HY 351: Ανάλυση και Σχεδίαση Πληροφοριακών Συστημάτων
CS 351: Information Systems Analysis and Design

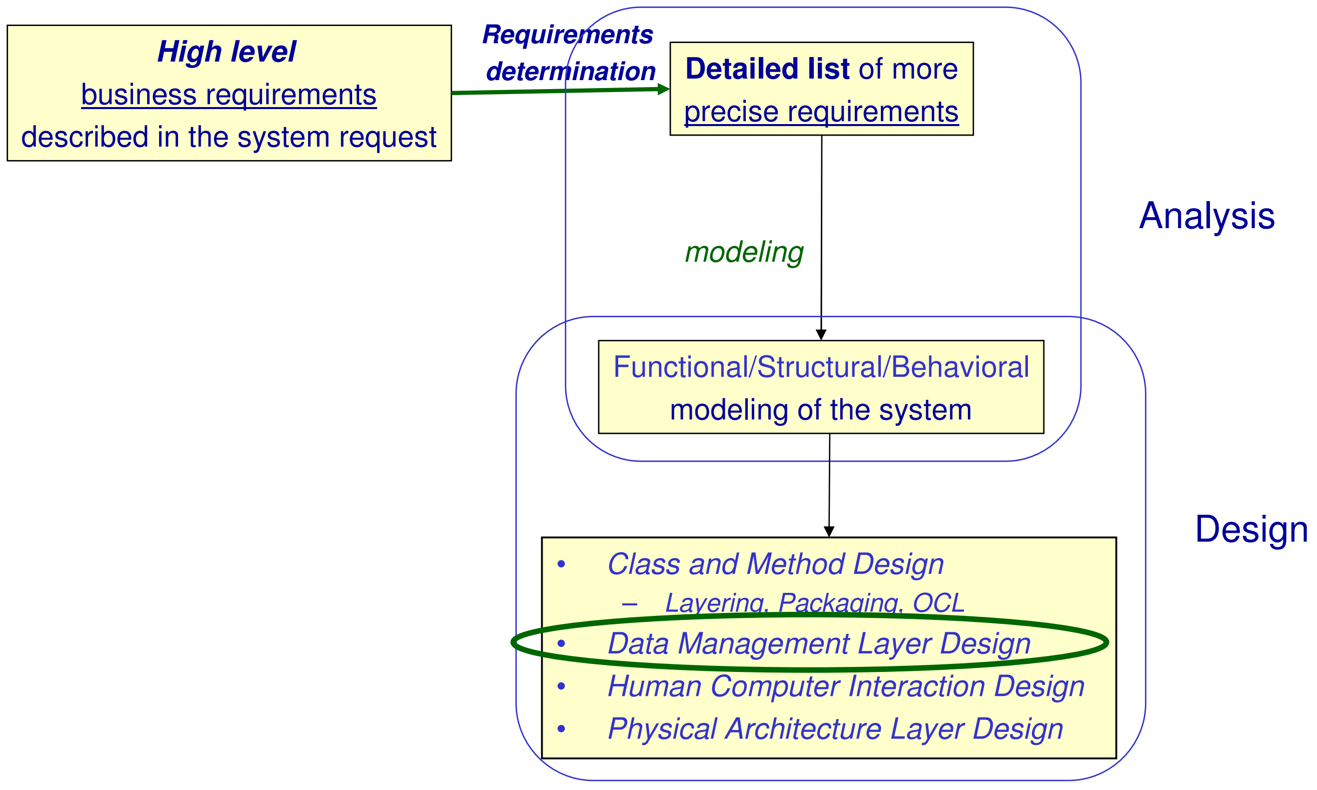
Data Management Layer Design (I)

.

Lecture : 15
Date : 1-12-2005

Yannis Tzitzikas
University of Crete, Fall 2005

Analysis and Design



High level
business requirements
described in the system request
Requirements determination
Detailed list of more
precise requirements
modeling
Functional/Structural/Behavioral
modeling of the system
Analysis
Design
• *Class and Method Design*
– *Layering, Packaging, OCL*
• *Data Management Layer Design*
• *Human Computer Interaction Design*
• *Physical Architecture Layer Design*

Outline

- Introduction
- Object-persistence formats
  - Files (Sequential and Random Access)
  - Databases (Relational, Object-Relational, Object-Oriented)

- Relational Databases,
- ER Model
- ER Model vs Class Diagrams
- ER Model => Relational Model

What is the Data Management Layer ?

The data management layer is about
how **data** is stored and handled by the programs that run the system

In an object-oriented system we are concerned with both **persistent objects** and **transient objects**.
- Persistence objects are those that must be stored using some kind of storage mechanism
- Transient objects will be erased from memory after they have been used.

How to design the data management layer ?

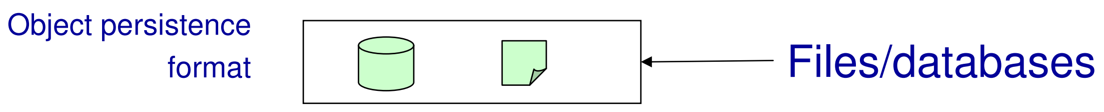
A design approach of 4 steps:
(A) Select the format of the storage
(B) Map problem domain objects to object-persistence formats
(C) Optimizing the object-persistence formats
(D) Design data access and manipulation classes

How to design the data management layer

(A) Select the format of the storage

There are four basic formats used for object persistence:

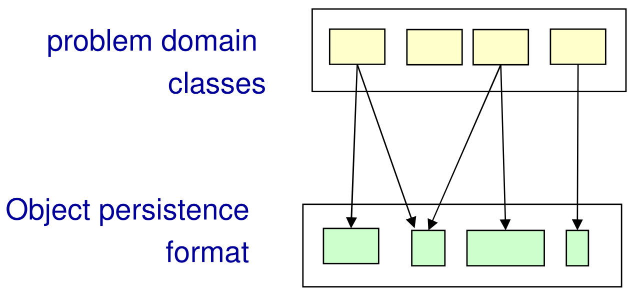
- files
- databases
  - relational database (RDB)
  - object-relational databases (ORDB)
  - object-oriented databases (OODB)



How to design the data management layer

(B) Map problem domain objects to object-persistence formats

- UML class diagrams define the data structures required by an application
- Some of these structures need to be persistently stored
- If we are going to use a DBMS we have to map these structures to data structures that are recognized by the database
- The latter depends on the underlying data model which can be relational, object-relational or object-oriented.



How to design the data management layer

(C) Optimizing the object-persistence formats

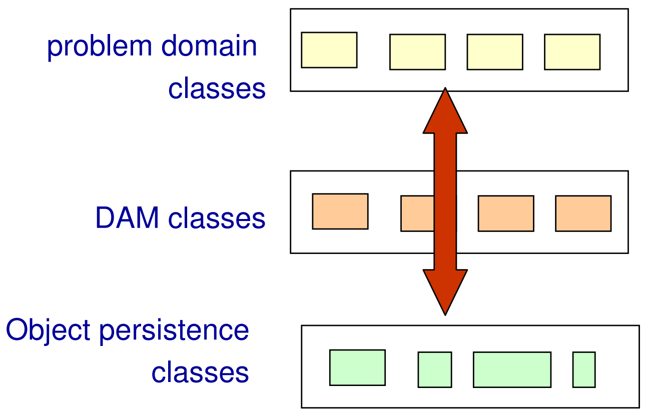
Dimensions of optimization:

- Storage efficiency (minimizing storage space)
- Speed of access (minimizing time to retrieve desired information)

How to design the data management layer

(D) Design data access and manipulation classes

- The **DAM (Data Access and Manipulation) classes** act as "<u>translators</u>" between the persistent objects and the problem domain objects.
- They should be able to read and write both persistent objects and problem domain objects.



(A) Select the format of the storage

Files

- Sequential Access
  - They allow sequential file operations (read, write, search)
  - Typically efficient for reports using all or most of the file’s data
  - Types
    - Unordered sequential files
    - Ordered sequential files
      - e.g. in ascending order by customer number
- Random Access
  - Data stored in unordered fashion
  - Typically efficient for finding individual records
  - However they do not support fast sequential accessing (e.g. report writing could be inefficient)

Most oo PLs support sequential and random access files as part of the language

e.g. FileInputStream, FileOutputStream, RandomAccessFile (in java.io package)

Moreover they offer mechanisms for converting objects into a form that can be written out to a file (serializing them) and for reading them back into memory from a file.

Types of Application Files

Master files
- store core information for the application (e.g. information about customers, orders, payments, etc)
- Usually new records are appended to these files

Transaction files
- store information that can be used to update the master file
- it can be destroyed after the update of the master file

Audit
- stores "before" and "after" images of data so that an audit can be performed if the integrity of data is questioned (e.g. in order to check who and when changed the salary of an employee)

History
- stores old information that is no longer used (old customers, old orders, etc)

Look-up
- contain static values, like the list of all countries, the list of all telephone codes of Greece. Typically used for validation purposes.

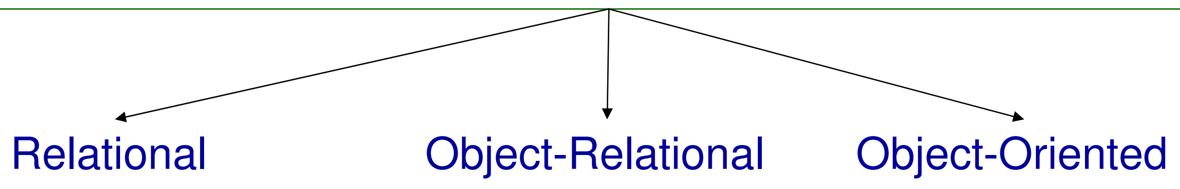
Configuration files and Backup files
- e.g. for localizing the system (so that labels, button captions and menu entries to be displayed in the language of the country where the system is being used)

Select the format of the storage
Databases

Databases

Basic functionalities offered by a DBMS

- Referential Integrity of data
- Query Language
- Concurrent access of data by large number of users and applications programs
- Transactions
- Authorization, Security
- Recovery
- ….



Relational, Object-Oriented and Object-Relational Databases

- Relational Databases
  - Based on the relational model (tables, 1NF, primary keys, foreign keys, relational algebra, SLQ, views, normalization)
  - Examples of relational DBMSs: *Sybase, DB2, Oracle, MySQL, MS Access (end-user DBMS)*
- Object-Relational Databases
  - Extend the relational model to include useful features from object-orientation, e.g. complex types.
  - Add constructs to relational query languages, e.g. SQL, to deal with these extensions
  - Example of ORDBMSs: *PostgreSQL*, *UniSQL, Oracle*
- Object-Oriented Databases
  - Extend OO programming to include features required for database system, e.g. persistent objects.
  - Examples of OODBMSs: *ObjectStore, Versant, Objectivity, O2,* Gemstone

IS Modeling vs Database Modeling

- Modeling an application program and modeling a database are sometimes disjoint activities
  - The former is done by application developers
  - The latter by database designers/administrators

However, we will see how from the application model (that we have already specified using UML), we can proceed and model the needed database.

Relational Databases

Relational Databases

- The relational model has been dominant for over 20 years
- It dominates in business information systems

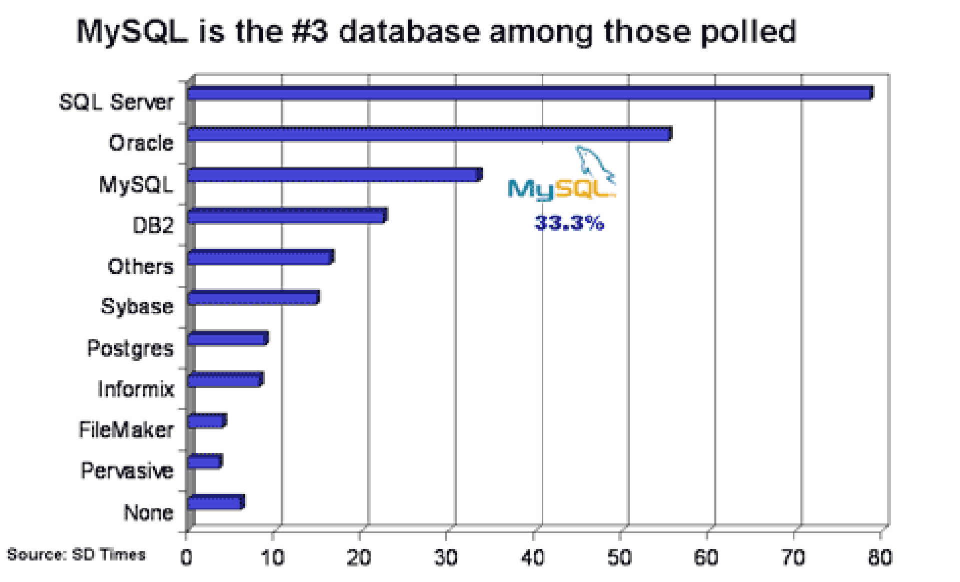
- It was standardized with SQL'92
- Modeling primitives
  - Tables consist of columns and rows
  - Cells can only contain values of atomic types (1NF)
    - object types, structured types, collections and references are not supported
    - references between tables are maintained by comparing values in columns

Key notions:

- 1NF, Primary key, Foreign key, Structured Query Language (SQL), Functional Dependency, Normalization.

