



The Relational Model

White Paper

Abstract

This paper describes the widely used method for organizing data.

The Relational Model

Revised by Chris Rindal
QA Technician, Tometa Software, Inc.

Tometa creates custom software for you

Tometa Software designs and develops robust software solutions for virtually all industries including in-house (vertical market) and retail software, some of which is on the shelves at your local software store. We focus our unique combination of creative, technical, and problem-solving skills on meeting our client's objectives. Because of our clarity of purpose, commitment to process, and broad professional skill sets, we are able to provide our clients with world-class solutions that are functionally superior and fully aligned with our client's strategic focus.

Balancing development speed, quality and cost is what we are all about. Tometa combines agile development practices with fixed pricing, so you know what the cost, end product, and delivery time table look like—up front. If we underestimate the effort, we complete the overrun on our dime. Simple as that. That's why large enterprise firms like Alcoa and NASDAQ choose Tometa.

Tometa's agile development expertise and low-overhead US location keep our prices comparable to offshore vendors – without offshore challenges. Using a fixed pricing model, we provide upfront visibility into a project's ultimate costs, end product and delivery schedule. Our clients like knowing that we have “skin in the game” – a fixed price that aligns our goals with yours, incenting us to get the job done right and fast.

Lastly, as a Microsoft Certified Gold Partner, Tometa Software, can customize its products or create custom web, client/server, and traditional applications. With programming experience in C#, C++, Visual Basic, CGI, HTML, RPG, Delphi, Java and many others; Tometa Software is uniquely positioned to meet your needs as a development firm.

[Check us out today](#)

The relational model is the most widely used and most evolved method of organizing data. All major database management systems assume that you will be working in some manner with the relational model. The relational model provides a simple and flexible way to describe data.

The relational model of data is everywhere. It is even visible in day-to-day life. For example, every record in a relational database has a unique identifier, and so ID numbers that uniquely identify something have become common place. Every person probably remembers several ID numbers on a day-to-day basis. It's just a small part of how the wide-spread implementation of relational databases has changed modern life.

Businesses adapt their structure to the relational model so solutions like PeopleSoft and SAP can be implemented. Anyone who has worked for a large company that has implemented one of these solutions can attest to how it has changed the day-to-day routine of their job. For example, a salesman now works with the same dataset as an accountant. While the abstraction of the data and forms they focus on may be different, it is all derived from the same information. This predefines what information needs to be collected into parts known as business objects.

It is unlikely a person can go through a day in their life without running across some way it has been touched by relational databases. It is a cornerstone of the information-based society we live in today. It seems almost monolithic in its use, however, that is because it is easy to learn and limitless in its uses.

This paper will discuss the basics of the relational model of database design, focusing primarily on the entity-relationship model and how it is used to describe data. Normalization and its purposes will be discussed briefly. The purposes of indexing will also be discussed.

Data Modeling

Data modeling is the initial purpose of any database. The data has to be modeled or described so that it can be input and output in a method that can be duplicated. In an office, several file cabinets store different files about different things. One file cabinet may store information about clients and another may store information about products. In those drawers, data may be defined into smaller categories so that it is more easily accessed. This is a basic data modeling system based on a hierarchy. The relational model uses a more complex algebraic system to describe how the data is stored. However, the developer of a relational database doesn't need to understand anything about the predicate calculus or formal triadic relations to build a database using the relational model.

In the relational model, data is broken into small pieces which are connected with each other by *relationships*. By breaking the data into smaller pieces, accuracy is ensured by making data entry a one time process then reusing the data by connecting it with other pieces. Using this process, the large

collection of unorganized data is simplified to its smallest pieces then related with many other small pieces to recreate the big picture in a more organized format.

The small pieces of data in the relational model are known as entities. Entities are organized using the *flat model*. The flat model organizes data into columns and rows like a spreadsheet. Information in the columns can be any digital format, from text to numbers to graphics. Information in an entity is usually collected based on a logical set of rules created by the designer following the principles of normalization. A collection of entities is known as a table. Information kept in tables make up the majority of the data stored by a database. The sample below shows very generally how restaurants and their menu items would be described by entities. The basic shape of items on the menu is broken into categories and then the pieces of information that make them up are defined.

