Informatics 1

School of Informatics, University of Edinburgh

Data and Analysis

Part I

Structured Data

Alex Simpson

January 2012

Part I — Structured Data

- For some application domains, data is *inherently structured*
  - For instance, the University records some common pieces of information about every student
- In such applications, it makes sense to organise the data in a way that directly corresponds to these *individuals* and their *properties*
- We will look at two main *data representation* models:
  - The *entity-relationship (ER)* model, and the *relational* model
- Finally, we will deal with some methods for data *manipulation* in the *relational model*, namely:
  - *Relational algebra*, the *Tuple-relational calculus* and the query language *SQL*

Part I — Structured Data

Data Representation:

Data Manipulation:

Required reading

For this section you need to read Chapter 2 of:

[DMS] R. Ramakrishnan and J. Gehrke
Database Management Systems
McGraw-Hill, Third Edition, 2003.

§§2.1–2.5 cover the technical material from this lecture in depth, while §§2.6–2.9 provide further context.

Try some of the exercises. Answers to the odd ones are on the book's website.

http://pages.cs.wisc.edu/ dbbook/

Initial stages of database design

1. Requirements analysis.

   Understand what data is to be stored in the database and what operations are likely to be performed on it.

2. Conceptual design

   Develop a high-level description of data to be stored and constraints that hold over it.

   This description is often given using the ER data model.

3. Logical design

   Implement the conceptual design by mapping it to a *logical data representation*. The outcome is a *logical schema*.

   The implementation is often performed by translating the ER data model into a *relational database schema* (see I.2).

The ER data model

- What is it used for?

  The ER model is a way to organise the description of *entities* (things in the real world) and the *relationships* between them

- Why is it useful?

  It readily maps into different *logical data models*, such as the relational model

- How is it used?

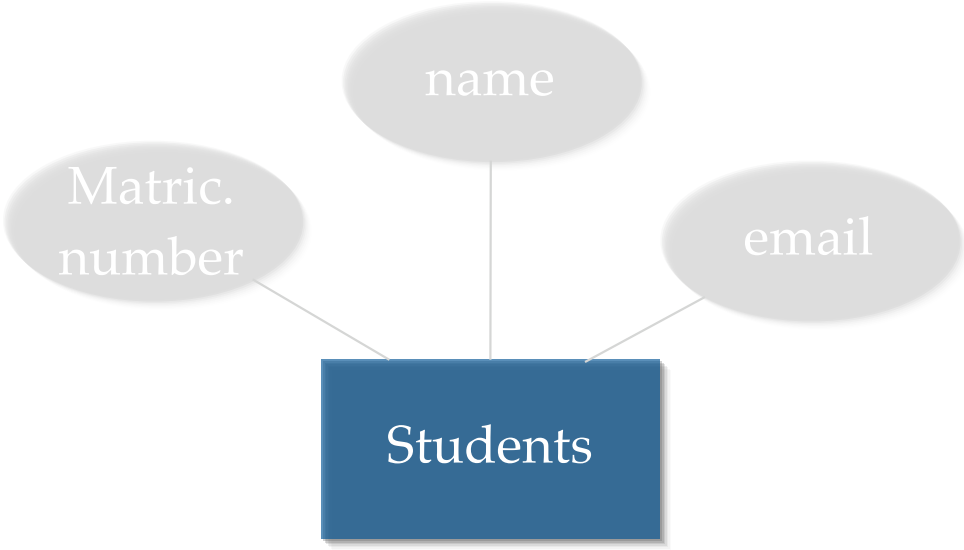
  As a way to visualise data and their dependencies, to clarify these and communicate them.

