

## Relational Database Concepts for Beginners

A database contains one or more tables of information. The rows in a table are called *records* and the columns in a table are called *fields* or *attributes*. A database that contains only one table is called a *flat* database. A database that contains two or more related tables is called a *relational* database. There are other more complex kinds of databases, but this paper is going to focus on the what and why of relational databases.

Here's an easy way to understand the benefits of dividing your data into multiple tables:

Imagine that you are responsible for keeping track of all the books being checked out of a library. You could use a single table (a flat database) to track all the critical information:

First Name	Last Name	Address	Phone	Book Title	Due Date
Bob	Smith	123 Main St.	555-1212	Don Quixote	7-14-09

This table meets the basic need to keep track of who has checked out which book, but does have some serious flaws in terms of efficiency, space required, and maintenance time. For example, as voracious reader Bob checks out more books over time, you will have to re-enter all of his contact information for every book.

First Name	Last Name	Address	Phone	Book Title	Due Date
Bob	Smith	123 Main St.	555-1212	Don Quixote	7-14-09
Alicia	Petersohn	136 Oak St.	555-1234	Three Men in a Boat	7-16-09
Bob	Smith	123 Main St.	555-1212	Things Fall Apart	8-15-09
Bob	Smith	123 Main St.	555-1212	Anna Karenina	8-15-09
Zayn	Murray	248 Pine Dr.	555-1248	Heidi	8-17-09
Bob	Smith	123 Main St.	555-1212	The Old Man and the Sea	9-10-09

To re-enter Bob's contact information wastes time, and increases the opportunity for error. Moreover, when an update is necessary (e.g. Bob's phone number changes), each of Bob's records must be located and corrected. If one of Bob's records has a different phone number from the rest, is it a correction, a record overlooked during the last update, or a data-entry mistake?

These problems can be decreased by *normalizing* our data – in other words, dividing the information into multiple tables with the goal of having “a place for everything, and everything in its place.” Each piece of information should appear just once, simplifying data maintenance and decreasing the storage space required.

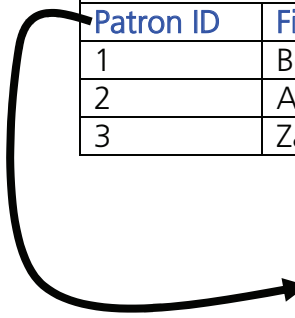
PATRONS TABLE			
First Name	Last Name	Address	Phone
Bob	Smith	123 Main St.	555-1212
Alicia	Petersohn	136 Oak St.	555-1234
Zayn	Murray	248 Pine Dr.	555-1248

CHECKOUT TABLE	
Book Title	Due Date
Don Quixote	7-14-09
Three Men in a Boat	7-16-09
Things Fall Apart	8-15-09
Anna Karenina	8-15-09
Heidi	8-17-09
The Old Man and the Sea	9-10-09

Now that the data are arranged efficiently, we need a way to show which records in the PATRONS table correspond to which records in the CHECKOUT table – in other words, who checked out which book. Instead of repeating everything we know about a patron whenever he checks out a book, we will instead give each library patron an ID, and repeat *only the ID* whenever we want to associate that person with a record in a different table.

PATRONS TABLE				
Patron ID	First Name	Last Name	Address	Phone
1	Bob	Smith	123 Main St.	555-1212
2	Alicia	Petersohn	136 Oak St.	555-1234
3	Zayn	Murray	248 Pine Dr.	555-1248

CHECKOUT TABLE		
Patron ID	Book Title	Due Date
1	Don Quixote	7-14-09
2	Three Men in a Boat	7-16-09
1	Things Fall Apart	8-15-09
1	Anna Karenina	8-15-09
3	Heidi	8-17-09
1	The Old Man and the Sea	9-10-09



Now the PATRONS and CHECKOUT tables can be related (how relationships are formally declared in various database software is beyond the scope of this paper). At this point, we need some new terms to talk about our related tables.

The **primary key** is a field whose values are unique in this table, and so can be used as identifiers for the records (*multi-field* or *composite* primary keys are beyond the scope of this paper, and are unlikely in an ArcGIS geodatabase). In table PATRONS, the Patron ID field is the primary key and so its values must remain unique. For example, the value "2" can appear only on one record - Alicia's - and Alicia can have only one Patron ID - "2."

