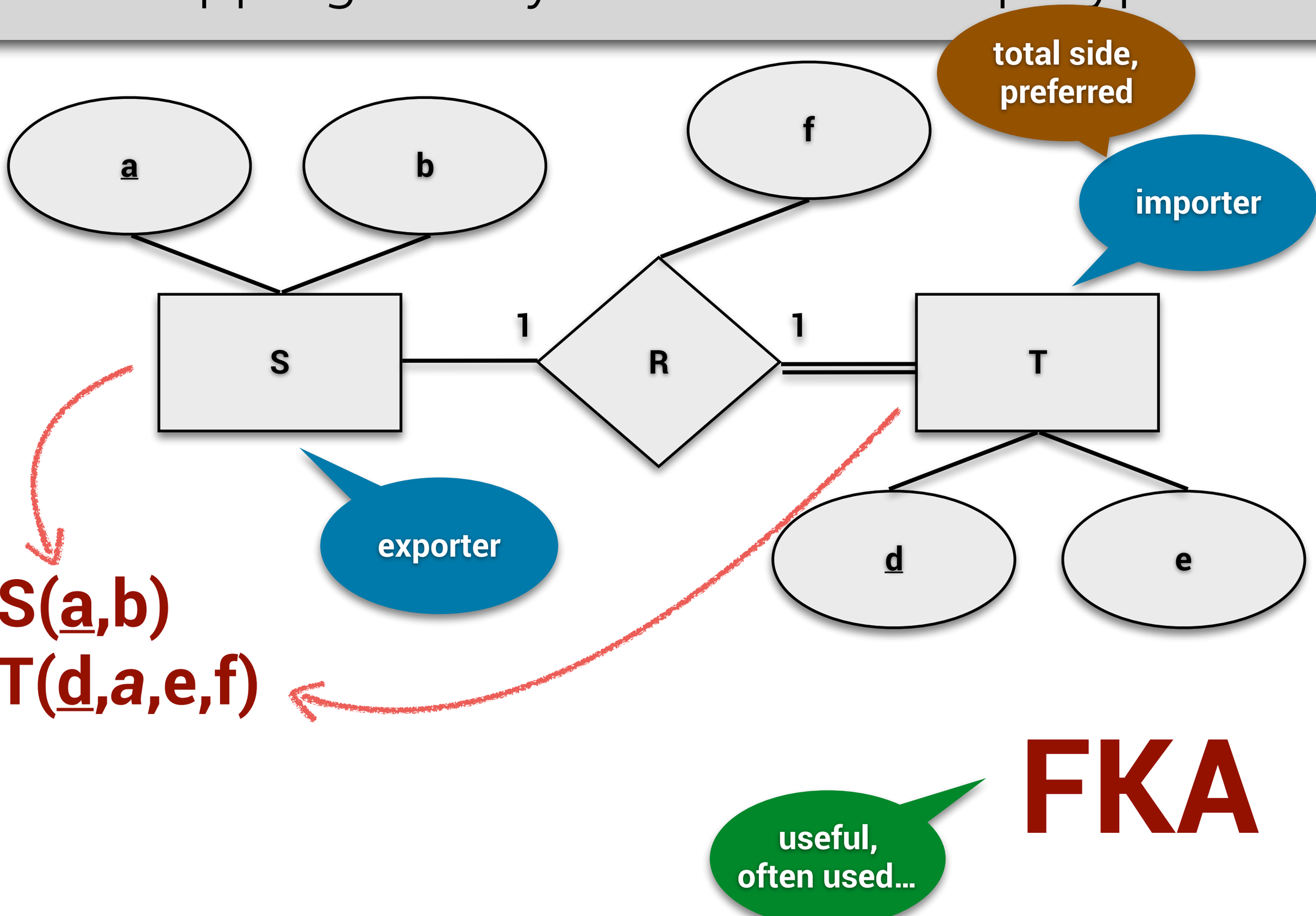
1: Mapping (Normal) Entities



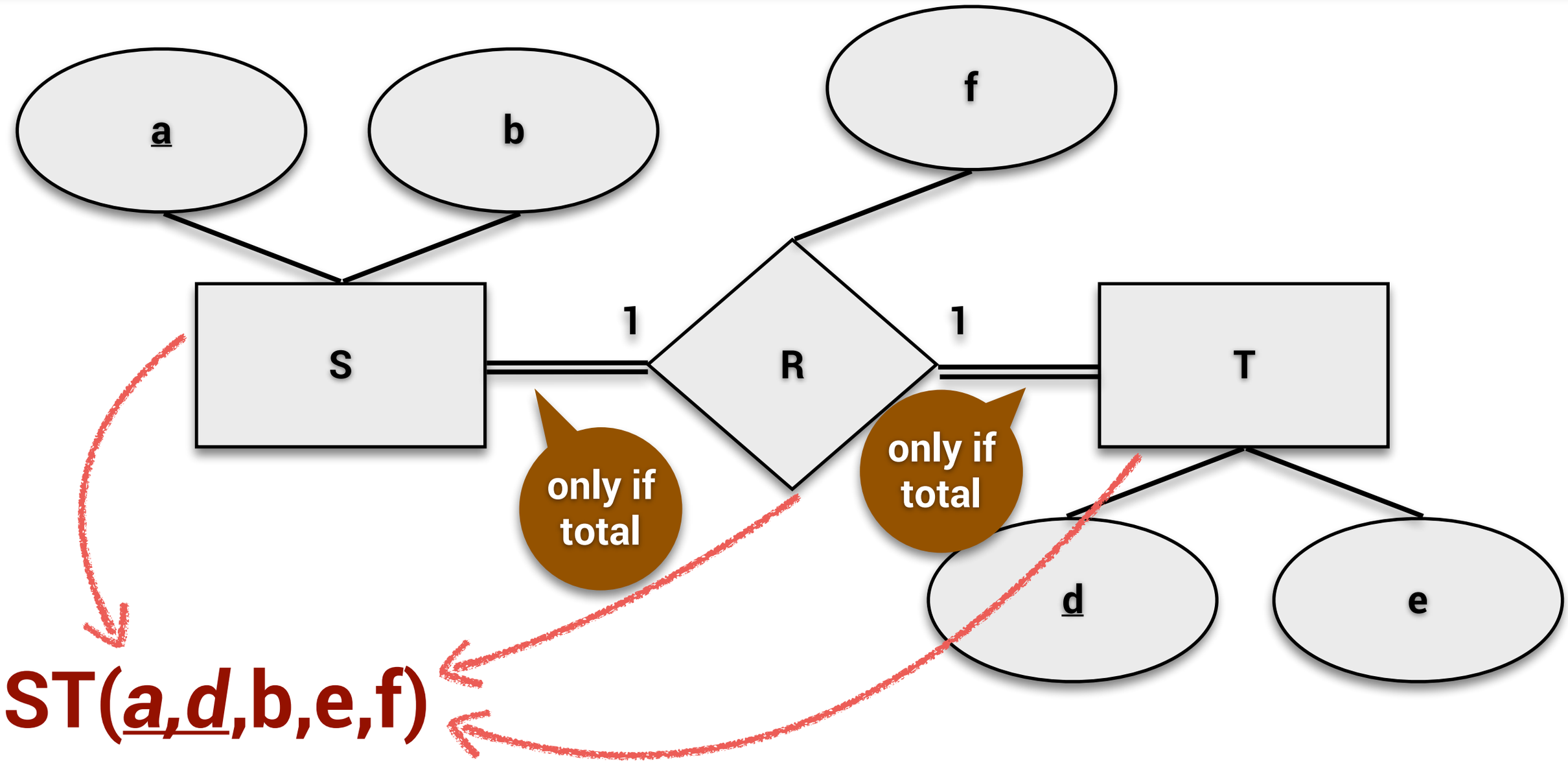
2: Mapping Weak Entities