Once the information is broken into entities and tables, putting it back together into its original form is done by creating a relationship between two tables. Having the information stored with no method of describing how it is associated with other data is not useful. In the entities we described before, what is on a menu is easily described, so are restaurants. However, we have no way to determine what restaurant has what menu items. This is solved by creating a relationship. The relationship will describe what restaurant each menu item belongs to.

The example shows that a menu item has a relationship with a restaurant. This relationship is represented in the database using *keys*. Keys serve as *unique identifiers* for entities. An entity is defined by a unique key to ensure that it has no records that share the same information in the table. Having no duplicate entries allows distinctions to be drawn between entities and allows *referential integrity*. Referential integrity is the guarantee that relationships are drawn through unique records. It insures that queries for information will always return the same information.

Unique identification is provided with a primary key. The primary key can be a unique number created for an entity which is known as a surrogate key. It can also be a record or a combination of records within an entity that uniquely identify it in every possible situation. In the case of the restaurant/menu example, both the menu items and the restaurant will be uniquely identified using surrogate keys since no combination of information in either entity is a unique identification.

The second half of a relationship is identified by a foreign key. A foreign key describes the primary key of the entity being related. By using these two keys, two entities from different tables can be joined together in a relationship. The example below illustrates what a relationship looks like when it is described by these keys.

By defining multiple foreign keys, an entity can have several relationships. A single entity can also be related to more than one entity in the same table. The

cardinality of a relationship describes how many entities can be involved in a relationship. Several cardinalities exist: one-to-one, one-to-many, many-to-many. One-to-one relationships force that foreign keys are distinct entries. One-to-many relationships allow duplicate foreign keys. Many-to-many relationships require a lookup table that combines foreign keys into the primary key of the table.

After a little practice is applied, the basics of the entity-relationship model become second nature. It isn't always intuitive at first how entities should be defined and then what other entities another entity should relate with. Information varies from project to project and database to database so the only way to become familiar with it is to practice it.

Normalization

Since what makes up an entity and what relates to what is ultimately a choice of the developer, absolute rules about what makes up an entity and what it is related to are hard to define. The process of coming up with absolute rules and applying them is called *normalization*. Proper normalization guarantees that from developer to developer, if the same rules are followed, the data will be organized nearly or exactly the same.

While going through all of the rules of normalization is beyond the scope of this paper, certain rules of normalization are required for good database design. In a well-formed database, the same piece of data will never have to be entered twice. If it is logical that information may be entered more than once, it is required by normalization to only create one entity for that information and use relationships to populate it through the database. This ensures that different users won't enter different entities that mean the same thing. This would break referential integrity. The purpose of commonly used normalization is to ensure referential integrity by only having unique data entered one time by using as many tables as required and having unquestionably unique primary keys for every entity. This sums up normalization to the 3rd form. While there are 6 forms of normalization, this is often as much normalization that is desirable. Certainly, this form of normalization guarantees that you will receive the most of the advantages of having the data in a relational database.

Indexing

The last property of the relational model we will look at is indexing. Indexing creates a rapidly searchable list of what is in a relational database. Indexing is applied to columns in entities. By preparing another table of information representing the information in the table being indexed, fast searches can be guaranteed by using binary trees. The process of indexing information is fairly processor intensive. Since data must be rendered into a rapidly searchable format, indexing is not a very dynamic property of relational databases. It also requires a proportional amount of memory to the initial memory use of the

database. In large databases, indexing is often a function that is performed during a maintenance cycle or performed when the server is at non-peak usage to avoid system conflicts.

Conclusion

The relational model is the de facto foundation for the majority of modern databases. It provides a simple method of breaking information up into small pieces that can be quickly searched and guarantees that the information can be recalled correctly every time. It can emphasize speed or size depending upon how it is designed and provides a great balance of both when designed correctly.