Is the Patron ID field in table CHECKOUT the primary key? We can see that it contains duplicate values, so the answer is No. If Patron ID were the primary key for CHECKOUT, each person would only be permitted to check out one book, and afterward would be forbidden to check out any more books, ever.

So if Patron ID is not the primary key for table CHECKOUT, which field is? We can't make Book Title the primary key, or we'd have a similar problem – each book could only be checked out once, and afterward no one would be permitted to check it out ever again. We can't make Due Date the primary key, or else only one book could be due each day. Since none of the existing fields works as a primary key, we will add a new field to hold an identifier for each record. We could name this field Checkout ID, or we could follow ESRI's convention of giving all primary key fields exactly the same name: ObjectID.

PATRONS TABLE				
ObjectID	First Name	Last Name	Address	Phone
1	Bob	Smith	123 Main St.	555-1212
2	Alicia	Petersohn	136 Oak St.	555-1234
3	Zayn	Murray	248 Pine Dr.	555-1248

CHECKOUT TABLE			
ObjectID	Patron ObjectID	Book Title	Due Date
1	1	Don Quixote	7-14-09
2	2	Three Men in a Boat	7-16-09
3	1	Things Fall Apart	8-15-09
4	1	Anna Karenina	8-15-09
5	3	Heidi	8-17-09
6	1	The Old Man and the Sea	9-10-09

Naming every primary key field "ObjectID" does make it easy to tell at a glance which field uniquely identifies the records in this table. We can also use this naming convention to provide hints about which fields are related. For example, Patron ObjectID in CHECKOUT is related to ObjectID in PATRONS.

To further increase efficiency, decrease required space, and improve ease of maintenance, we can separate the book information into its own table.

PATRONS TABLE				
ObjectID	First Name	Last Name	Address	Phone
1	Bob	Smith	123 Main St.	555-1212
2	Alicia	Petersohn	136 Oak St.	555-1234
3	Zayn	Murray	248 Pine Dr.	555-1248

CHECKOUT TABLE			
ObjectID	Patron ObjectID	Book ObjectID	Due Date
1	1	1	7-14-09
2	2	6	7-16-09
3	1	3	8-15-09
4	1	5	8-15-09
5	3	2	8-17-09
6	1	4	9-10-09

BOOKS TABLE			
ObjectID	Title	Author	Year
1	Don Quixote	Miguel Cervantes	1605
2	Heidi	Johanna Spyri	1880
3	Things Fall Apart	Chinua Achebe	1958
4	The Old Man and the Sea	Ernest Hemingway	1952
5	Anna Karenina	Leo Tolstoy	1873
6	Three Men in a Boat	Jerome K. Jerome	1889

Now ObjectID in BOOKS is related to Book ObjectID in CHECKOUT.

When two tables have an unequal relationship, we call the independent table the *parent* and the dependent table the *child*. You can identify the parent table by determining which table could contain a record without needing a corresponding record in the table on the other side of the relationship. For example, is it possible to have an unpopular library book which never gets checked out? Yes. Is it possible to check out a book that doesn't exist? No. Since BOOKS can contain

records that aren't referenced by CHECKOUT, BOOKS is the parent in this relationship, and CHECKOUT is the child.