Entities and entity sets

Any distinguishable object (for example, in the real world) can be an *entity*

A collection of the same sort of entities is an *entity set*

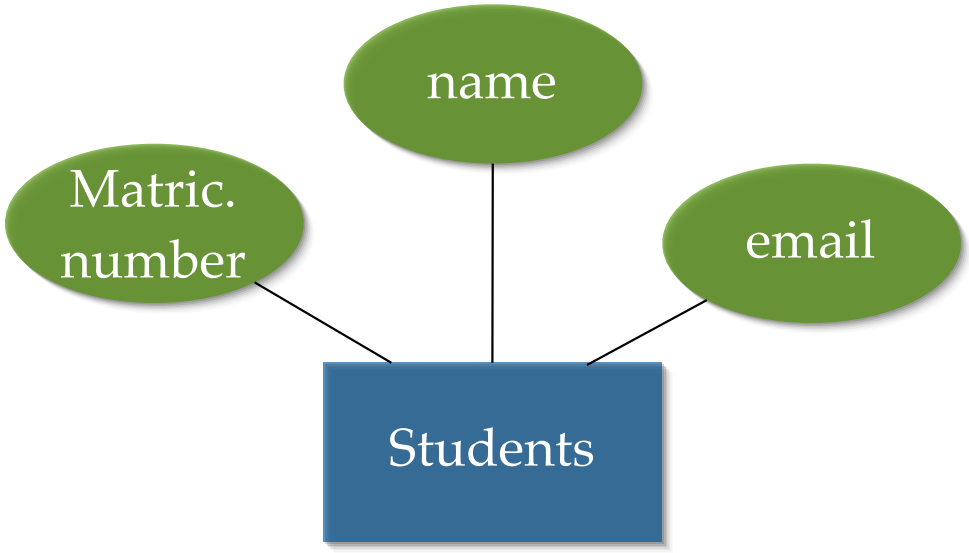
Entity sets are shown in an *ER diagram* by *boxes*, labelled with the entity set's name



Attributes

Each entity of the same entity set has some characteristic *attributes*

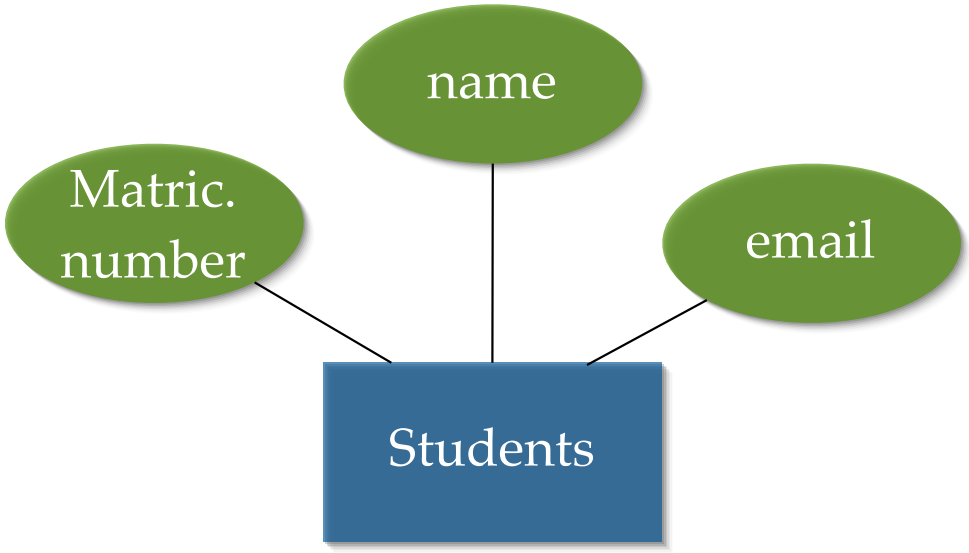
Attributes are represented by *ovals*, labelled with the attribute's name, connected to the entity set they belong to.



Domains

Each attribute has a *domain* from which allowable values are derived

E.g., Matric. number is a *positive integer*
name and email might be *strings of up to 64 characters*

