Entity–Relationship Data Model

SWEN 304

Trimester 2, 2019

Lecturer: Dr Hui Ma

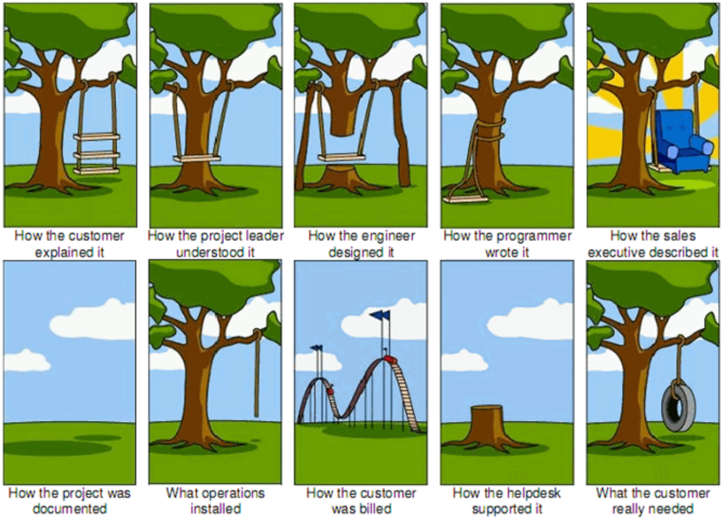
Engineering and Computer Science

TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI
VICTORIA
UNIVERSITY OF WELLINGTON

Database Design - Motivation

- According to the Project Management Institute,
    - 61% of all IT projects failed or were stopped before completion (in 2007)
    - more than 75% of all projects exceeded their budgets by 30% (in 2008)
- **Why do IT projects fail to meet customer expectations?**
- There can be many reasons:
    - the client is not sure what they want
    - the requirements were not properly documented
    - the lack of appropriate development methodology
    - the desired functionality was difficult to develop
    - the budget and resources were not sufficient
    - the lack of team and work management
    - . . .

Database Design - Motivation


How the customer explained it
How the project leader understood it
How the engineer designed it
How the programmer wrote it
How the sales executive described it
How the project was documented
What operations installed
How the customer was billed
How the helpdesk supported it
What the customer really needed

Database Design – Four Phases

- **Database Design** is the process of constructing a **detailed schema of a database** that can support the organization's business needs and objectives for the database system under development
- The database design process has four phases:
  1. **Requirements Collection and Analysis**
  2. **Conceptual Design**
  3. **Logical Design**
  4. **Physical Design**

Phase 1: Requirements Collection and Analysis

- **Requirements collection and analysis** is the process of collecting and analyzing data requirements of the organization so as to provide database solutions that fulfill business needs of the organization
- Compilation of data requirements includes:
  - a description of the data to be used or generated
  - details of how data is to be used or generated
  - any additional requirements for database system under development
  - . . .

Phase 2: Conceptual Design

- **Conceptual design** is the process of constructing a database schema used in an organization, independent of any physical or implementation considerations
    - modelling at a high-level of abstraction
    - simple enough
    - often with graphical representation
    - used to communicate the logical structure of a database with domain experts and potential users
- The data model is built using the input from the requirements specification
- **Note:** The conceptual design is based on the **entity-relationship model** in this course

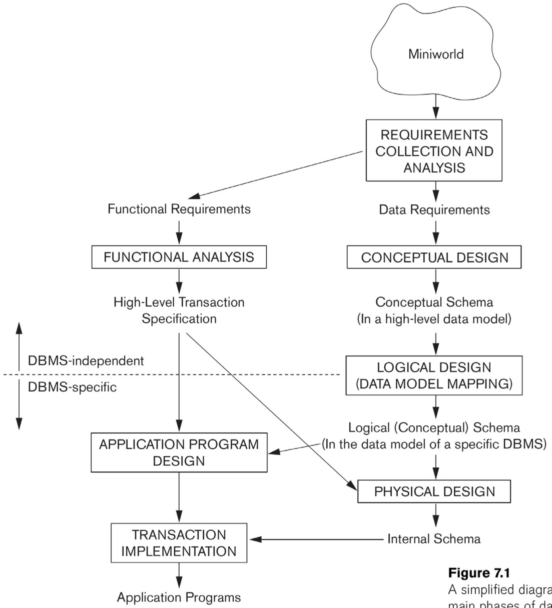
Phase 3: Logical Design

- **Logical design** is the process of constructing a database schema used in an organization, as represented by a particular data manipulation technology, but still independent of physical considerations
- The conceptual database schema is mapped onto a logical database schema, which can be further refined to meet the data requirements
    - popular data manipulation technogies used in logical design are those provided by the relational model or some other model (for example: OO, XML, JSON)
    - in case of the relational model, the database schema includes a description of all tables with their primary and foreign keys, so that SQL can be applied for data definition, data manipulation, and data querying
- **Note:** The logical design is based on the **relational model** in this course

