Part I — Structured Data

Data Representation:

I.1 The entity-relationship (ER) data model

I.2 The relational model

Data Manipulation:

I.3 Relational algebra

I.4 Tuple relational calculus

I.5 The SQL query language

Related reading: Chapter 3 of [DMS], §§ 3.1,3.2,3.4,3.5

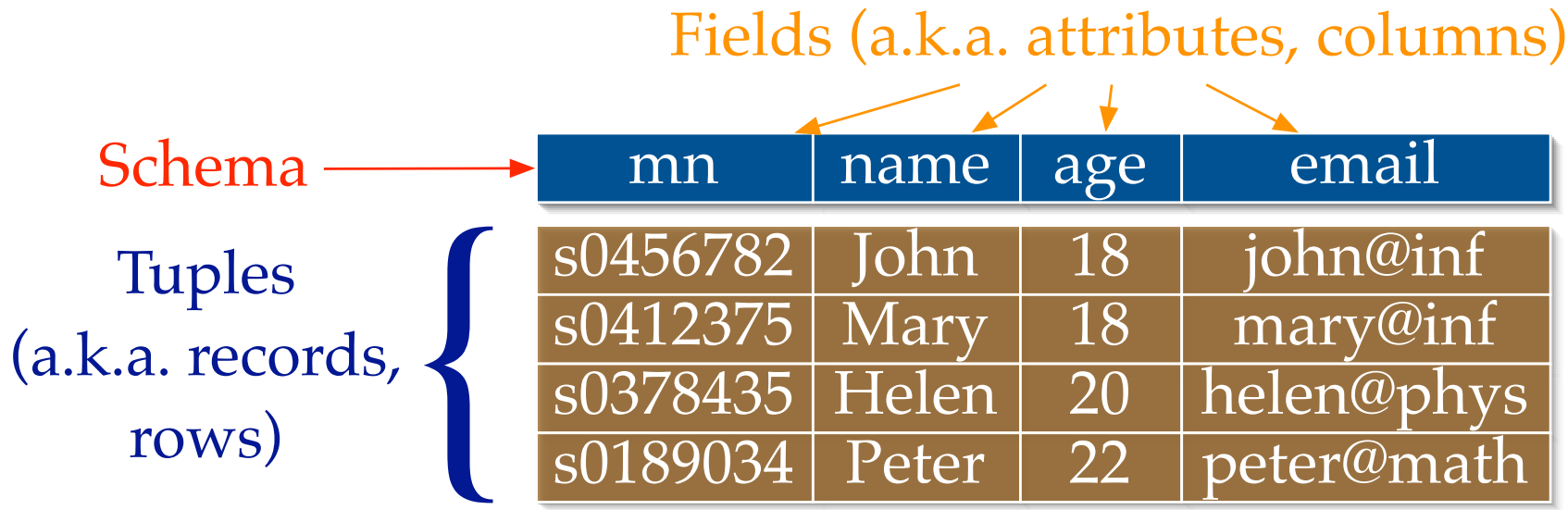
History of relational model

- The *relational model* was introduced in 1970 by Edgar F. Codd, a British computer scientist working at IBM's Almaden Research Center in San Jose, California.
- IBM was initially slow to exploit the idea, but by the mid 1970's IBM was at the forefront of the commercial development of relational database systems with its System R project, which included the development and first implementation of SQL. (Codd was sidelined from this project!)
- Around the same time, the relational model was developed and implemented at UC Berkeley (the Ingres project)
- Nowadays relational databases are a multi-billion pound industry.
- A major reason for the success of the relational model is its simplicity
- Codd received the 1981 *Turing Award* for his pioneering work on relational databases

Building blocks

- The basic construct is a *relation*.
  - It consists of a *schema* and an *instance*
  - The *schema* can be thought of as the format of the relation
  - A *relation instance* is also known as a *table*
- A *schema* is a set of fields, which are (name, domain) pairs
  - *fields* may be referred to as attributes, or columns
  - *domains* are referred to as types
- The rows of a table are called *tuples* (or *records*) and they are value assignments from the specified domain for the fields of the table
- The *arity* of a relation is its number of columns (fields)
- The *cardinality* of a table is its number of rows (tuples)

Example



mn	name	age	email
s0456782	John	18	john@inf
s0412375	Mary	18	mary@inf
s0378435	Helen	20	helen@phys
s0189034	Peter	22	peter@math

SQL: The Structured Query Language

- SQL is the standard language for interacting with relational database management systems
- Substantial parts of SQL are *declarative*: code states what should be done, not necessarily how to do it.
- When actually querying a large database, database systems take advantage of this to plan, rearrange, and optimize the execution of queries.
- Procedural parts of SQL do contain *imperative* code to make changes to the database.
- While SQL is an international standard (ISO 9075), individual implementations have notable idiosyncrasies, and code may not be entirely portable.