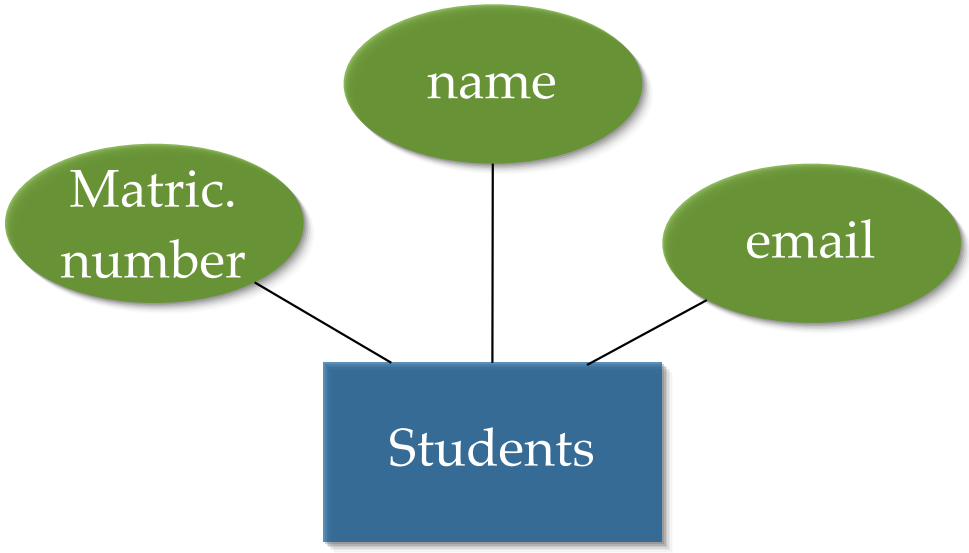


Keys

A *key* is a minimal set of attributes whose values allow us to uniquely identify an entity in an entity set

If there is more than one such minimal set, they are called *candidate keys*

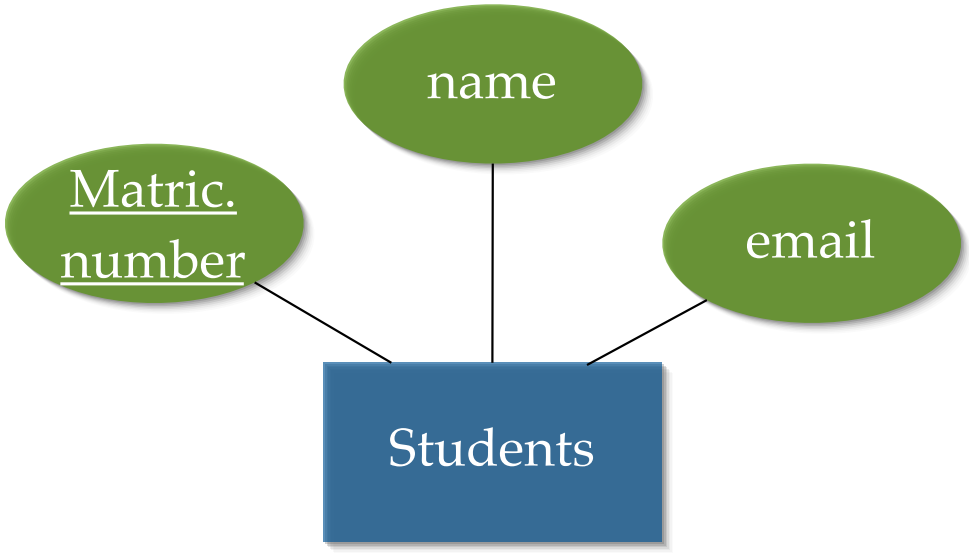
E.g., either Matric. number or email can act as keys.



Primary keys

If multiple candidate keys exist, we choose one and make it the *primary key*.

The attributes occurring in the primary key are *underlined* in the ER diagram. If there are several then they form a *composite key*.



