REPRESENTING RELATIONSHIPS ON E-R DIAGRAMS

In this lesson, we'll be talking about the following:

1. What is the difference between a **strong** and **weak** relationship?
2. What are **participation** and **cardinality** constraints?
3. How can **M:N** (many-to-many) relationships be converted into 1:M (one-to-many) relationships?
4. What is the difference between **unary, binary,** and **ternary relationship?**

Relationships are the glue that holds the tables together. They are used to connect related information between tables. Relationships can be either strong or weak. They are further classified based on their **participants** (which entities are related?), their **connectivity** (is the relationship one-to-many, one-to-one, or many-to-one) and optionally their **cardinality** (a measure of the precise minimum and maximum number of entities on "each side" of the relationship). Finally, they may be classified by **degree,** or the number of entity types that are connected by the relationship.

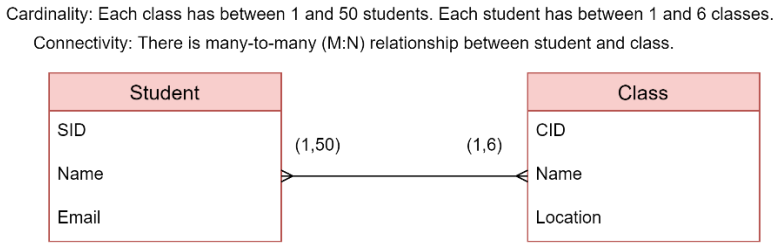
RELATIONSHIP STRENGTH

Relationship strength is based on how the primary key of a related entity is defined.

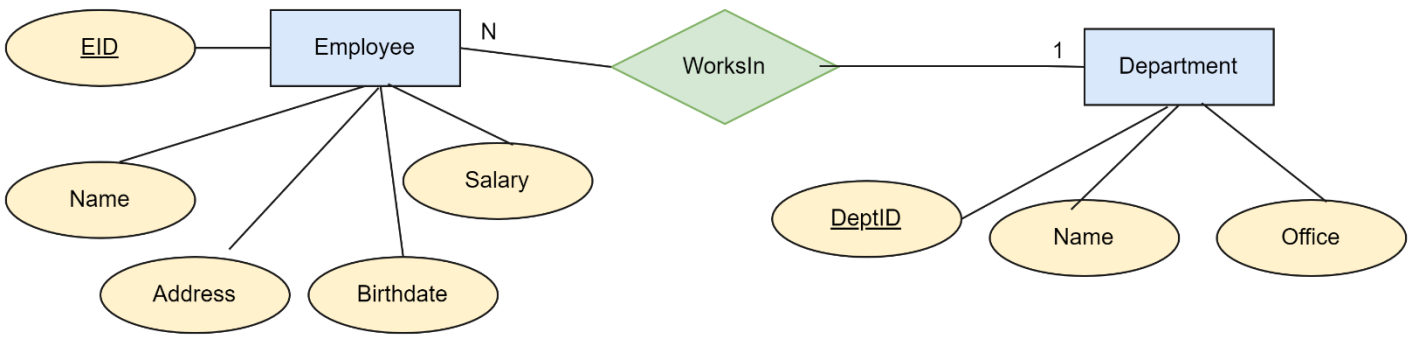
1. A **weak, or non-identifying, relationship** exists if the primary key of the related entity does NOT contain a primary key component of the parent entity. In the Company database, for example, the Order is only "weakly" identified by customer (since the CustID is not part of the primary key for Order):
    a. Customer(**CustID**, CustName)
    b. Order(**OrderID**, CustID, Date)
2. A **strong, or identifying, relationship** exists when the primary key of the related entity contains the primary key component of the parent entity. So, for example, the Class would be strongly identified by Course:
    a. Course(**CrsC ode**, DeptCode, Description)
    b. Class(**CrsCode, Section**, ClassTime…)

