Informatics 1B, 2008

School of Informatics, University of Edinburgh

Data and Analysis

Note 2

The Relational Model

Alex Simpson

Part I — Structured Data

Data Representation:

Note 1 The entity-relationship (ER) data model

Note 2 **The relational model**

Data Manipulation:

Note 3 Relational algebra

Note 4 Tuple relational calculus

Note 5 The SQL query language

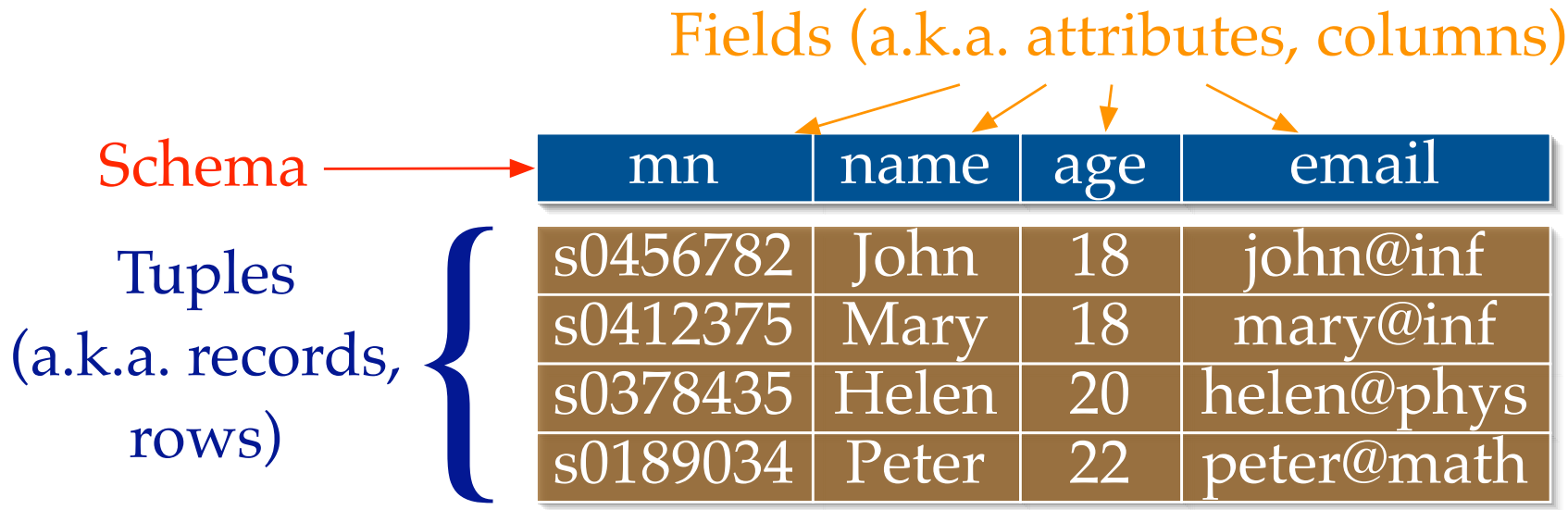
History of relational model

- 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 Berkely (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
- In 1981, Codd received the 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


Fields (a.k.a. attributes, columns)
Schema
Tuples
(a.k.a. records,
rows)
mn
name
age
email
s0456782
John
18
john@inf
s0412375
Mary
18
mary@inf
s0378435
Helen
20
helen@phys
s0189034
Peter
22
peter@math

Data definition in SQL

- SQL stands for *Structured Query Language*
- 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