3.i: Mapping Binary 1:1 Relationship Types



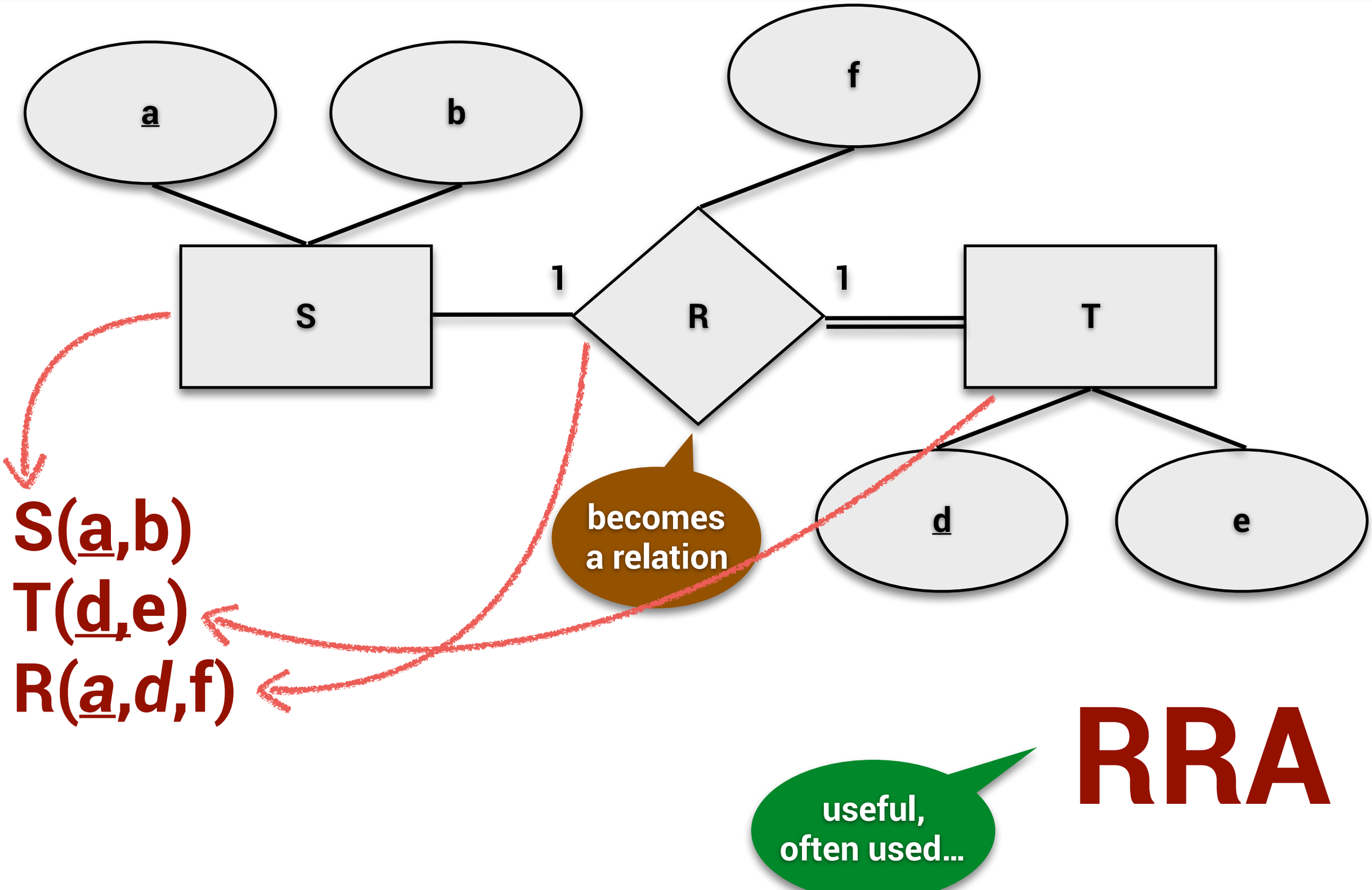
3.ii: Mapping Binary 1:1 Relationship Types