Relational DBMSs

- "Relational Databases Rule the Roost" published in SD Times in July 2004:



Tables

- A relational table is defined by a fixed set of columns
- Columns have built-in or user-defined types (i.e. domains)
- Tables can have any number of rows (tuples)
- There are no duplicate rows in a table
- A column value may be allowed to be NULL
- Every table has a primary key
  - A key is a minimal set of columns such that the values in these columns uniquely identify a single row in a table
  - A table can have many such keys
  - One of these selected by the user is the primary key (the rest are called candidate or alternate keys)

Domains and Rules

- A domain defines the legal set of values that a column can take
  - It can be anonymous: e.g. gender char(1)
  - It can be named, e.g. gender Gender
    - create domain Gender char(1);
  - A named domain can be used in the definition of many columns in different tables
    - Changes to the domain definition are automatically reflected in column definitions

- Columns and domains can have “business rules” that constrain them
  - Default value (e.g. if no value is provided for city, assume “Heraklio”)
  - Range of values (e.g. range of ages: 10-90)
  - List of values (e.g. the allowed color is “green”, “yellow”, “red”)
  - Case of value (e.g. the value must be in lowercase)
  - Format of value (e.g. the value must start with the letter “F”)

A diagrammatic technique for Tables and its definition in SQL

Employee			
emp_id	CHAR(7)	<pk>	not null
family_name	VARCHAR(30)	<ak>	not null
first_name	VARCHAR(20)		not null
date_of_birth	DATE	<ak>	not null
gender	Gender		not null
phone1	VARCHAR(12)		null
phone2	VARCHAR(12)		null
salary	DEC(8,2)		null

```
--=======================================
-- Domain: “Gender”
--=======================================
create distinct type “Gender” as CHAR(1) with
      comparisons;
--=======================================
-- Table “Employee”
--=======================================
create table “Employee” (
        “emp_id “          CHAR(7)              not null,
        “ family_name“  VARCHAR(30)        not null,
        “ first_name “     VARCHAR(20)        not null,
        “ date_of_birth“  DATE                     not null,
        “ gender“          “Gender”               not null
             constraint “C_gender” check (“gender” in
             (‘F’,’M’,’f’,’m’)),
        “ phone1“                         VARCHAR(12),
        “ phone2“                         VARCHAR(12),
        “ salary “                          DEC(8,2),
primary key (“emp_id”),
unique (“date_of_birth”, “family_name”)
);
```

Referential Integrity

- A foreign key is defined as a set of columns in one table whose values are either NULL or are required to match the values of the primary key in the same or another table.
- This primary-to-foreign key correspondence is called the referential integrity.

Employee			
emp_id	CHAR(7)	<pk>	not null
family_name	VARCHAR(30)	<ak>	not null
first_name	VARCHAR(20)		not null
date_of_birth	DATE	<ak>	not null
gender	Gender		not null
phone1	VARCHAR(12)		null
phone2	VARCHAR(12)		null
salary	DEC(8,2)		null
dept_id	SMALLINT	**<fk>**	null

Department			
dep_id	SMALLINT	<pk>	not null
dept_name	VARCHAR(50)	<ak>	not null
address	VARCHAR(120)		null

dept_id=dept_id

Upd(R);Del(N)

```
alter table “Employee”
      add foreign key “RefToDepartment” (“dept_id”)
                references “Department” (“dept_id”)
                on delete set null;
```

Referential Integrity

What should happen if a department row is updated or deleted?
Specifically, if dept_id is updated or when a row of department is deleted ?

Declarative referential integrity constraints associated with delete and update operations

- Upd(R); Del(R)
  - **Restrict** the update or delete information
    - Here: do not allow this operation if there are tuples of Employee linked to that department
- Upd(C);Del(C)
  - **Cascade** the operation
    - Here: update or delete all linked employees rows
- Upd(N);Del(N)
  - Set **null**
    - Here: set dept_id of the linked Employee rows to NULL
- Upd(D); Del(D)
  - Set **default**
    - Here: set dept_id of the linked Employee rows to the default value

Triggers

- Declarative referential integrity constraints allow only simple business rules to be recorded.
- A more expressive solution is **triggers** (standardized in SQL:1999)
- A trigger is a small program (e.g. written in an extended SQL) that is executed automatically (triggered) as a result of a modification operation on a table on which the trigger has been defined.
  - A modification can be any of the SQL modification statements: **insert**, **update**, or **delete**.
- A trigger can be used to implement business rules
  - E.g. updates are not allowed in weekends
  - After deleting a department all deptIds of the Employee rows (that have the deleted deptId) should be set to Null.

Example of Trigger (Sybase)

Internal table

```
create trigger keepdpt
    on Department
    for delete
    as
    if @@rowcount = 0
          return /* avoid firing trigger if no rows affected */
    if exists
          (select * from Employee, deleted where Employee.dept_id = deleted.dept_id)
          begin
                    print 'Test for RESTRICT DELETE failed. No deletion'
                    rollback transaction
                    return
          end
          return
go
```

This trigger implements the Del(R) declarative constraint

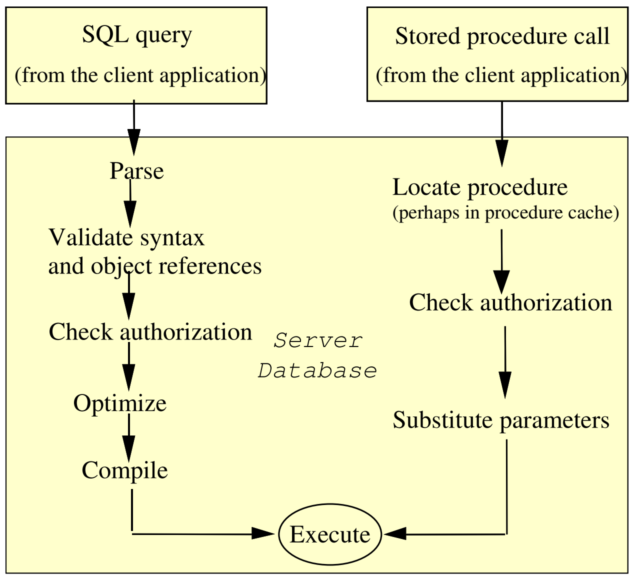
Stored Procedures

- A stored procedure is given a name, can have input and output parameters, and it is compiled and stored in the database.
- It is written in an extended SQL that allows variables, loops, branches, and assignment statements
- Stored procedures turn a database into an active programming system.
  - Stored procedures (first introduced by Sybase now part of every major commercial DBMS)
- Triggers are a special kind of stored procedures
  - They trigger themselves on insert, update and delete events on a table, and cannot be otherwise called.
  - So for each table we can have at most 3 triggers, while we can have unlimited number of stored procedures.

Stored Procedures (II)

A client program can call a stored procedure rather than sending a complete query to the server.

- Sending a query requires parsing it and checking its syntax (at the server side)
- Stored procedures are more efficient (less network traffic, parsing and compilation steps are done only once)
- A stored procedure can be exploited by many clients



SQL

- See HY360

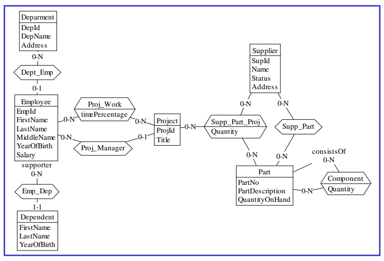
Views

- Is a stored and named SQL query
- This is a very useful feature for
  - Providing different perspectives of the data
  - For database security (restring users to the contents of certain views)
  - For alleviating the query formulation effort (SQL queries that use views instead of tables)

The traditional way to design a relational database is to start from the Entity-Relationship model.

Below we will review ER model and we will compare it with UML class diagrams

The Entity-Relationship Model



- Introduction
- The Entity-Relationship model
  - – Entities, Relationships, Attributes,Generalization
- ER diagrams vs UML Class Diagrams
- Conceptual Database Design (ER Design)
  - – Documentation for ER Diagrams
    - business rules, data dictionary
- ER model => Relational model

The Entity Relationship Model

- The **Entity Relationship (ER)** model is a conceptual model for describing the data requirements for a new information system in direct and easy to understand *graphical notation.*
- It views the real world as entities and relationships.
- A basic component of the model is the Entity-Relationship diagram which is used to visually represents data objects.

- ER Model History
  - – The Entity-Relationship (ER) model was originally proposed by Peter in 1976 [Chen76] as a way to unify the network and relational database views.
  - – Since Chen wrote his paper the model has been extended and today it is commonly used for database design

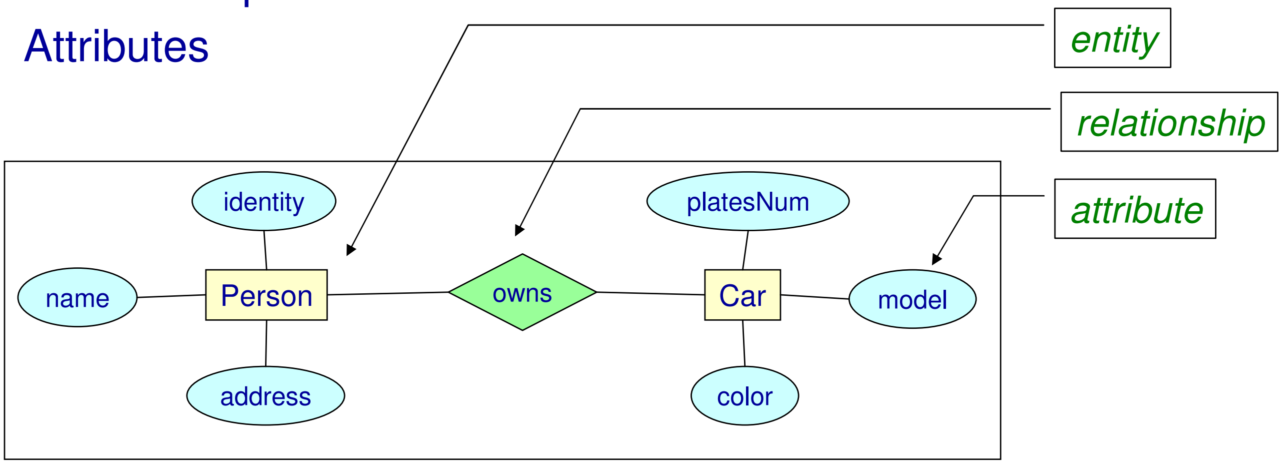
The Utility of the ER model

For the database designer, the utility of the ER model is:

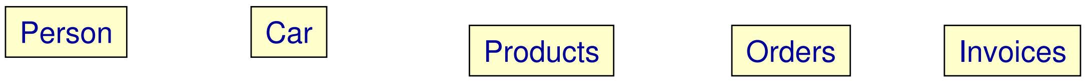
- it maps well to the relational model. The constructs used in the ER model can easily be transformed into relational tables.
- it is simple and easy to understand with a minimum of training. Therefore, the model can be used by the database designer to communicate the design to the end user.
- In addition, the model can be used as a design plan by the database developer to implement a data model in a specific database management software.