Phase 4: Physical Design

- **Physical design** is the process of implementing the logical data model in a database management system (DBMS)
- If the logical data model is the relational model, then the physical design is to establish relations in the DBMS that involves:
  - selecting the files in which to store the tables
  - deciding which indexes should be used to achieve efficient access to data items
  - describing the integrity constraints and security measures
  - ...
- Decisions made during the physical design phase often affect non-functional properties of the database at run time, such as performance, accessibility, security and user-friendliness
- **Note:** Details of physical design are beyond the scope of this course

Database Design Process - Overview



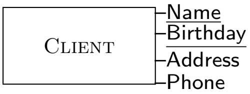
Miniworld
REQUIREMENTS COLLECTION AND ANALYSIS
Functional Requirements
Data Requirements
FUNCTIONAL ANALYSIS
CONCEPTUAL DESIGN
High-Level Transaction Specification
Conceptual Schema (In a high-level data model)
DBMS-independent
DBMS-specific
LOGICAL DESIGN (DATA MODEL MAPPING)
Logical (Conceptual) Schema (In the data model of a specific DBMS)
APPLICATION PROGRAM DESIGN
PHYSICAL DESIGN
TRANSACTION IMPLEMENTATION
Internal Schema
Application Programs

Figure 7.1
A simplified diagram to illustrate the main phases of database design.

Entity-Relationship Model - Terminology

- the **Entity-Relationship model** (ER, for short) is the most popular conceptual data model, and the de facto standard in conceptual design
- Originally proposed by Peter P. Chen in 1976
- The target of the database is regarded as consisting of **entities** and **relationships**
- **Entities** are basic objects that can be identified in the target of the database
- **Attributes** are properties that describe entities
- Entities that are described by the same set of attributes are classified into an **entity set**
- An entity set can be abstracted into an **entity type**
- **Example:**

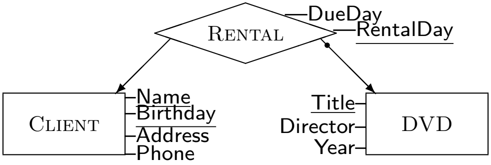
  A entity type CLIENT with attribute set {Name, Birthday, Address, Phone}, and key {Name, Birthday}



Entity-Relationship Model - Terminology

- **Relationships** are associations between entities, or objects that are derived from entities
- Relationships have entities as their **components**
- Relationships can also have **attributes** as properties that describe them
- Relationships that are described by the same set of components and attributes are classified into a **relationship set**
- A relationship set can be abstracted into a **relationships type**
- **Example:**

  A relationship type RENTAL with component set $\{$CLIENT, DVD$\}$, attribute set $\{$RentalDay, DueDay$\}$, and key $\{$DVD, RentalDay$\}$