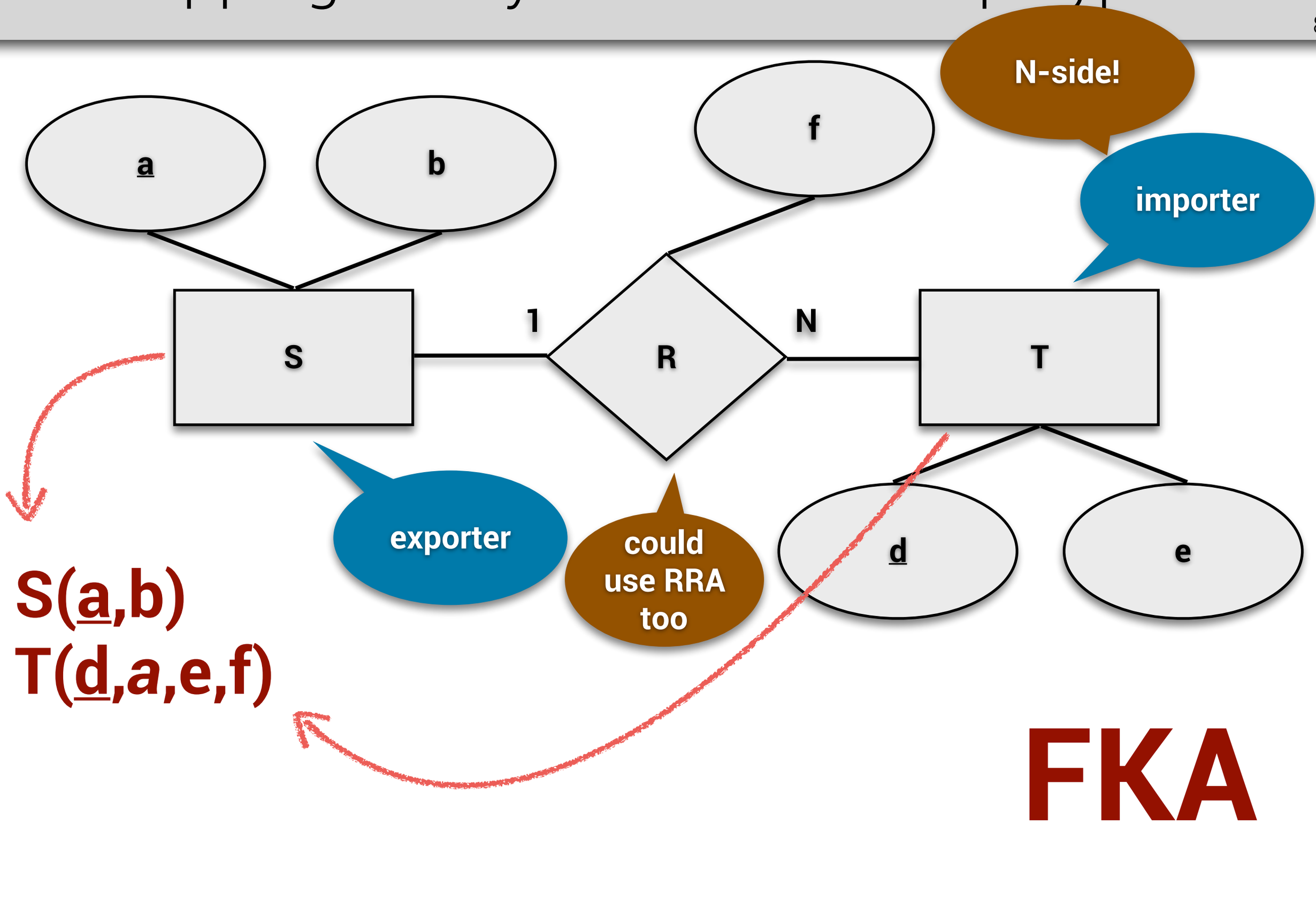
MRA

not often used...

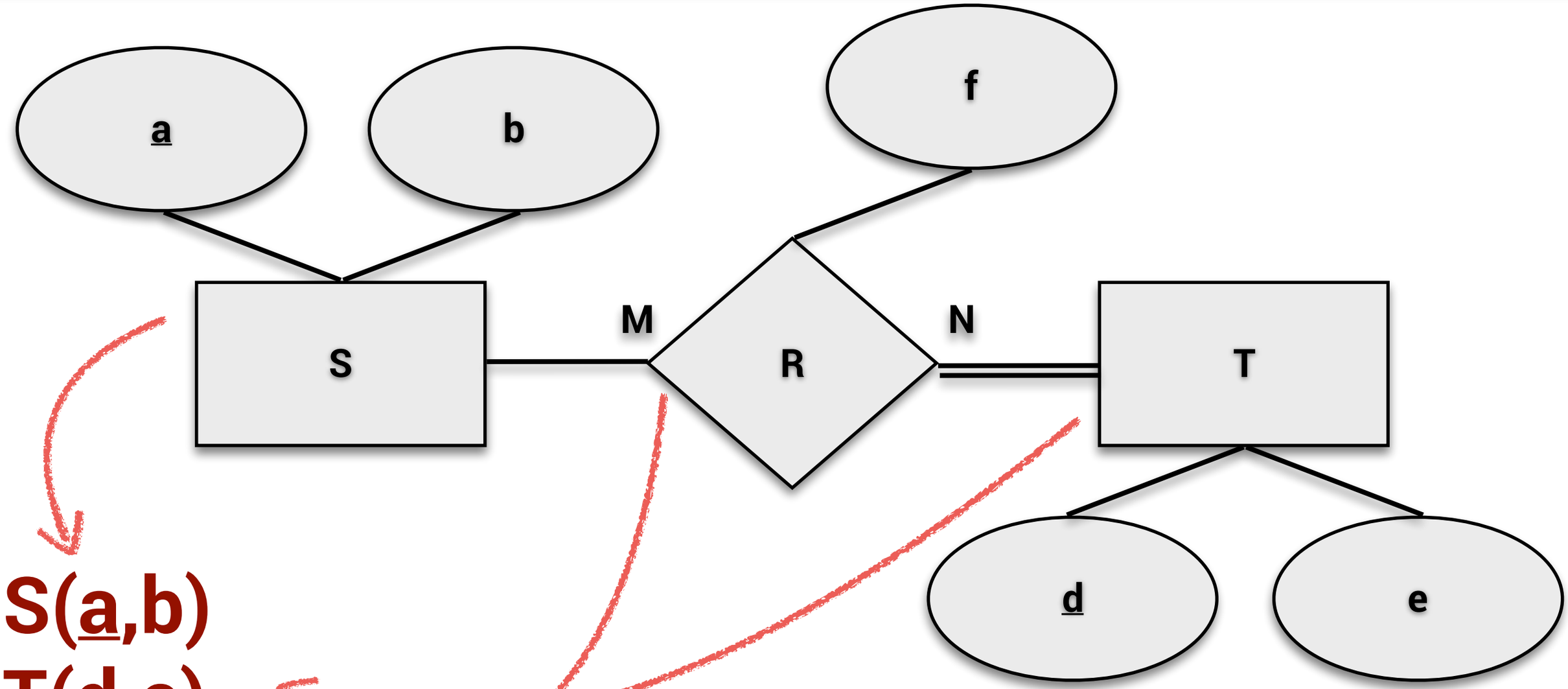
3.iii: Mapping Binary 1:1 Relationship Types



