CS 186 Discussion Section
Week 2

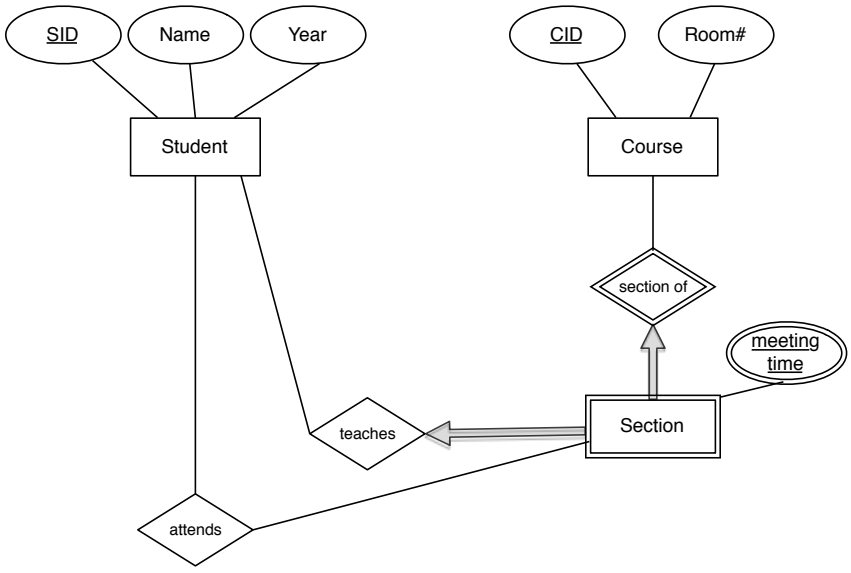
Peter Alvaro & Kuang Chen

February 6, 2009

1 Mapping the Entity-Relation model to the Relational Model

These section notes are brought to you by the letters E and R, and the number 2.

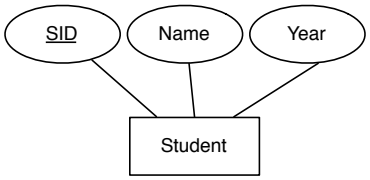
1.1



Recall our model of courses, students and discussion sections from last week. Although mapping is not a hard science, a small set of rules can guide our translation:

1. *Translating Entities*

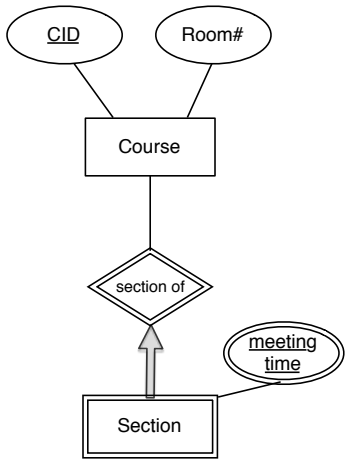
   An entity is translated directly into a table in a straightforward fashion:



```
create table student (
       SID int,
       name varchar(20),
       year int,
       CONSTRAINT pk_student PRIMARY KEY (SID)
);
```

2. *Translating Weak Entities*

   A weak entity is also converted into a table, borrowing they key from its identifying relationship such that the composite key of identifying key, partial key form the primary key of the new table (again, imagine that "meeting time" is underlined with a dotted line in the ER model below):



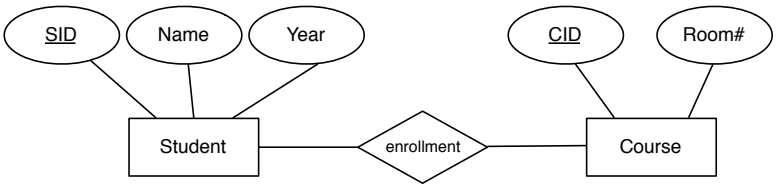

```
create table course (
    CID int,
    room_number int,
    constraint pk_course PRIMARY KEY (CID)
);

create table section (
    CID int,
    meeting_time int,
    constraint pk_section PRIMARY KEY (CID, meeting_time),
    constraint fk_course FOREIGN KEY (CID) references course
);
```

3. *Translating Relations*

   This is a bit more subtle. It is easiest to think about this in terms of another subset of rules:

   (a) *Many-to-many Relationships*

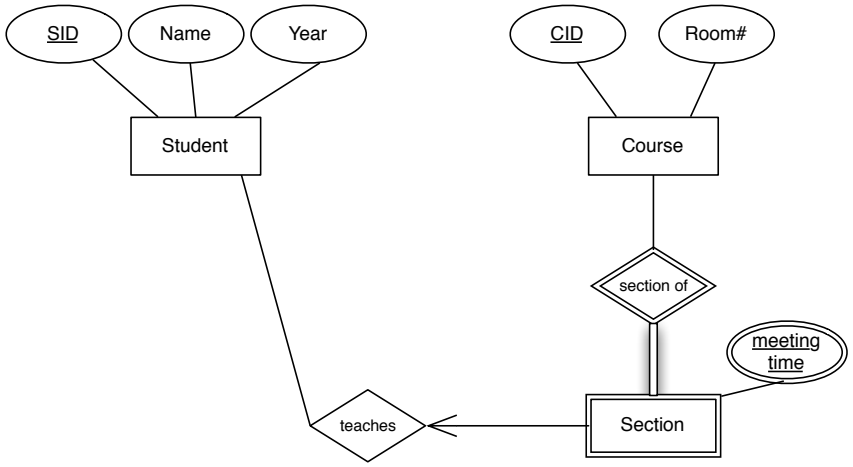
   A many-to-many relationship is modeled as a new table, whose primary key is the composite of the primary keys of the entities that it relates:




```
create table enrollment (
    SID int,
    CID int,
    constraint pk_enrollment PRIMARY KEY (SID, CID),
    constraint fk_student FOREIGN KEY (SID) references student,
    constraint fk_course FOREIGN KEY (CID) references course
);
```

(b) *Many-to-one Relationships*

There are two ways to express a many-to-one relationship like the one below (not a great model, as it allows a section to have no TA, but bear with us).



i. *with a relation*

We can create a relation as in (1) above. To enforce the many-1 constraint, we must limit the primary key to only include the many-entity:

```
create table teaches (
    TA int,
    CID int,
    meeting_time int,
    constraint pk_enrollment PRIMARY KEY (CID, meeting_time),
    constraint fk_student FOREIGN KEY (TA) references student,
    constraint fk_course FOREIGN KEY (CID) references course
);
```

Since a section (uniquely identified by a CID, meeting time pair) may only appear once in this table, we have prevented a course from having more than one TA.

ii. *by rolling the 1-entity into the many-entity*

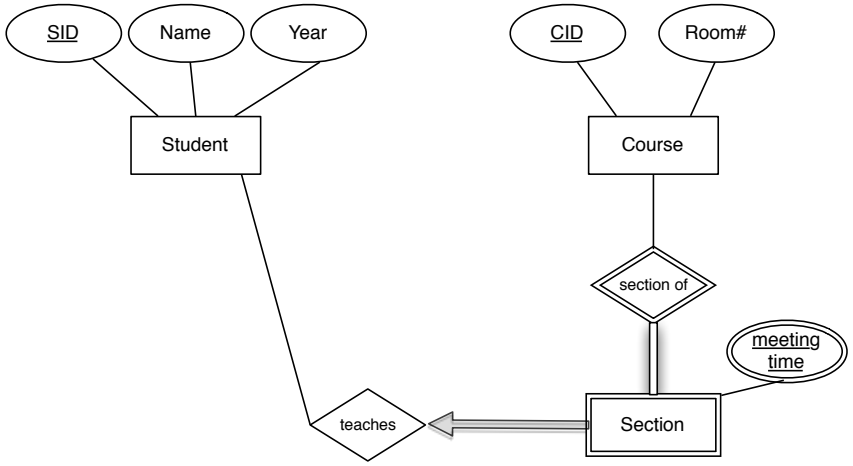
It is often preferable to avoid creating another table altogether. We can express the same constraint by simply adding a TA column to the section entity, which is a foreign key to student.

```
create table section (
    CID int,
    meeting_time int,
    TA int,
    constraint pk_section PRIMARY KEY (CID, meeting_time),
    constraint fk_course FOREIGN KEY (CID) references course,
    constraint fk_ta FOREIGN KEY (TA) references student
);
```

See what has happened? There is only one slot in which to put a student id representing the TA, so we're able to enforce the constraint within the section table.

(c) *Participation Constraints*

If we have a constraint that is participation AND 1-many, the problem is easily solved by adding a NOT NULL constraint to the TA column in Approach 2 above:




```
create table teaches (
    TA int NOT NULL,
    CID int,
    meeting_time int,
    constraint pk_enrollment PRIMARY KEY (CID, meeting_time),
    constraint fk_student FOREIGN KEY (TA) references student,
    constraint fk_course FOREIGN KEY (CID) references course
);
```

By making the TA column NOT NULL, we prevent a section from not having a specified TA.

If, on the other hand, we have a general participation constraint (many-many), the problem is much more complicated. Clearly Approach 2 won't work, because we can't store multiple students in the TA column of the section table.

Unfortunately, Approach 1 won't work either, because nothing about how the teaches table is defined (and nothing we can do to it!) will prevent the scenario where a section exists for which no row exists in teaches with that (CID, section) pair. The constaint we want can be phrased as "for every section

defined in the section table, there MUST be at least one row in the teaches table for that section, with a non-null TA". Note that this (English) constraint references more than one table, while integrity contraints pertain to only a single table.

More sophisticated technique including check constraints and triggers can be used to solve this problem in a relational database implementation. We'll discuss these later in the course.