Data definition in SQL

- A special subset of SQL called the *Data Definition Language (DDL)* is used to declare table schemata
- Relations are called *tables* in SQL
- It is a typed language
  - For simplicity, we will assume there are only three types: (i) **`integer`** for integer numbers, (ii) **`real`** for real numbers (floating point), and (iii) **`char(`**n**`)`** for a string of maximum length n
  - There is also a special **`null`** value, which we see briefly later.

General form of a DDL statement

create table *table name* **(** *attribute name* *attribute type*
[, *attribute name* *attribute type* **]***
⟨*integrity constraints*⟩ **)**

Example 1

```
create table Students (
      mn              char(8),
      name            char(20),
      age             integer,
      email           char(15),
      primary key     (mn) )
```

The example defines the **Students** table.

The last line implements a *primary key constraint*, it declares **mn** to be the chosen primary key for **Students**.

This constraint requires that the **Students** table contains at most one row with any given **mn** value. This is enforced by the system.

Any attempt to insert a new row with an **mn** value that already exists in some other row of the table will fail.

```
create table Students (
      mn              char(8),
      name            char(20),
      age             integer,
      email           char(15),
      primary key     (mn) )
```

General form of a DDL statement

create table *table name* **(** *attribute name* *attribute type*
[, *attribute name* *attribute type* **]***
⟨*integrity constraints*⟩ **)**

Example 2

```
create table Takes (
      mn              char(8),
      code            char(20),
      mark            integer,
      primary key     (mn, code),
      foreign key     (mn) references Students,
      foreign key     (code) references Courses )
```

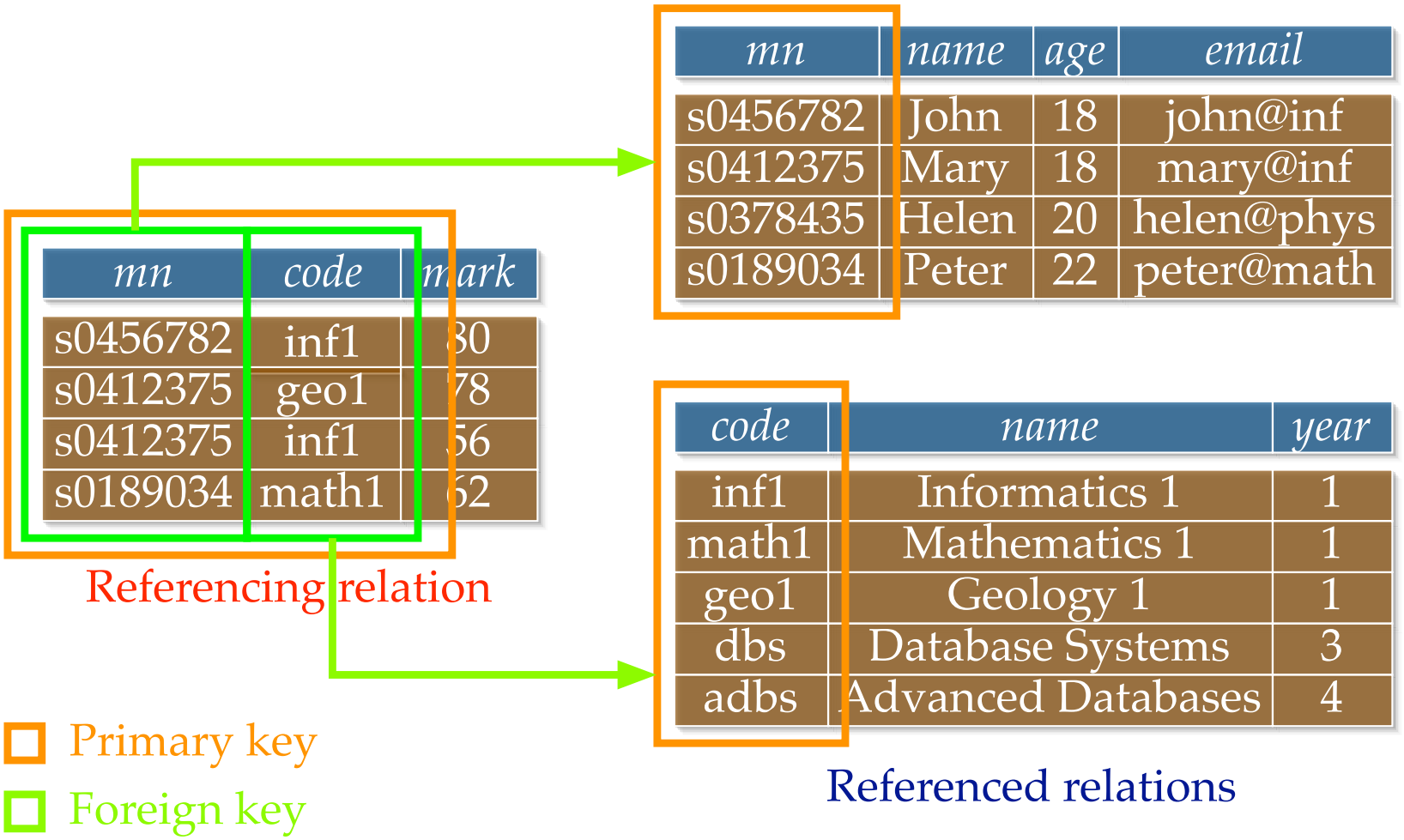
In this case, the primary key is a composite key using a pair of fields.