4: Mapping Binary 1:N Relationship Types



5: Mapping Binary N:M Relationship Types

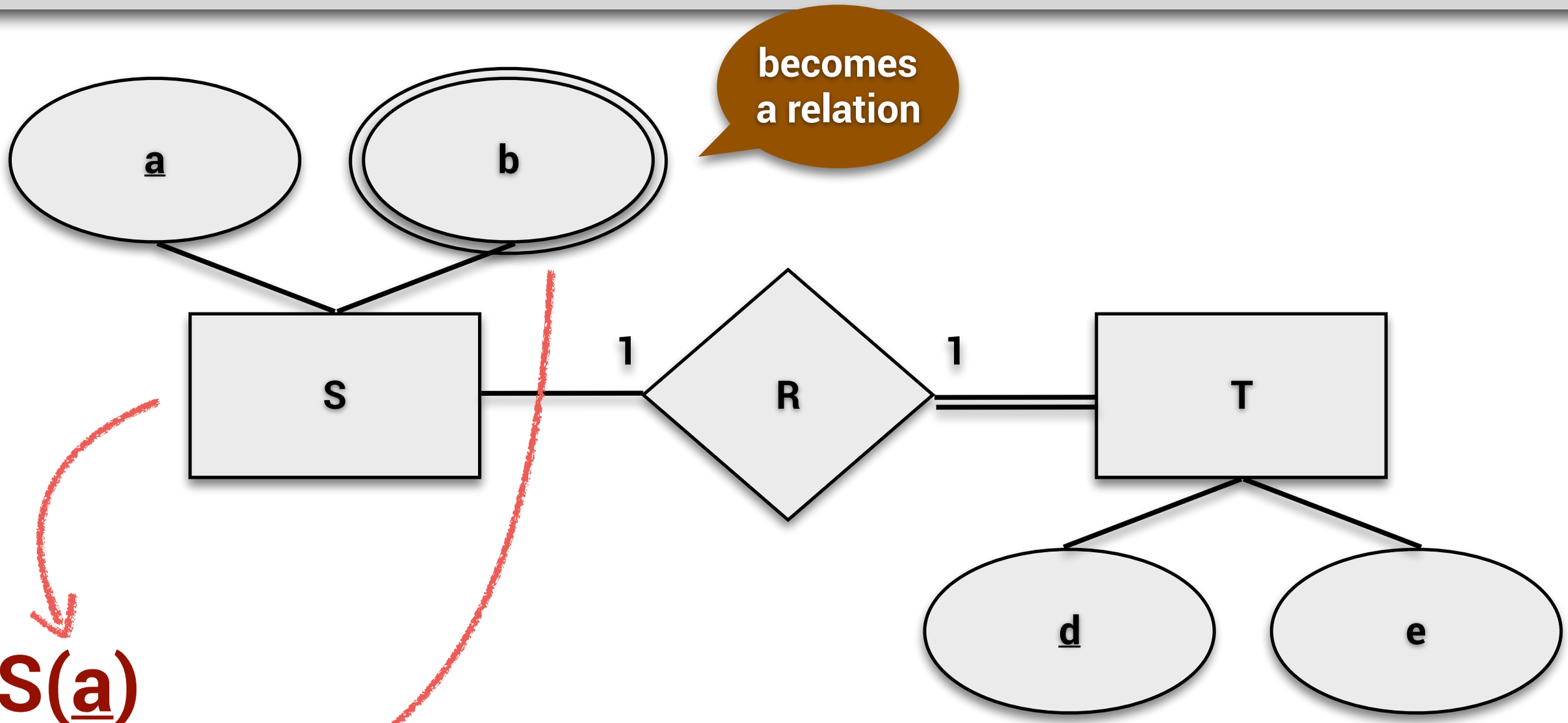


S(a,b)
T(d,e)
R(a,d,f)

notice! composite PK!

