1 INTRODUCTION

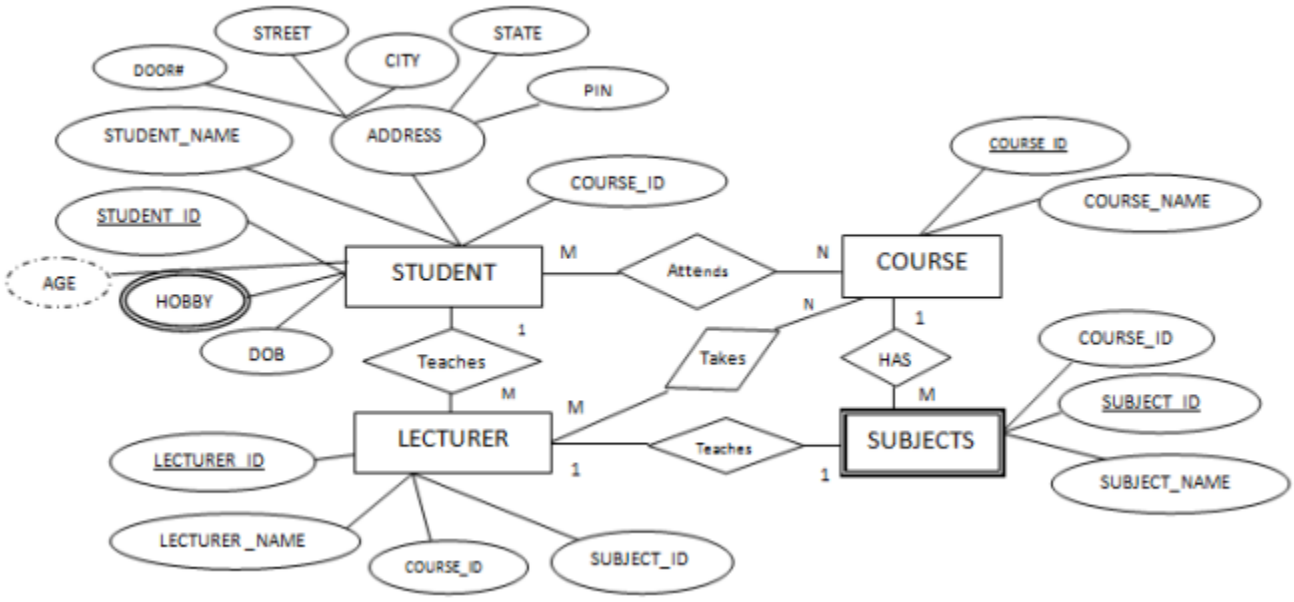
As the database grows, the ER diagram representation becomes more complex and crowded. It is creates a difficult situation to understand the requirement and their structure as a whole. Similarly, if ER diagram is represented at very high level, it again creates a difficulty in understanding the system. But representation at high level and till the minute levels is very necessary to understand the system well. These concepts are well defined by generalization and specialization. Sometimes, we would have divided the entities into two or more entities to be more accurate in design. But when compared to the whole database or user, it can be combined to one entity. Such a process is called as aggregation.

Once designing ER diagram is complete, we need to put it into logical structure. But how it can be done? Let us discuss this in the last section.

2 TRANSFORM ER DIAGRAM INTO TABLES

Since ER diagram gives us the good knowledge about the requirement and the mapping of the entities in it, we can easily convert them as tables and columns. i.e.; using ER diagrams one can easily created relational data model, which nothing but the logical view of the database.

There are various steps involved in converting it into tables and columns. Each type of entity, attribute and relationship in the diagram takes their own depiction here. Consider the ER diagram below and will see how it is converted into tables, columns and mappings.



The basic rule for converting the ER diagrams into tables is

- Convert all the Entities in the diagram to tables.

All the entities represented in the rectangular box in the ER diagram become independent tables in the database. In the below diagram, STUDENT, COURSE, LECTURER and SUBJECTS forms individual tables.

