# 1 Introduction

So far, we've learned how to use already built databases: SQL queries! We also learned the internals of a database management system which includes many abstraction levels. But what happens if we are given a high-level description of what data we want to store with our database? How do we design a database to fit out needs? We will discuss these question in this module!

2 Entity-Relationship Models

When designing a database, we often use Entity-Relationship Models (aka "E-R" models). These models have 2 main components.

The first main component is an **entity**: a real-world object described by a set of attribute values. An entity is usually depicted as a rectangle in our E-R model. In addition, the attributes/fields associated with the entity are depicted as ovals. For example, the following figure shows an employee as an entity in our E-R model with the following fields: a SSN number, a name, and a lot. Note that the attribute "ssn" is underlined because it is the identifying attribute for this entity!
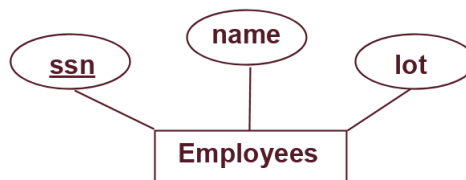
The second main component is a **relationship**: association among two or more entities. A relationship is usually depicted as a diamond our an E-R model. Again, the attributes associated with the relationship are depicted as ovals. For example, the following figure shows "works in" as a relationship between the entities employee and department. The "works in" relationship has the attribute "since."
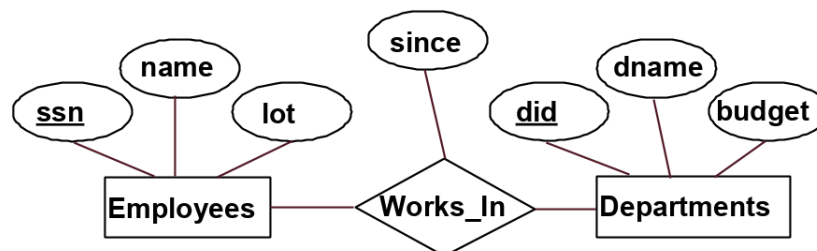

Note that the same entity set can participate in different relationship sets, or in different "roles" in the same relationship set.

2.1   Relationship Constraints

We have connected our entities to relationships with a thin black line up until now. This line means denotes a **many-to-many** relationship. This means each entity can participate in 0 or more times in the relationship and vice versa. For example, for the figure above, this means that many employees can work in many departments (0 or more) and departments can have many employees (0 or more).

In contrast, we want our model to reflect that each department has at most one manager. We do this by using a **key constraint**, which denotes a **1-to-many** relationship. The key constraint is depicted by a thin arrow. For example, for the following figure, this means that each department has at most one manager (0 or 1) while employees can be managers for many (0 or more) departments.
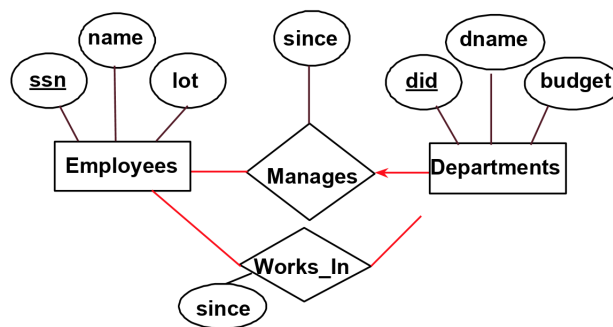
What if we want our model to show that each employee works in at least one department? We do this by using a **participation constraint**, which denotes **at least one** relationship. The participation constraint is depicted by a thick line. For example, in the following figure, we added two thick lines. One of them denotes that each employee works in at least one department. The other denotes that each department has at least one employee.
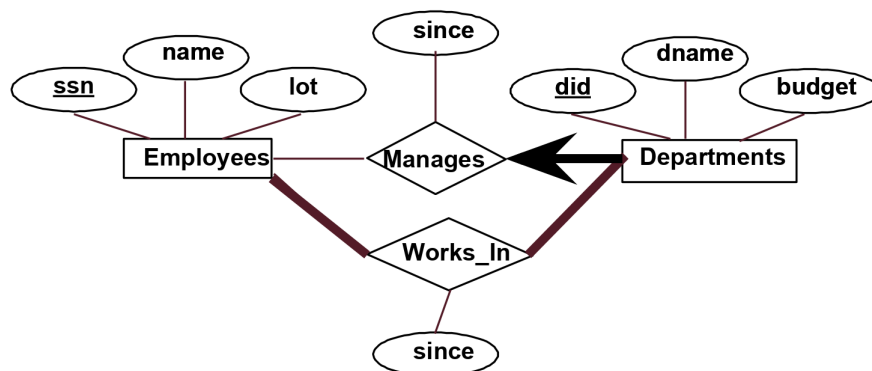
In addition, you can see we added a thick arrow pointing from departments to manages. This is both a key and participation constraint: each department has at least and at most one manager.