RRA

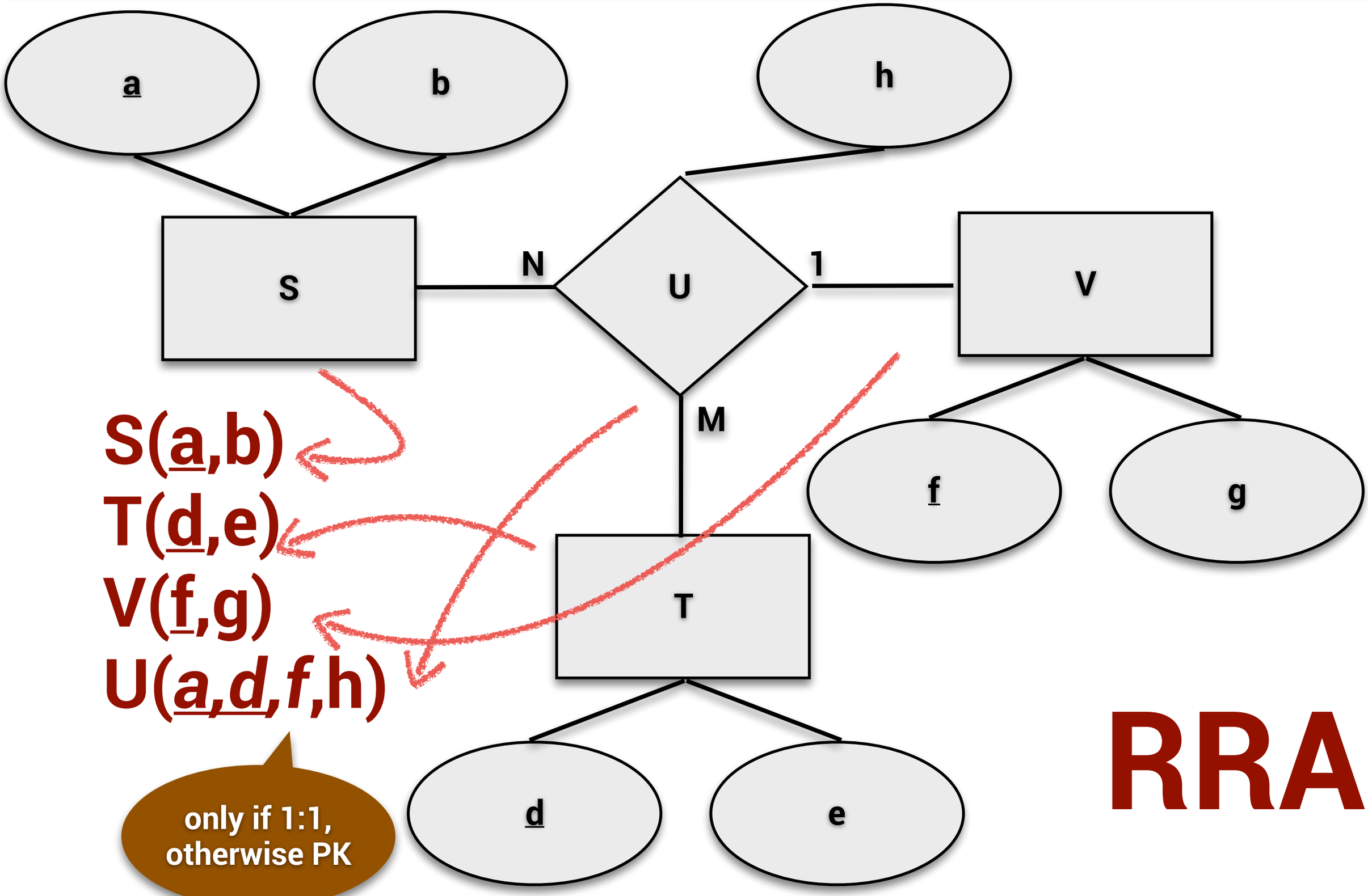
6: Mapping Multivalued Attributes



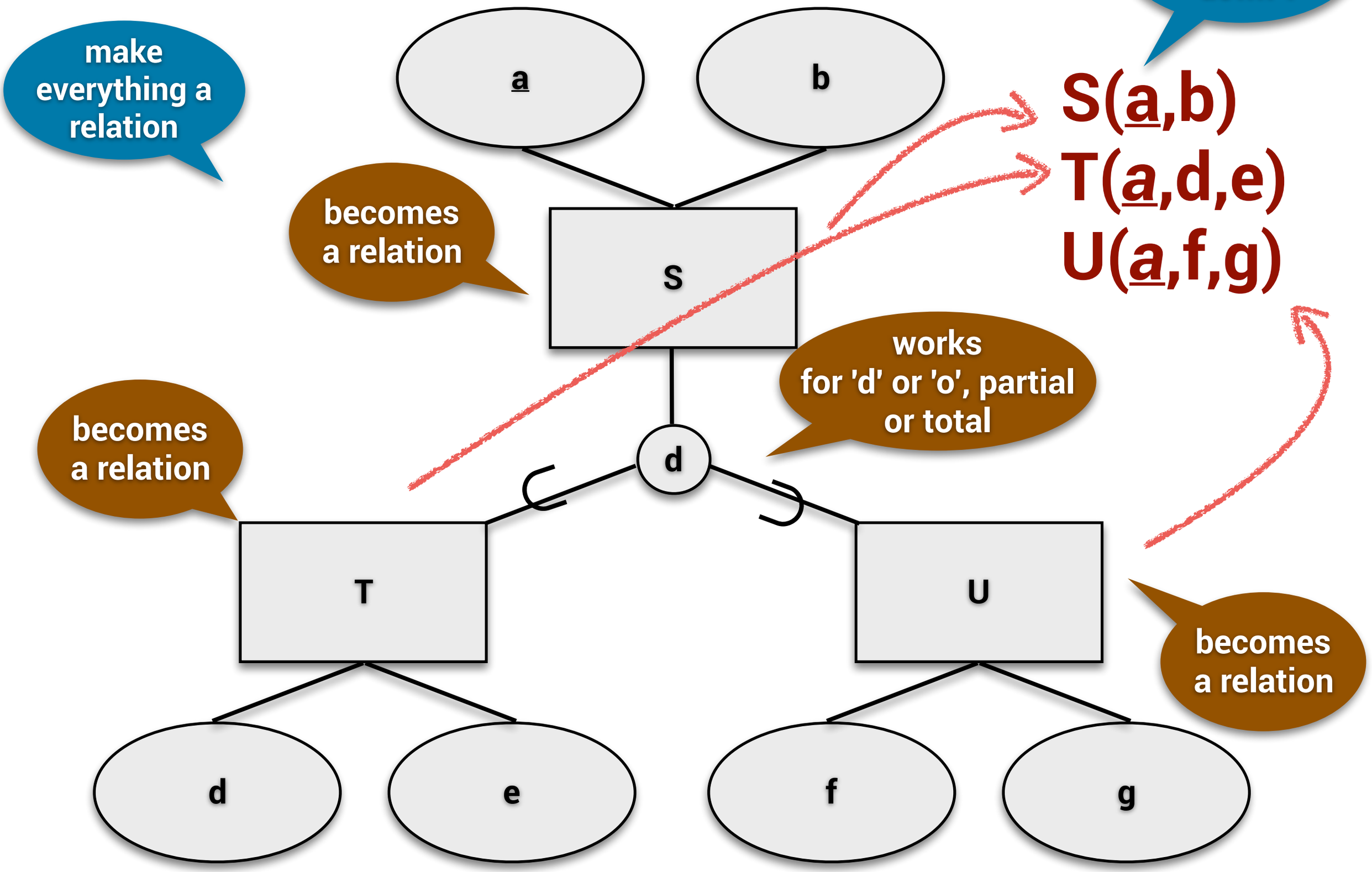
S(a)
B(a,b)

notice! composite PK!

7: Mapping N-ary Relationship Types



Spec/Gen Case-by-Case [8A]