- All single valued attributes of an entity is converted to a column of the table

All the attributes, whose value at any instance of time is unique, are considered as columns of that table. In the STUDENT Entity, STUDENT_ID, STUDENT_NAME form the columns of STUDENT table. Similarly, LECTURER_ID, LECTURER_NAME form the columns of LECTURER table. And so on.

- Key attribute in the ER diagram becomes the Primary key of the table.

In diagram above, STUDENT_ID, LECTURER_ID, COURSE_ID and SUB_ID are the key attributes of the entities. Hence we consider them as the primary keys of respective table.

- Declare the foreign key column, if applicable.

In the diagram, attribute COURSE_ID in the STUDENT entity is from COURSE entity. Hence add COURSE_ID in the STUDENT table and assign it foreign key constraint. COURSE_ID and SUBJECT_ID in LECTURER table forms the foreign key column. Hence by declaring the foreign key constraints, mapping between the tables are established.

- Any multi-valued attributes are converted into new table.

A hobby in the Student table is a multivalued attribute. Any student can have any number of hobbies. So we cannot represent multiple values in a single column of STUDENT table. We need to store it separately, so that we can store any number of hobbies, adding/ removing / deleting hobbies should not create any redundancy or anomalies in the system. Hence we create a separate table STUD_HOBBY with STUDENT_ID and HOBBY as its columns. We create a composite key using both the columns.

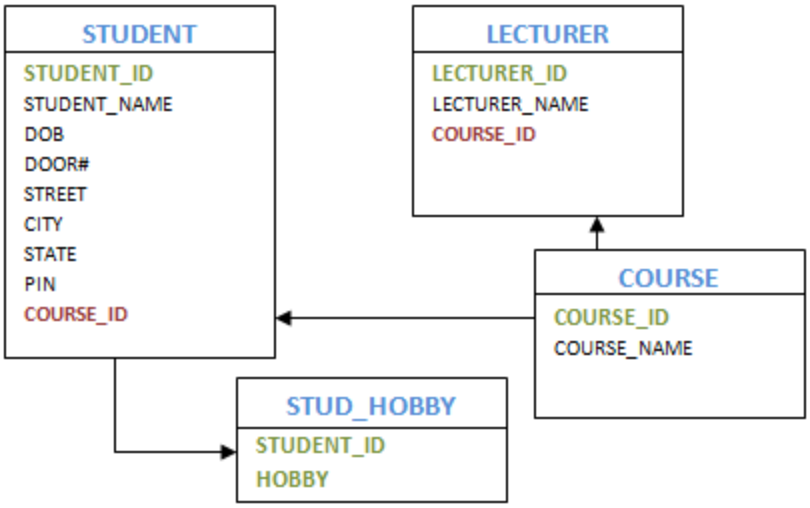
- Any composite attributes are merged into same table as different columns.

In the diagram above, Student Address is a composite attribute. It has Door#, Street, City, State and Pin. These attributes are merged into STUDENT table as individual columns.

- One can ignore derived attribute, since it can be calculated at any time.

In the STUDENT table, Age can be derived at any point of time by calculating the difference between DateOfBirth and current date. Hence we need not create a column for this attribute. It reduces the duplicity in the database.

These are the very basic rules of converting ER diagram into tables and columns, and assigning the mapping between the tables. Table structure at this would be as below:

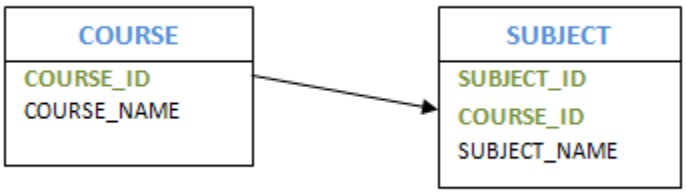


Let us see some of the special cases.

