Entity-Relationship Data Model

SWEN304 / SWEN435

Trimester 1, 2024

Lecturer: Kevin Shedlock

Engineering and Computer Science

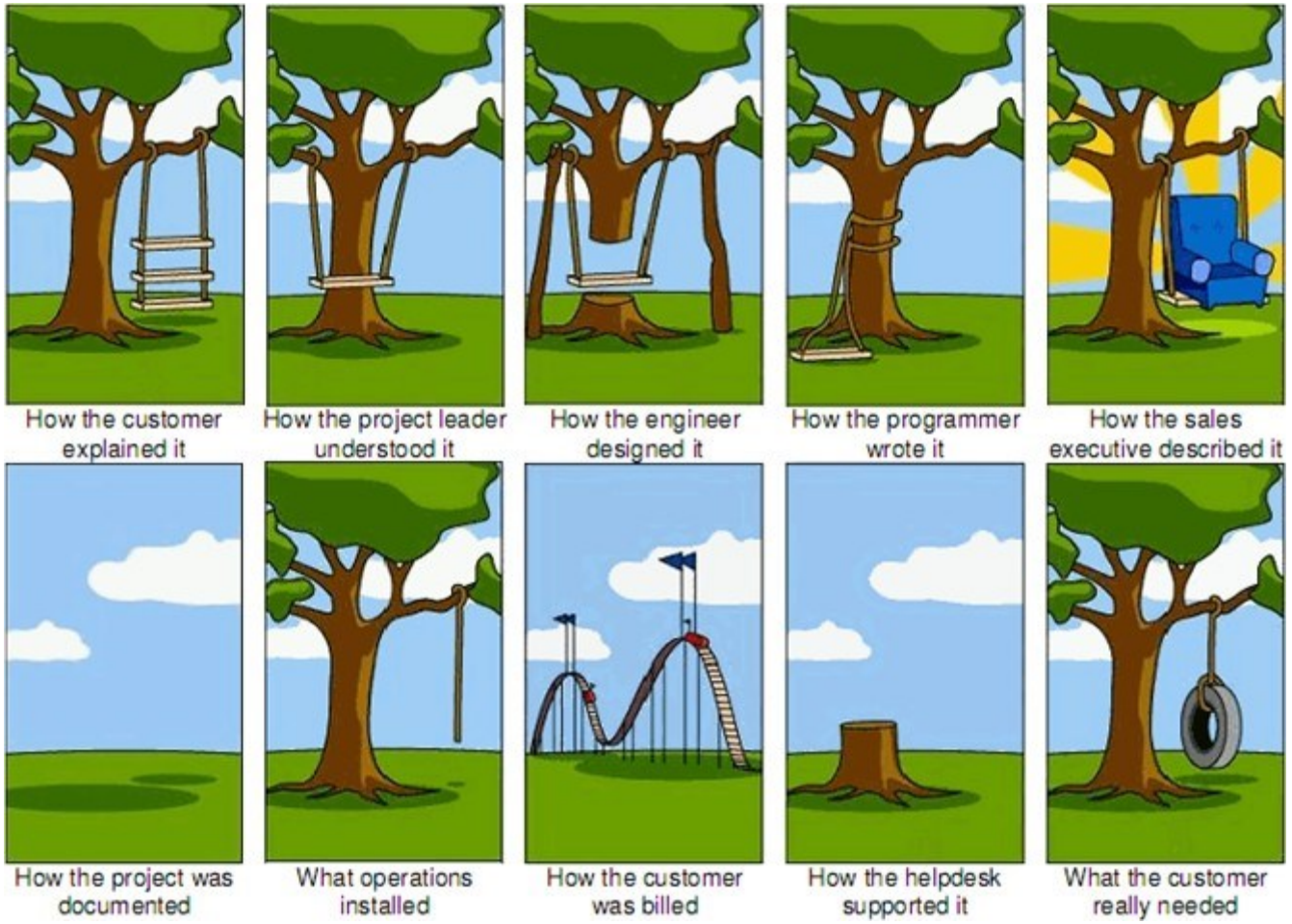
VICTORIA UNIVERSITY OF WELLINGTON TE HERENGA WAKA

Database Design

We will study:

1. Database Design Process
   - Requirements collection and analysis
   - Conceptual design
   - Logical design
   - Physical design
2. Entity-Relationship Model
   - Entities and relationships
   - Entity-Relationship diagram
3. Extensions of the Entity-Relationship Model
   - Specialisation
   - Aggregation and higher-order relationship types
   - Generalisation and clusters
4. MovieDB Examples

Review: Database Design as a Challenge?



Review: Success Factors in Database Design

- Work interactively with domain experts and users
- Use a structured approach throughout the data modelling process
- Employ a data-driven approach
- Incorporate structural/integrity considerations into data models
- Combine conceptualisation, refinement and validation techniques into the data modelling methodology
- Use diagrams to represent the DB schema as much as possible
- Build a data dictionary to supplement the diagrams
- Be willing to repeat steps

Review: Database Requirements Complexity

- Understanding Requirements - a thorough understanding of the organization's requirements, including **data needs**, **business rules**, and **processes**.

- **Gathering and interpreting** the requirements accurately can be challenging, especially when dealing with stakeholders with diverse perspectives and priorities.