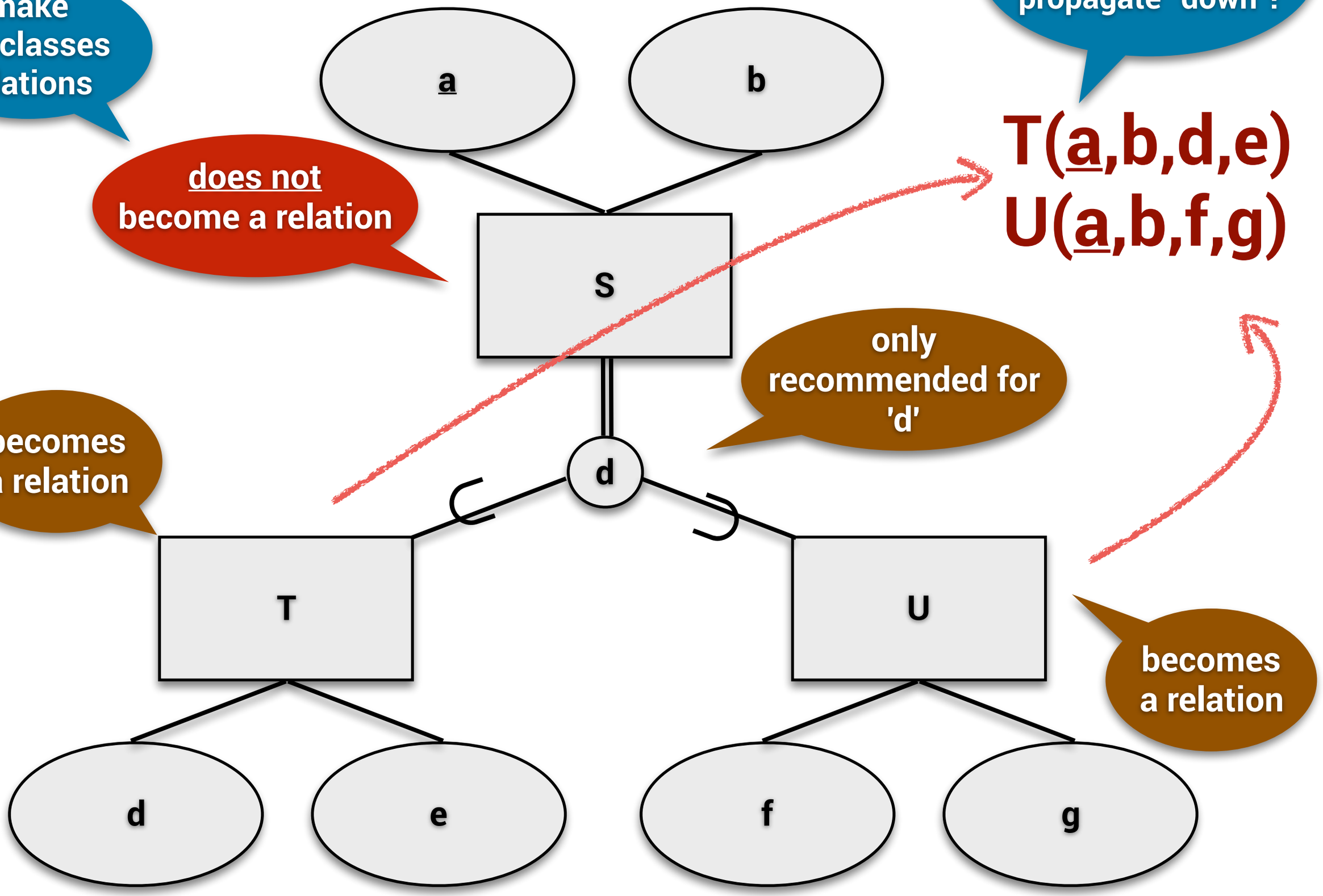


Spec/Gen Case-by-Case [8B]

make subclasses relations

does not become a relation

notice! PK and non-key attributes propagate "down"!



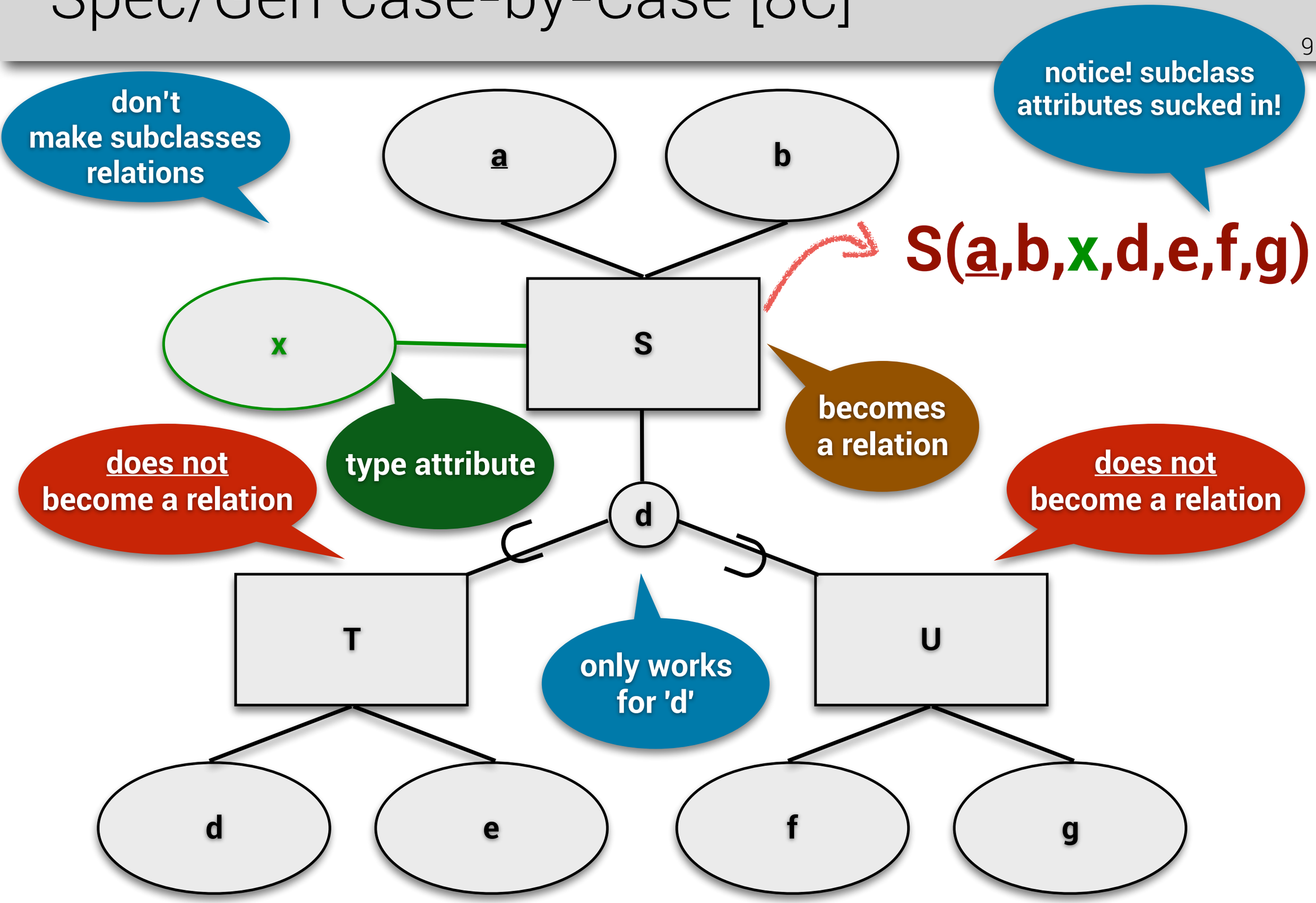
T(a,b,d,e)
U(a,b,f,g)

only recommended for 'd'

becomes a relation

becomes a relation

Spec/Gen Case-by-Case [8C]