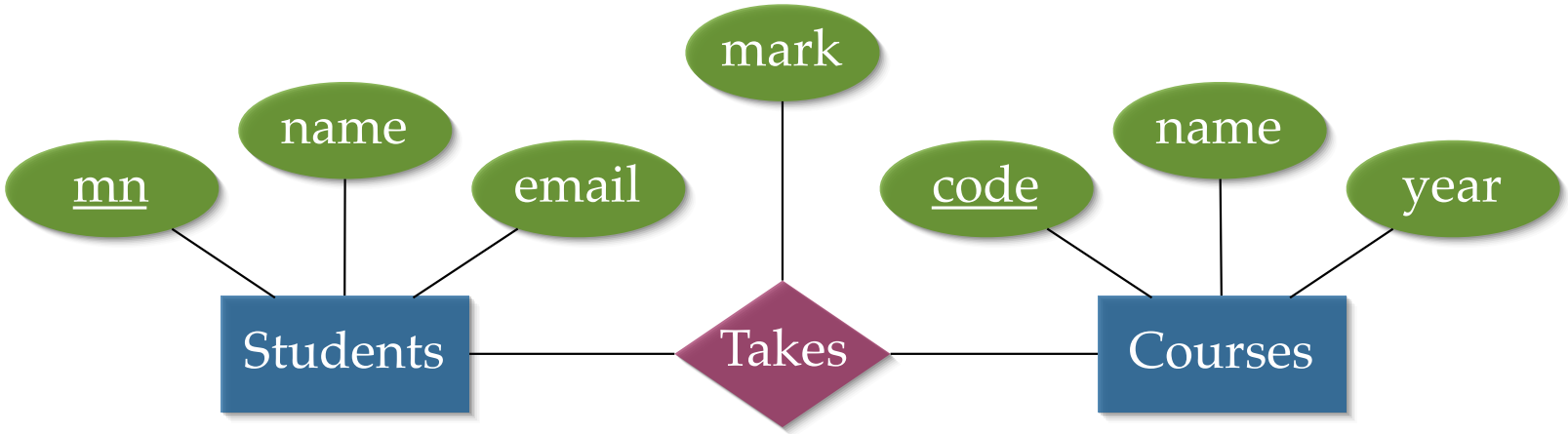
Relationships and relationship sets

Relationships model associations between entities

These are grouped into *relationship sets* of relationships between entities from specified entity sets.

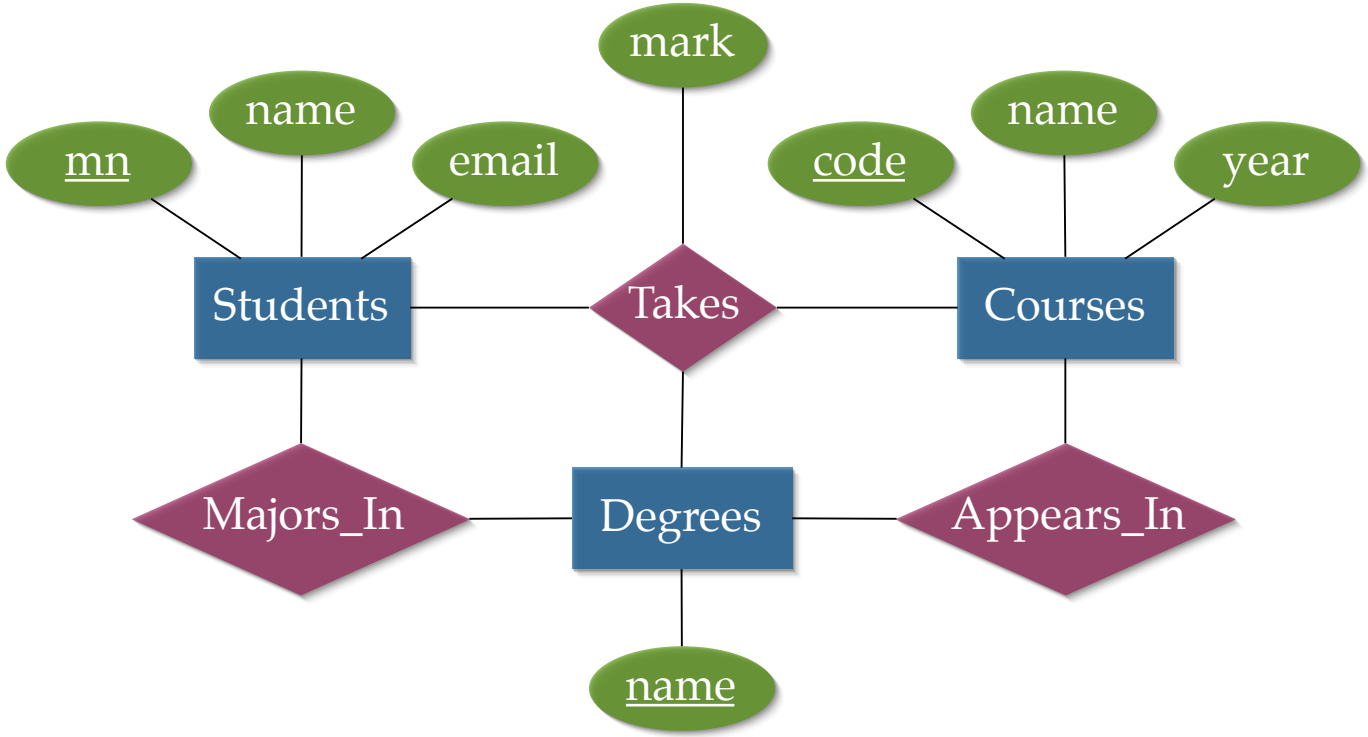
Relationship sets are represented as *diamonds* in ER diagrams