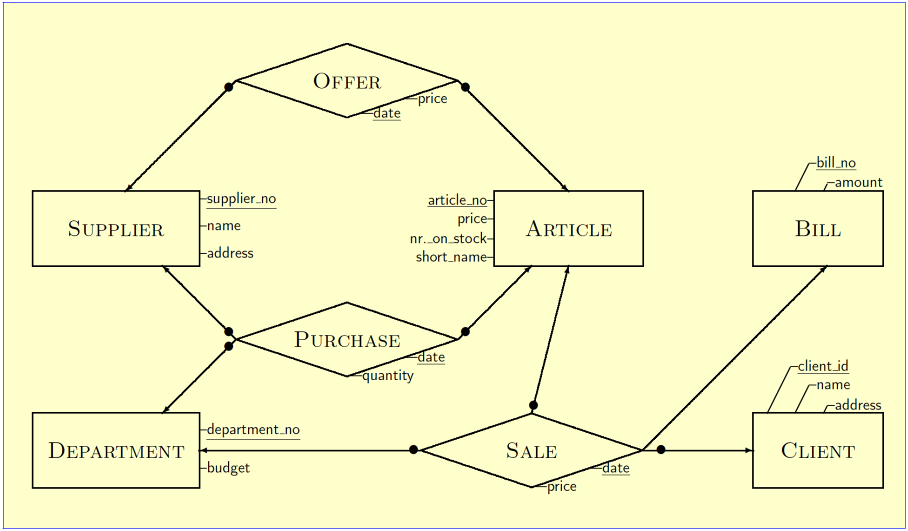


Entity-Relationship Diagram

- For the communication between database designers, database programmers, domain experts and potential users, it is helpful to visualise the database schema graphically
- A **entity-relationship (ER) diagram** is a directed graph with a node for every entity type or relationship type
- **Convention** (proposed by Chen): draw entity types as rectangles, and draw relationship types as diamonds
- To capture more details of the conceptual design, database designers often show ...
    - attributes in the ER diagram by attaching them to their entity or relationship type
    - keys in the ER diagram by underlining key attributes and by putting dots on key components
    - attribute domains in the ER diagram by adding them to the attributes

Entity-Relationship Diagram - Example

- **An example:**



Entity-Relationship Schema - Example

- **In our example we have the following entity types:**
- SUPPLIER
  - *attributes:* supplier_no, name, address
  - *key:* supplier_no
- ARTICLE
  - *attributes:* article_no, price, nr_on_stock, short_name
  - *key:* article_no
- BILL
  - *attributes:* bill_no, amount
  - *key:* bill_no
- DEPARTMENT
  - *attributes:* department_no, budget
  - *key:* department_no
- CLIENT
  - *attributes:* client_id, name, address
  - *key:* client_id

Entity-Relationship Schema - Example

- **In our example we have the following relationship types:**
- OFFER
  - *components:* SUPPLIER, ARTICLE
  - *attributes:* date, price
  - *key:* SUPPLIER, ARTICLE, date
- PURCHASE
  - *components:* SUPPLIER, ARTICLE, DEPARTMENT
  - *attributes:* date, quantity
  - *key:* SUPPLIER, ARTICLE, DEPARTMENT, date
- SALE
  - *components:* DEPARTMENT, ARTICLE, CLIENT, BILL
  - *attributes:* date, price
  - *key:* DEPARTMENT, ARTICLE, CLIENT, date

Entity-Relationship Schema

- An **entity-relationship schema** (**ER schema**, for short) is a finite set $\mathcal{S}$ of entity and relationship types, such that for every relationship type R in $\mathcal{S}$ all its components belong to $\mathcal{S}$, too

- An **instance** $\mathcal{S}^t$ of an ER schema $\mathcal{S}$ assigns each entity type E an entity set E^t and each relationship type R a relationship set R^t, such that for each relationship r in the relationsip set R^t and for every component E of R the entity $r(E)$ belongs to the entity set E^t

Keys

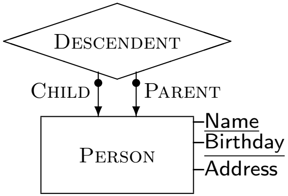
- **Keys** ensure that the entities in an entity set (or the relationships in a relationship set) can be **uniquely identified**
- In our example the entity types have keys as follows:
  - clients are uniquely identified by their client_id
  - departments are uniquely identified by their department_no
  - bills are uniquely identified by their bill_no
  - articles are uniquely identified by their article_no
  - suppliers are uniquely identified by their supplier_no
- In our example, the relationship types have keys as follows:
  - offers are uniquely identified by supplier, article and date
  - purchases are uniquely identified by department, supplier, article and date
  - sales are uniquely identified by article, department, client and date

n-ary Relationship Types

- An entity type E is described by its **attributes** $attr(E)$ and its **key** $key(E)$
- A relationship R is described by its **components** $comp(R)$, its **attributes** $attr(R)$ and its **key** $key(R)$
- A relationship type with n components is called **n-ary**, in particular
    - a **unary** relationship type has one component
    - a **binary** relationship type has two components
    - a **ternary** relationship type has three components

Recursive Relationship Types

- A relationship type is called **recursive** if it contains the same entity type more than once as a component
- **Example:** suppose we want to model relationships between a child and a parent (that is, between two persons)
- Can use a relationship type DESCENDENT that contains the entity type PERSON twice as a component