Basic Constructs of ER Model

- Entities
- Relationships
- Attributes

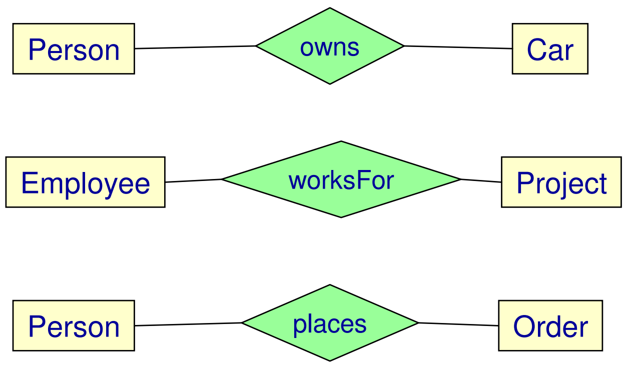


Entities



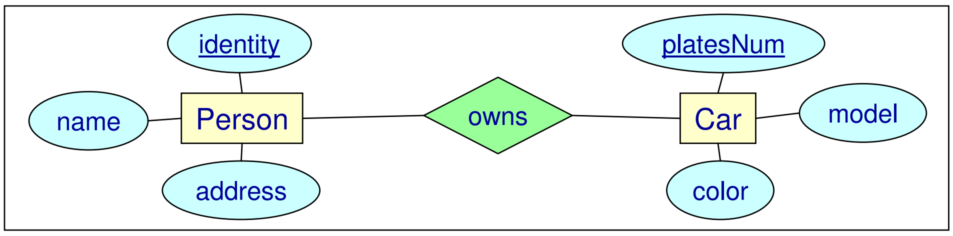
- *Entities* are the principal data object about which information is to be collected.
- They are usually recognizable concepts, either concrete or abstract, such as person, places, things, or events which have relevance to the database.
- An *entity occurrence* (also called an instance) is an individual occurrence of an entity. An occurrence is analogous to a row in the relational table.

Relationships



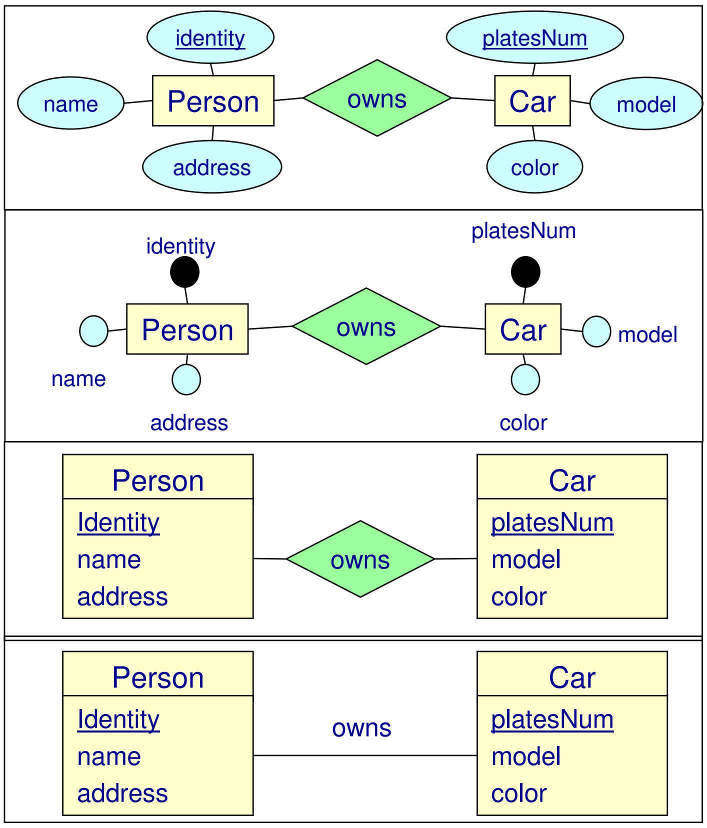
A Relationship represents an association between two or more entities.

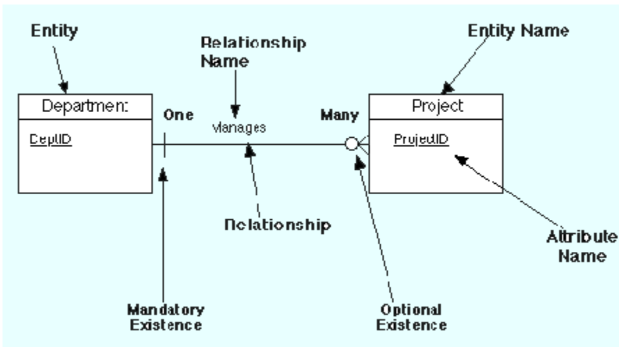
Attributes



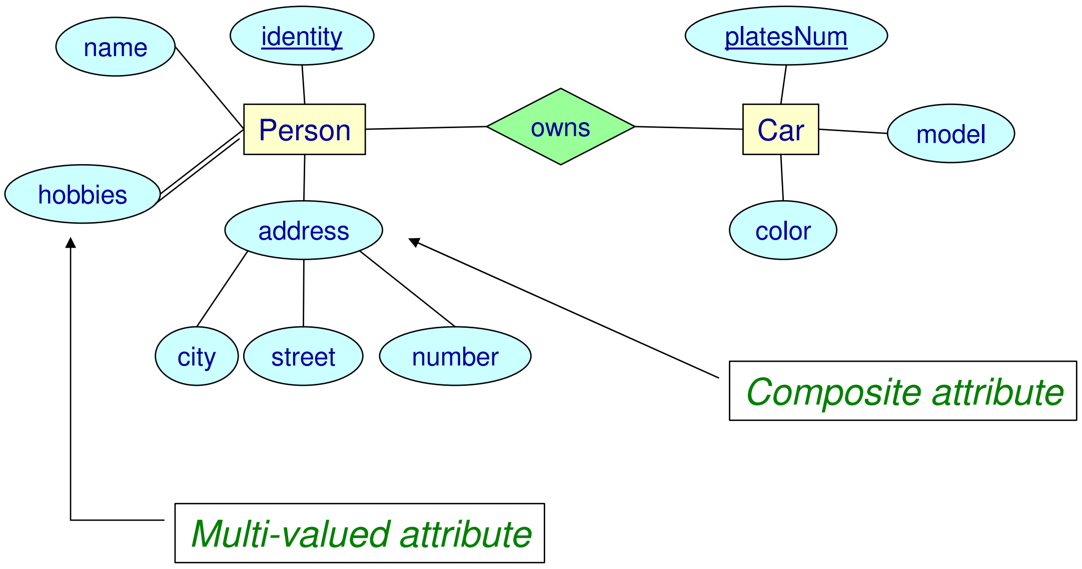
- *Attributes* describe the entity of which they are associated.
- A particular instance of an attribute is a *value*, e.g. «Yannis» is one value of the attribute Name.
- The *domain* of an attribute is the collection of all possible values an attribute can have. The domain of Name is a character string.
- Attributes can be classified as:
  - **identifiers**
    - An identifier (more commonly called *key)*, uniquely identifies an instance of an entity. We underline them in diagrams
  - **descriptors**
    - A descriptor describes a non-unique characteristic of an entity instance.

Different notations for ER diagrams

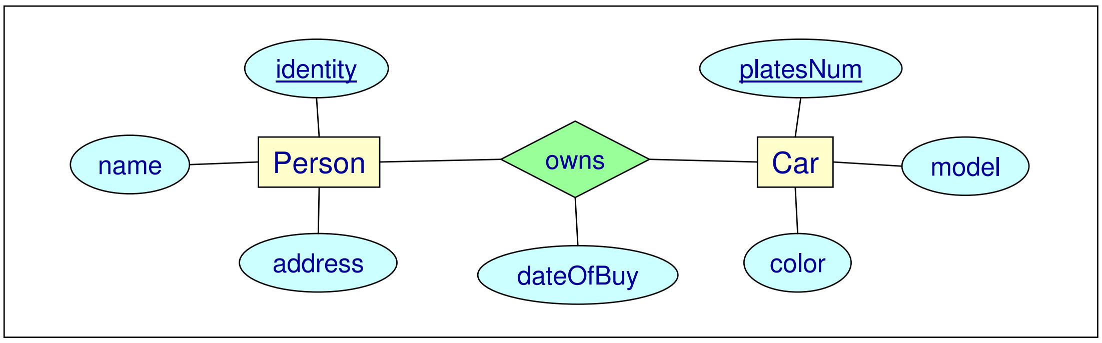




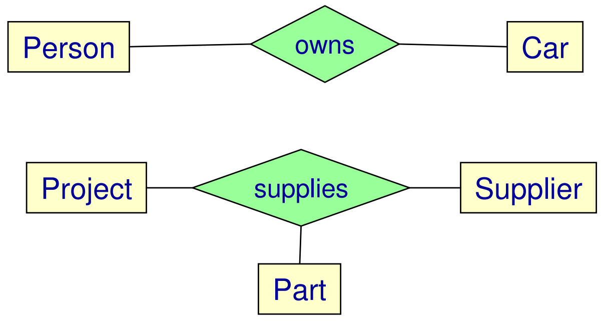
Composite and Multi-valued Attributes



Relationships can also have Attributes

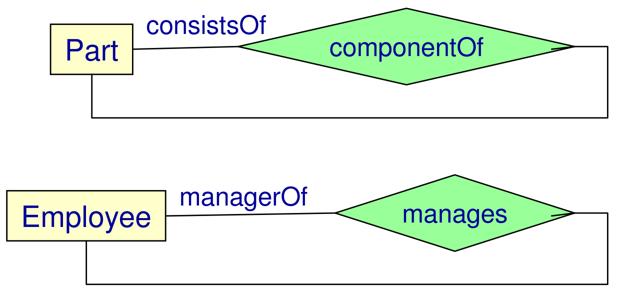


Degree of a relationship



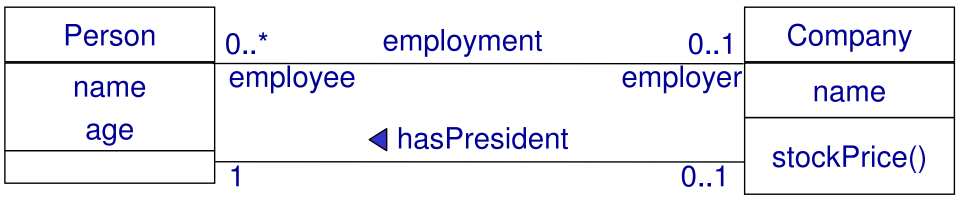
degree 2: binary relationship

degree 3: ternary relationship



Recursive relationships

Reminder: Multiplicity Constraints of Class Diagrams



- Multiplicity constraints
  - how many objects may participate in a given relationship?
  - multiplicity indicates lower & upper bounds

*****	≡	**0..***	≡ **0..** ∞	// no constraint
1	≡	**1..1**		// mandatory and single-valued association
0..1				// optional single-valued association
1..*				// mandatory and multi-valued association

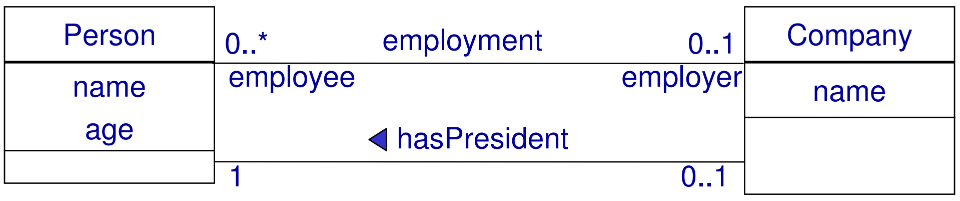
  - other more general multiplicity constraints

    1..11 (for soccer teams)

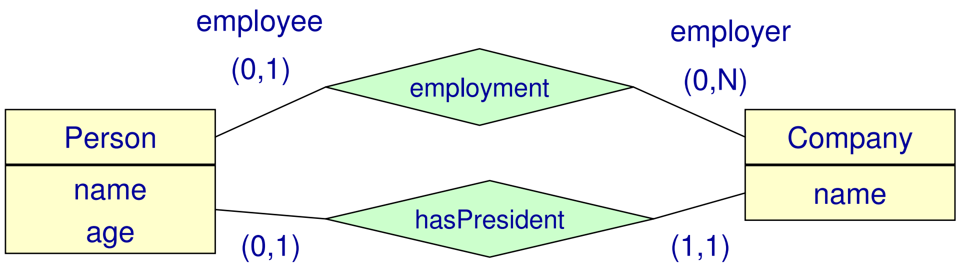
    3..4 (wheels of a car)

Multiplicity (or Cardinality) Constraints (ER vs UML)