Relationships may have *attributes* of their own.



There is no bound on the number of entities participating in a relationship.

Correspondingly, there is no bound on the number of relationships an entity can participate in



Instances

Entity instances and *relationship instances* are what we obtain after instantiating the attributes of an entity or a relationship

Examples

An entity instance from the Students entity set:

(**123**, Winston, wsmith@example.org)

An entity instance from the Courses entity set:

(**08015**, Informatics 1: D & A, **2010**)

A relationship instance from the Takes relationship set:

(**123**, Winston, wsmith@example.org, **08015**, Informatics 1: D & A, **2010**, **88**)

Key constraints

A *key constraint* is a particular kind of connection between a relationship set and an entity set.

Definition. Suppose $\boldsymbol{R}$ is a relationship between $\boldsymbol{n}$ entity sets, $\boldsymbol{E_1, \ldots, E_n}$. There is a *key constraint* on the entity set $\boldsymbol{E_k}$ if, however we instantiate the attributes of $\boldsymbol{E_k}$, that entity instance participates in at most one relationship instance.

Notation. A key constraint is shown on an ER diagram by an *arrow* from $\boldsymbol{E_k}$ to $\boldsymbol{R}$.

Example. Consider the entity sets of Students, directors of studies (DoS), and the Directed-By relationship between them.

- Given a Students instance, we can determine the Directed-By instance it appears in. That is, each student has a unique DoS.

One-to-many and many-to-many relationships

A *one-to-many* relationship R between entity sets E_o and E_m means that, for each instance $e_m \in E_m$, there is at most one instance $e_o \in E_o$ such that e_o and e_m appear together in some relationship instance $r \in R$.

More simply: each instance $e_o \in E_o$ may be associated (in R) with many instances $e_m \in E_m$, but each instance $e_m \in E_m$ must be associated (in R) with at most one instance $e_o \in E_o$.

If R is a binary relationship between E_o and E_m, then being one-to-many is equivalent to there being a key constraint on E_m.

A *many-to-many* relationship R between entity sets E_o and E_m means that there are no constraints on the number of times entity instances $e_o \in E_o$ and $e_m \in E_m$ may appear in relationship instances $r \in R$.

Examples

The Directed_By relationship between the Students and DoS entity sets is a many-to-one relationship.

- Each student has a single DoS, but
- each DoS may have many students

The Takes relationship between Students and Courses is a many-to-many relationship

- Each student takes many different courses;
- Each course may be taken by many different students