Review: Database Design Complexity

- Database design is a multi-disciplinary task that requires **consideration between the trade-offs,** and constraints matching technical expertise, domain knowledge, and effective communication skills.

- Data Modelling Complexity – **translating real-world entities** and relationships accurately into a structured database schema that makes design decisions can be complex.

Review: Database Security Complexity

- Security and Privacy - the **security** and **privacy of data** is critical in database design. Design decisions must consider factors such as access control, encryption, and compliance with regulatory requirements i.e. balancing **security requirements** with **usability** and **performance** can be challenging.

Extensions of the Entity-Relationship Model

- An Extended Entity-Relationship (EER) model is an enhancement to the original Entity-Relationship (ER) model proposed by Chen.
- An Extended Entity-Relationship provides additional constructs to represent complex relationships and constraints more accurately.
- The Extended Entity-Relationship model is suitable for capturing complex data requirements and constraints in modern database systems.

Extensions of the Entity-Relationship Model

- Extensions enrich the power of the ER model by giving database modellers features that add more information to the model. Some of these key features include:
  - Specialization and generalization
  - Constrain Specification
  - Aggregation
  - Inheritance
  - Categories and Union Types
  - Subtypes, supertypes

Extensions of the Entity-Relationship Model

- The ER model with extensions provides effective and convenient means of describing the target of the database at a high level of abstraction

- Some extensions we discuss here:
  - Specialisation
  - Aggregation and higher-order relationship types
  - Generalisation and clusters

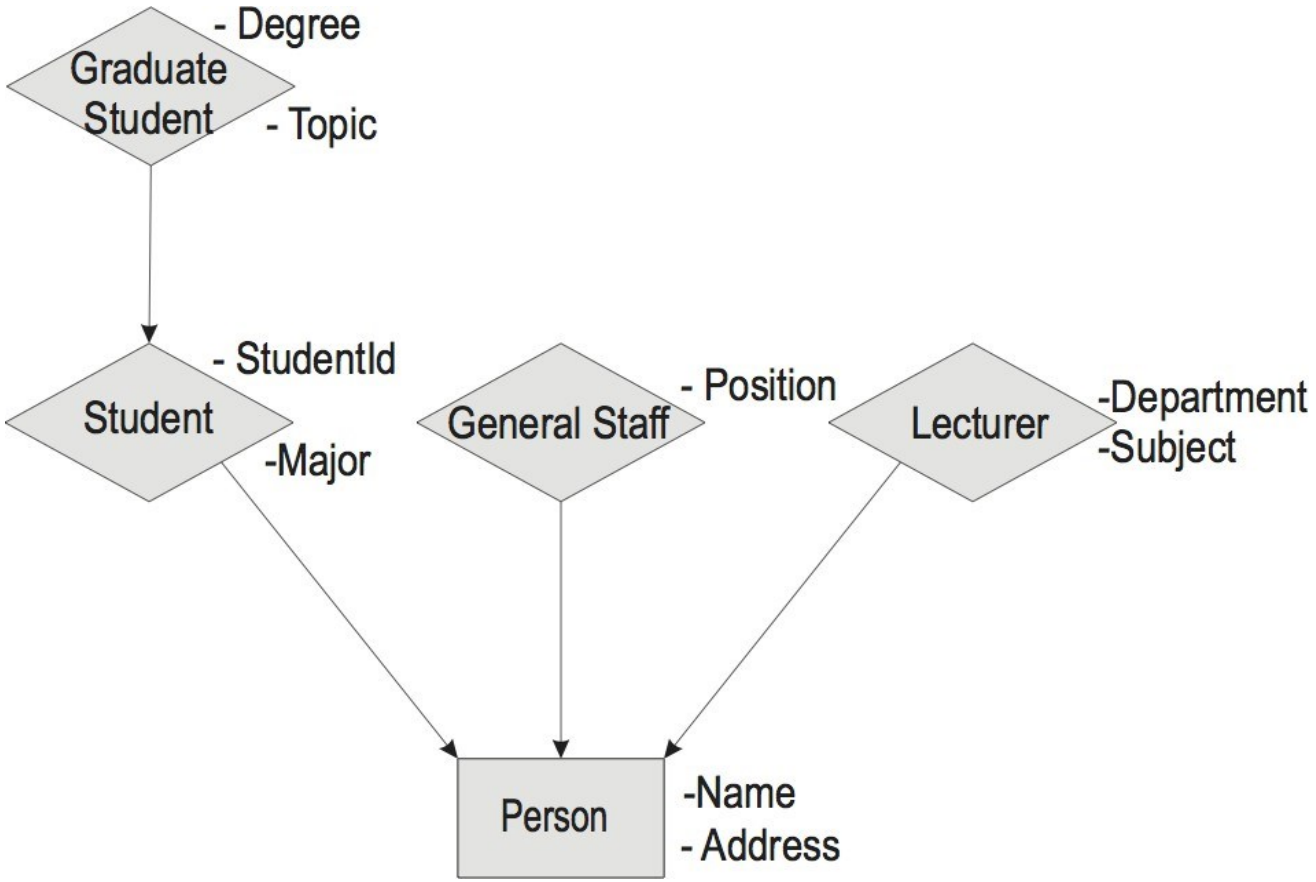
Specialisation

- Sometimes, an object can be represented by more than just a single abstract concept
  - A Student is also a person
  - Tutors are also students, and thus they are also people

- This idea is known as specialisation is a subtype of the more general supertype. Subtype inherits all features of the supertype **and** may add new properties.