2.1 Converting Weak Entity

Weak entity is also represented as table. All the attributes of the weak entity forms the column of the table. But the key attribute represented in the diagram cannot form the primary key of this table. We have to add a foreign key column, which would be the primary key column of its strong entity. This foreign key column along with its key attribute column forms the primary key of the table.

In our example above, SUBJECTS is the weak entity. Hence, we create a table for it. Its attributes SUBJECT_ID and SUBJECT_NAME forms the column of this table. Although SUBJECT_ID is represented as key attribute in the diagram, it cannot be considered as primary key. In order to add primary key to the column, we have to find the foreign key first. COURSE is the strong entity related to SUBJECT. Hence the primary key COURSE_ID of COURSE is added to SUBJECT table as foreign key. Now we can create a composite primary key out of COURSE_ID and SUBJECT_ID.



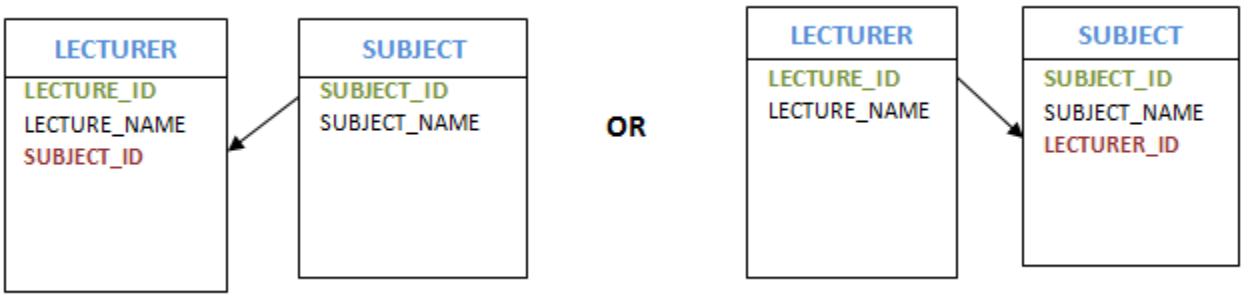
2.2 Representing 1:1 relationship

Imagine SUBJECT is not a weak entity, and we have LECTURER teaches SUBJECT relation. It is a 1:1 relation. i.e.; one lecturer teaches only one subject. We can represent this case in two ways

1. Create table for both LECTURER and SUBJECT. Add the primary key of LECTURER in SUBJECT table as foreign key. It implies the lecturer name for that particular subject.

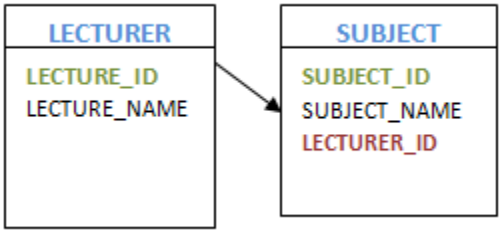
2. Create table for both LECTURER and SUBJECT. Add the primary key of SUBJECT in LECTURER table as foreign key. It implies the subject taught by the lecturer.

In both the case, meaning is same. Foreign key column can be added in either of the table, depending on the developer's choice.



2.3 Representing 1:N relationship

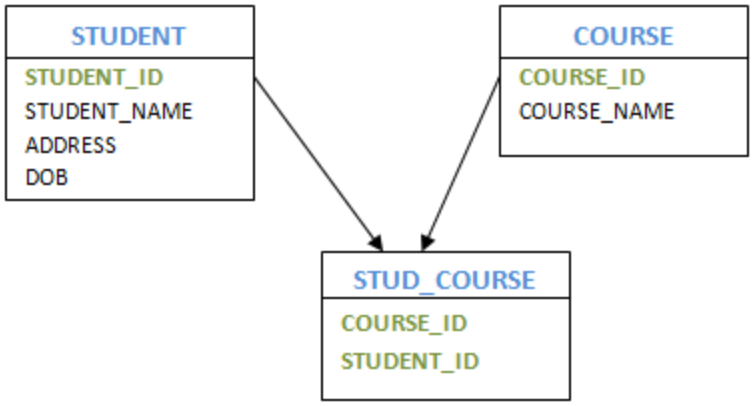
Consider SUBJECT and LECTURER relation, where each Lecturer teaches multiple subjects. This is a 1: N relation. In this case, primary key of LECTURER table is added to the SUBJECT table. i.e.; the primary key at 1 cardinality entity is added as foreign key to N cardinality entity



