Design a Ward

We've seen how entities can relate to each other. Now let's try organizing some data we have here into entities and the attributes that will belong to those entities. We will also look at the relationships between them and how that all might look in an ERD.

Here we have a file with information. The information here represents church data. We want to place this data into a relational database so we need to organize it into entities to begin with. Normalization or putting the data into normal forms will come into play as we organize our entities. But what we want to focus on is entity relationships and entity vs attributes for this activity.

As we look at the data, what entities do you see? Entities being recognizable concepts like people, place, things, events, etc. What I see are members and the wards they belong to and the missions they've been on and the callings they have and the ordinances they've received. So let's start with members. We have two members here Jared and Doris.

I'll **place a table** on our work area and **name it member**. We want a **primary key** for sure that will uniquely identify each ward member. We know the church actually uses one **called record number**. So we will add that. I will just be throwing in data types for each attribute but not really discussing those in this lesson. Which columns of the data would fit with members? I know for sure their **first and last name and date of birth** go with that entity. So let's **add those for now**. OK, so there is a start for our member's table. There may be other attributes later but let's start with those. Here is a good example of checking 3rd normal form. Does the 3 non-key attributes of first name, last name and date of birth really belong to that record number. Does the value of Doris go with that record number? Yes. But if we throw in the primary teacher calling as part of the member table, does primary teacher only go with that record number? No, record number keeps track of members not callings. The calling of Primary teacher is not reliant on just that primary key or Doris' record number.

Now let's look at **another entity, ward**, on our spreadsheet we only have the name of the ward and that's about it, but realize we could also have attributes like what the boundaries of the ward are, the date the ward was first organized, etc. that could belong to the entity ward. For now we will just put the name because that is what's on the spreadsheet. We also need a **primary key for wards** I'm sure there is a unique key for wards in real life but I'll call it ward id for now. And I'll **add the name of the ward**.

Now let's look at the relationship between these two tables. How does the entity members relate to the entity wards? Wards can have many members and each member belongs to one ward. So we have a **one-to-many relationship** here. With the many part of the relationship belonging to the members. Many members belong to one ward. So our crows feet notation will look like this. Notice the foreign key that shows up in members. This establishes the relationship link between the two

tables. Workbench automatically places the foreign key there for you on the many side of the relationship.

Good, what other entities might we have? Right next to members data, in the file, we have mission information. The mission name and the start and end date of when that member was on the mission. So let's start with an **entity called mission**. Let's give it a **primary key called mission id and the name of the mission**. The start and end date aren't the start and end date of the mission itself but of the member's mission. So let's leave the mission table like this for now. What is the relationship between the members and the missions table? Missions can have many members who go on that mission. At first thought you could say members only go on one mission. So that would be a one-to-many. Missions can have many members serve there and each member serves one mission. But if you think about it, members can go on many missions. You could go on a mission when you're 18 or 19 and again when you are older right? So we really have a many-to-many relationship here. And what has to happen with a many to many relationship? You need a linking table. If we choose the **many to many tool** in Workbench and we click on each entity it creates a linking table for us. It automatically includes the two primary keys as the composite key that will uniquely identify each row of the linking table. That composite key is made up of the two primary keys of each of the tables. In this case two foreign keys become a composite key that is acting as the primary key—two columns together that uniquely identify each row. We can also rename the table if we'd like. Workbench adds a name that combines the two tables. Here I'll edit it just a bit.

So where does the beginning and ending date of the mission go? If the mission table would contain data about every mission in the church it's mission name, maybe the boundaries of the mission or even the date the mission was created—the beginning and ending date of a single missionary in that mission doesn't really belong there. If we put the beginning and ending date of the mission in the members table there is a possibility of repeating columns with two or more beginning dates and two or more ending dates in the same table (if they'd gone on more than one mission) and when we learn normal forms we see that this will violate first normal form with repeating data columns or attribute values so we know it doesn't go there. But if we look at the linking table each row will be a unique mission linking one member to one mission so we could **add two more columns to the rows on that table to describe the beginning and ending date of each mission the member goes on**. This is a good example of 2nd normal form. Each attribute that isn't part of the primary key has to belong to both parts of the composite primary key. The beginning and end data only fit with that member on that mission. So yes, both columns definitely go with the entire composite primary key.

Next we have the callings of the members with a **calling table**. I'll give it a **primary key of calling id and add the name of the calling** (for example the values for that attribute or calling name when we add it later in the table, would be something like relief society president, or in our case primary teacher for Doris and Elder's Quorum 1st counselor for Jared). In our example each member has one calling. So we could have a one-to-one relationship one member has one calling and one calling would only have one member. But since we already started with multiple missions we could continue that with callings as well. Members can have many callings over time

(and even 2 callings at the same time) and callings can have many members. Let's **add the many to many** relationship and we see a linking table come up.

