Data Modelling

Relational set operators:

The data in relational tables are of limited value unless the data can be manipulated to generate useful information.

Relational Algebra defines the theoretical foundation of manipulating table content using the eight relational operators:

SELECT, PROJECT, JOIN, INTERSECT, UNION, DIFFERENCE, PRODUCT, AND DIVIDE.

The relational operators have the property of closure that is applying operators on relations produce relations.

1. SELECT, also known as RESTRICT, yields values for all the rows found in a table that satisfy a given condition. SELECT yields a horizontal subset of a table.
2. PROJECT yields all values for selected attributes. PROJECT yields a vertical subset of a table
3. UNION: combines all rows from two or more tables, excluding duplicate rows. In order to be used in a UNION, the tables must have the same attribute characteristics, that it the attributes and their domains must be compatible. When two or more tables share the same number of columns and when their corresponding columns share the same or compatible domains, they are said to be union-compatible.

4. INTERSECT: yields only the rows that appear in both tables. As with UNION, the tables must be union-compatible to yield valid results.
5. DIFFERENCE: yields all rows in one table that are not found in the other table, that is, it subtracts one table from the other. As with the UNION, the tables must be UNION-compatible to yield valid results.
6. PRODUCT: yields all possible pairs of rows from two tables- also known as Cartesian product. Therefore, if one table has six rows and the other table has three, the PRODUCT yields a list composed of 6 x 3= 18 rows.
7. JOIN: allows information to be combined from two or more tables. JOIN allows the use of independent tables linked by common attributes.

   A natural join links tables be selecting only the rows with common values in their common attributes. A natural join is the result of a three stage process:

   a. First, a PRODUCT of the tables is created
   b. Second, a SELECT is performed on the output of Step a) to yield only the rows whose values are equal.
   c. A PROJECT is performed on the results of Step b to yield a single copy of each attribute, thereby eliminating duplicate columns.

   The final outcome of a natural join yields a table that does not include unmatched pairs and provides only copies of the matches.

Relationships within the Relational Database:

A relationship describes association among entities. For example, a relationship exists between customers and an agent, in that an agent can serve many customers and each customer may be served by only one agent. Data Models use three types of relationships: one-to-many, many-to-many, and one-to-one. Database designers usually use these notations: 1: M or 1..*, M:N or ".." or 1:1 or 1..1 respectively.

- One-to-many (1:M or 1..*) relationship: A painter creates many different paintings, but each painting is only made by one painter. Thus the painter is the "one" is related to the paintings (the "many"). Therefore "PAINTER paints PAINTINGS" is represented as 1:M.
- Many-to-many (M:N or "..") relationship: An employee may learn many job skills and each job skill may be learned by many employees. Thus, "EMPLOYEE learns SKILL" as M:N.
- One-to-one (1:1 or 1..1) relationship: A retail company's management structure may require that each store be managed by a single employee. In turn, each manager can only manage a single store. Therefore, the relationship " EMPLOYEE manages STORE" is labeled 1:1
- The 1:M relationship is the relational modeling ideal. Therefore, this relationship type should be the norm in any relationship database design.
- The 1:1 relationship should be rare in any relational database design

- M:N relationships cannot be implemented as such in the relational model, they have to be changed into two 1:M relationships.

The 1:M Relationship: The 1:M relationship is the norm for relational databases. For example, one painter usually has many paintings. Each painting was painted by only one painter, but a painter could have many paintings.