- The subtype may be modelled as a unary relationship type whose single component is just the supertype (but may have additional attributes). Every object of the subtype gives rise to at most one object of a supertype

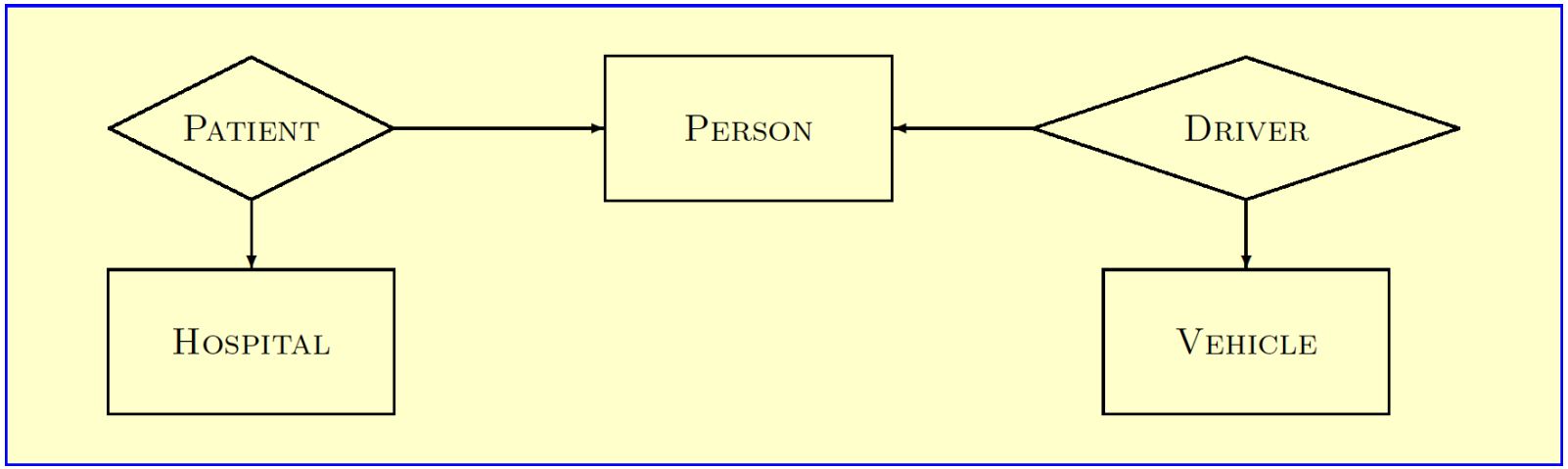
An Example for a Specialisation Hierarchy



Abstract concepts can be derived from other abstract concepts. Above, the more specific student derives from the more general person

An Example for Aggregation

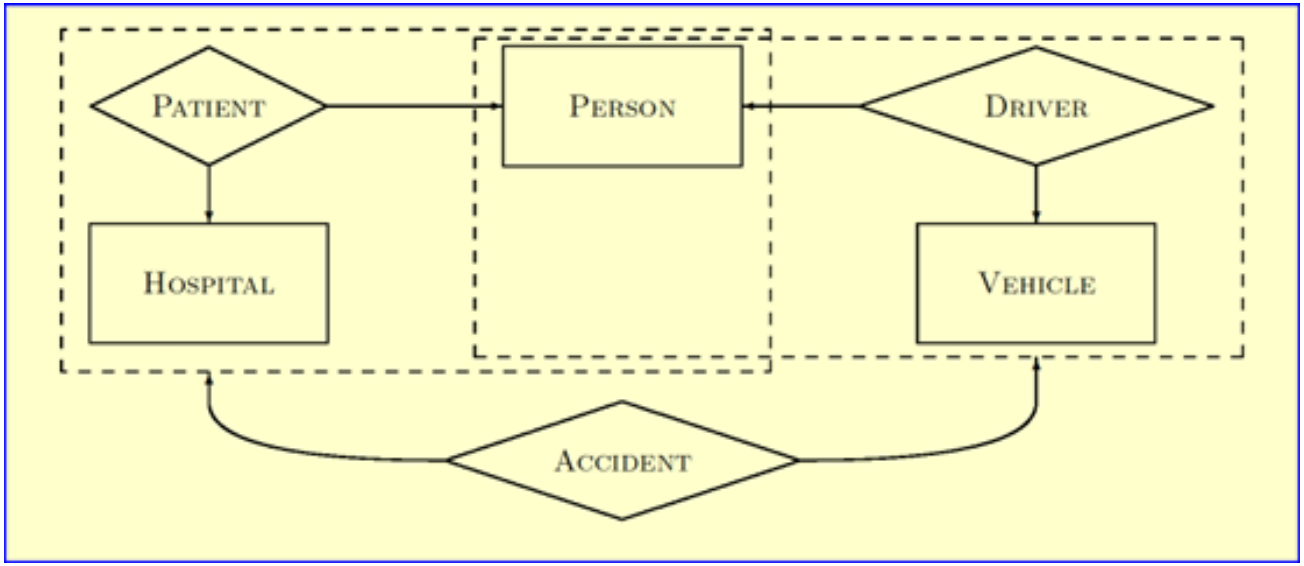
- The following ER diagram contains two relationship types: Patient and Driver



- How can we model that a person is in hospital because they had an accident when driving?