- **Role names** are used to distinguish the different occurrences of the same entity type in a recursive relationship type (in our example CHILD and PARENT)

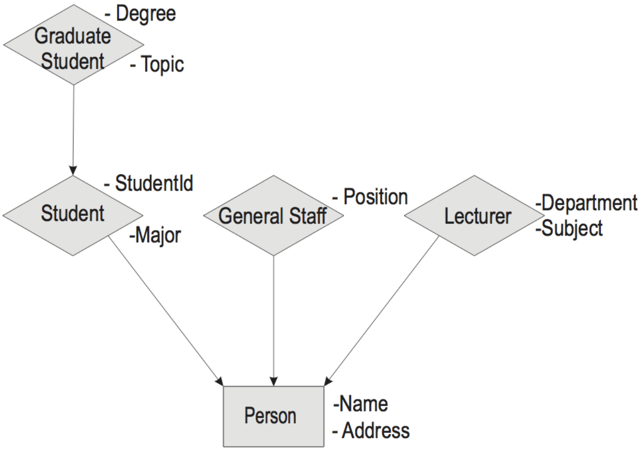
Extentions of the Entity-Relationship Model

- Several extensions have been proposed to overcome inconveniencies and to enrich the expressive power of the ER model
- After all, the ER model with its 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
    - Higher-order relationship types
    - Generalisation and clusters

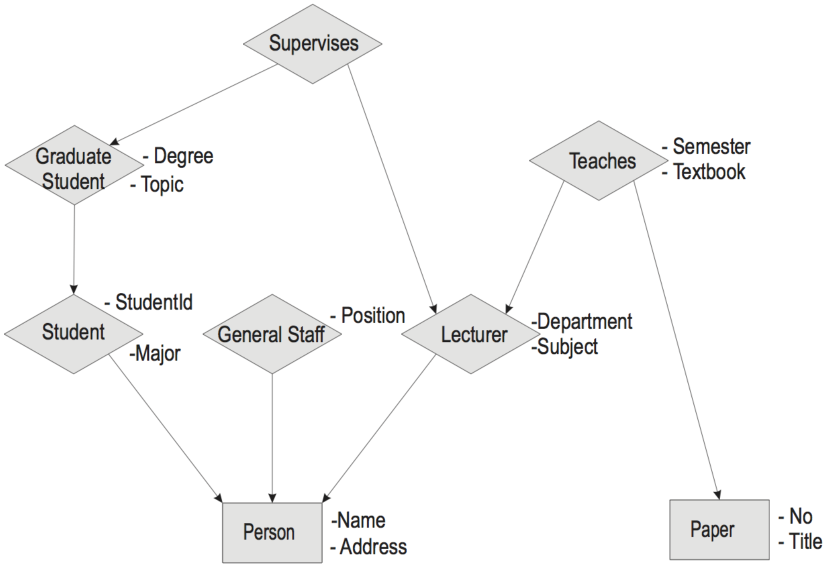
Specialisation

- Sometimes, an object in the target of the database can be represented by more than just a single abstract concept
- **Examples:**
  - Students are also persons
  - Graduate students are also students, and thus hey are also persons
- Abstract concepts can be derived from other abstract concepts, for example the **more specific** concept STUDENT from the **more general** concept PERSON
- This idea is known as **specialisation**:
  - the derived object type is a **subtype** of the more general **supertype**
  - the subtype inherits all features of the supertype, but often adds some additional properties
  - the subtype may be modelled as a unary relationship type whose single component is just the supertype (but may have additional attributes)
- **Note:** every object of the subtype gives rise to at most one object of supertype