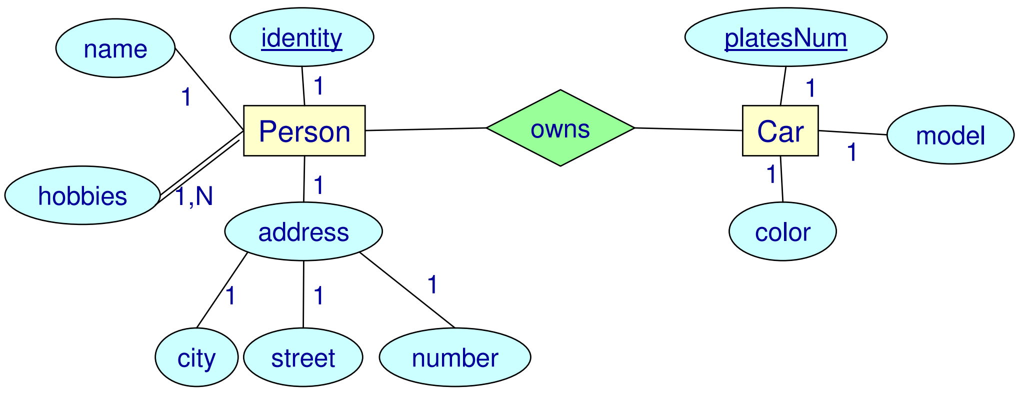
UML



ER

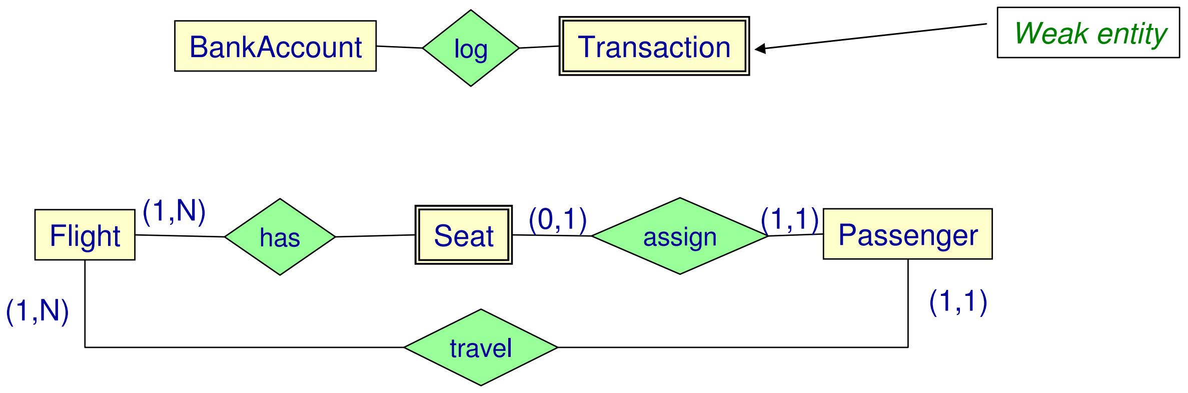


Attributes and Cardinalities



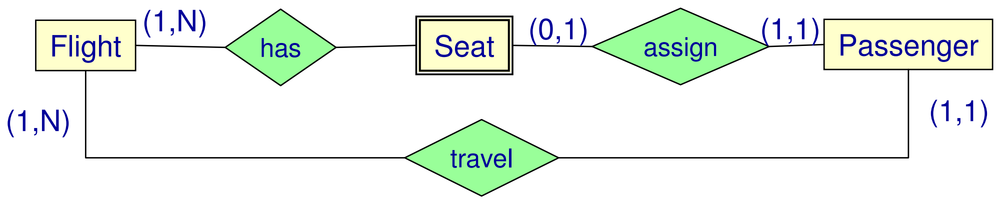
Default cardinality for attributes: (1,1)

Strong and Weak Entities (Independent and Dependent Entities)



- *Strong (*or *Independent)* entity
  - does not rely on another entity for identification.
- *Weak* (or *dependent)* entity
  - relies on another entity (which it is related though a relationship) for identification.

Strong and Weak Entities

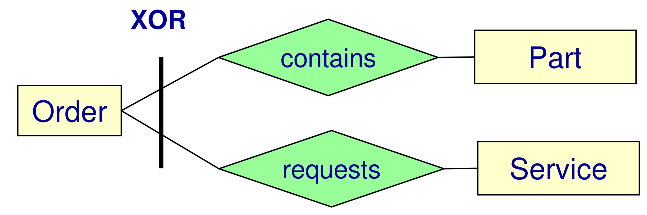


Weak entities

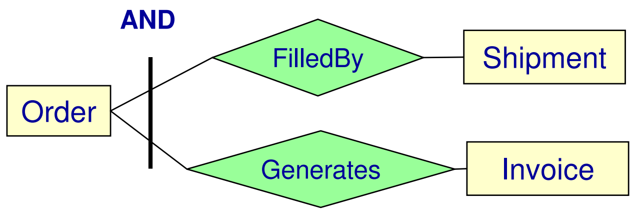
- they do not have their own identifiers
- they can only have *partial identifiers*, ı.e. attributes that can identify the instances of the weak entity that are associated with the same instance of the strong entity (the strong entity is called the «owner» of these instances)

- The identifiers of a weak entity are formed by the identifiers of the strong entity plus the partial identifiers of the weak entity
- A weak entity can be the owner of other weak entities
- A weak entity can be associated with more than one strong entities (through different relationships)

AND/XOR for Relationships

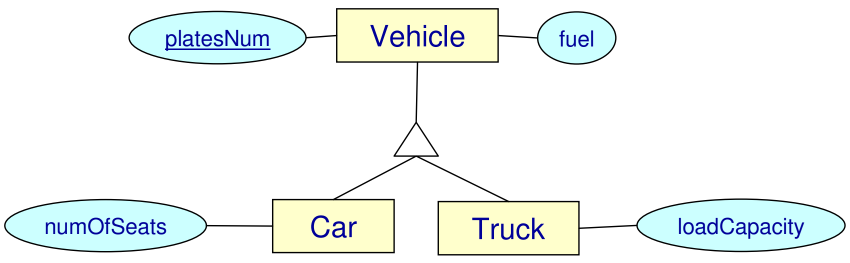


- Orders either order a part, or request a service. Not both



- For any given order, whenever there is at least one invoice there is also at least one shipment and vice versa.

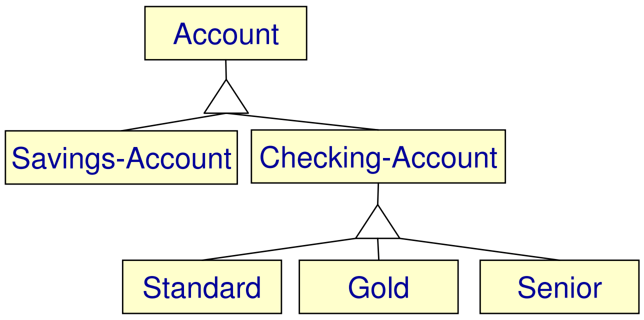
Generalization (or specialization) Hierarchies



- Generalization occurs when two or more entities represent categories of the same real-world object
- A generalization hierarchy is a form of abstraction that specifies that two or more entities that share common attributes can be generalized into a higher level entity type called a *supertype* or *generic* entity.
- The lower-level of entities become the *subtype* to the supertype.
- Subtypes are *dependent entities*.

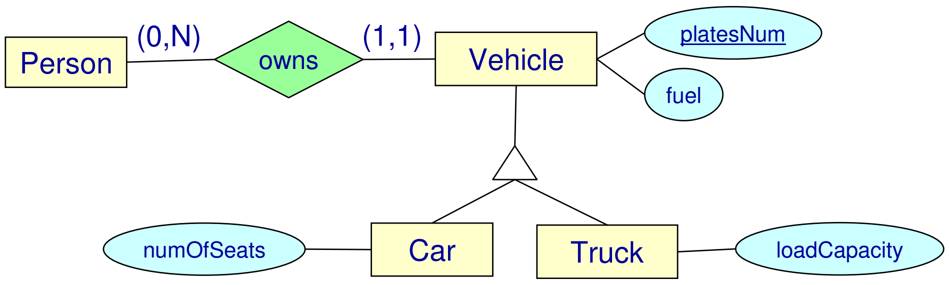
(Specialization is the dual counterpart of generalization)

Generalization Hierarchies



- Generalization hierarchies can be nested. That is, a subtype of one hierarchy can be a supertype of another. The level of nesting is limited only by the constraint of simplicity.

Generalization and Inheritance

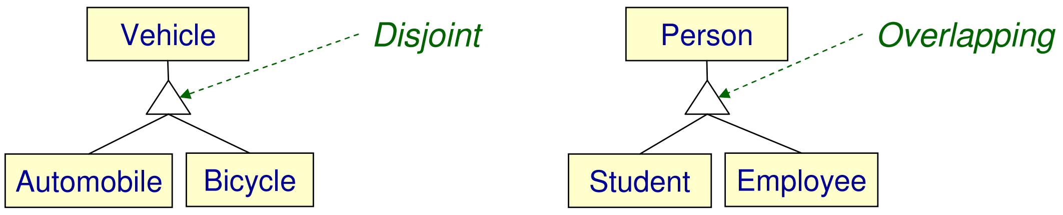


- The subtypes inherit
  - attributes
  - participation in relationship types (with the same cardinality constraints)

Disjoint and Overlapping Subtypes

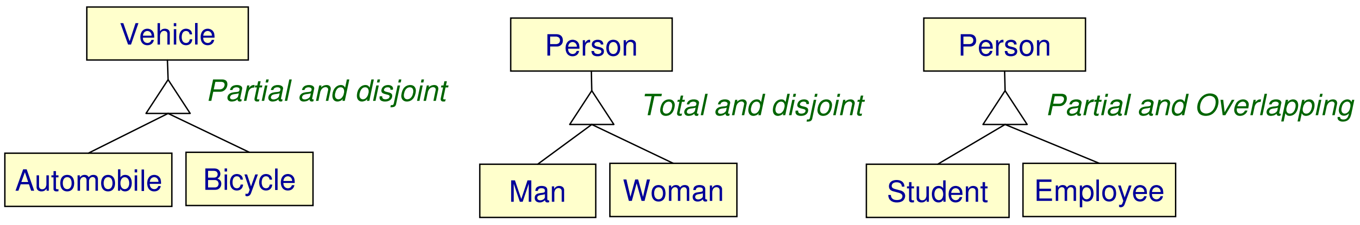
Subtypes can be either **mutually exclusive** (or *disjoint*), or **overlapping** (or *inclusive*).

- A mutually exclusive category is when an entity instance can be in only one category.
  - A vehicle cannot be automobile and bicycle.
- An overlapping category is when an entity instance may be in two or more subtypes.
  - A student can also be an employee.

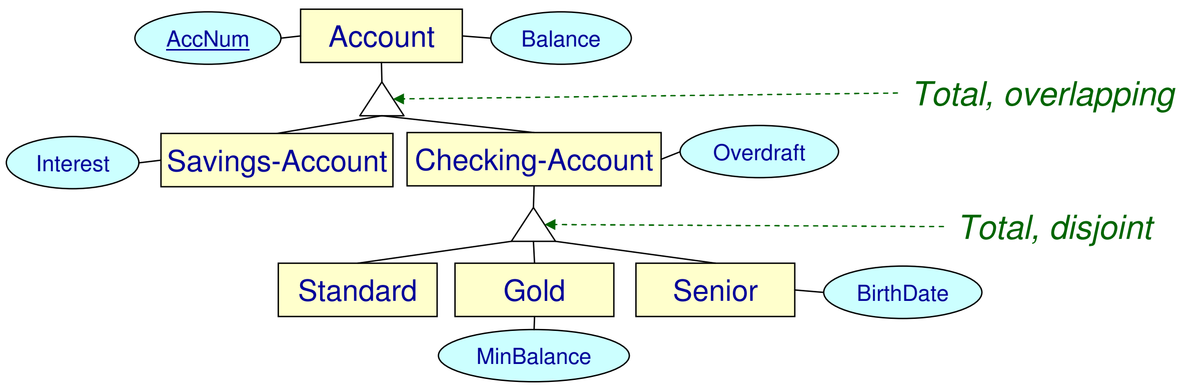


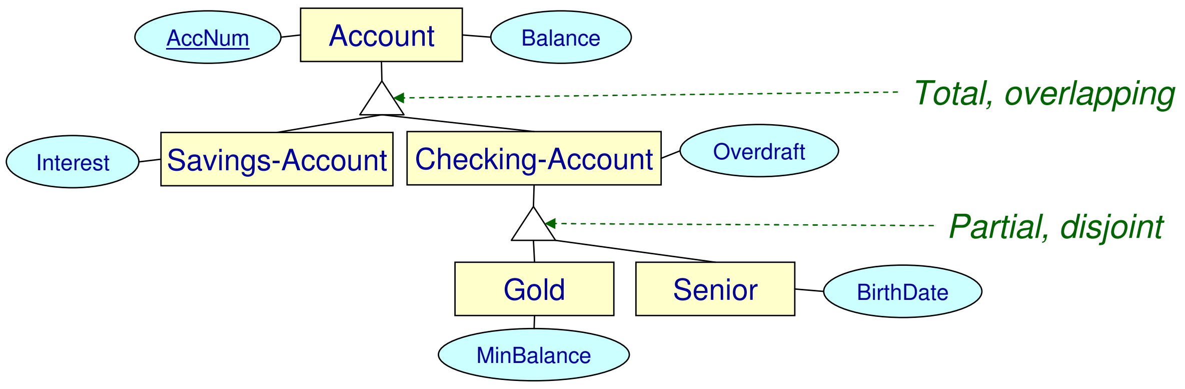
Partial vs Total Subtypes

- Partial
  - an entity occurrence may not necessarily belong to one subtype
- Total
  - every entity occurrence should belong to one subtype