An Example for Aggregation

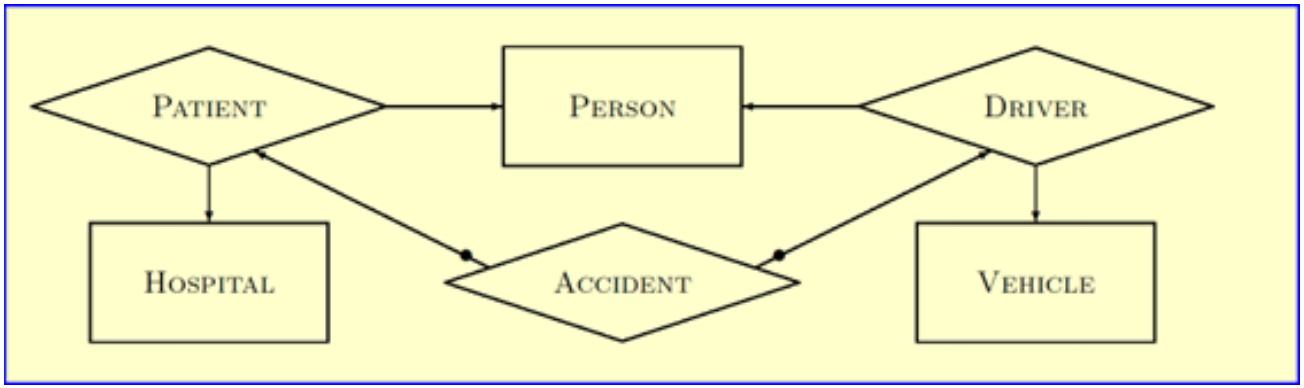
- **Idea:** We could use a new relationship type Accident to model these relationships
- Depicting the components of Accident?



- It would be convenient to regard parts of the ER diagram as entity types (called **aggregation**)

An Example of Relationship Types

- Idea: We could use the relationships types Patient and Driver as components of Accident



- It would be convenient to allow relationship types to participate in other relationship types (these are called higher-order relationship types)

- In our example the relationship types have orders:
  - Patient and Driver have order 1
  - Accident has order 2