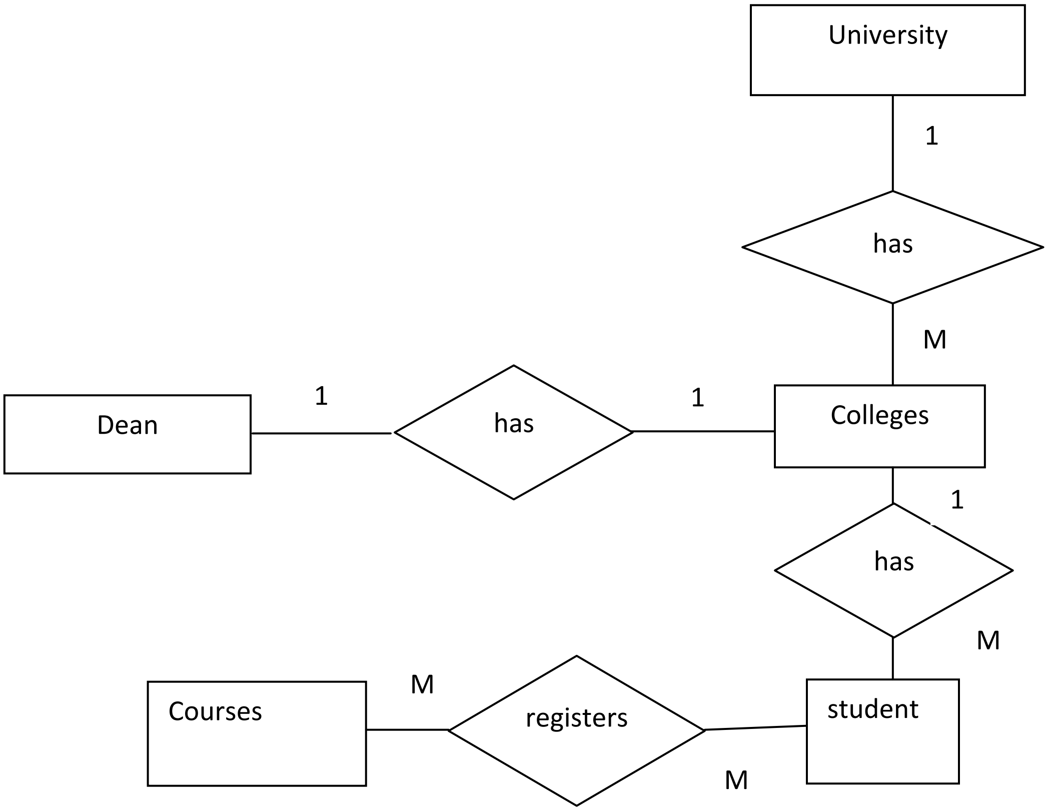
The 1:1 Relationship:
One entity in a 1:1 relationship can only be related to one other entity and vice versa. For example, every department has only one Chair and each chair can only head one department.
1:1 relationships should be rare.

The M:N Relationship
A many to many (M:N) relationship is not supported directly in the relational environment. They are usually implemented by creating a new entity in 1:M relationships with the original entities.
Example: Each CLASS is taken by many students, and each STUDENT can take many CLASSes.
There may be many rows in the CLASS table for any given row in the STUDENT table. Additionally, there can be many rows in the STUDENT table for any given row in the CLASS table.

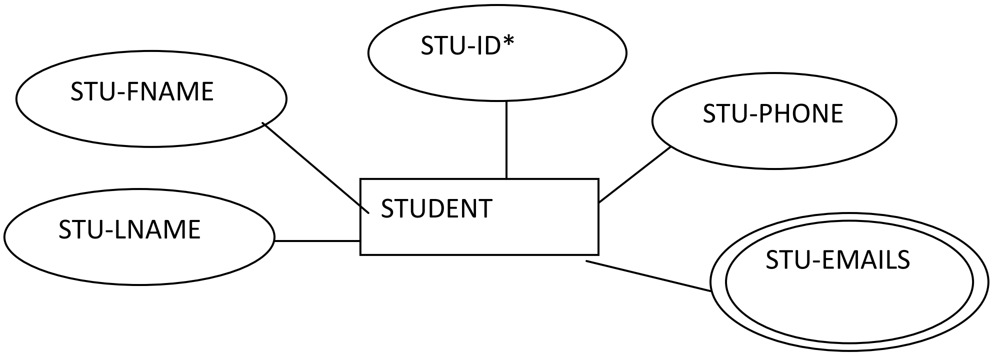
M:N relations create a lot of redundancy , in that the same tuple occurs many times in a given table, so tuples and their attributes are repeated many times, occupying space and leading to errors and efficiency problems.

Bridge Entity :
The problem inherent to the many-to-many relationship can be avoided by creating composite entity , also called bridge entity or associative entity. Such tables are used to link the tables that were originally related in an M: N relationship. The composite entity structure includes –as foreign keys- at least the primary keys of the tables that are to be linked.
The designer has two options when defining a composite table primary key: use a combination of those foreign keys or create a new primary key.