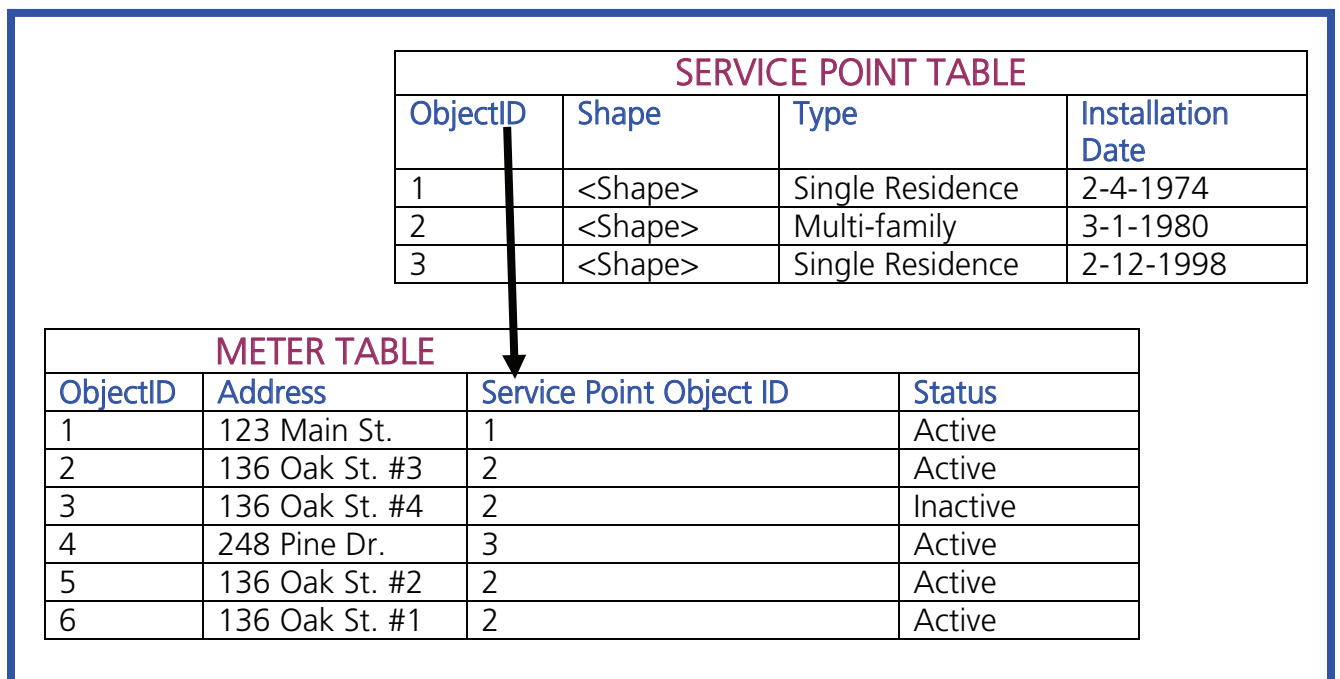
If somehow the child table contains a record that does not have a corresponding record in the parent table, that record is called an *orphan*. Orphaned records are a problem that generally requires attention from the database administrator.

Another way to identify the child table is to find the field which refers to the other table's ObjectID. BOOKS does not contain an ObjectID field for the CHECKOUTS, but CHECKOUTS does contain a field to store Book ObjectIDs. Therefore, CHECKOUTS is the child table in this relationship.

The last new concept to consider is *cardinality*, which describes how many records in one table can be related to records in another table. Two tables might have a cardinality of 1-1 (one to one), 1-∞ (one to many), 1-3 (one to three), ∞ - ∞ (many to many), etc. The PATRONS – CHECKOUT relationship has a 1-∞ cardinality, because 1 patron may have any number of checkouts, from zero to infinity. Put another way, the CHECKOUT – PATRONS relationship has a cardinality of ∞ - 1. If the cardinality of PATRONS – CHECKOUT were 1-1, then each patron could check out only one book. If it were ∞ - ∞, then several patrons together might share joint responsibility for one or more checkouts.

The BOOKS – CHECKOUT relationship is also 1 - ∞, since one book may be checked out multiple times.