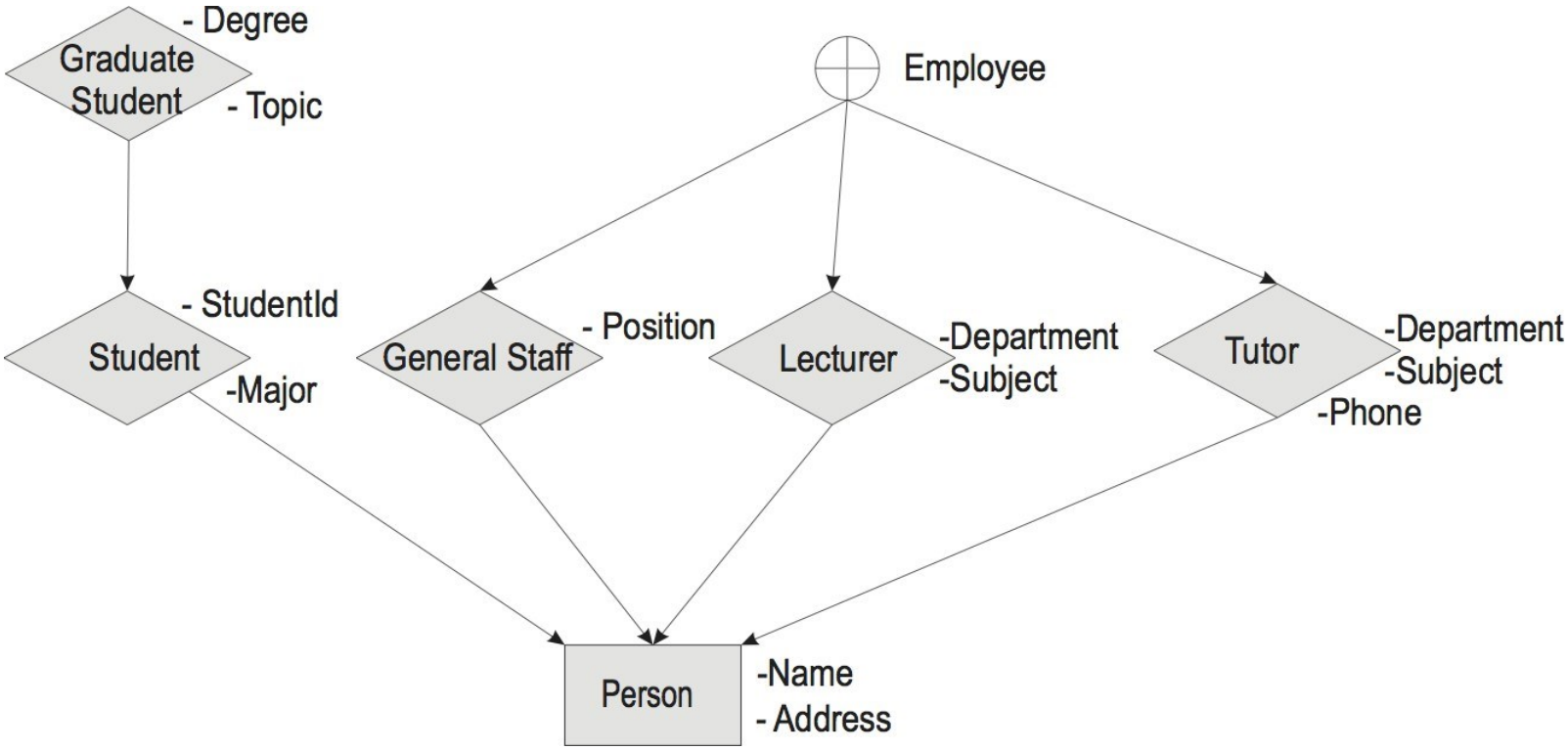
Higher-order Relationship Types

- A relationship type *R* is of **order 1** if all its components are entity types
  - *In a (simple) ER diagram all relationship types are of order 1*
- *R* is of **order k** if its components have maximum order $k - 1$
  - This extension is possible in an extended ER diagram
  - It gives the database designer more flexibility, e.g. to model aggregation
- For convenience, entity relationship types are all called **object types**
  - Entities and relationships are called objects
  - Entity sets and relationship sets are called object sets
  - Entity types may be regarded as object types of order 0
  - Object types of order *k* are just relationship types of order *k*

Generalization and Clustering

- Cluster refers to a way of organizing and structuring data within a database system using enhanced entity-relationship (EER) modelled techniques.
- An EER cluster typically refers to a grouping of related entities, attributes, and relationships within the EER diagram. Clustering helps in organizing the database schema in a logical and efficient manner (easier to understand)
- A Cluster can represent a logical grouping of entities and their relationships within a specific domain or application context.

Cluster



This example shows the EER cluster representing knowledge of its Students who are enrolled as graduates with TTUPA. The student is part of the Person supertype, and the *Employee* cluster that includes General Staff; Lecturer and Tutor. Each with their own known attributes.

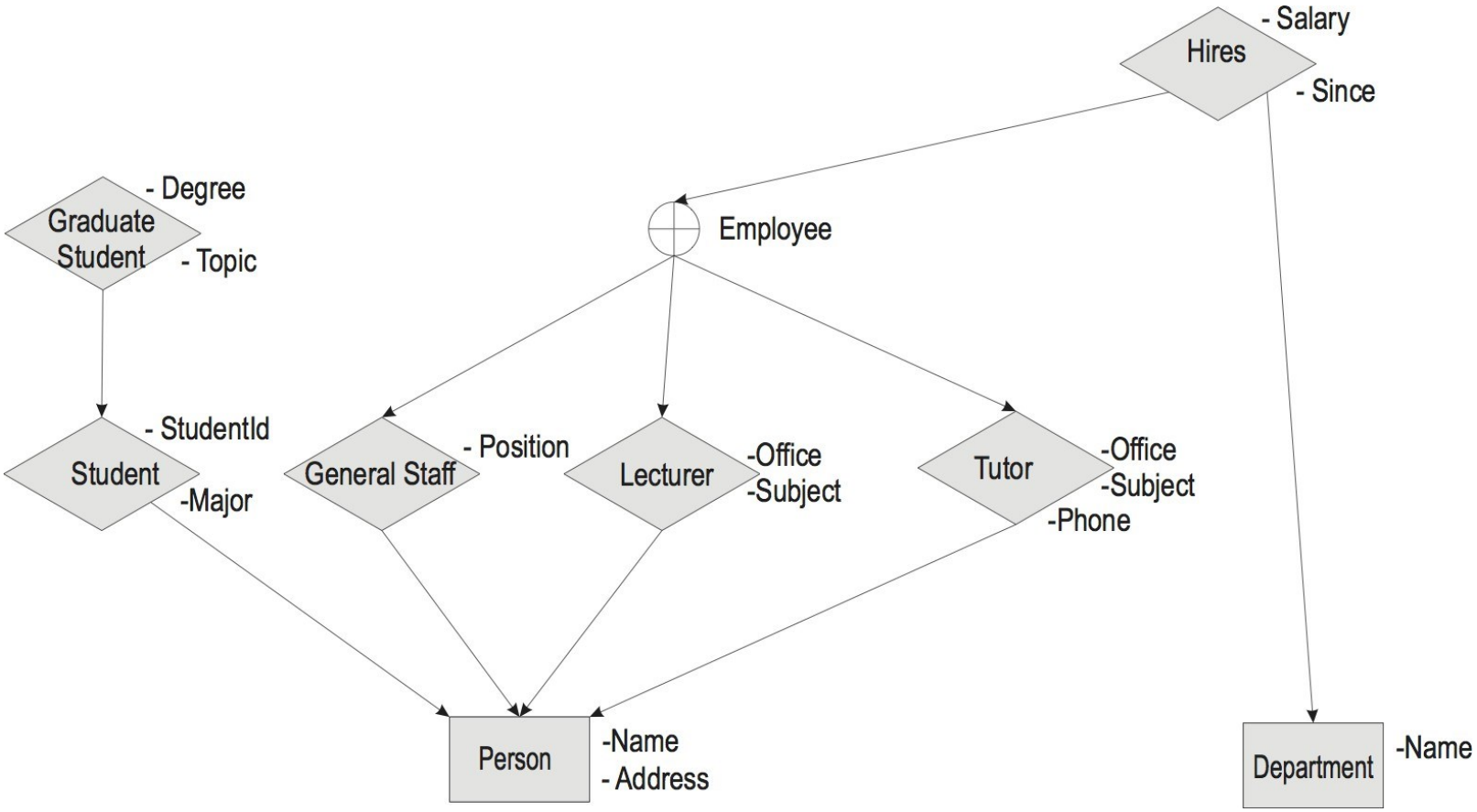
Cluster

- Sometimes a database designer wants to model **alternatives**, for example having a relationship to students *or* lecturers
- We can abstract further by generalising a set of abstract concepts