RELATIONSHIP CONNECTIVITY AND CARDINALITY

The **cardinality** of a relationship is a measure of "how many" entities can be involved on either side of the relationship. So, for example, if each Student must take between 1 and 4 courses, but a course may enroll between 1 and 50 students. This is often left out of diagrams in favor of the more general **connectivity** (which can be 1:1, 1:M, or M:M). So we have the following in Crow's Foot notation:

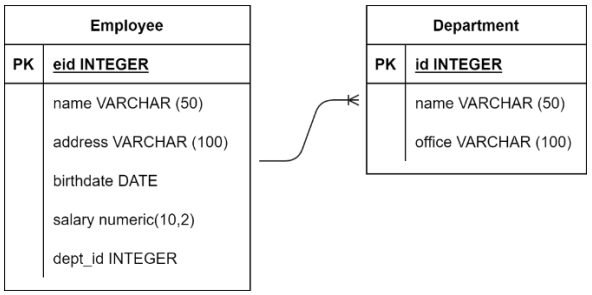


A **one-to-many (1:M)** relationship should be the norm in any relational database design and is found in all relational database environments. For example, one department has many employees. The figure below shows the relationship of one of these employees to the department.

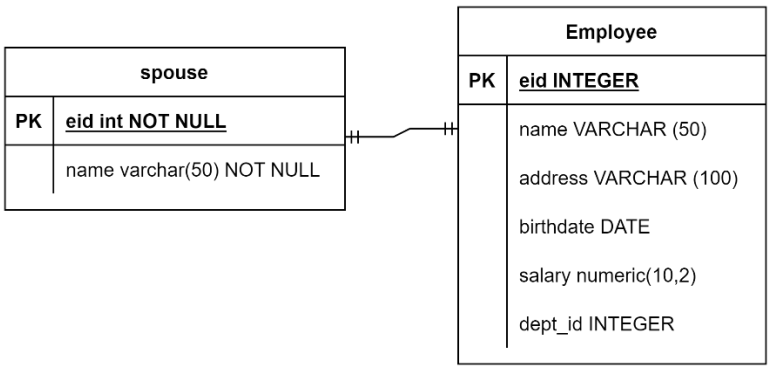


A **one-to-one (1:1)** relationship is the relationship of one entity to only one other entity, and vice versa. It should be rare in any relational database design. In fact, it could indicate that two entities actually belong in the same table. An example from the COMPANY database is one employee is associated with one spouse, and one spouse is associated with one employee.

Crow's Foot: One-to-Many Relationship



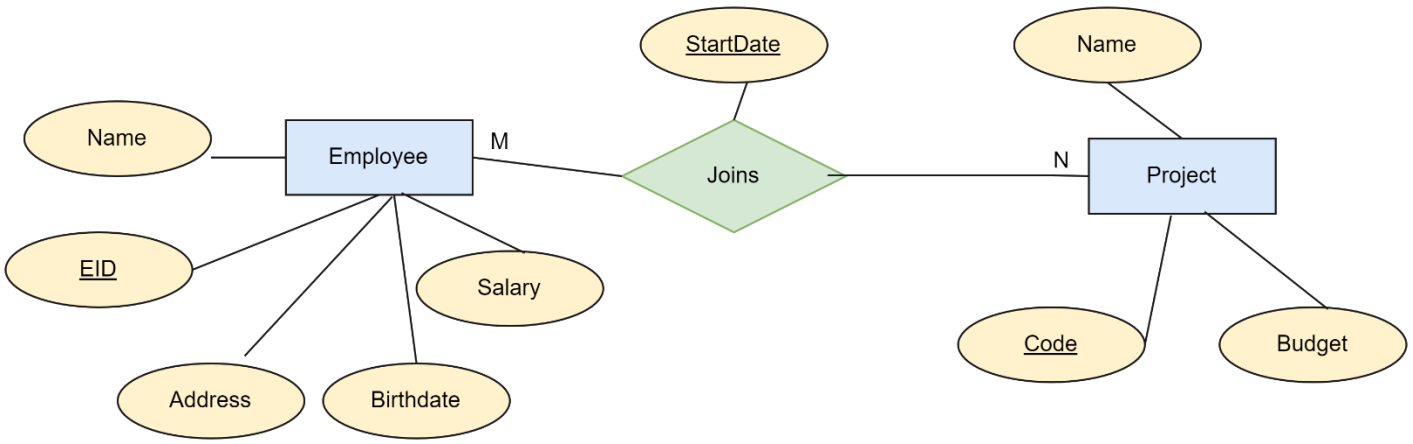
Crow's Foot: One-to-One



Many-to-many (M:N) relationships can be represented within the ER model, but CANNOT be implement with relational databases. So, whenever you encounter these sorts of relationships. Instead, you'll want to do the following:

1. It can be changed into two 1:M relationships using **bridge entities.**
2. The bridge entity table (the **linking table)** must contain at least the primary keys of the original tables.
3. The linking table contains multiple occurrences of the foreign key values.
4. Additional attributes may be assigned as needed.
5. It can avoid problems inherent in an M:N relationship by creating a composite entity or bridge entity. For example, an employee can work on many projects OR a project can have many employees working on it, depending on the business rules. Or, a student can have many classes and a class can hold many students.

For example, in the following figure shows a many-to-many (written as **M:N**) relationship between Employees and Projects. One employee may be involved in many projects and each project, in turn may have many employees. Moreover, there is an attribute (StartDate) which is associate with the *relationship itself,* reflecting the date that each employee joined each project.



If we turned this into a relational database, we could use the following relational schema:

- EMPLOYEE(EID, Name, Address, Birthdate, Salary)
- PROJECT(Code, Name, Budget)
- JOIN(EID, Code, StartDate)

Converting M:N relationships to two 1:M relationships. To convert you need to the following:

1. For each M:N between entity types A and B, identify a relationship R that can be used to characterize this relationship.
2. Create a new relation (table) S to represent the relationship R.

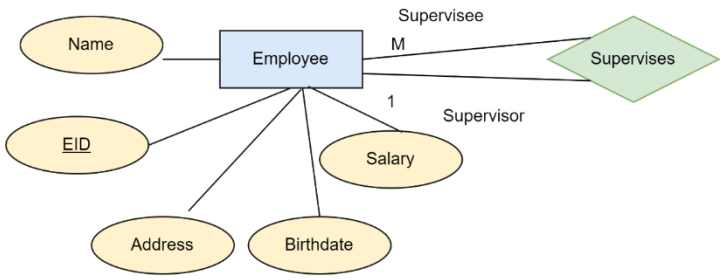
3. Table S needs to contain the PKs of BOTH A and B. These together can be the PK in the S table OR these together with another simple attribute in the new table S can be the PK. (So, in the above example, Join contains the primary keys of both employee and project AND an additional simple attribute).

RELATIONSHIP DEGREE

Most of the relationships we have talked about so far have been **binary** relationships, which relate exactly two different types of entities. These are the "norm" with both ERM and the Relational Model. However, there are relationships with different **degrees,** that relate different numbers of entity types.