The *foreign key constraints* given here enforce two further properties:

- Whenever a tuple is inserted, the value for the **mn** field must be a value that appears in the primary key column of the **Students** table
- Similarly, the value for the **code** field must be a value that appears in the primary key column of the **Courses** table

```
create table Takes (
      mn              char(8),
      code            char(20),
      mark            integer,
      primary key     (mn, code),
      foreign key     (mn) references Students,
      foreign key     (code) references Courses )
```

Key constraints example



Summary

We have seen two forms of constraint:

`primary key` **`(`***declaration*`)`**

`foreign key` **`(`***declaration*`)`** **`references`** *table*

- Primary key constraints declare primary keys.
- Foreign key constraints link columns of one table to the primary key columns of another table.

Both are declared by the user, but enforced by the system itself.
(Attempting to enter a tuple that violates the constraint results in failure.)

N.B. In the ER model, **`Students`** was an entity set and **`Takes`** a relationship. In the relational model, *both* are (necessarily!) implemented as tables.

Translating an ER diagram to a relational schema

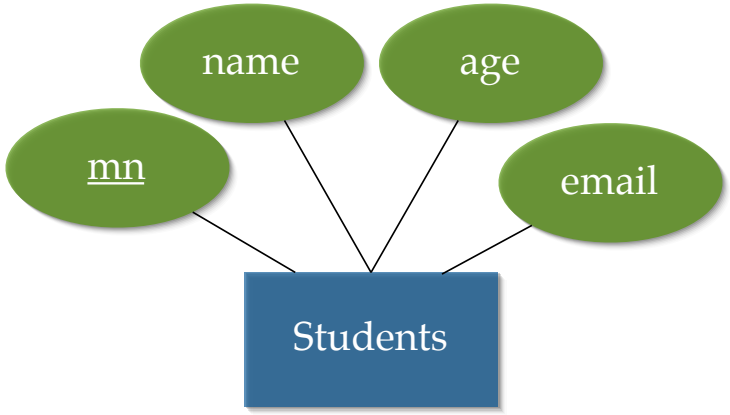
Given an ER diagram, we can look for a relational schema that closely approximates the ER design.

The translation is *approximate* because it is not always feasible to capture all the constraints in the ER design within the relational schema. (In SQL, certain types of constraint, for example, are inefficient to enforce, and so usually not implemented.)

There is more than one approach to translating an ER diagram to a relational schema. Different translations amount to making different implementation choices for the ER diagram.