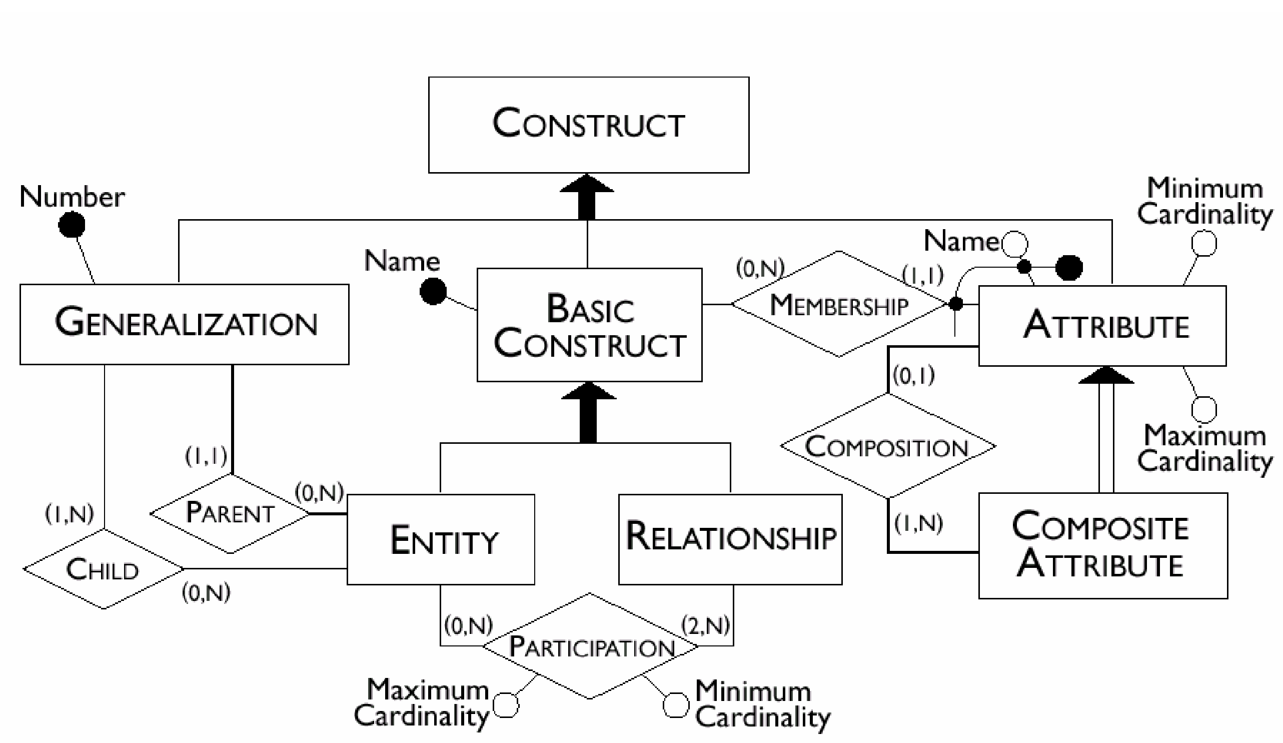


Example





The ER metamodel (as an E-R Diagram)



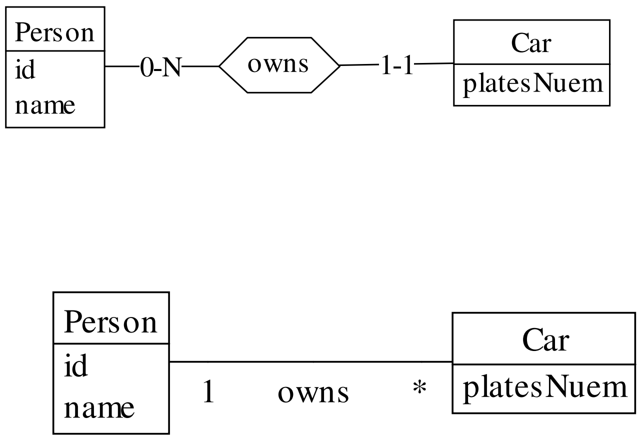
ER Diagrams vs UML Class Diagrams

What is the difference between ER diagrams and UML Class Diagrams?

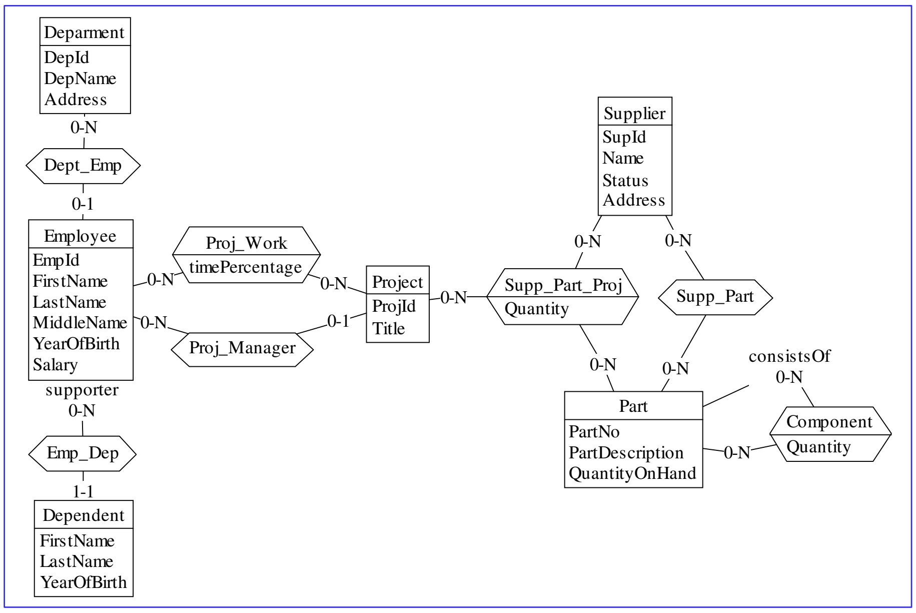
- Class Diagrams are a superset of ER diagrams.
- ER diagrams focus only on data, while Class Diagrams go a step further by allowing modeling the behaviour.
  - In the context of database design, these logical operations can be turned into triggers or stored procedures.

- ER diagrams allow N-ary (N>2) relationships
  - Class Diagrams mainly comprise binary but n-ary could be used too
- ER diagrams allow the specification of identifiers
  - class diagrams do not
    - we could however use a stereotype or tagged values to indicate them
- Class diagrams allow dynamic classification
  - ER diagrams do not
- Class diagrams can have methods and constraints (e.g. pre/post-conditions expressed in OCL)
  - ER diagrams do not

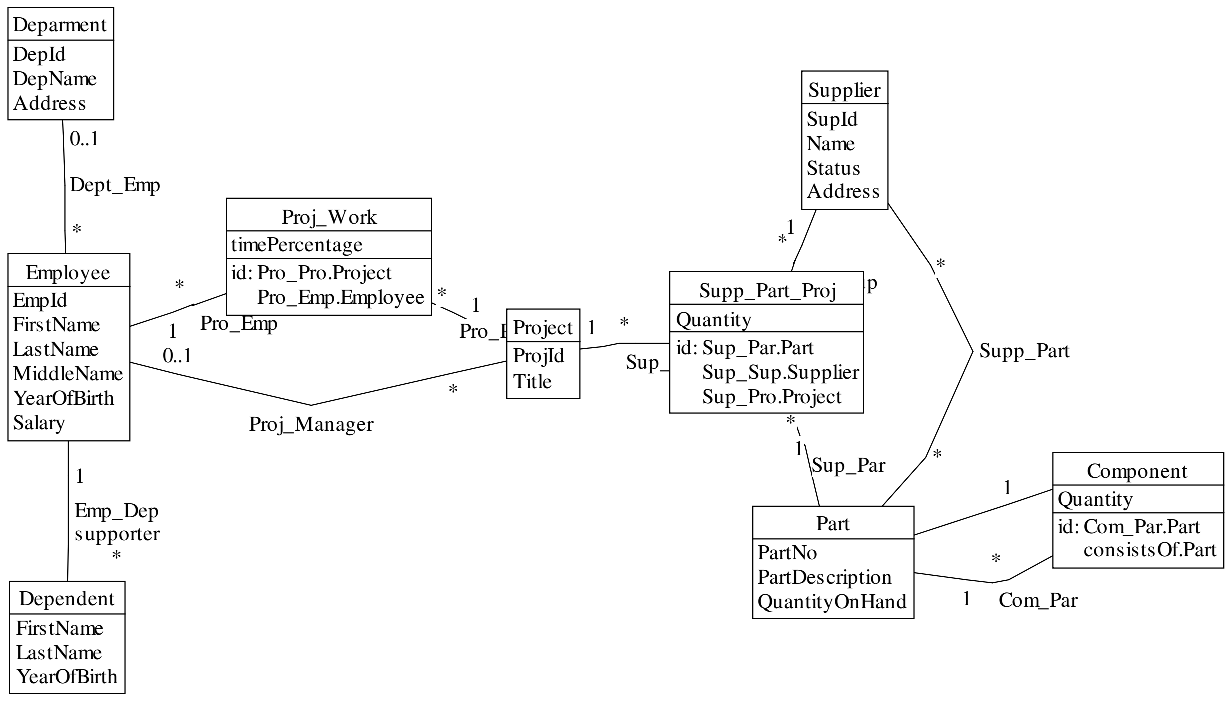
ER vs UML: Notations



The first ER Diagram



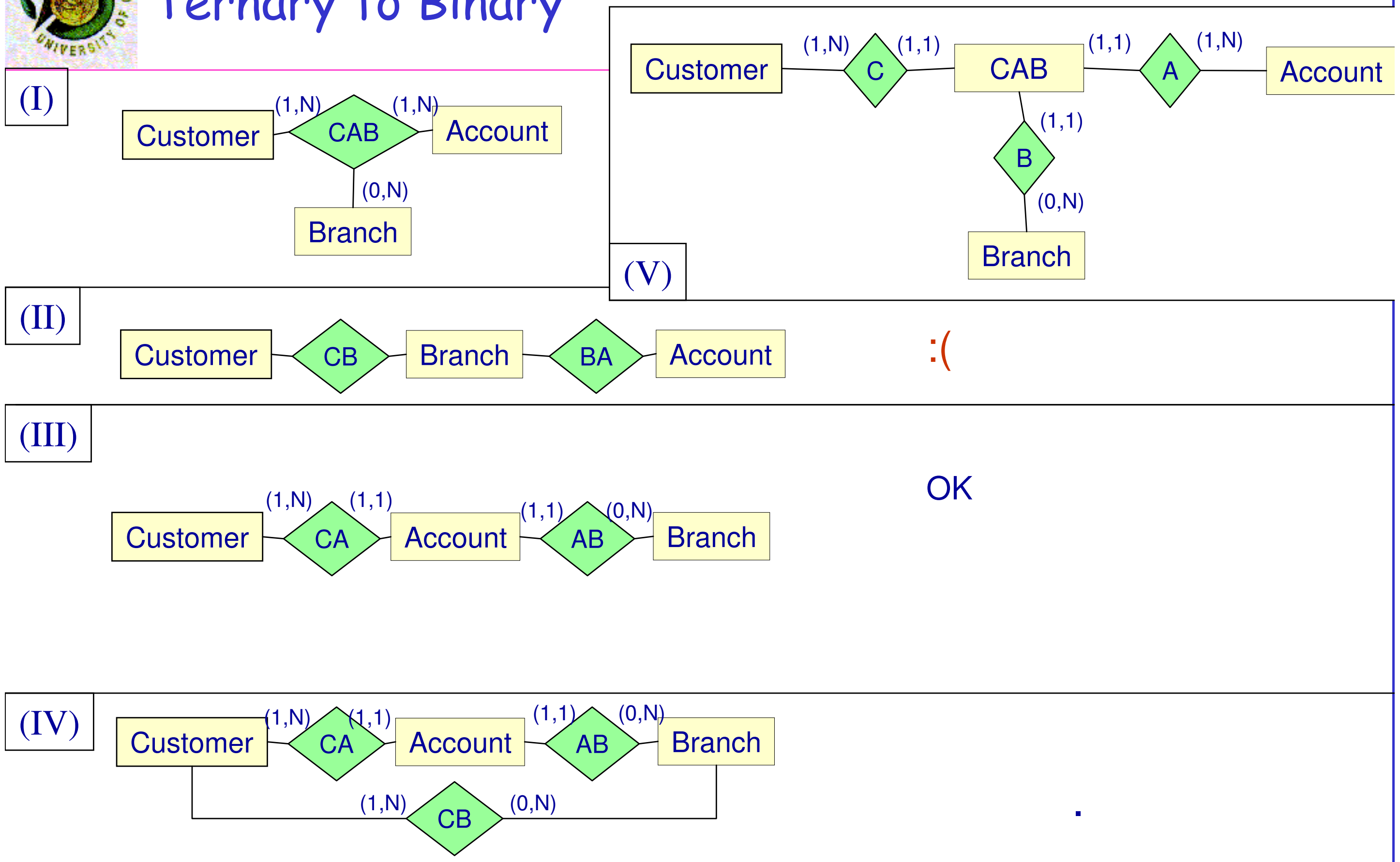
The first ER Diagram as UML Class Diagram



