Translating an ER diagram to a relational schema

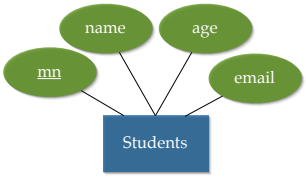
Given an ER diagram, we can look for a relational schema that closely approximates the ER design.

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.

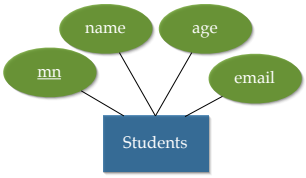
Mapping entity sets



Algorithm

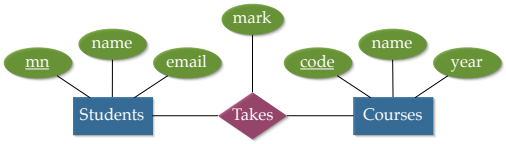
- Create a table for the entity set.
- Make each attribute of the entity set a field of the table, with an appropriate type.
- Declare the field or fields comprising the primary key

Mapping entity sets




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

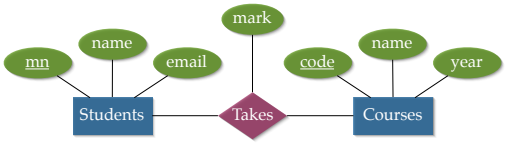
Mapping relationship sets (no key constraints)



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

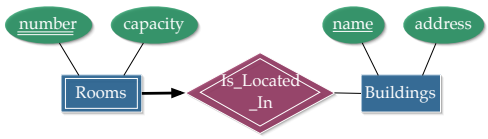
Algorithm

- 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 )
```

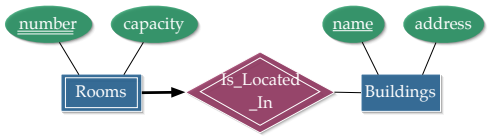
Mapping weak entity sets and identifying relationships



Algorithm

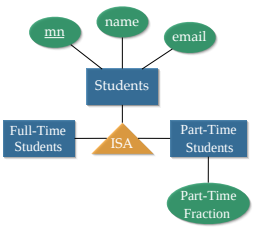
- Create a table for the weak entity set.
- Make each attribute of the weak entity set a field of the table.
- Add fields for the primary key attributes of the identifying owner.
- Declare a foreign key constraint on these identifying owner fields.
- Instruct the system to automatically delete any tuples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

Mapping weak entity sets and identifying relationships



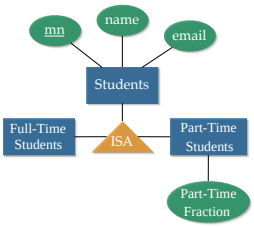

```
create table Rooms (
     number          char(8),
     capacity        integer,
     building_name   char(20),
     primary key     (number,building_name),
     foreign key     (building_name) references Buildings
                     on delete cascade )
```


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 )
```