It is possible to make these translations complete (work for any diagram) and automatic (in a push-button graphical tool); but in this course we shall just consider a few examples illustrating some of the main ideas.

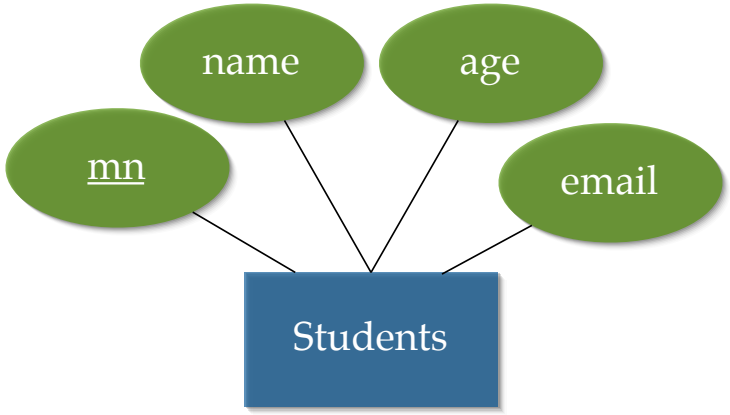
Mapping entity sets



Algorithm

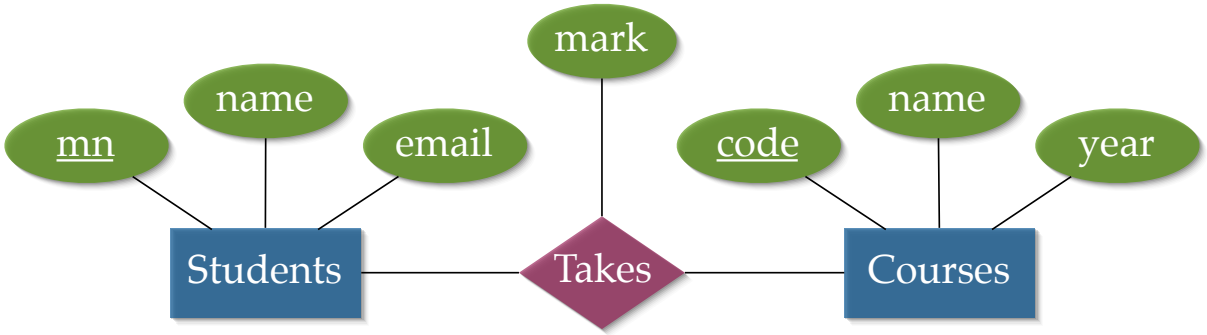
- Create a table for the entity set.
- Make each attribute of the entity set a field of the table, with an appropriate type.
- Declare the field or fields comprising the primary key

Mapping entity sets




```
create table Students (
      mn              char(8),
      name            char(20),
      age             integer,
      email           char(15),
      primary key     (mn) )
```

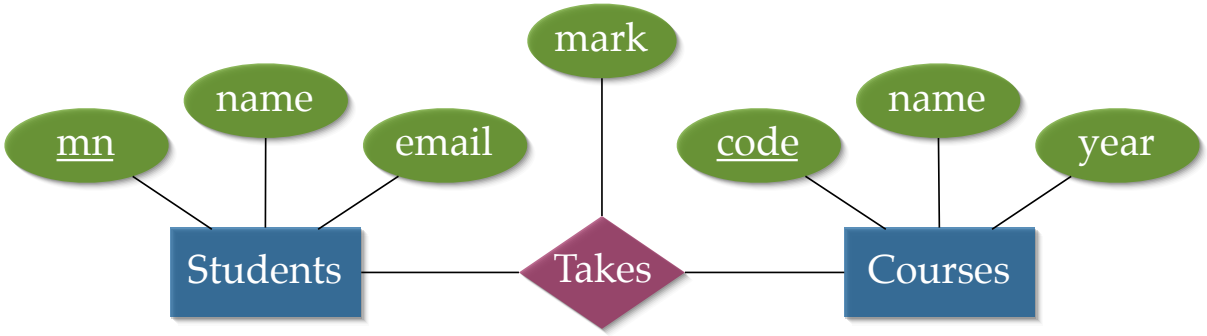
Mapping relationship sets (no key constraints)



Algorithm

- Create a table for the relationship set.
- Add all primary keys of the participating entity sets as fields of the table.
- Add a field for each attribute of the relationship.
- Declare a primary key using all key fields from the entity sets.
- Declare foreign key constraints for all these fields from the entity sets.