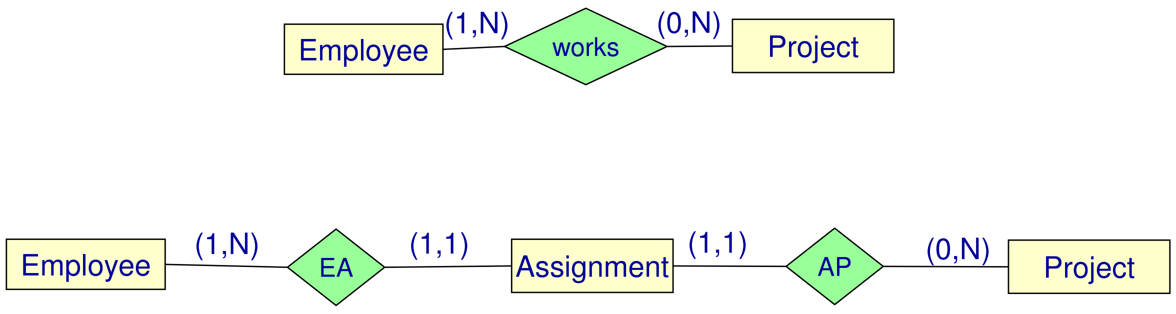
As translated automatically by DB-MAIN

Some ER Transformations

Ternary to Binary

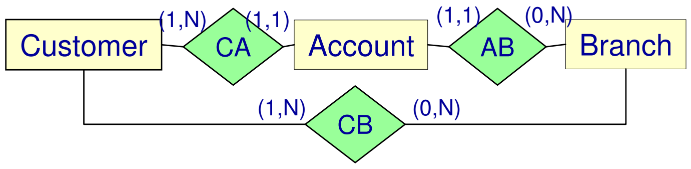


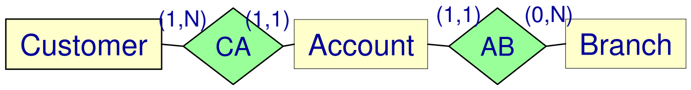
Resolving Many-to-Many Relationships



Eliminate redundant relationships

- A redundant relationship is a relationship between two entities that is equivalent in meaning to another relationship between those same two entities that may pass through an intermediate entity.





Conceptual Database Design

Conceptual Database Design (ER Diagram Design)

Questions

- What are the *entities* and *relationships* in the enterprise?
- What information about these entities and relationships should we store in the database?
- What are the *integrity constraints* or *business rules* that hold?

- There is no standard process for doing so.
- Some methodologies propose a staged development process
  - first model entities and relationships
  - then key attributes,
  - finally non-key attributes
- Other experts argue that in practice, using a phased approach is impractical because it requires too many meetings with the end-users

The OO Analysis and Design methodology (on which this course focuses) has given us one (use cases-> reqs gathering and determination, domain class diagrams,...)

Documentation of an ER Diagram

In many cases the diagram is not enough

We complement it with

- documentation that describes the properties of the data that cannot be expressed using the constructs of the model
- A widely-used documentation concept for conceptual schemas is the business rule.

A ***business rule*** can be:

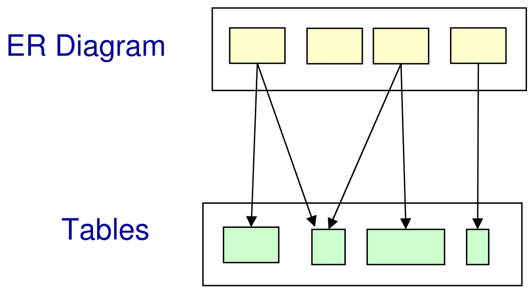
- the description of a concept relevant to the application (also known as a business object)
- an integrity constraint on the data of the application
- a derivation rule, whereby information can be derived from other information

The Data Dictionary

- Comprises two tables: the first describes the entities; the second the relationships
- Business rules that describe constraints
  - **<concept> must/mustnot <expression on concepts>**
- Business rules that describe derivations
  - **<concept> is obtained by <operation on concepts>**

We have already seen OCL which is a formal language for expressing all these.

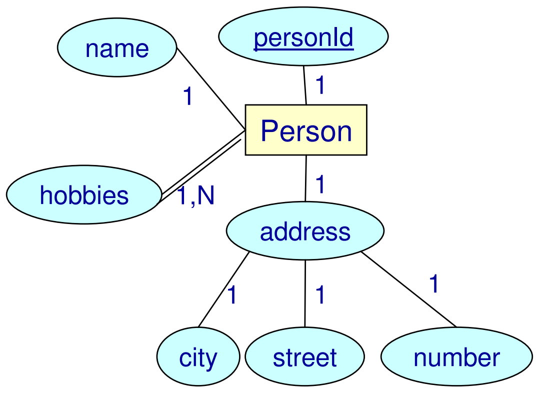
From an ER model to a Relational Database Schema



Entities and Relationships must be converted so they can be stored in tables

ER=>Relational (1)

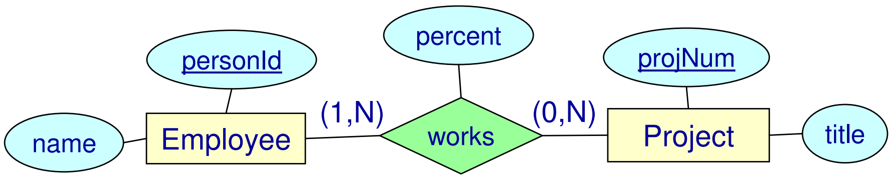
- Entity E => Table T
  - single-valued attributes of E => attributes of T
  - identifier attributes => candidate keys of T
- A multi-valued attribute of E => Table T
  - attributes of T: identifier of E plus the multi-valued attribute



Person(<u>personId</u>, name, city, street,number)
Hobbies(<u>personId</u>, hobby)

ER=>Relational (2)

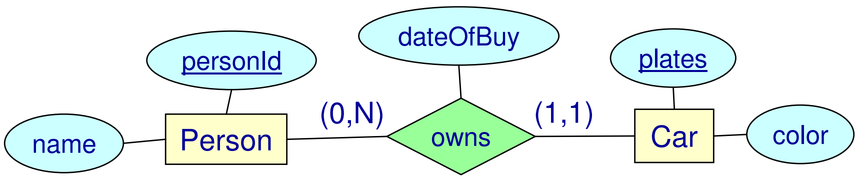
- M-N Relationship R => Table T
  - T contains all identifier attributes of the entities that participate in R
  - T also contains the attributes of R



Employee(<u>personId</u>, name)
Project(<u>projNum</u>, title)
Works(<u>personId</u>, <u>projNum</u>,percent)

ER=>Relational (3)

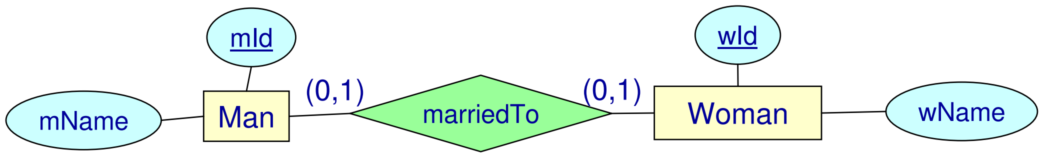
- N-1 Relationship between E1 and E2 => no new table
  - we add to the table of E2 the key of E1 (foreign key)
  - if the participation of E2 is (1,1) and not (1,0) then this attribute cannot have null values



Person(<u>personId</u>, name)
Car(<u>plates</u>, color, personId, dateOfBuy) // personId: foreign key

ER=>Relational (4)

- 1-1 Relationship between E1 and E2 => no new table
  - If (0,1) (0,1) we add to one of the tables that correspond to E1 or E2 the key of the other.



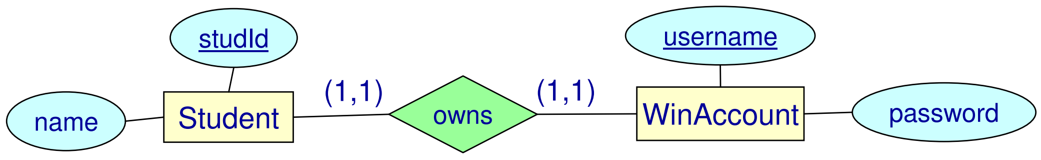
Man(<u>mId</u>, mName,wId) // wId: foreigh key
Woman(<u>wId</u>, wName)

Man(<u>mId</u>, mName)
Woman(<u>wId</u>, wName, mId) // wId: foreigh key

Equivalent alternatives

ER=>Relational (5)

- 1-1 Relationship between E1 and E2 => no new table
  - if (1,1)(1,1) then both tables can be combined into one



Student(<u>studId</u>, name, username, password)

WinAccount(<u>username</u>, password, studId, name)

Equivalent alternatives

ER=>Relational (all in one)

- Entity E => Table T
  - single-valued attributes => attributes of relation
  - identifier attributes => candidate keys of the relations
- A multi-valued attribute of E => Table T
  - attributes of T: identifier of E plus the multi-valued attribute
- M-N Relationship R => Table T
  - T contains all identifier attributes of the entities that participate in R
  - T also contains the attributes of R
- N-1 Relationship between E1 and E2 => no new table
  - we add to the table of E2 the key of E1 (foreign key)
  - if the participation of E2 is (1,1) and not (1,0) then this attribute cannot have null values
- 1-1 Relationship between E1 and E2 => no new table
  - If (0,1) (0,1) we add to one of the tables that correspond to E1 or E2 the key of the other. we add to the table of E2 the key of E1 (foreign key)
  - if (1,1)(1,1) then both tables can be combined into one

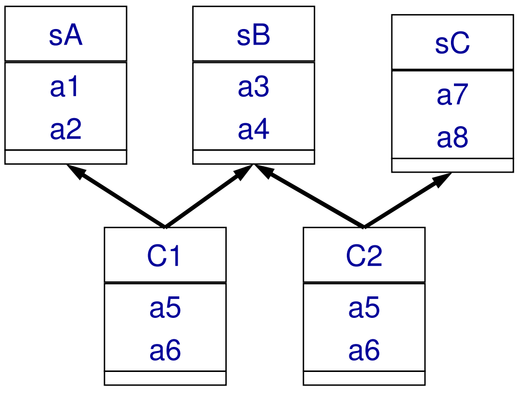
How to map generalization/specialization hierarchies to the relational model?