An Example for a Specialisation Hierachy

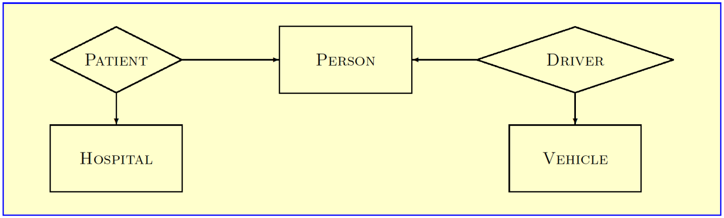


Same Example with further Relationship Types



An Example for Aggregation

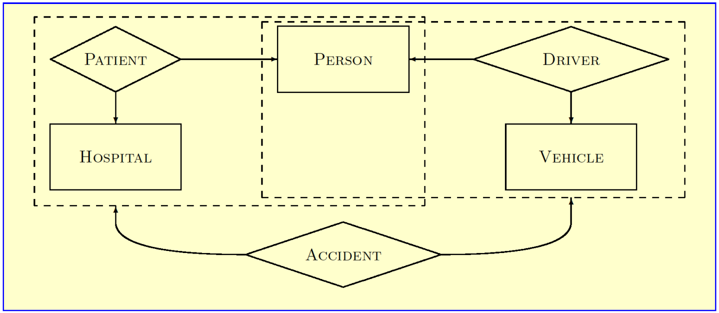
- The following ER diagram contains two relationship types PATIENT and DRIVER



- Suppose we want to model that a person is in hospital because she/he had an accident with a vehicle

An Example for Aggregation

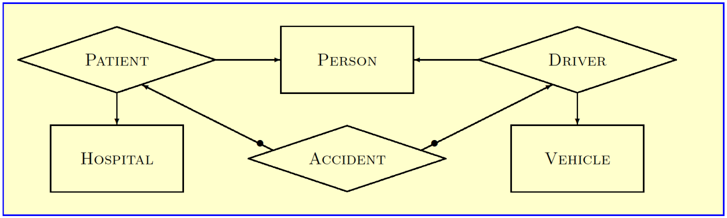
- Can use a new relationship type ACCIDENT to model these relationships
- But what are the components of ACCIDENT?



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

An Example of Relationship Types participating in Relationship Types

- In fact, we could simply 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 the following order:
    - 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
  - Recall: *In a (simple) ER diagram all relationship types are of order 1*
- A relationship type 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, for example to model aggregation
- For the sake of convenience, entity types and relationship types together 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

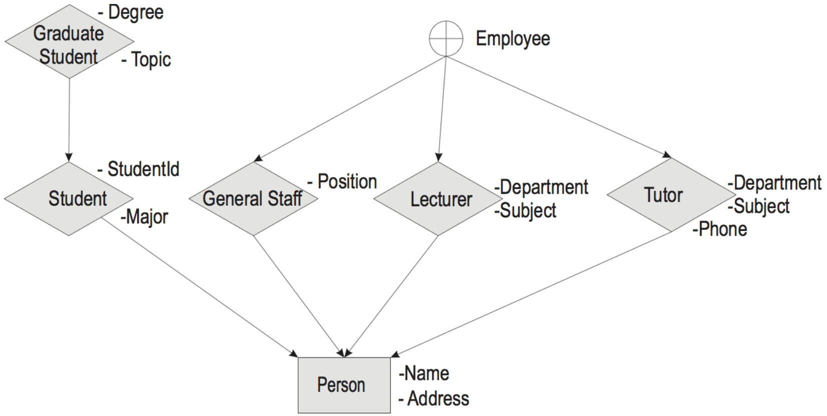
Cluster

- sometimes it is necessary to model **alternatives**, for example having a relationship to students *or* lecturers
- Abstract concepts can be obtained by generalization from a collection of abstract concepts

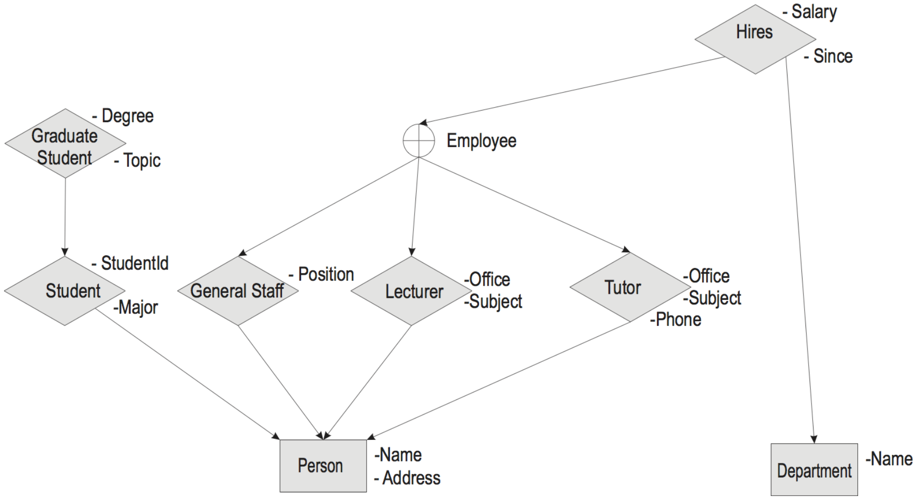
- A **cluster** C represents an alternative among a collection of object types
  - let C be an alternative between given object types $C_1, \ldots, C_m$
  - for simplicity, the cluster can be denoted as $C = C_1 \oplus \cdots \oplus C_m$
  - we call $C_1, \ldots, C_m$ the **components** of the cluster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