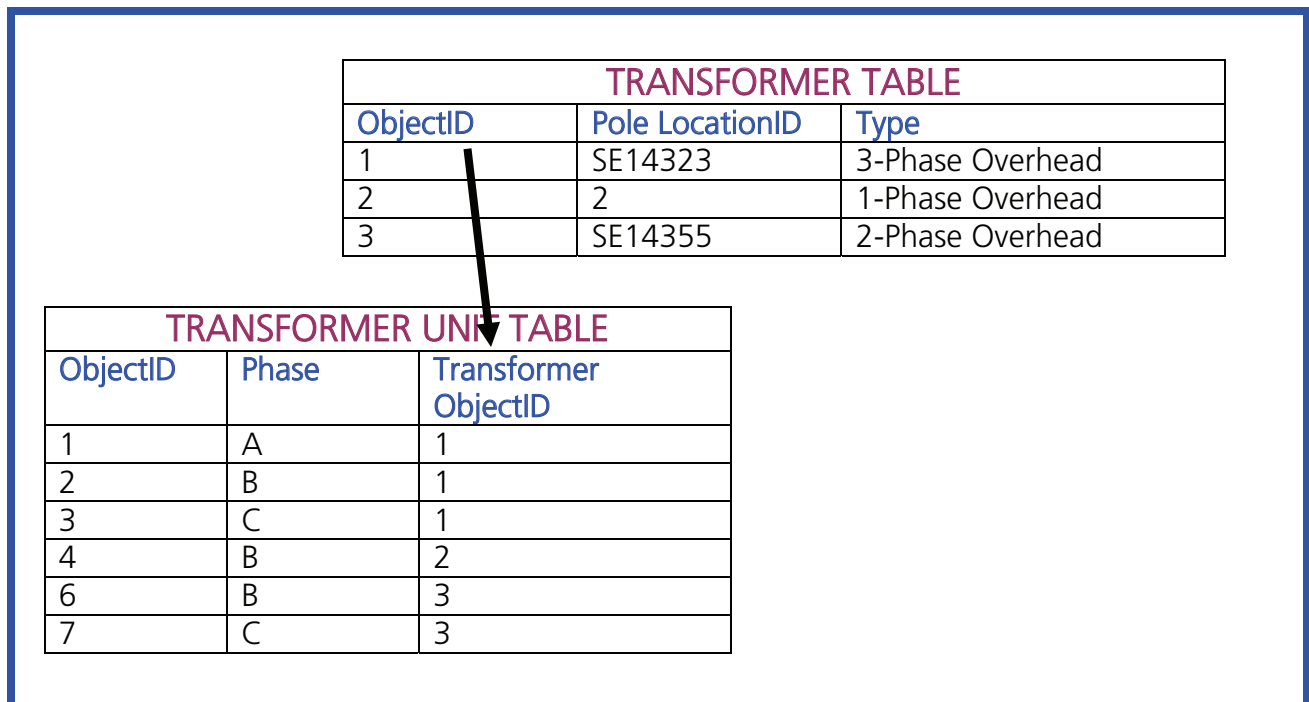
If we really were designing the *data model* (tables, fields, relationships, etc.) for a library, we would continue by separating even more data (such as the authors) into other tables until the model was as efficient as possible. Since we are modeling utility data instead, let's see how these ideas apply to meters and service points:



The SERVICE POINT table stores one record per location, but each location could have multiple Meters (for example, all the meters for an apartment building may be accessible from the same closet). The relationship between SERVICE POINT and METER is 1-∞ (one Service Point can include any number of meters). METER is the child table, because it contains a field to store Service Point ObjectIDs.