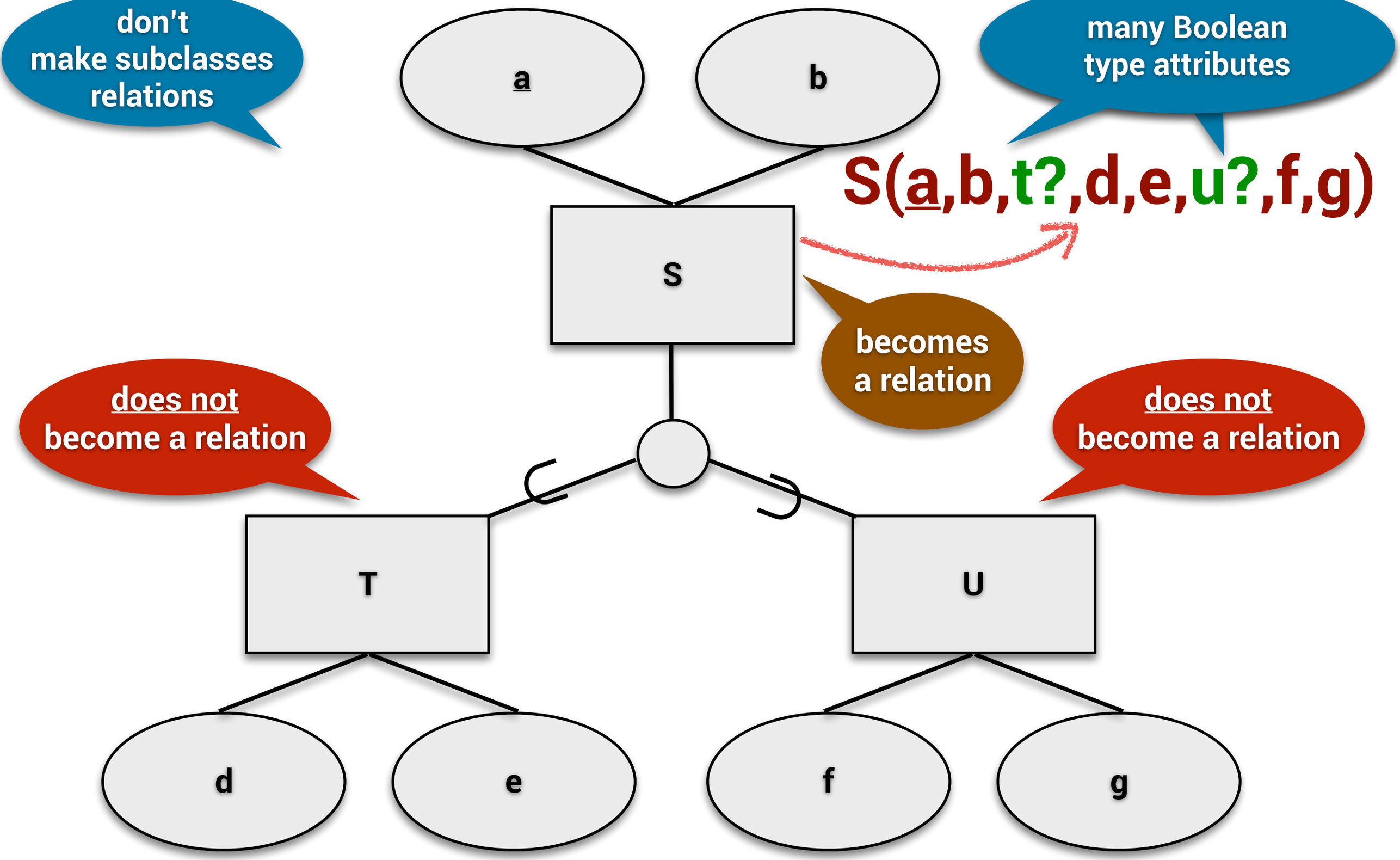


Spec/Gen Case-by-Case [8D]

messy?...