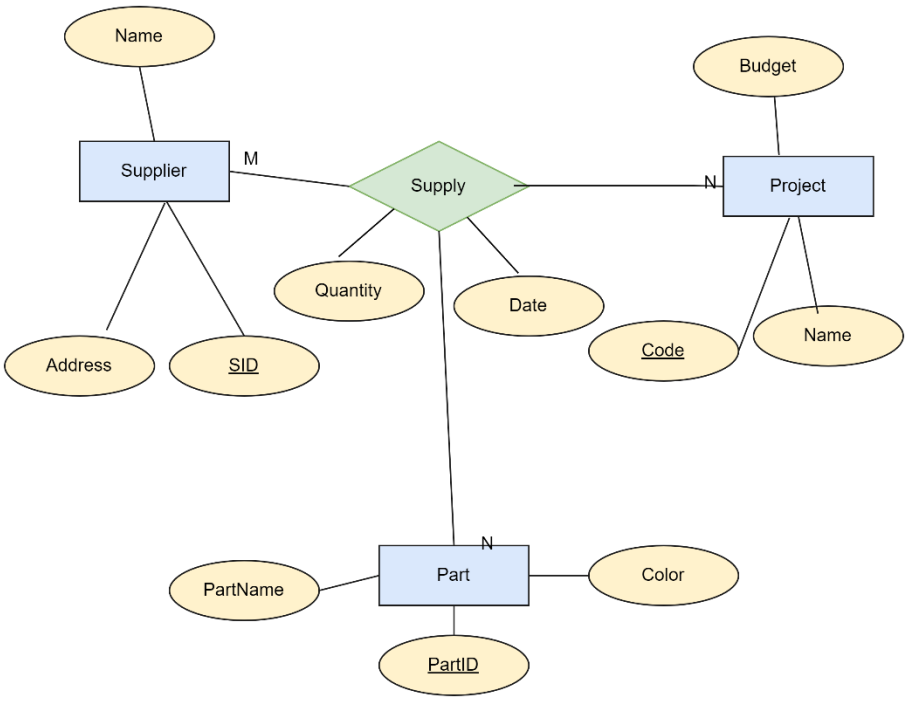
A **unary relationship**, also called **recursive**, is one in which a relationship exists between occurrences of the same entity set. In this relationship, the primary and foreign keys are the same, but they represent two entities with different roles. For some entities in a unary relationship, a separate column can be created that refers to the primary key of the same entity set.
The relational scheme here is (not that each employee may one have one supervisor, but that a supervisor may supervise many employees. We capture this by encoding the EID of the supervisor on the "many" side of the relationship, which in this case ends up being the entries representing the supervisees.).

- EMPLOYEE(EID, Name, Address, Birthdate, Salary, Supervisor-EID)



A **ternary relationship** is a relationship type that involves many to many relationships between three tables. It is an example of **n-ary relationship** where *n=3* (here *n* = the number of different entity types that are related). The following figure provides an an example of mapping a ternary relationship type. Note *n-ary*)

- For each n-ary (> 2) relationship, create a new relation to represent the relationship.
- The primary key of the new relation is a combination of the primary keys of the participating entities that hold the N (many) side.
- In most cases of an n-ary relationship, all the participating entities hold a many-to-many relationship with one another.



The relational schema (which reflects the fact we need to create a SUPPLY table) is as follows:

- SUPPLIER (SID, Name, Address)
- PROJECT (Code, Name, Budget)
- PART(PartID, PartName,Color)
- SUPPLY(SID, Code, PartID, Quantity, Date)

REVIEW QUESTIONS AND ACTIVITIES

Problem 1. A manufacturing company produces products. The following product information is stored: product name, product ID and quantity on hand. These products are made up of many components. Each component can be supplied by one or more suppliers. The following component information is kept: component ID, name, description, suppliers who supply them, and products in which they are used. The business rules are as follows:

1. A supplier can exist without providing components.
2. A component does not have to be associated with a supplier.
3. A component does not have to be associated with a product. Not all components are used in products.
4. A product cannot exist without components.

Create an ERD to show how you would track this information**.** Show entity names, primary keys, attributes for each entity, relationships between the entities and cardinality.

Problem 2 (A challenge!). Create an ERD for a car dealership. The dealership sells both new and used cars, and it operates a service facility. Base your design on the following business rules:

1. A salesperson may sell many cars, but each car is sold by only one salesperson.
2. A customer may buy many cars, but each car is bought by only one customer.
3. A salesperson writes a single invoice for each car he or she sells.
4. A customer gets an invoice for each car he or she buys.
5. A customer may come in just to have his or her car serviced; that is, a customer need not buy a car to be classified as a customer.
6. When a customer takes one or more cars in for repair or service, one service ticket is written for each car.
7. The car dealership maintains a service history for each of the cars serviced. The service records are referenced by the car's serial number.
8. A car brought in for service can be worked on by many mechanics, and each mechanic may work on many cars.
9. A car that is serviced may or may not need parts (e.g., adjusting a carburetor or cleaning a fuel injector nozzle does not require providing new parts).

Create an ERD to show how you would track this information**.** Show entity names, primary keys, attributes for each entity, relationships between the entities and cardinality.