2.4 Representing M:N relationship

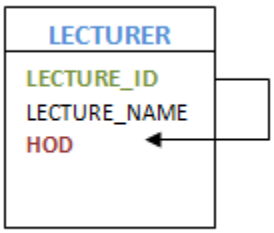
Consider the example, multiple students enrolled for multiple courses, which is M:N relation. In this case, we create STUDENT and COURSE tables for the entities. Create one more table for the relation 'Enrolment' and name it as STUD_COURSE. Add the primary keys of COURSE and STUDENT into it, which forms the composite primary key of the new table.

That is, in this case both the participating entities are converted into tables, and a new table is created for the relation between them. Primary keys of entity tables are added into new table to form the composite primary key. We can add any additional columns, if present as attribute of the relation in ER diagram.



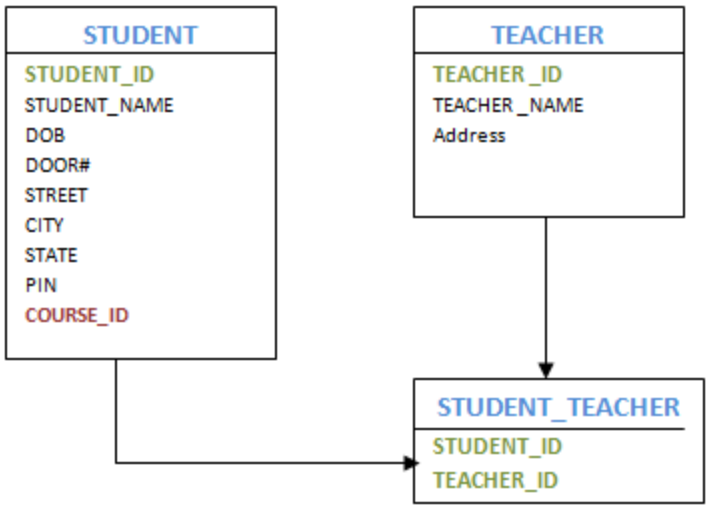
2.5 Self Referencing 1:N relation

Consider the example of HOD and Lecturers. Here one of the Lecturers is a HOD of the department. i.e.; one HOD has multiple lecturers working with him. In this case, we create LECTURER table for the Lecturer entity. Create the columns and primary keys as usual. In order to represent HOD, we add one more column to LECTURER table which is same column as primary key, but acts as a foreign key. i.e.; LECTURER_ID is the primary key of LECTURER table. We add one more column HOD, which will have LECTURER_ID of the HOD. Hence LECTURER table will show HOD's Lecturer ID for each Lecturer. In this case, primary key column acts as a foreign key in the same table.



2.6 Self Referencing M:N relation

Consider Student and Teacher example as 'Manny students have Many Teachers teaching the subjects'. Here relation between Student and Teacher is M:N. In this case, create independent tables for student and teacher, and set their primary keys. Then we create a new table for the relationship 'have' as STUDENT_TEACHER, which will have student and teacher combination, and any other columns if applicable. Basically, student-teacher combination is the two primary key columns from respective tables, hence establishing the relationship between them. Both the primary keys from both tables act as a composite primary key in the new table. This reduces the storing of redundant data and consistency in the database.


STUDENT
STUDENT_ID
STUDENT_NAME
DOB
DOOR#
STREET
CITY
STATE
PIN
COURSE_ID
TEACHER
TEACHER _ID
TEACHER _NAME
Address
STUDENT_TEACHER
STUDENT_ID
TEACHER_ID