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 )
```

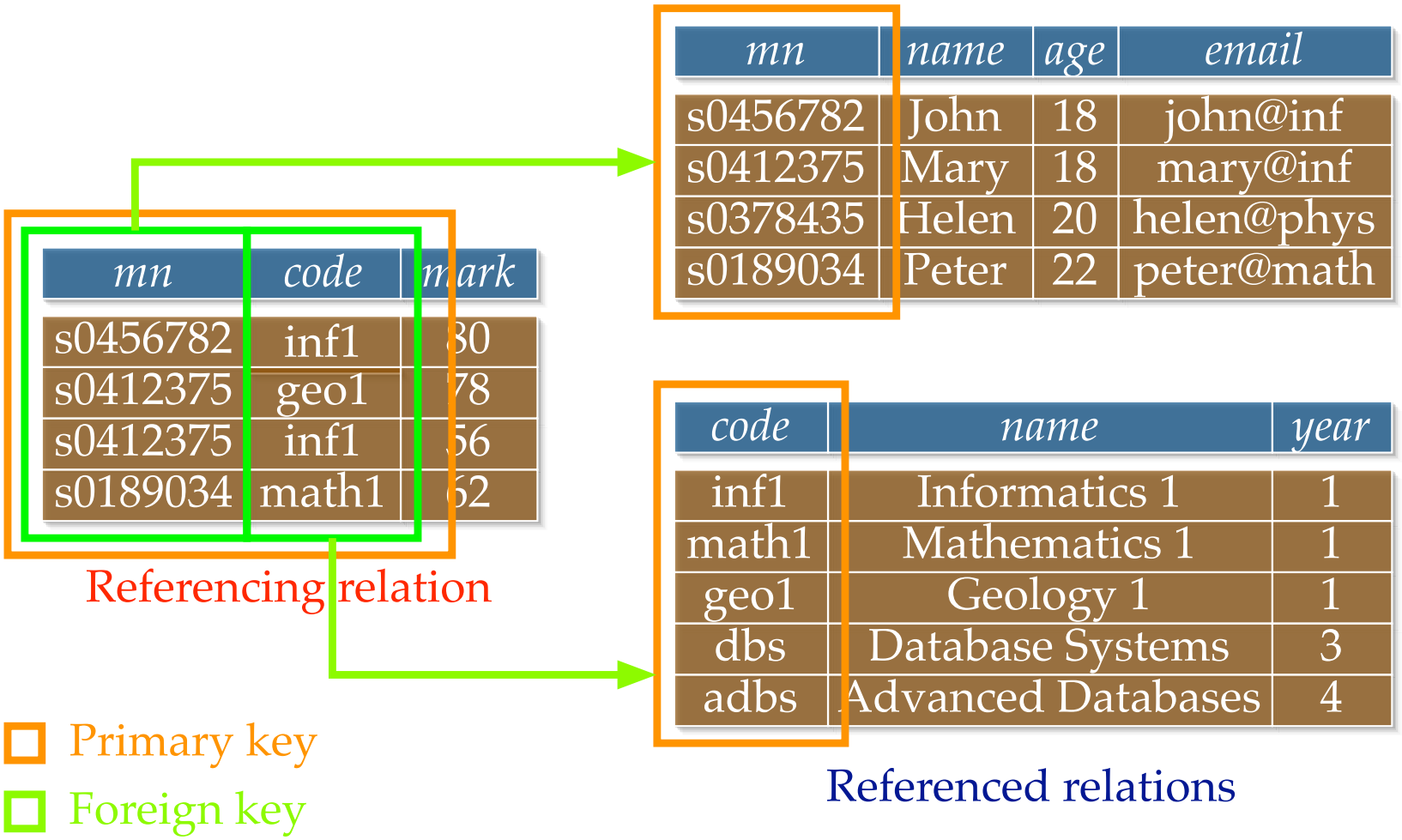
In this case, the primary key is a pair of fields.

The *foreign key constraints* 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 relationhip. In the relational model, *both* are (necessarily!) implemented as tables.

Translating an ER diagram to a relational schema

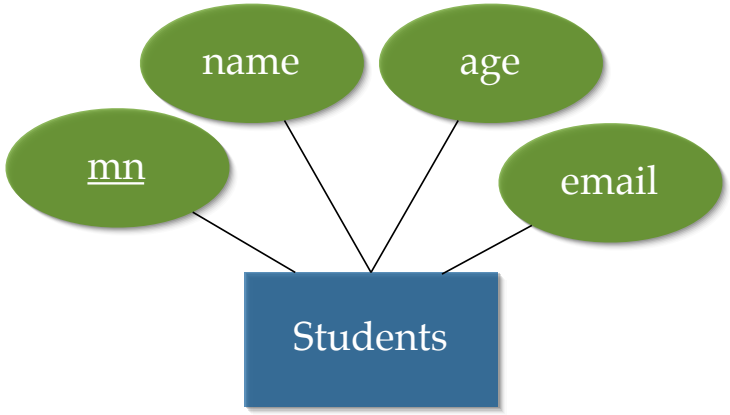
Given an ER diagram, we find a relational schema that closely aproximates the ER design.

The translation is *approximate* because it is not 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.

In D&A, we just consider a few examples illustrating some of the main ideas.

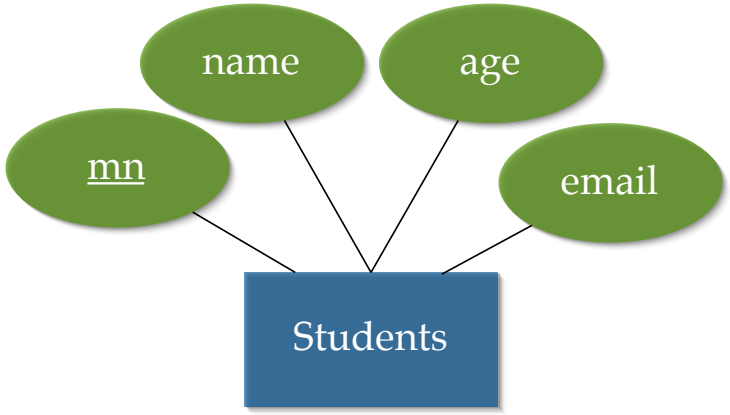
Mapping entity sets



Algorithm

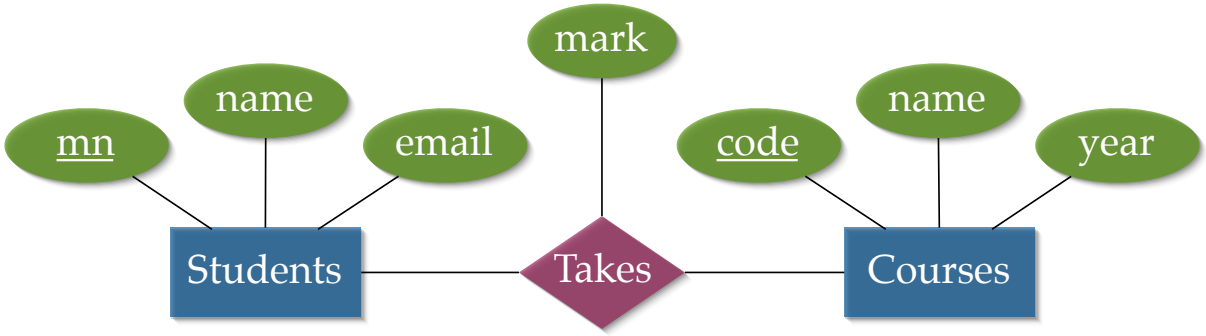
- A table is created for the entity set