Data Modelling Representation:
Graphical representation of entities and their relationships in a database structure is popular because it complements the relational data model concepts. ER models are represented in an entity relationship diagram(ERD), which uses graphical representations to model database components.
There are 3 types of notation: the original Chen notation, the Crow's Foot notation and the class diagram notation which is part of the Unified Modeling Language(UML)



Chen Notation for relationships between entities



Chen Notation for entities

In Chen notation, an entity is represented by a rectangle box, enclosing the entity name. The attributes of the entity are enclosed in ovals and are attached to their entity type by straight lines. Composite attributes are attached to their component attributes by straight lines. Multivalued attributes are displayed in double ovals.

Components used in the creation of an ERD:

Crow Foot Notation:

Entity: A person, place or thing about which we want to collect and store multiple instances of data. It has a name, which is a noun, and attributes which describe the data we are interested in storing. It also has an identifier, which uniquely identifies one instance of an entity. The attribute which acts as the identifier is marked with an asterisk.

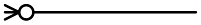
Relationship: Illustrates an association between two entities. It has a name which is a verb. It also has cardinality and modality.

is assigned

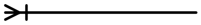
Maximal cardinality and **minimal cardinality** are the indicators of the associations around a relationship.

- Maximal Cardinality refers to the maximum number of times an instance in one entity can be associated with instances in the related entity.
- Maximal Cardinality can be 1 or Many and the symbol is placed on the outside ends of the relationship line, closest to the entity,
- For a cardinality of 1 a straight line is drawn.
- For a cardinality of Many a foot with three toes is drawn.

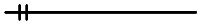
- Modality or minimal cardinality refers to the minimum number of times an instance in one entity can be associated with an instance in the related entity.
- Modality can be 1 or 0 and the symbol is placed on the inside, next to the cardinality symbol.
- For a modality of 1 a straight line is drawn.
- For a modality of 0 a circle is drawn.



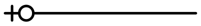
zero or more



1 or more

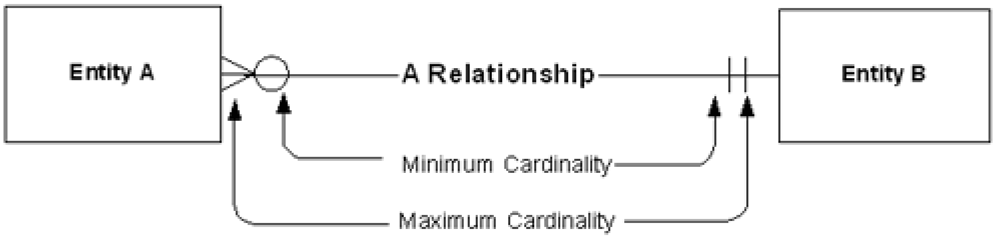


1 and only 1 (exactly 1)

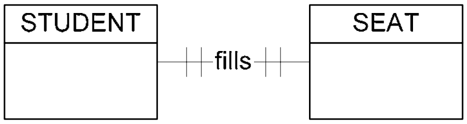


zero or 1

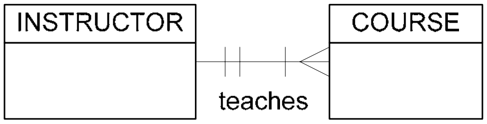
Cardinality and modality are indicated at both ends of the relationship line. Once this has been done, the relationships are read as being 1 to 1 (1:1), 1 to many (1:M), or many to many (M:N).