Mapping relationship sets (no key constraints)




```
create table Takes (
      mn              char(8),
      code            char(20),
      mark            integer,
      primary key     (mn, code),
      foreign key     (mn) references Students,
      foreign key     (code) references Courses )
```

Mapping relationship sets (with key constraints)

Algorithm

- Create a table for the relationship set.
- Add all primary keys of the participating entity sets as fields of the table.
- Add a field for each attribute of the relationship.
- Declare a primary key using the key fields from the source entity set only.
- Declare foreign key constraints for all the fields from the source and target entity sets.

Mapping relationship sets (with key constraints)


```
create table Directed_By (
      mn              char(8),
      staff_id        char(8),
      primary key     (mn),
      foreign key     (mn) references Students,
      foreign key     (staff_id) references DoS )
```

Note that this has captured the key constraint, but not the participation constraint. That requires an alternative approach, and a further kind of declaration.

Mapping relationship sets (with key constraints, 2nd method)

Algorithm

- Create a table for the source and target entity sets as usual.
- Add every primary key field of the target as a field in the source.
- Declare these fields as foreign keys.

Because the Directed_By relation is many-to-one, we don't in fact need a whole table for the relation itself. However, this does slightly "pollute" the source entity table.

Mapping relationship sets (with key constraints, 2nd method)


```
create table Students (
      mn              char(8),
      name            char(20),
      age             integer,
      email           char(15),
      dos_id          char(8),
      primary key     (mn),
      foreign key     (dos_id) references DoS )
```

Note that this has still not included the participation constraint on **Students** in **Directed_By**, but we are now nearer to doing so...

Null values

In SQL, a field can have the special value **`null`**

A **`null`** value means that a field is either unknown/missing/unavailable, or undefined/makes no sense here.

Some implementations do not allow the null value to appear in *primary key* fields.

They may allow nulls to appear in *foreign key* fields.

Null values can be disallowed from specific fields with a **`not null`** declaration.

In certain circumstances, by disallowing **`null`**, we can enforce a *participation constraint*.

Mapping relationship sets with key+participation constraints

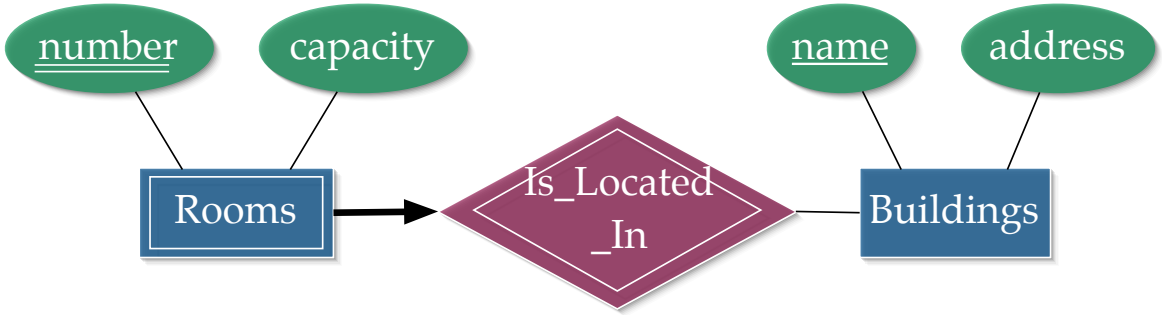
Algorithm

- Create a table for the source and target entity sets as usual.
- Add every primary key field of the target as a field in the source.
- Declare these fields as **`not null`**.
- Declare these fields as foreign keys.