- A **cluster** C represents an alternative among a collection of object types
  - let C be an alternative between given object types $C_1, \ldots, C_m$
  - for simplicity, the cluster can be denoted as $C = C_1 \oplus \cdots \oplus C_m$
  - we call $C_1, \ldots, C_m$ the **components** of the cluster C

- **Example:** Employee = General Staff $\oplus$ Lecturer $\oplus$ Tutor $\oplus$ Every object of cluster type C is an object of *exactly one* of the object types in the alternative
- A cluster is of **order** k if its components have maximum order $k - 1$

A Cluster participating in a Relationship Type



This example shows the EER cluster representing that is part of the Hires relationship that is part of the entity Department. Again each with their respective attributes.

How to handle Clusters?

- Clusters used in conceptual design to model alternatives. the relational data model does **not** provide similar concept
- Transform an EER schema with clusters into an equivalent EER schema without clusters
- Only necessary as pre-processing before actual transformation to the relational data model
- In general: clusters provide database designers a convenient way to model objects in the target of database
- Best not to avoid clusters as the size of an EER schema increases dramatically and becomes harder to interpret

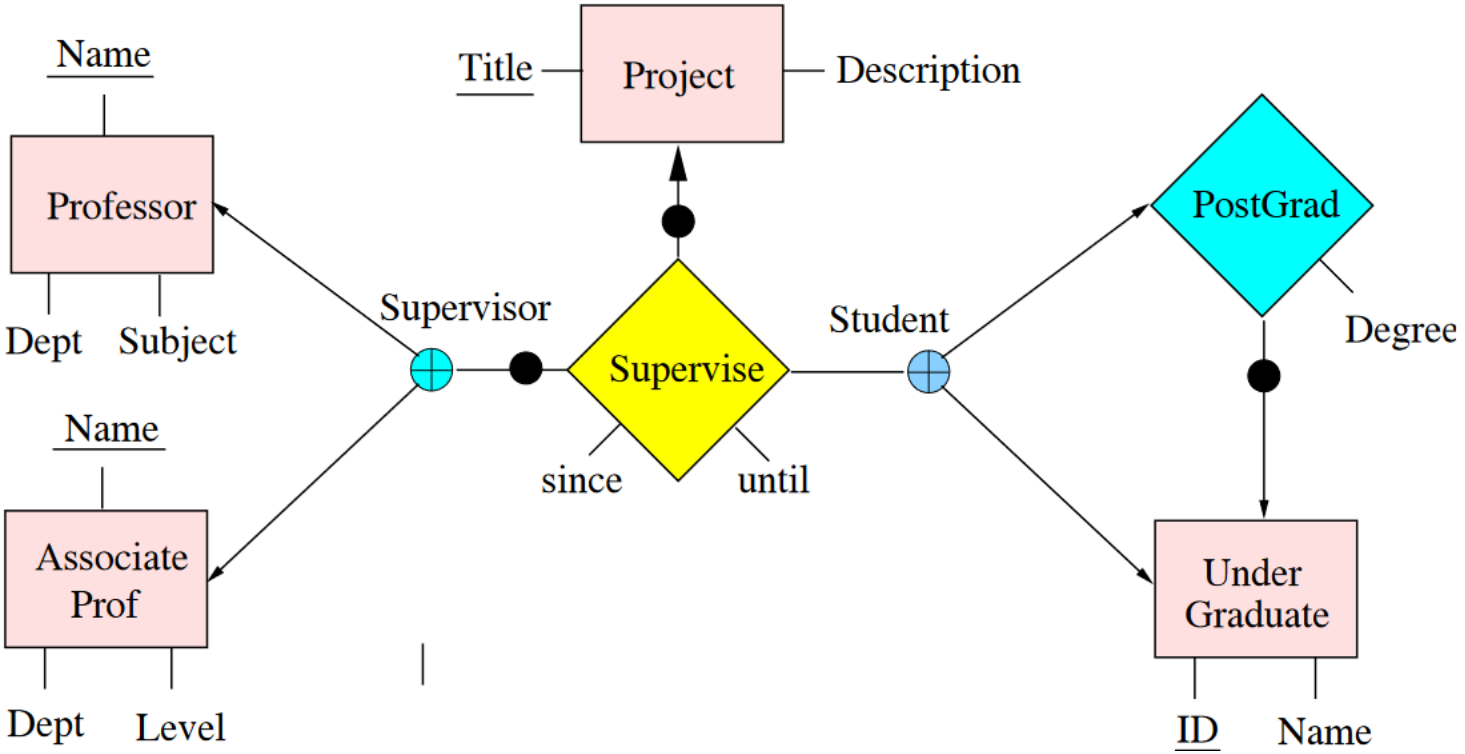
How to handle Clusters?