- Each attribute of the entity set becomes an field of the table with an appropriate type
- A primary key is declared

Mapping entity sets




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

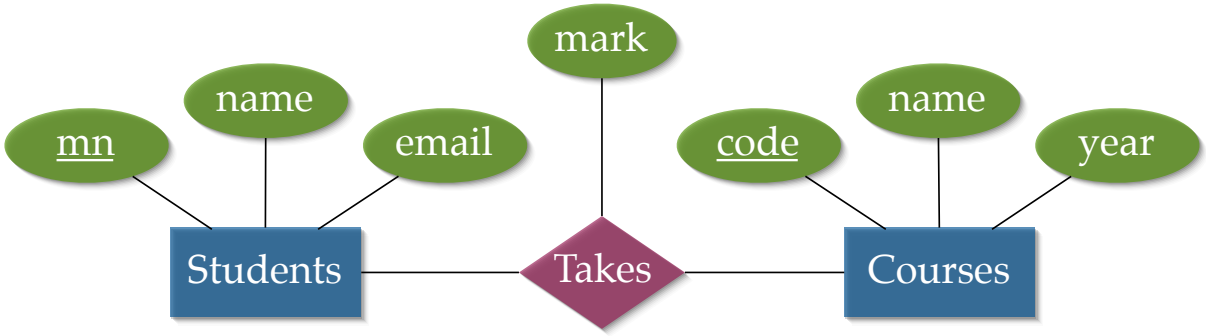
Mapping relationship sets (no key constraints)



Algorithm

- A table is created for the relationship set
- The table contains the primary keys of the participating entity sets
- Descriptive attributes of the relationship are added
- A composite primary key is declared on the table
- Foreign key constraints are declared

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

- A table is created for the relationship set
- The primary key of the "source" entity set is declared as the primary key of the relationship set
- Foreign key constraints are declared for both 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 )
```