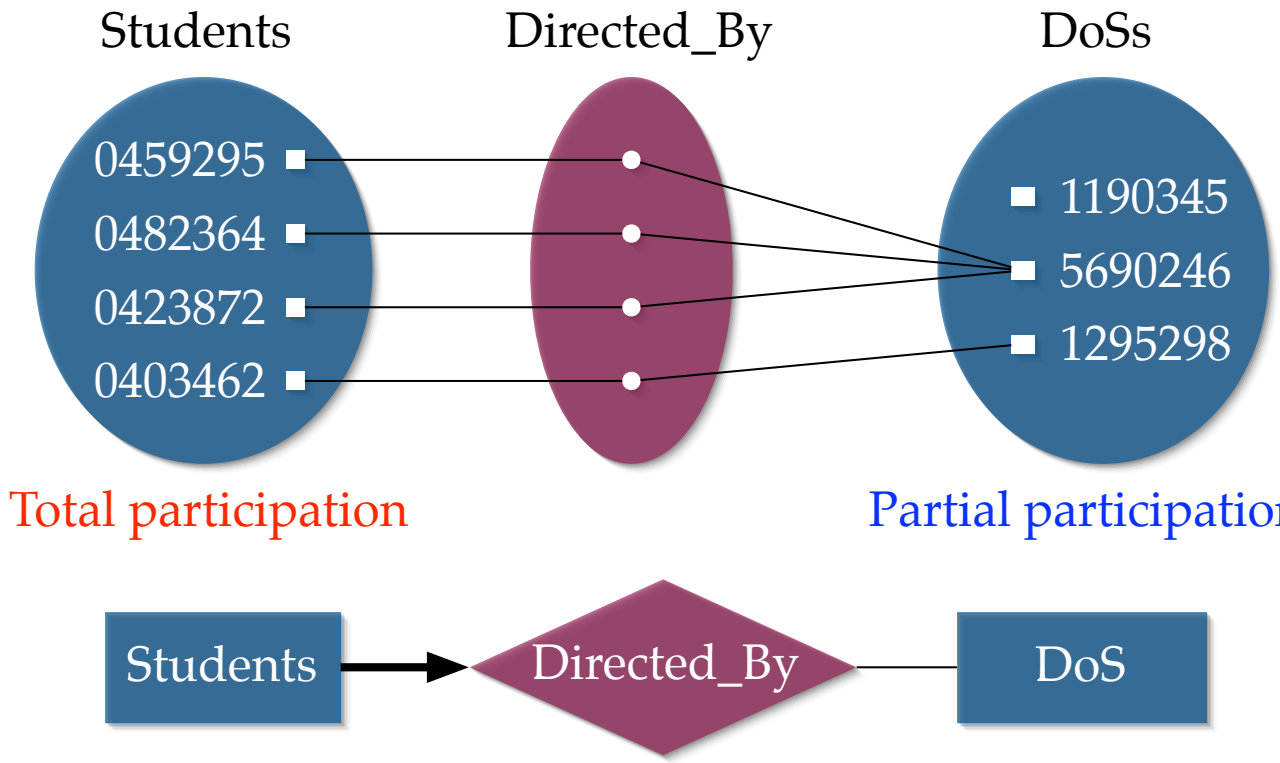
Participation constraints

Participation constraints express the degree to which in which entities participate in a relationship.

Total participation on entity set E for relationship R means that every entity instance $e \in E$ appears in at least one relationship instance of R.

Partial participation on entity set E for relationship R means that there exist entities $e \in E$ that do not appear in instances of R.