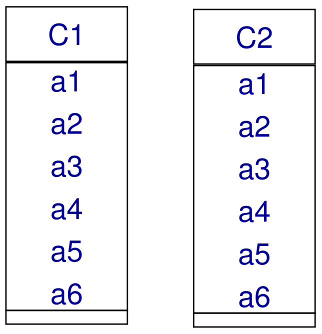
Recall the Lecture about
“Class and Method Design:
How to eliminate inheritance”

Class and Method Design>Eliminating Inheritance>

Method 1: Flattening

Assuming sA, sB and sC are abstract

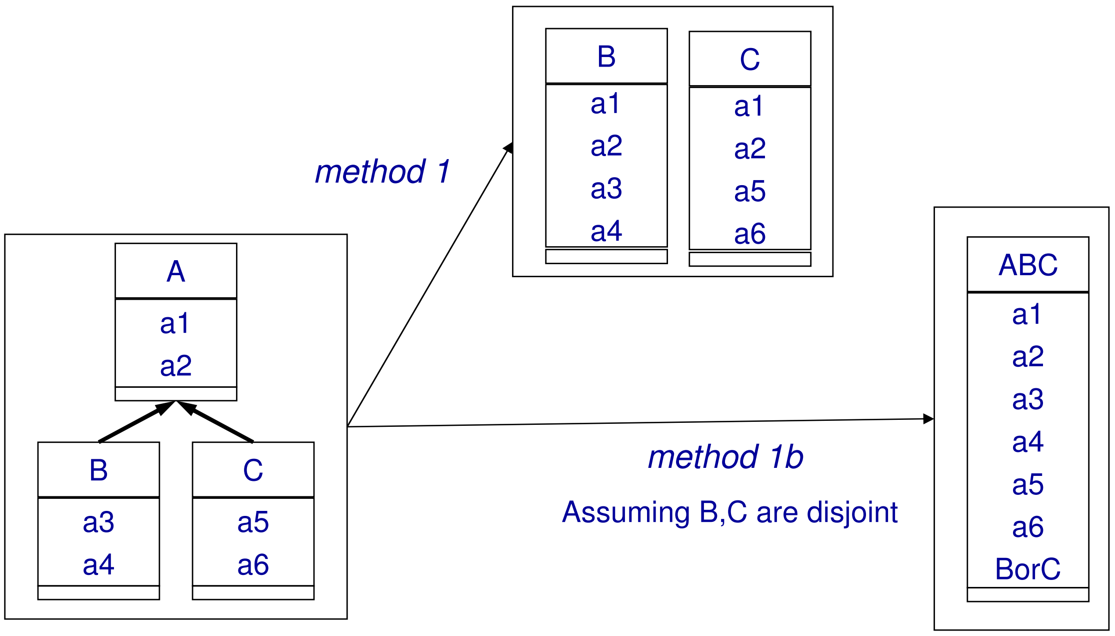




Class and Method Design>Eliminating Inheritance>

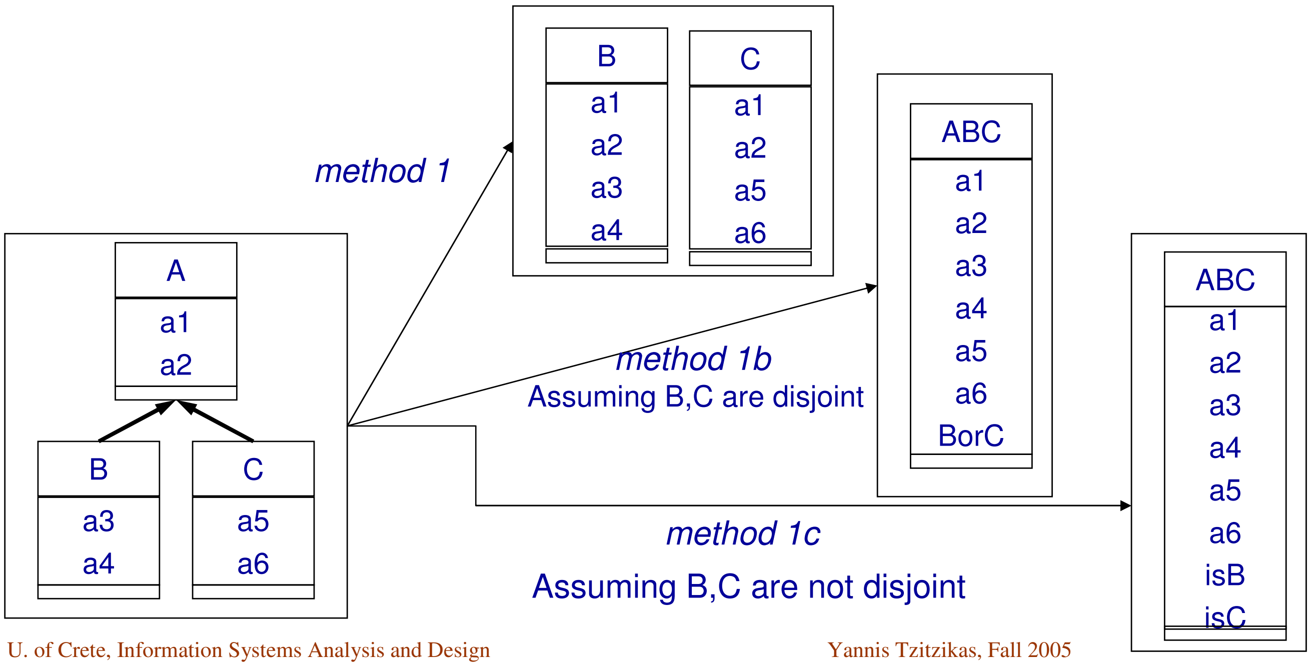
Method 1b: Flattening all in one table

- If the subtypes <u>are disjoint</u>
- Create one table with all attributes
- Define an extra attribute to discriminate the subtype



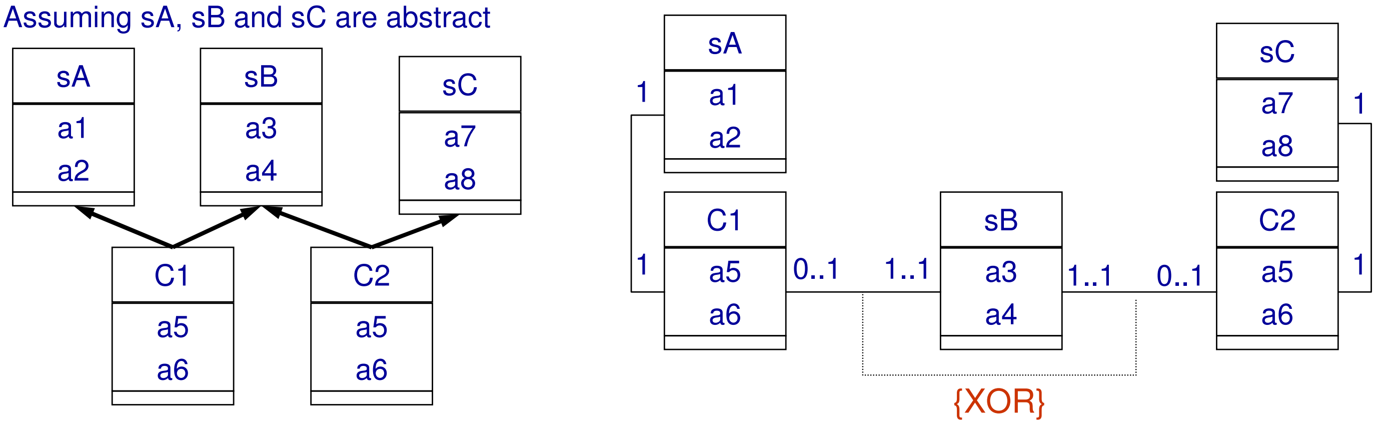
Method 1c: Flattening all in one table

- If the subtypes are not disjoint
- Create one table with all attributes
- Define an extra flag attribute for each subtype

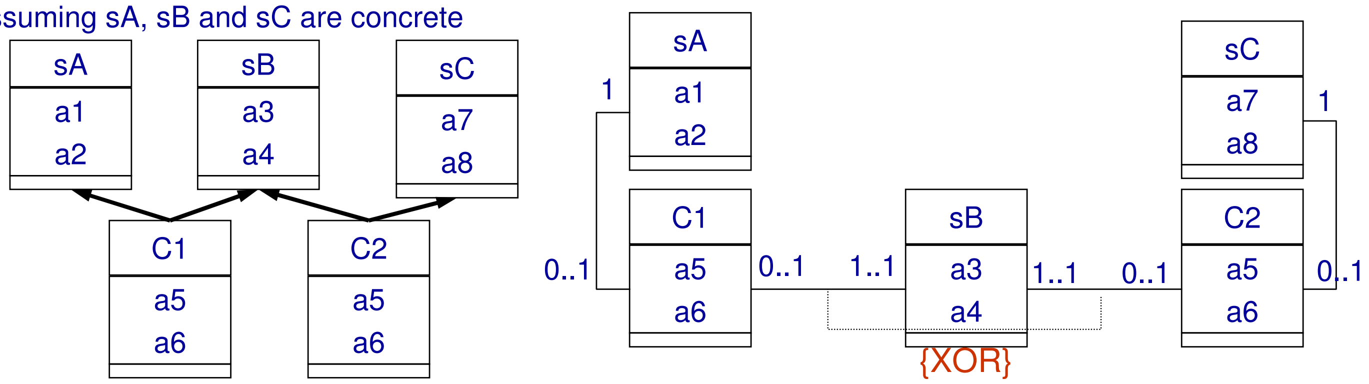


Method 2: Convert all IsA-links to Associations

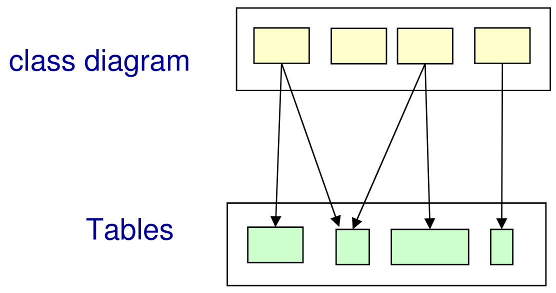
Assuming sA, sB and sC are abstract



Assuming sA, sB and sC are concrete



From a Class Diagram to a Relational Database Schema



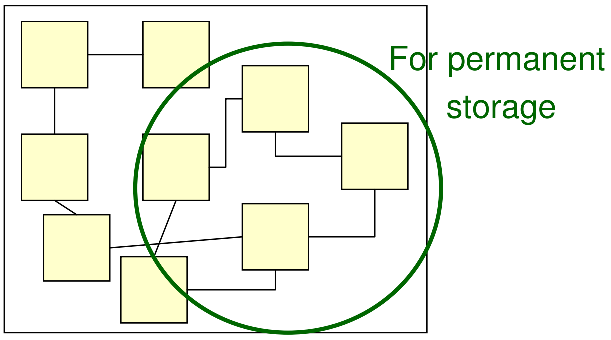
Objects must be converted so they can be stored in tables

Class Diagram => Relational Model

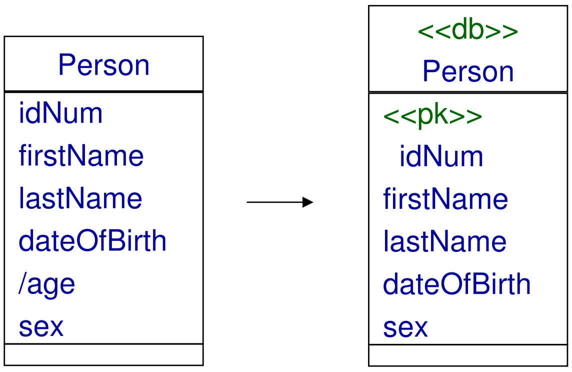
- The structural part of the class diagram can be mapped to a relational schema by employing the techniques that we described earlier for mapping an ER diagram to a Relational Schema.
- The behavioral part of the class diagram (e.g. operation) could be turned into triggers or stored procedures.