Really, for that matter we really could argue a many to many relationship with members and wards as well. Members can belong to many wards over time and wards can have many members. So these are questions we'd have to ask those who need this database. Is this database going to keep track of a member at one single point in their life or do you want to keep track of all the past history data of each member? For now, let's say members can have more than one calling at a time, we know that happens. And let's assume that any given calling only has one member that has that calling. If we did realize that we needed to keep a historical account of every calling of each member throughout their life time, we could try a many-to-many relationship here which would give us a linking table of members_callings and we could add beginning and end dates to each calling that the member served in (kind of like their missions), but for now to stay true to the data in the spreadsheet, let's do a one to many with the many part meaning the member could have a number of callings at the same time. For example Doris could be a ministering sister and a primary teacher. So I'll **delete the linking table and try a one-to many relationship with the many portion for the callings.** Notice the foreign key here.

The next column in the spreadsheet is the name of the organization. I can assume from how they have it here that they are talking about the organization of the calling, not the organization that the member belongs to. For example Doris' calling says Primary Teacher so the organization of Primary belongs to her calling. Doris is probably a member of the relief society herself though. We can actually use the organization table later to relate to each member as well as their calling. We'll call the **entity organization** and add an **organization id and a name of the organization**. We could also add attributes like a description of each organization, the age range of people in that organization. For example, Primary are ages 18 months to 11 years. Relief Society 18 years old and on, etc. But we'll leave it for now. This type of table is what they call a lookup table. There are only so many organizations in the church. (elders quorum, relief society, primary, etc.) There would be a set number of rows in this type of table.

What's the relationship between organization and calling? Each organization has many callings and each calling belongs to one organization. **So we have a one-to-many with the many part of the relationship on the callings**. Again, notice the foreign key that relates the tables.

While we are here, I could also relate the member and organization table together. This demonstrates the beauty of relational databases. Not only do members have callings that belong to organizations, they themselves belong to organizations. So let's look at the relationship between members and organizations. Members belong to one organization (for example, Doris is a member of the relief society and Jared belongs to the Elders Quorum) but organizations can have many members. So it is a **one to many relationship with the many portions belonging to the members table**. We get a foreign key relationship.

Since there is a set number of organizations, why couldn't we have just put the name of the organization that the person belonged to as an attribute in the members table.

This could also be true for the callings table as well. We could eliminate the organizations table all together and just add the attribute of organization as a part of the calling and member table. The calling table would have a field or column of organization and the member table would have field of organization. And maybe that would have been OK. It just depends on what the output of the database that needs to be produced for the users. So here's a classic case of entity vs. attribute. Do we make a whole new entity called organization or just use it as an attribute with the name organization with values that would be elders quorum, relief society, etc. Or do we make a new lookup table that contains each organization as a row in the table? These again are decisions you have to make as you are designing the database. In the future, what if the Relief Society changes its name to "Adult Women Group" then it would have been easier to change the name in the lookup table and not in the member and calling tables. Or maybe you could have just run a mass update in your table later to change all the text values. I like the way we have it here because I can relate two different tables to the same entity without having to place the same type of attribute twice in two different tables. As we build queries later you may see additional benefits to more detailed queries that you can run on the data as well depending on how you design your tables here. So let's leave it this way for now.

OK we are getting there. So now we just have a list of ordinances that Doris and Jared have received or not received. Again we have the choice here. Should these be attributes of the member entity? We could put more attributes into the members entity for example, blessing with a value of a date or with it left null if they weren't blessed. We could put baptism at an attribute in members again with a date of when it happened or null if they weren't baptized and on and on with all the ordinances. That would probably work OK, but what if you also needed to keep track of things like who were the witnesses of the baptism or where was the location of the baptism or what temple were they sealed in. Then the date isn't enough information. For our example it might be however, so you could leave it this way. But for arguments sake, let's say our database users need to have the additional information about witnesses, locations etc in addition to just the dates.

So let's **make an ordinances table**. I'm going to give it the **primary key of ordinance_id and name of the ordinance**. Let's leave it there for now. What's the relationship here? **Members can receive many ordinances and ordinances can be given to many people. Many-to-many.** So we get a linking table. This is **where I would put the date of the ordinance** and the location and the witnesses etc. Again the ordinance table becomes a look up tables. There are only so many ordinances in the church (a set number or rows for this table) But the linking table could be huge with all the members associated with their ordinances.

So we can see how crazy keeping track of just one member and all that is associated with them in the church can be. We could have expanded this even further to mission presidents that go with missions and bishops to wards and temples to ordinances etc. It could have gotten huge. It just all depends on what the organization needs to keep track of. But this was a good start. And remember it's not set in stone we can come back and alter the design if we need to, but it's easier to alter it now before we start putting the actual data into each tables.

So, there we have a few ways to look at entities and their relationships and if we want certain data to be their own entities or just attributes of entities. We can also see how separating out entities goes hand in hand with normalizing our data.