Use the following procedure to replace clusters:

- clusters that are not component of any relationship type can be removed
- consider relationship type R with a cluster component $C = C_1 \oplus \ldots \oplus C_n$
- replace R by n new relationship types $R_1, \ldots, R_n$:
- for $i = 1, \ldots, n$, obtain R_i from R by replacing every occurrence of C by C_i
- if R_i still contain clusters, then repeat the process for each cluster
- the final extended ER schema is cluster-free
- now we can apply the previously described transformation to the relational data model

Example – Transforming Clusters

- Consider the following EER diagram:



SUPERVISE={SUPERVISOR, STUDENT, PROJECT},{since, until},{SUPERVISOR, PROJECT}) with

- cluster SUPERVISOR = PROFESSOR ⊕ ASSOCIATEPROF
- cluster STUDENT = POSTGRAD ⊕ UNDERGRADUATE

Example – Transforming Clusters

- By replacing the cluster SUPERVISOR we obtain:
  - PROF_SUPERVISION:
    ({PROFESSOR, STUDENT, PROJECT},{since,until},{PROFESSOR, PROJECT})
  - APROF_SUPERVISION:
    ({ASSOCIATEPROF, STUDENT, PROJECT},{since,until},{ASSOCIATEPROF, PROJECT})
- By replacing the cluster STUDENT subsequently we obtain:
  - PROF_POSTGRAD_SUPERVISION:
    ({PROFESSOR, POSTGRAD, PROJECT},{since,until},{PROFESSOR, PROJECT})
  - PROF_UNDERGRAD_SUPERVISION:
    ({PROFESSOR, UNDERGRADUATE, PROJECT},{since,until},{PROFESSOR, PROJECT})
  - APROF_POSTGRAD_SUPERVISION:
    ({ASSOCIATEPROF, POSTGRAD, PROJECT},{since,until},{ASSOCIATEPROF, PROJECT})
  - APROF_UNDERGRAD_SUPERVISION:
    ({ASSOCIATEPROF, UNDERGRAD, PROJECT},{since,until},{ASSOCIATEPROF, PROJECT})

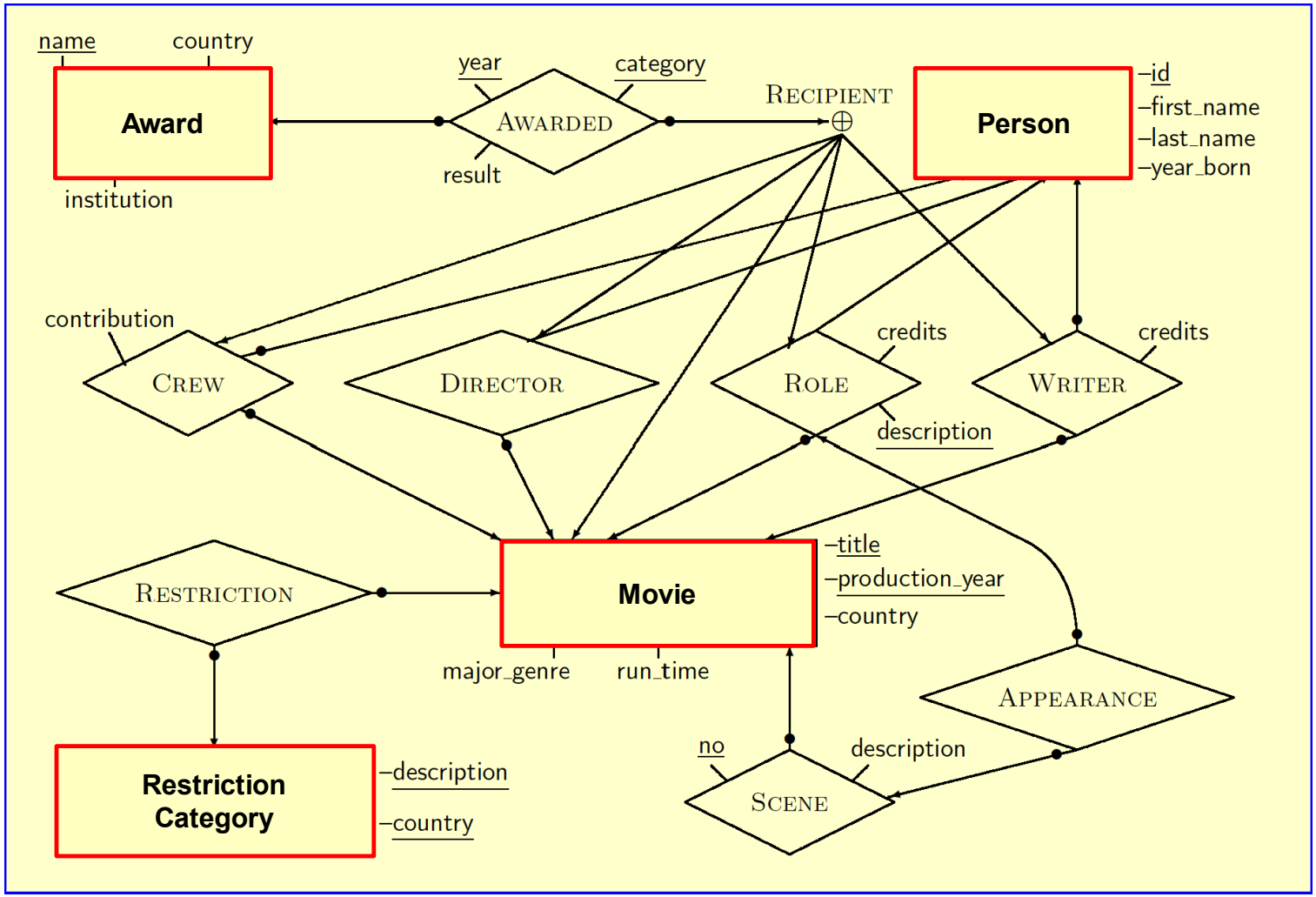
Extended Entity-Relationship Diagram, Schema and State