N.B. The participation constraint on **Students** in **Directed_By** has not been implemented. To implement this constraint another approach is needed.

Null values

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

A **`null`** value means that a field is undefined or missing

Null values are *not allowed* to appear in *primary key* fields,

They *are allowed* to appear in *foreign key* fields.

Null values can be disallowed from other fields using a **`not null`** declaration

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

Mapping relationship sets with key+participation constraints

Algorithm

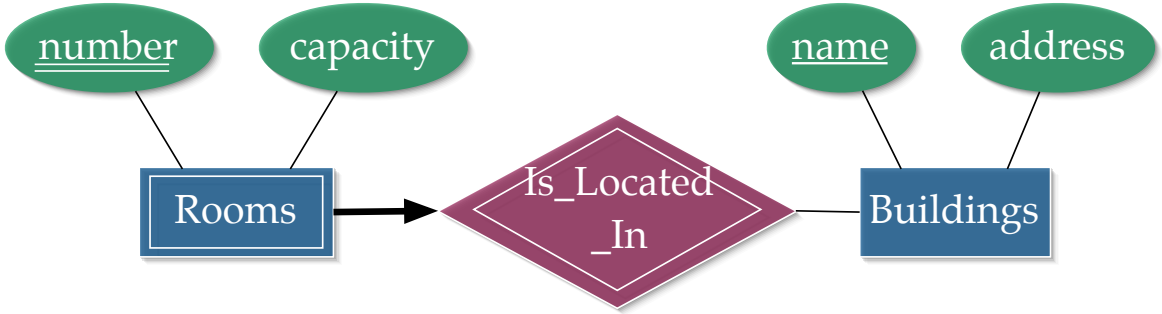
- Include a foreign key field for the "target" entity set within the table for the "source" entity set.
- Give this field a **`not null`** declaration.

N.B. By omitting the **`not null`** declaration, we obtain an alternative way of implementing the key constraint without the participation constraint.