don't make subclasses relations

many Boolean type attributes



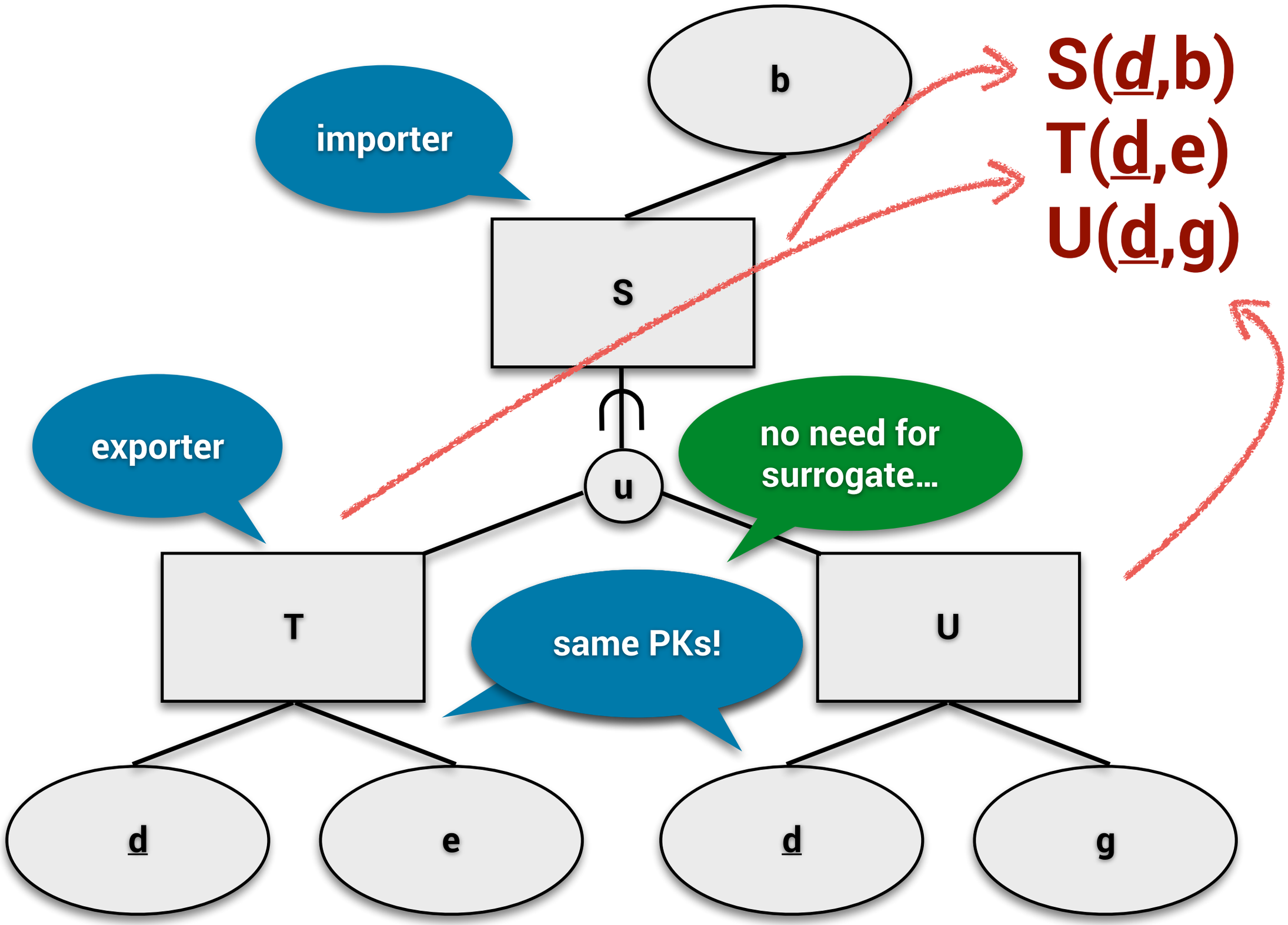
S(a,b,t?,d,e,u?,f,g)

becomes a relation

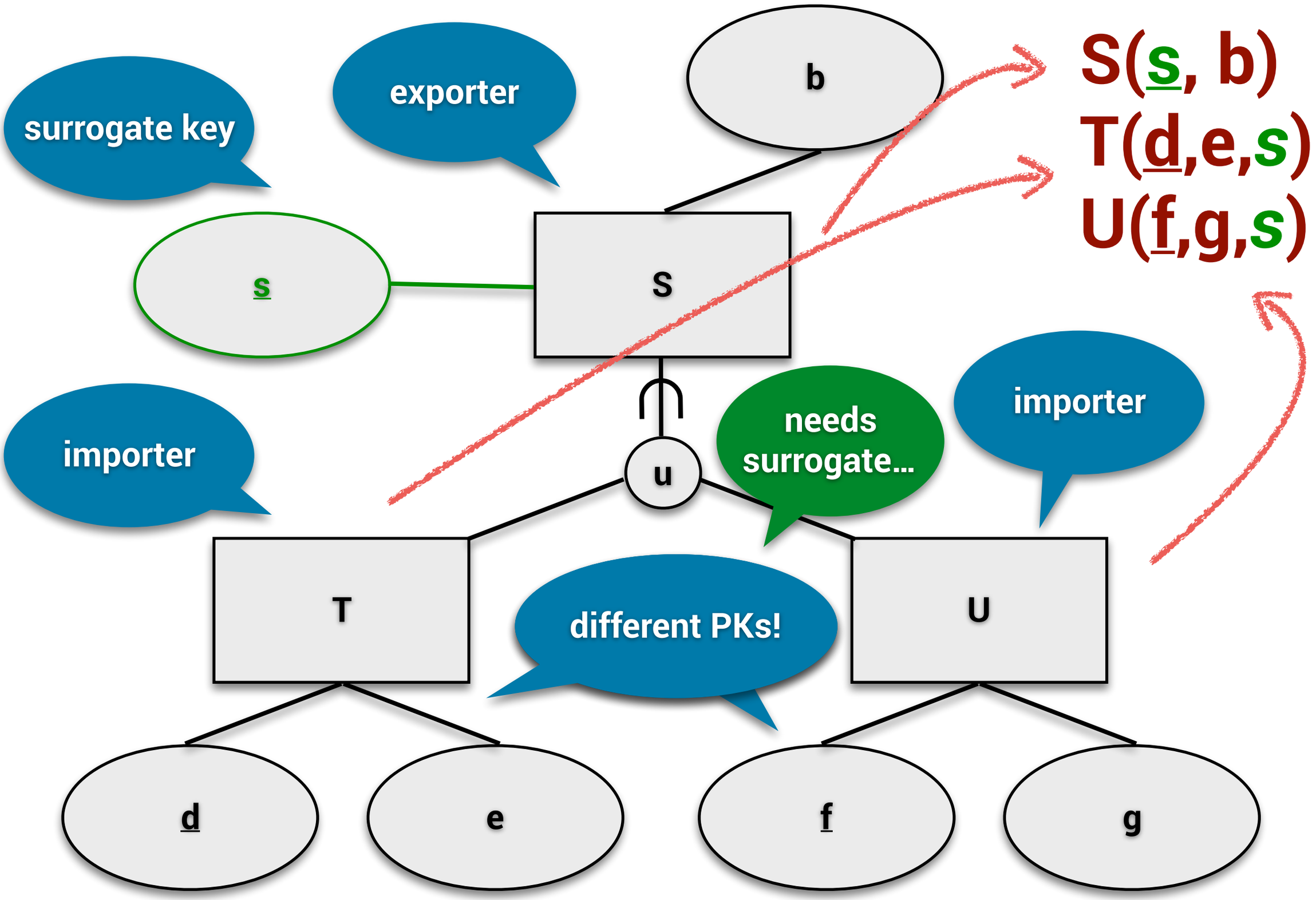
does not become a relation

does not become a relation

Categories (Union Types)Case-by-Case [9a]



Categories (Union Types)Case-by-Case [9b]



What Next?

- Sandra Sampaio takes over.
- She'll cover:
  - Functional dependencies and normal forms
  - Normalization
  - Advanced SQL
  - Transactions
  - File Organizations and Indices
- A final lecture, in the last week of term by Alvaro A A Fernandes will discuss DBMS architectures.