Class Diagram => Relational Model

- However UML class diagrams define the data structures required by the entire application. Not every class of the class diagram needs persistent storage.
- So
  - we have to identify the classes that need persistence
  - we can mark those classes (e.g. using a stereotype)
  - we can create a class diagram that contains only these
  - we cam mark the identifiers (keys) of those classes (e.g. using a stereotype)



Class diagram



(C) Optimizing the object-persistence formats (assuming the Relational Model)

(C) Optimizing the object-persistence formats

Dimensions of optimization:

- Storage efficiency (minimizing storage space, reduce redundant data)
- Speed of access (minimizing time to retrieve desired information)

A well-formed logical data model does not contain redundancy or many null values

- muplitple possible interpretation of null values may lead to mistakes

Normal forms

- There are six: 1NF, 2NF, 3NF, BCNF, 4NF, 5NF
- A table that is in a higher NF is also in all lower NFs
- A table must be at least in 1NF (i.e. with no structured or multi-valued columns).
- A table in a low NF can exhibit so-called update anomalies: undesirable side effects as a result of a modification operation.
  - E.g. if the same information is repeated many times (which may drive to inconsistencies and storage space redundancies)
- We can normalize a table to a higher NF by splitting it vertically
  - (the original table can be reconstructed using join operations)
- A db with very frequent updates should be in a high NF
- A rather static db can be in a low NF (more efficient, less joins)

Functional Dependencies (in brief)

$X \to Y$: $t1[X] = t2[X] \Rightarrow t1[Y] = t2[Y]$

- some trivial fds: $A \to A$, $X \to Y$ and $Y \subseteq X$
- if K is the primary key of a relation R then $K \to R$

Armstrong's axioms

- $Y \subseteq X \Rightarrow X \to Y$
- $X \to Y \Rightarrow WX \to WY$
- $X \to Y, Y \to Z \Rightarrow X \to Z$

Normal Forms (in brief)

A table is in:

- 1NF: If the domain of each attribute consists of atomic values only
  - i.e. structured or multi-valued attributes are not allowed.
- 2NF: if it is in 1NF and every non-key attribute is functionally dependent on the whole primary key
  - If the primary key consists of more than one attribute and there is a column that depends on only a part of the primary key, then the table is not in 2NF.
- 3NF: if it is in 2NF and no nokey attribute is transitively dependent on the primary key
  - If there is an attribute that depends on a non-primary key column then the table in not in 3NF
- BCNF:
- 4NF: based on multivalued dependencies
  - .....

More at HY360

For practical purposes it is usually adequate to normalize data into 3NF

Normal Forms: Examples

Not in 2NF:

Room(<u>buildingNum</u>, <u>roomNum</u>, street, streetNum, citypostalcode, city, numOfSeats)

is not in 2NF because buildingNum → street,...

In 2NF:

Room(<u>buildingNum, roomNum</u>, numOfSeats)

Building(<u>buldingNum</u>, street, streetNum, citypostalcode, city)

Not in 3NF:

Building(<u>buldingNum</u>, street, streetNum, citypostalcode, city)

is not in 3NF because citypostalcode → city

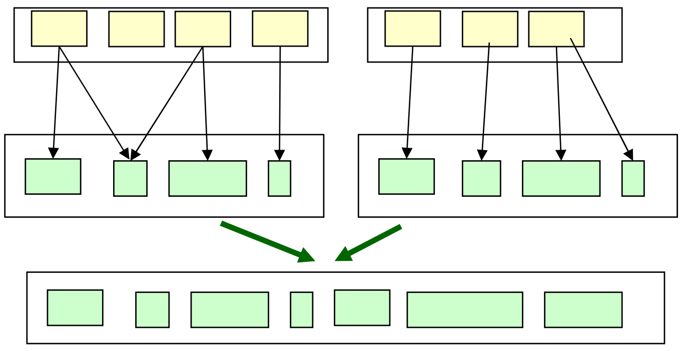
In 3NF:

Building(<u>buldId</u>, street, streetNum,citypostalcode)

CPostCode(<u>citypostalcode</u>, city)

we moved the attribute that depend on non-key attributes to another relation

View Integration

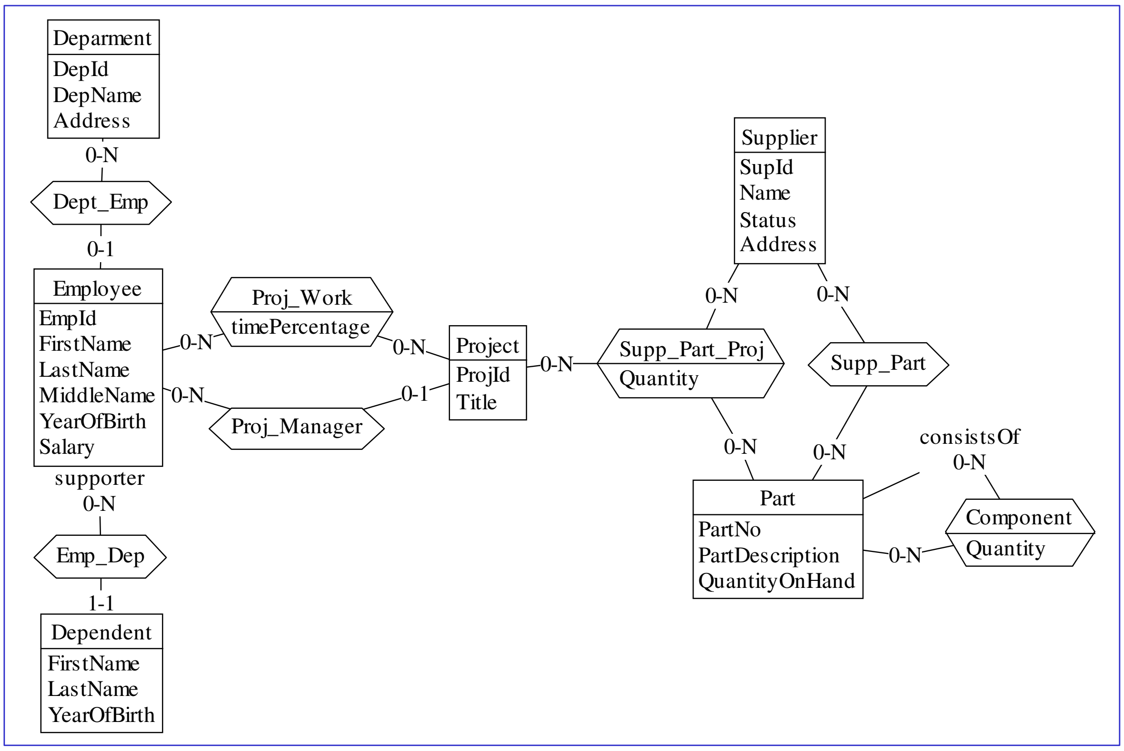


- As part of the logical database design, normalized tables likely have been created from a number of separate ER diagrams.
- We should merge these tables and remove any redundancy. This task is usually called *view integration*.
- Common view integration problems:
  - synonyms
    - 2 or more attributes have different name but the same meaning
  - homonyms
    - 2 or more attributes have the same name but different meaning
- We have to identify such cases and fix them

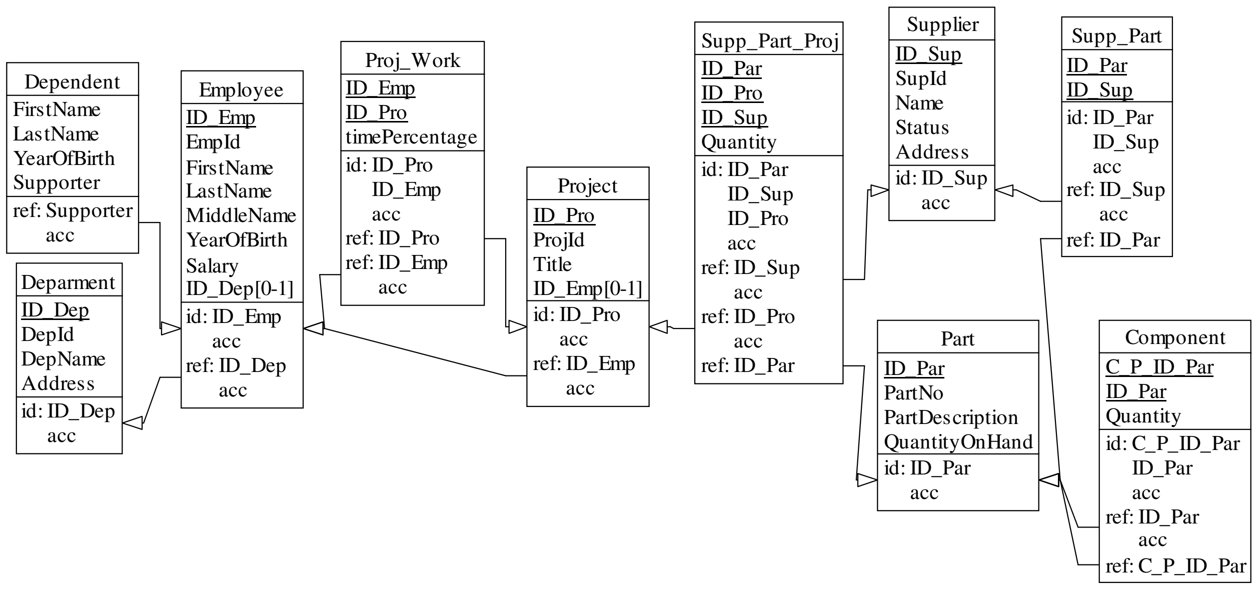
CASE Tools

- CASE tools targeting system design and implementation normally provide a data-modeling technique that targets a vast variety of specific DBMSs.
- They provide a capability for constructing a combined logical/physical model and immediately generating the relevant SQL code.
- They also support a number of functions that are useful for view integration.
- Using them we can save a lot of time
- The CASE tool DB-MAIN will be presented in the tutorial of Dec 12

ER=>Relational Example: First ER Diagram



ER=>Relational
Example: Its translation to the relational model



Generated by DB-MAIN

ER => SQL DDL
Tables and Constraint Section

Header section

```
-- ***********************************************
-- * Standard SQL generation                    *
-- *------------------------------------------*
-- * Generator date:  Nov  8 2004               *
-- * Generation date: Mon Nov 21 15:12:39
     2005 *
-- ***********************************************
-- Database Section
-- _______________

create database SCHEMA compact;

-- DBSpace Section
-- _______________
```

Tables section

```
-- Tables Section
-- _____________

create table Component (
    C_P_ID_Par char(10) not null,
    ID_Par char(10) not null,
    Quantity char(1) not null,
    constraint ID_Component primary key
      (C_P_ID_Par, ID_Par));

create table Deparment (
    ID_Dep char(10) not null,
    DepId char(1) not null,
    DepName char(1) not null,
    Address char(1) not null,
    constraint ID primary key (ID_Dep));

....
```

ER => SQL DDL Index Section

Constraint section

```
-- Constraints Section
-- ___________________

alter table Component add constraint FKconsistsOf
     foreign key (ID_Par)
     references Part;
alter table Component add constraint FKCom_Par
     foreign key (C_P_ID_Par)
     references Part;
alter table Dependent add constraint FKEmp_Dep
     foreign key (Supporter)
     references Employee;
alter table Employee add constraint FKDept_Emp
     foreign key (ID_Dep)
     references Deparment;
alter table Project add constraint FKProj_Manager
     foreign key (ID_Emp)
     references Employee;
alter table Proj_Work add constraint FKPro_Pro
     foreign key (ID_Pro)
     references Project;
alter table Proj_Work add constraint FKPro_Emp
     foreign key (ID_Emp)
     references Employee;
alter table Supp_Part add constraint FKSup_Sup_1
     foreign key (ID_Sup)
     references Supplier;
```

Index section

```
-- Index Section
-- _____________

create index ID_Component
     on Component (C_P_ID_Par,
       ID_Par);

create index FKconsistsOf
     on Component (ID_Par);

create index ID
     on Deparment (ID_Dep);

create index FKEmp_Dep
     on Dependent (Supporter);

create index ID
     on Employee (ID_Emp);

create index FKDept_Emp
     on Employee (ID_Dep);

create index ID
     on Part (ID_Par);
```

Instructions for the physical Data layer of the db

ER: Reading and References

- **Systems Analysis and Design with UML Version 2.0** (2nd edition) by A. Dennis, B. Haley Wixom, D. Tegarden, Wiley, 2005. Chapter 11
- **Requirements Analysis and System Design** (2nd edition) by Leszek A. Maciaszek, Addison Wesley, 2005, Chapter 8
- Slides from
  - CS360 University of Crete: HY360 (www.csd.uoc.gr/~hy360)
  - University of Texas at Austin (Data Modeling)
- More about the transition "Natural Language Specifications => ER Diagram" can be found at:
  - A. Min Tjoa, Linda Berger: Transformation of Requirement Specifications Expressed in Natural Language into an EER Model. ER 1993: 206-217
  - H. M. Harmain and Robert J. Gaizauskas, CM-Builder: An Automated NL-Based CASE Tool, Automated Software Engineering", 45-54, 2000