- **Example:** $\textsc{Employee} = \textsc{Lecturer} \oplus \textsc{Tutor} \oplus \textsc{General\_Staff}$
- 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 $k - 1$

An Example for a Cluster



An Example for a Cluster participating in a Relationship Type



Extended Entity-Relationship Schema and Diagram

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

- An **instance** $\mathcal{S}^t$ of an extended ER schema $\mathcal{S}$ assigns each object type O an object set O^t, such 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

- A **extended entity-relationship diagram** (**extended ER diagram**, for short) 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 $\oplus$

Critical Success Factors in Database Design

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

Transformation EER $\rightarrow$ Relational (1)

- we describe the transformation of an EER schema into a relational database schema
- start with entity types (object types of order 0) and then work one's way up gradually
- Transformation of **entity types**:
  - an entity type $E = (attr(E), key(E))$ leads to a relation schema E'
  - the attributes of E are preserved
  - the domain assignment for the attributes of E is preserved, too
  - the key of E becomes the primary key of the relation schema E'

Example - Transforming Entity Types

- three entity types:
  - $\textsc{Department} = (\{\text{No,Budget}\}, \{\text{No}\})$
  - $\textsc{Supplier} = (\{\text{No,Name,Address}\}, \{\text{No}\})$
  - $\textsc{Article} = (\{\text{No,Shortname,QuantityOnStock}\}, \{\text{No}\})$

- result in three relation schemata:
  - $\textsc{Department} = \{\text{No,Budget}\}$ with primary key $\{\text{No}\}$
  - $\textsc{Supplier} = \{\text{No,Name,Address}\}$ with primary key $\{\text{No}\}$
  - $\textsc{Article} = \{\text{No,Shortname,QuantityOnStock}\}$ with primary key $\{\text{No}\}$

Transformation EER $\rightarrow$ RDM (2)

- for a relationship type $R = (comp(R), attr(R), id(R))$ and each component $C \in comp(R)$ choose pairwise disjoint sets of new attribute names not occurring in $attr(R)$:

  $$k\_attr(C) = \{C.A \text{ where } A \text{ is a key attribute of } C'\}$$

  where C' is originating from a prior transformation of component C

- **Example:** we derive new sets of attribute names for entity types:
  - $k\_attr(\textsc{Department}) = \{\text{Department\_No}\}$
  - $k\_attr(\textsc{Supplier}) = \{\text{Supplier\_No}\}$
  - $k\_attr(\textsc{Article}) = \{\text{Article\_No}\}$

Transformation EER $\rightarrow$ RDM (3)

- Transformation of **relationship types**:
  - a relationship type $R = (comp(R), attr(R), id(R))$ leads to relation schema R' with

$$attr(R') = \bigcup_{C \in comp(R)} k\_attr(C) \cup attr(R)$$

  - the domain assignment of the attributes is preserved
  - R' has the primary key

$$\bigcup_{C \in key(R) \cap comp(R)} k\_attr(C) \cup (key(R) \cap attr(R))$$

  - each component $C \in comp(R)$ defines a foreign key

$$[C.A_1, \dots, C.A_n] \subseteq C'[A_1, \dots, A_n]$$

    on R' for $key(C) = \{A_1, \dots, A_n\}$

Example - Transforming Relationship Types

- Recall:
  - $k\_attr(\textsc{Department}) = \{\text{Department\_No}\}$
  - $k\_attr(\textsc{Supplier}) = \{\text{Supplier\_No}\}$
  - $k\_attr(\textsc{Article}) = \{\text{Article\_No}\}$