Example

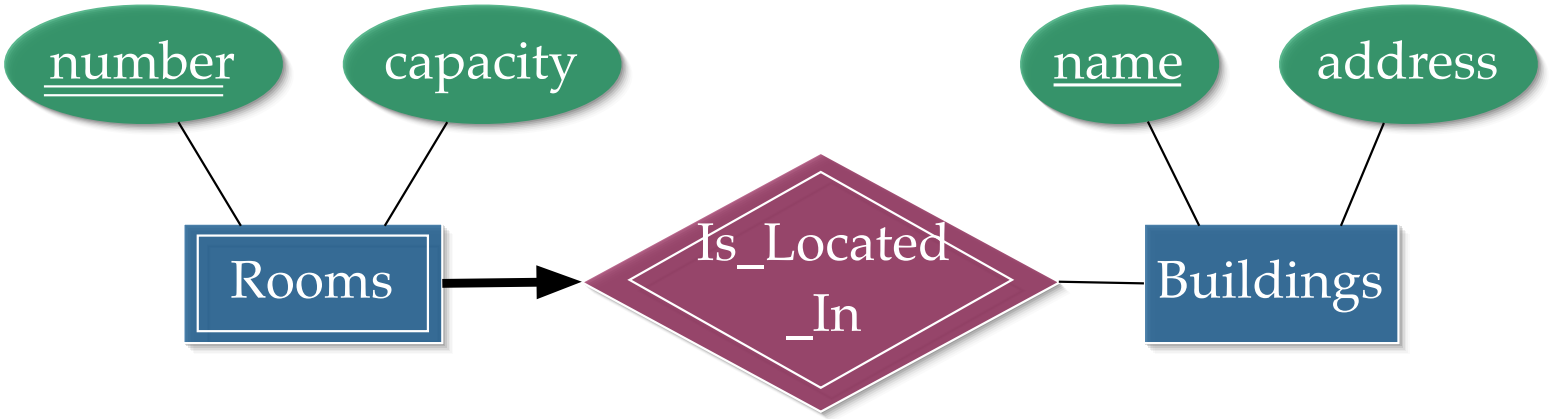


Notation. A *thick line* from an entity to a relationship represents total participation. (Here there is a key constraint too.)

Weak entity sets

In certain cases, it is impossible to designate a primary key for the entities of an entity set.

Instead, we create a key by adding in the key of another entity set.

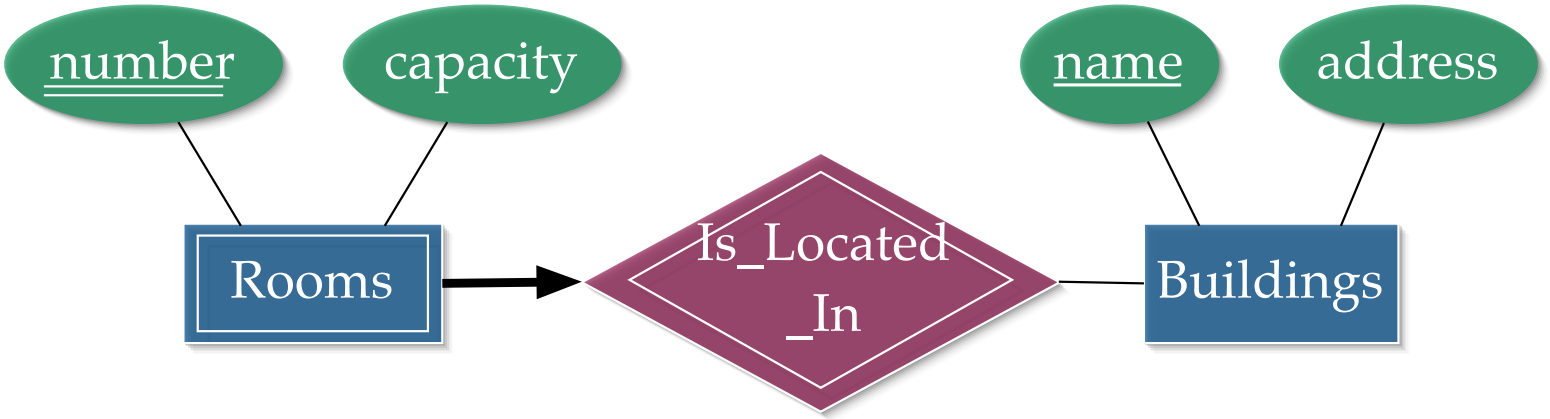


Notation

Double border for the weak entity and its identifying relationship

Double underlines for the attributes of the weak entity set contributing to the composite key

The *identifying relationship* must be many-to-one and total.



Weak entity set: Definition

- A *weak entity set* is an entity set for which a primary key consisting only of its own attributes cannot be identified
- The *key* is formed by a combination of its own attributes and the key attributes from another entity set with which it has a relationship
- The entity set from which attributes are borrowed is called the *identifying owner*
- The relationship between the weak entity set and its identifying owner is called an *identifying relationship*.
- The identifying relationship must be many-to-one and total.

Hierarchical entities and inheritance

A *subclass* like Full-Time Students or Part-Time Students will *specialize* a *superclass* (Students) by *inheriting* attributes from the superclass.

Subclasses may also have additional attributes of their own.