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 )
```

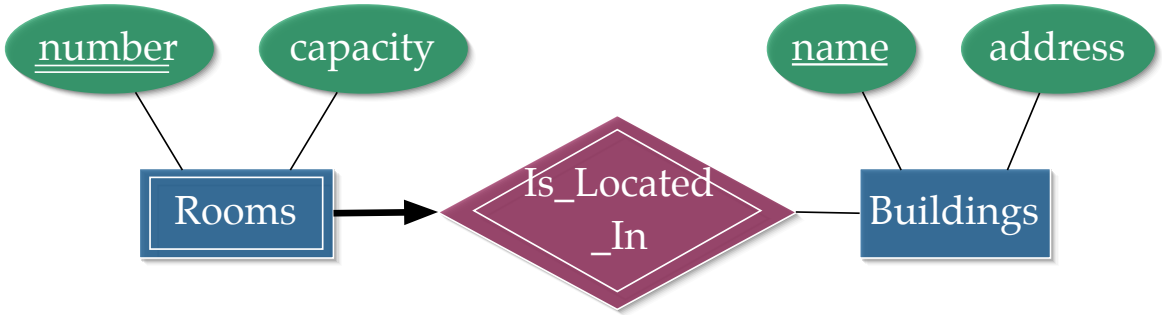
Mapping weak entity sets and identifying relationships



Algorithm

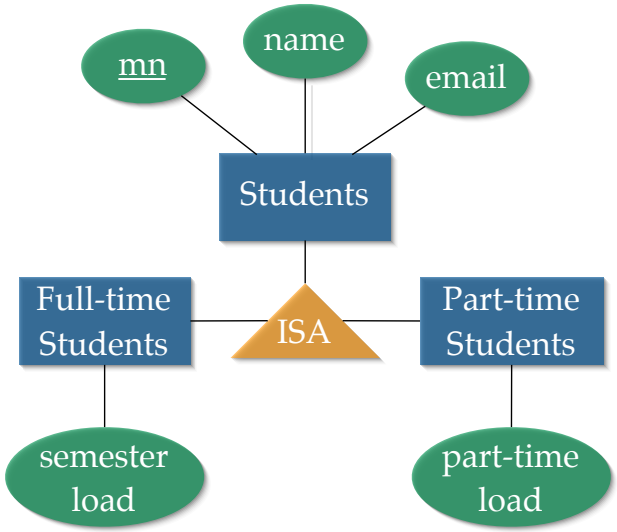
- Create a table for the weak entity set
- Add an attribute set, for the primary key of the entity set's identifying owner's
- Add a foreign key constraint on the identifying owners primary key
- Instruct the system to automatically delete any tuples in the table for which there are no owners

Mapping weak entity sets and identifying relationships



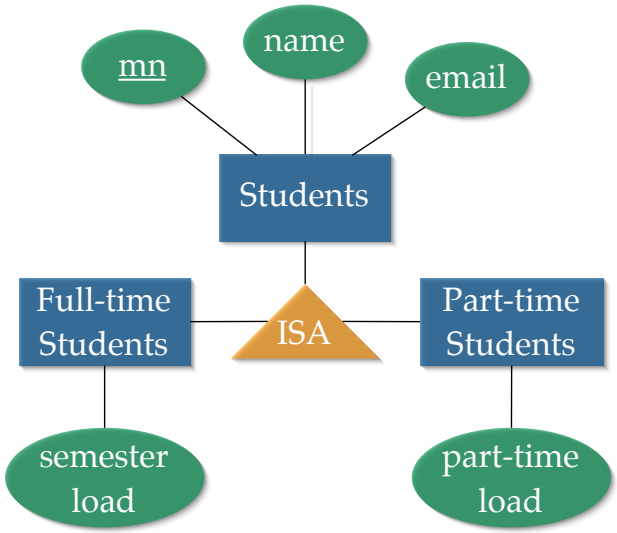

```
create table Rooms (
      number           char(8),
      capacity         integer,
      building_name    char(20),
      primary key      (number,building_name),
      foreign key      (building_name) references Buildings
                       on delete cascade )
```

Mapping hierarchical entities



- Declare a table for the superclass of the hierarchy
- For each subclass, declare another table, containing the superclass's primary key and the subclass's extra attributes
- Each subclass has the same primary key as its superclass
- Declare foreign key constraints

Mapping hierarchical entities




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