- relationship type
  $\textsc{DayOffer} = (\{\textsc{Supplier}, \textsc{Article}\}, \{\text{Date,Price}\}, \{\textsc{Supplier}, \textsc{Article}, \text{Date}\})$
  becomes:
  - $\textsc{DayOffer} = \{\text{Supplier\_No,Article\_No,Date,Price}\}$ with
  - primary key: $\{\text{Supplier\_No,Article\_No,Date}\}$
  - foreign keys: $[\text{Supplier\_No}] \subseteq \textsc{Supplier}[\text{No}]$
    $[\text{Article\_No}] \subseteq \textsc{Article}[\text{No}]$

Example - Transforming Relationship Types

- first we derive new sets of attribute names for components of relationship type $\textsc{Purchase}$:
  - $k\_attr(\textsc{DayOffer}) =$ {DayOffer_Supplier_No,DayOffer_Article_No,DayOffer_Date}
- then relationship type $\textsc{Purchase} =$ $(\{\textsc{Department}, \textsc{Day\_Offer}\}, \{\text{Quantity}\}, \{\textsc{Department}, \textsc{Day\_Offer}\})$ becomes:
  - $\textsc{Purchase} =$ {Department_No,DayOffer_Supplier_No,DayOffer_Article_No, DayOffer_Date,Quantity} with
  - primary key: {Department_No,DayOffer_Supplier_No, DayOffer_Article_No,DayOffer_Date}
  - foreign keys: [Department_No] $\subseteq$ $\textsc{Department}$[No]
    [DayOffer_Supplier_No,DayOffer_Article_No,DayOffer_Date]
    $\subseteq$ $\textsc{DayOffer}$[Supplier_No,Article_No,Date]

How to handle Clusters

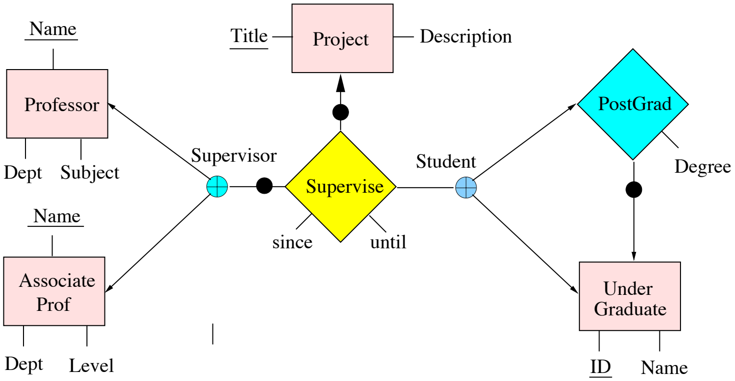
- cluster types used in conceptual design to model alternatives
- the relational data model does not provide similar concept
- transform EER schema with clusters into equivalent EER schema without clusters
- only necessary as pre-processing before actual transformation to the realational data model
- in general: clusters provide database designers a convenient way to model objects in the target of database
- do not recommend to avoid clusters as size of EER schema increases dramatically and becomes harder to comprehend

Transformation EER $\rightarrow$ RDM (3)

- replacing clusters in EER schema is straightforward:
  - cluster types in EER schema $\mathcal{S}$ that are not component of any relationship type can be removed from $\mathcal{S}$
  - consider relationship type R with cluster component $C = C_1 \oplus \cdots \oplus C_n$
  - replace R by n new relationship types $R_1, \ldots, R_n$:
  - for $i = 1, \ldots, n$: R_i obtained from R by replacing every occurrence of C by C_i
  - if R_i still contain clusters, then repeat process of replacing these clusters by its components
  - the final EER schema is cluster-free and previous transformation to RDM can be applied

Example - Transforming Clusters

- consider the following EER diagram:



- SUPERVISE=({SUPERVISOR, STUDENT, PROJECT},{since, until},{SUPERVISOR, PROJECT}) with
  - cluster SUPERVISOR = PROFESSOR $\oplus$ ASSOCIATEPROF
  - cluster STUDENT = POSTGRAD $\oplus$ UNDERGRADUATE

Example - Transforming Clusters

- using SUPERVISOR we obtain:
  - PROF_SUPERVISION:
    ({PROFESSOR, STUDENT, PROJECT},{since,until},{PROFESSOR, PROJECT})
  - APROF_SUPERVISION:
    ({ASSOCIATEPROF, STUDENT, PROJECT},{since,until},{ASSOCIATEPROF, PROJECT})
- using 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})