- A **extended entity-relationship diagram** (**EER diagram**) is a directed graph with a node for every entity type or relationship type
  - Convention: draw entity types as rectangles, draw relationship types as diamonds, draw clusters as $\oplus$

- An **extended entity-relationship schema** (**EER schema**) is a finite set S of object types (entity types, relationship types, clusters), such that for every object type O in S all its components belong to S, too

- A **state/instance** S^t of an EER schema S assigns each object type O an object set O^t, so that for each object o in the object set O^t and for every component O' of O the object $o(O')$ belongs to the object set O'^t

Example: EER Diagram of the MovieDB



Example: EER Schema of the MovieDB

Entity Types

- Person
  - *attributes:* id, first name, last name, year born
  - *primary key:* id
- Movie
  - *attributes:* title, production year, country, run time, major genre
  - *primary key:* title, production year
- Award
  - *attributes:* name, institution, country
  - *primary key:* name
- Restriction_Categrory
  - *attributes:* description, country
  - *primary key:* description, country

Example: EER Schema of the MovieDB

Relationship Types (order 1):

- Director
  - *components:* Person, Movie
  - *attributes:*
  - *primary key:* Movie
- Writer
  - *components:* Person, Movie
  - *attributes:* credits
  - *primary key:* Person, Movie
- Crew
  - *components:* Person, Movie
  - *attributes:* contribution
  - *primary key:* Person, Movie

Example: EER Schema of the MovieDB

Relationship Types (order 1, cont):

- Role
  - *components:* Person, Movie
  - *attributes:* description, credits
  - *primary key:* Movie, description
- Scene
  - *components:* Movie
  - *attributes:* scene no, description
  - *primary key:* Movie, description
- Restriction
  - *components:* Restriction Category, Movie
  - *attributes:*
  - *primary key:* Restriction Category, Movie

Example: EER Schema of the MovieDB

Cluster (order 2):

- Recipient = Movie $\oplus$ Crew $\oplus$ Director $\oplus$ Writer $\oplus$ Role

Relationship Types (order 2):

- Appearance
  - *components:* Role, Scene
  - *attributes:*
  - *primary key:* Role, Scene

Relationship Types (order 3):

- Awarded
  - *components:* Award, Recipient
  - *attributes:* year, category, result
  - *primary key:* Award, Recipient, year, category

Transformation Entity-Relationship to Relational Data Model

- The relational data model is the basis for defining a relational database with SQL

- How to transform from the entity-relationship to the relational data model?

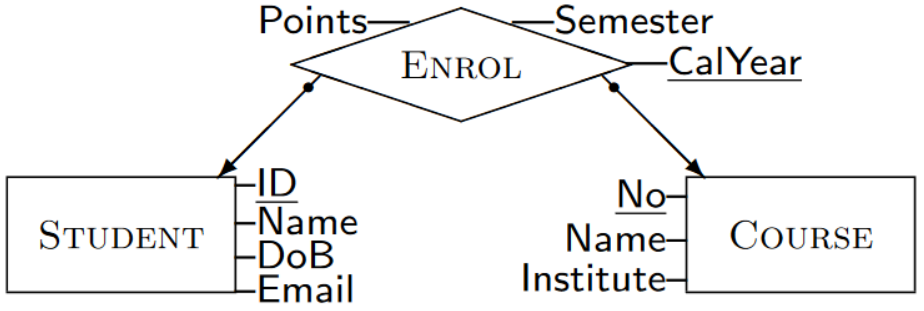
- We will transform an EER schema to a relational schema, starting with the entity types and then working upwards

Transformation Entity-Relationship to Relational Data Model

Consider a relationship type $R = (comp(R), attr(R), key(R))$. For each component E of R we choose a set:

$$keyAttr(E) = \{E\_A : \text{where } A \text{ is a primary key attribute of } E^{\text{trans}}\}$$

where E_A are pairwise disjoint new attribute names that do not occur in $attr(R)$ and E^{trans} originates from a prior transformation of component E



Example:

Given a relationship type ENROL with component set {STUDENT, COURSE}, attribute set {CalYear, Semester, Points}, and primary key {STUDENT, COURSE, CalYear}

. . . we choose the following sets:
$keyAttr(\text{STUDENT}) = \{\text{Student\_Id}\}$ and $keyAttr(\text{COURSE}) = \{\text{Course\_No}\}$