Mapping relationship sets with key+participation constraints


```
create table Students (
      mn              char(8),
      name            char(20),
      age             integer,
      email           char(15),
      dos_id          char(8) not null,
      primary key     (mn),
      foreign key     (dos_id) references DoS )
```

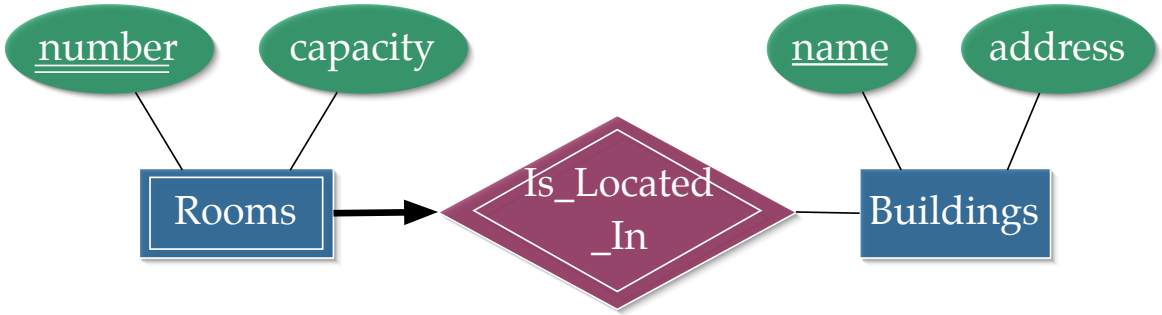
Mapping weak entity sets and identifying relationships



Algorithm

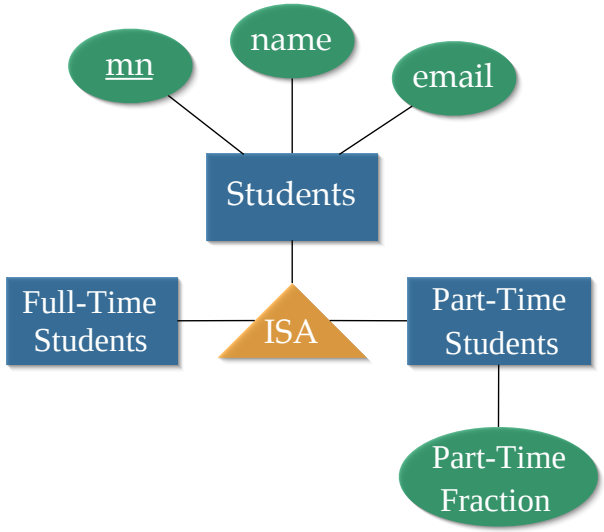
- Create a table for the weak entity set.
- Make each attribute of the weak entity set a field of the table.
- Add fields for the primary key attributes of the identifying owner.
- Declare a foreign key constraint on these identifying owner fields.
- Instruct the system to automatically delete any tuples in the table for which there are no owners

Mapping weak entity sets and identifying relationships



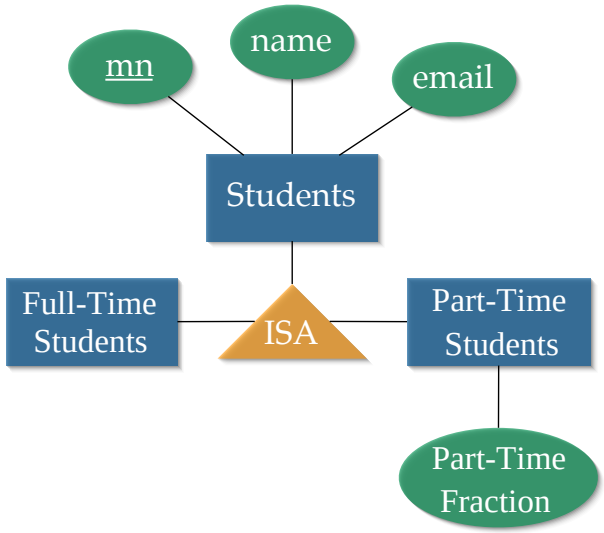

```
create table Rooms (
      number           char(8),
      capacity         integer,
      building_name    char(20),
      primary key      (number,building_name),
      foreign key      (building_name) references Buildings
                       on delete cascade )
```

Mapping hierarchical entities



- Declare a table as usual for the superclass of the hierarchy.
- For each subclass declare another table using superclass's primary key and the subclass's extra attributes.
- Declare the primary key from the superclass as the primary key of the subclass, and with a foreign key constraint.

Mapping hierarchical entities




```
create table PT_Students (
      mn              char(8),
      pt_frac         integer,
      primary key     (mn),
      foreign key     (mn) references Students )
```