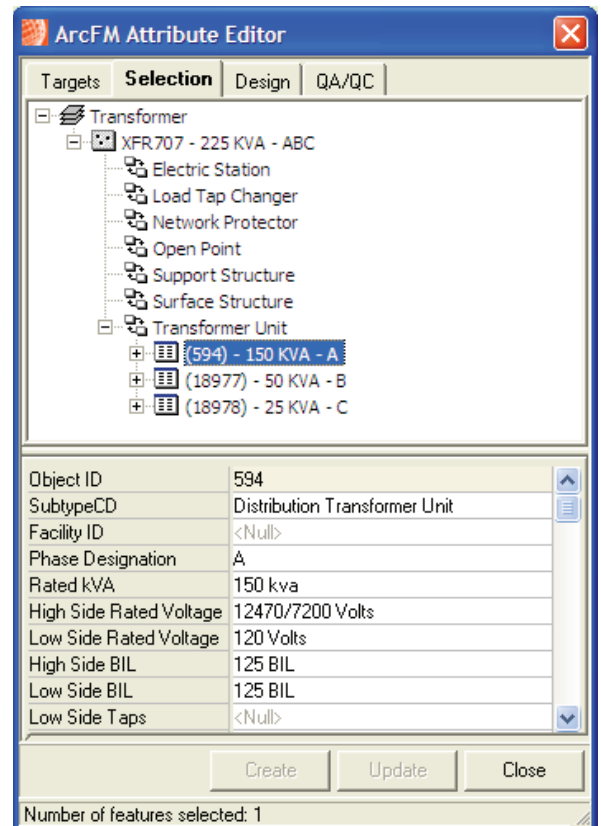
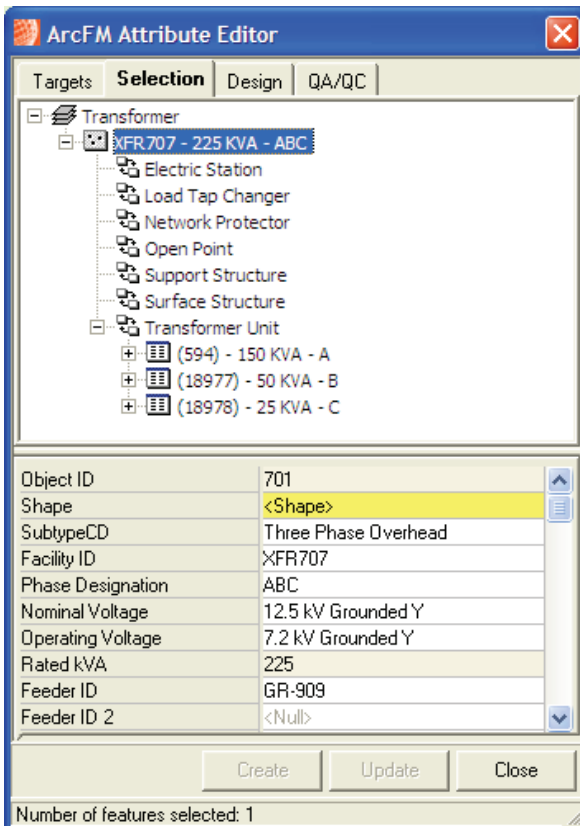
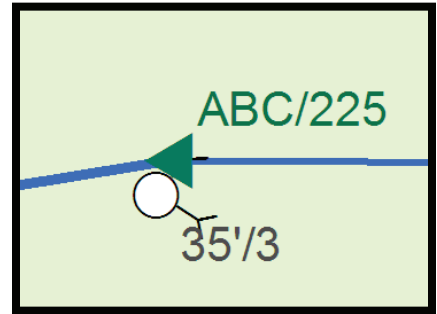
Also notice that SERVICE POINT contains a Shape field, but METER does not. ArcGIS stores a map feature's dimensions and location in the Shape field (although the specifics of this information may be hidden under the generic alias <Shape>). The presence of a Shape field in the SERVICE POINT table tells us that Service Points are *features* – objects that can be displayed on a map. The absence of a Shape field in METER tells us that Meters are ordinary objects that cannot be displayed on a map.

When several related objects are likely to be in close proximity, having a Shape field only at the parent level decreases map clutter. For example, a high-rise apartment building might have one Service Point and 80 Meters. It is much more efficient to sketch one Service Point and then create 80 unmapped Meters related to that Service Point, than to sketch 80 individual Meters. In the same way, when sketching a three phase transformer, it is usually better to sketch one parent Transformer, which relates to three child Transformer Units (one per phase).