A **weak entity** can be identified uniquely only by considering the primary key of another (owner) entity. Owner entity set and weak entity set must participate in a one-to-many relationship set (one owner, many weak entities). Weak entity set must have total participation in this identifying relationship set. We depict a weak entity by bolding the rectangle and relationship as shown below. In addition, weak entities have only a "partial key" (dashed underline).
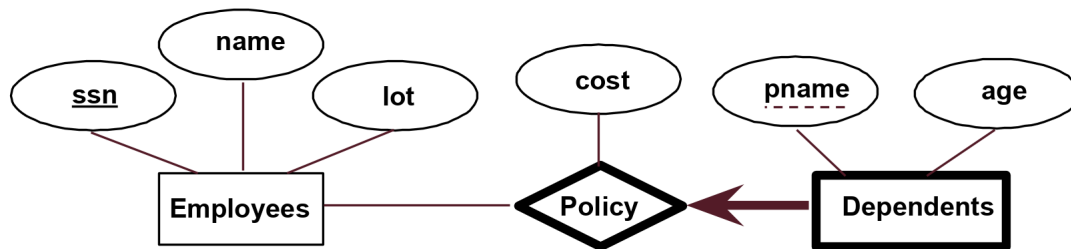
Now that we have our E-R Model drawn out, how do we actually organize it into relations? We can translate each entity and relationship to its own table and have primary keys and foreign keys depending on the relationship between tables. In general, there are many ways to set up relations to represent the same E-R model.

3 FDs and Normalization

Now that we have set up our database, we want to optimize! One thing we want to avoid is redundancy in our schema. This is wasted storage and also leads to insert/delete/update anomalies. Our solution to this problem is going to be **Functional Dependencies**. A functional dependency $X \to Y$ means that the X column determines Y column in a table R. This means given any two tuples in table R, if their X values are the same, then their Y values must be the same (but not vice versa).

Let's look at a couple of examples to better understand functional dependencies and what "determines" means. In the example below, two of the rows have the same X value of 1 but different Y values of 5 and 6, so we cannot say that X determines Y:

X	Y
1	5
1	6
2	1

For the following example, we see that all rows with the same X value of have the same Y value. All the rows with $X = 1$ will have $Y = 2$, all the rows with $X = 2$ will have $Y = 4$, and all the rows with $X = 3$ will have $Y = 4$. In this case, we can say that X determines Y (aka $X \to Y$):

X	Y
1	2
1	2
1	2
2	4
3	4

Note that in the above example, even though $Y = 4$ for two rows, their X values do not have to be the same to say that X determines Y. We only care that if the X values are the same that the Y values are also the same, not vice versa!

Primary keys are special cases of FDs. A **superkey** is a set of columns that determine all the columns in the table. A **candidate key** is a minimal set of of columns that determine all the columns in the table. For example, if columns K, L determine all the columns in the table and K is also a primary key of the table (aka column K alone determins all the columns in the table) then K, L is superkey and K is a superkey and a candidate key.

We will decompose our relation to avoid redundancies by using functional dependencies. But first, we define the closure of F denoted as $F+$ to be the set of all FDs that are implied by F.

Let's take a look at a couple of examples to better understand these definitions. Given a relation R with columns $\{A, B, C, D, E\}$ and functional dependencies $F = \{B \rightarrow CD, D \rightarrow E, B \rightarrow A, E \rightarrow C, AD \rightarrow B\}$, let's examine the following questions:

- Is $B \rightarrow E$ in $F+$? To find the answer to this, we will evaluate the closure of B. $B+ = \{B, C, D, E, \dots\}$, we get this because B determines CD from the first functional dependency from the set F. Then we see that D determines E so we see that E is in the closer of B. The answer is YES.

- Is D a key for relation R? We will determine the answer to this by evaluating the closer of D. $D+ = \{D, E, C\}$. We get this because D determines itself. From the second FD in F, we see that D determines E. Then we see that E determines C. Since $D+$ is not equal to all the columns of the relation R, we know that D is NOT a key for R. The answer is NO.

- Is AD a key for R? We again evaluate the closure of $AD+$. Using the same procedure as the last two, we get that $AD+ = \{A, D, E, C, B\}$. Therefore, the answer is YES.

- Is AD a candidate key for R? We can check if X is a candidate key checking if $(X - A)+ = R$ for each attribute A in X. So we check $A+ = \{A\}$ and $D+ = \{D, E, C\}$. Since none of them are R, we know that AD is a candidate key! The answer is YES.

- Is ADE a candidate key for R? We know that AD determines all the columns for R already so ADE is NOT a candidate key. The answer is NO.

3.1   Decomposing a Relation

We will now decompose the relation into Boyce-Codd Normal Form. Relation R with FDs F is in BCNF if for all $X \rightarrow A$ in $F+$, $A \subseteq X$ (called a trivial FD) or X is a superkey for R. The procedure is as follows: if $X \rightarrow Y$ violates BCNF, R into $R - Y$ and XY.

Let's take a look at example to see how this works. Let us look at relation $R = \{C, S, J, D, P, Q, V\}$ with the key C and $F = \{JP \rightarrow C, SD \rightarrow P, J \rightarrow S\}$.

- To deal with $SD \rightarrow P$, decompose into SDP, $CSJDQV$.

- To deal with $J \rightarrow S$, decompose $CSJDQV$ into JS and $CJDQV$.

- So we end up with: SDP, JS, and $CJDQV$.