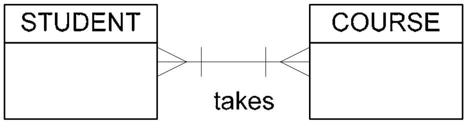
1:1



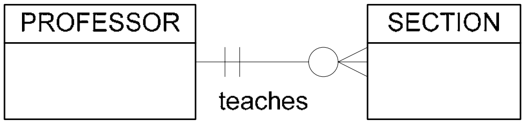
1:M



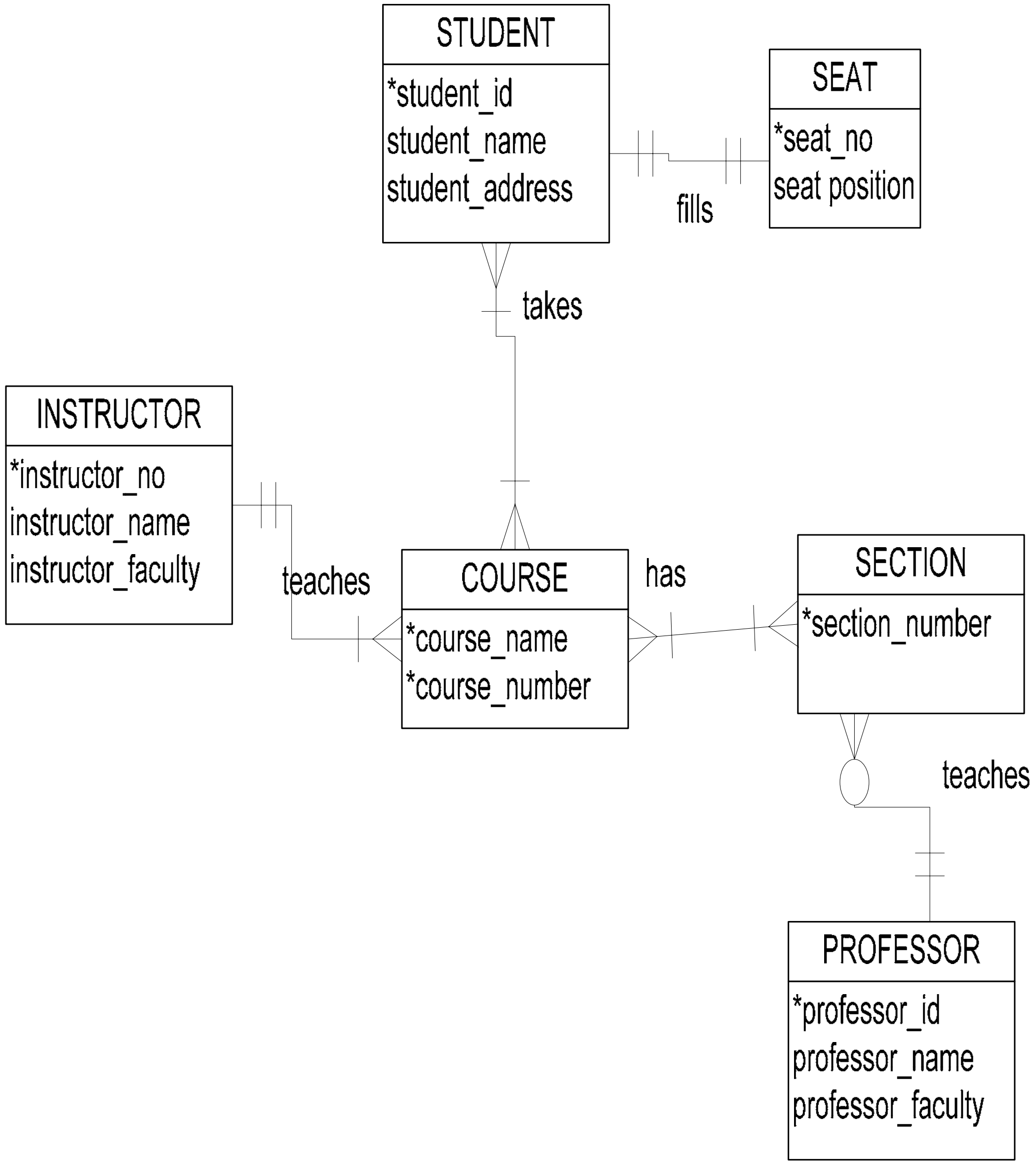
M:M



1:M

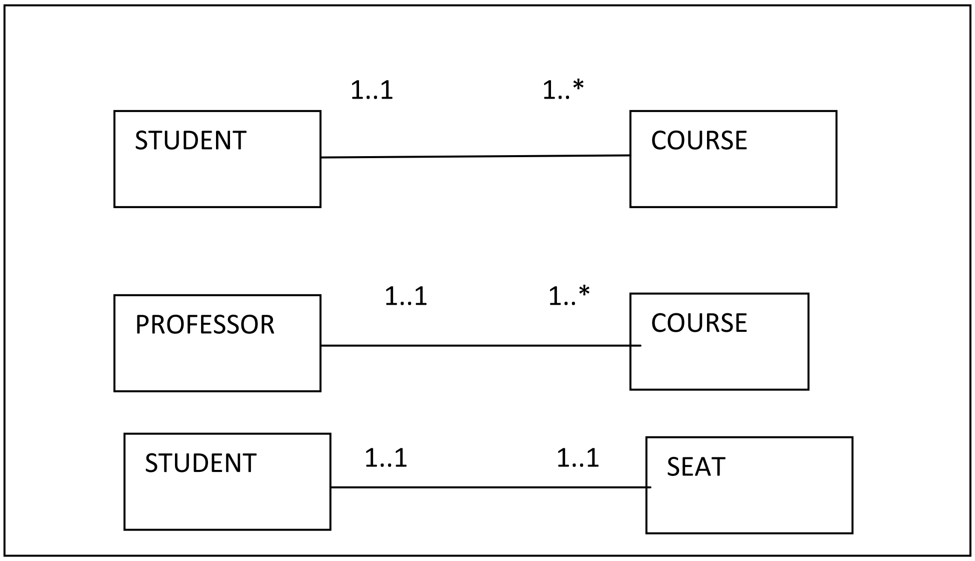


Typically, ERDs are much more complex than this, involving quite a number of entities and relationships. If we join all of the above relationships together and add a few attributes, a small collection of data might be depicted in the following way using Crow’s Foot Notation:



(from http://www2.cs.uregina.ca/~bernatja/crowsfoot.html)

UML Diagram:

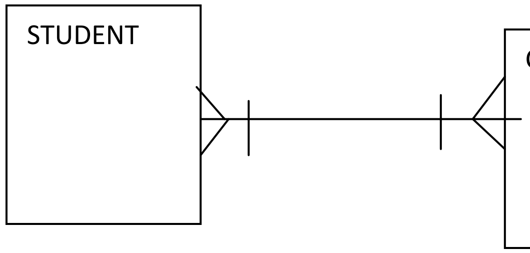


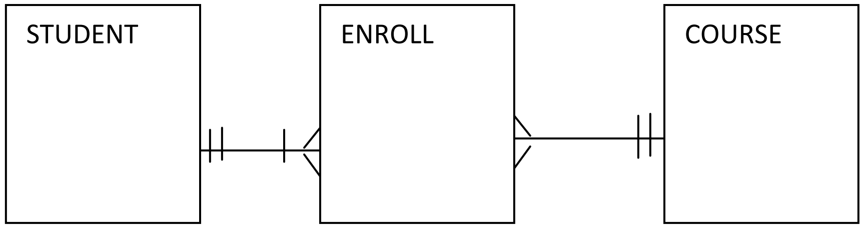
Changing the M: N relationship to two 1: M relationships:

The M:N relationship holds between two entities E1 and E2 in which each tuple from E1 is in relation with many entities of E2. Conversely, many one entity from E2 is also in relation with many entities from E2. For example, a STUDENT tuple Si takes several courses and so is linked with several tuples, C1, C2, ..Cn from COURSE and a COURSE Ci is taken by several students S1, S2, .. Sn.

In order to represent this relationship in a relational database, we need to introduce a new bridge entity or composite entity ENROLL to link the tables COURSE and STUDENT. In this example, the ENROLL table primary key is the combination of its foreign keys COURSE_NUMBER and STUDENT_ID.

Another option would be to create a single-attribute new primary key, using a different value to identify each ENROLL table row uniquely.





The composite entity represented by ENROLL must contain at least the primary keys of the COURSE and STUDENT entities for which it serves as connector. Now the COURSE and STUDENT table only contain one row per entity. The ENROLL table contains multiple occurrences of the foreign key values, but these are controlled redundancies that should not produce anomalies as long as referential integrity constraints are enforced. Additional attributes may be added to ENROLL as needed, for example, ENROLL_GRADE .

Data Redundancy:

Data redundancy leads to data anomalies, which can destroy the effectiveness of the database. Relational databases make it possible to control data redundancies by using common attributes that are shared by tables, or foreign keys.

The proper use of foreign keys is important in controlling data redundancy but it cannot eliminate it because foreign key values are repeated many times in the tables.

The real test for redundancy is not how many copies of a given attribute are stored, but whether the elimination of an attribute will eliminate information. Therefore, if you delete an attribute and the original information can still be generated through relational algebra, then the inclusion of the attribute is redundant.