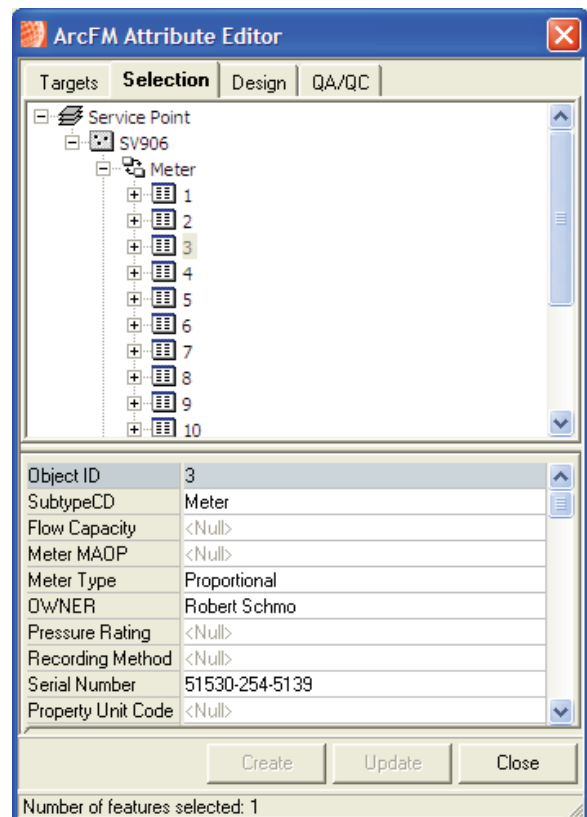
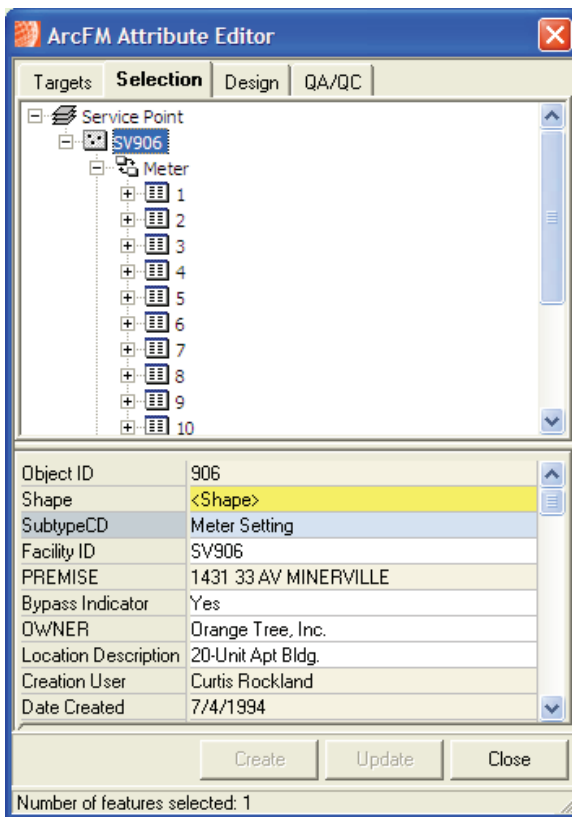
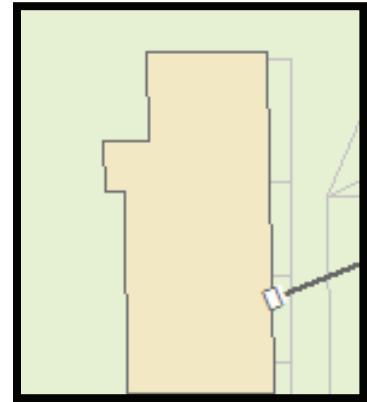
How does all this look in ArcFM and ArcMap?

The green triangle shown here is one way a three-phase overhead Transformer might be symbolized on the map (this Transformer is on a 35-foot class 3 Pole with an Anchor Guy). Although only one symbol appears on the map, if we select this Transformer we can see that it is related to three Transformer Units – one for each phase. We can also see that the Transformer has attributes, and each of the Transformer Units has its own attributes. The Transformer has a Shape field where its geographic location is stored. The Transformer Units do not have a Shape field, so we know they cannot appear on the map.



A list of all related tables (both parents and children) appears below the feature. Thus we can see that Transformers can be related not only to Transformer Units, but also Electric Stations, Load Tap Changers, and so on. Whenever a record related to the selected feature exists in one of these tables, an expandable plus/minus box appears to the left. Since Transformer Unit is the only line with a plus/minus box, we know that this particular Transformer currently has related records only in the Transformer Unit table.

The small white box shown here is one way a Service Point might be symbolized on the map. This single Service Point includes twenty Meters – one for each unit in this apartment building. Since only the Service Point has a Shape field, only it can be shown on the map. The Meter records store all the information we need to track for each customer, but cannot appear on the map.



And now you know the fundamental database concepts of primary keys, parents, children, cardinality, and relationships, and their application to utility ArcGIS databases.