UNDERSTANDING NORMALIZATION

A solid database structure is the foundation of any successful database application, and you will inevitably encounter problems if your database is poorly designed. Regardless of how you access your data, the information you retrieve is only as good as the data upon which it is based. In this paper, I'll cover a traditional database design topic that seems to be quite a hurdle for many database designers and developers: *Normalization*. Normalization is the process of refining table structures into a proper state so that they can store data as efficiently as possible. Here you'll discover why Normalization is crucial and learn how to normalize your tables using Normal Forms. First you'll learn about modification anomalies and Dependency Theory – two issues crucial to understanding Normal Forms. Then, I'll discuss each Normal Form in detail and you'll learn how they resolve problems such as multi-valued dependencies, transitive dependencies and poor data integrity.

The Normalization Process

One of the most challenging aspects of traditional database design is **Normalization**. Much has been written about it, many people have struggled with it, and some have actually understood it. Yet, despite its reputation as a difficult process to learn and implement, it's actually pretty easy once you understand three fundamental concepts:

- The overall premise behind Normalization
- Modification anomalies
- Data dependencies

If you learn about these concepts first, you'll find it easier to grasp the theories behind each of the Normal Forms. We'll start by defining Normalization.

What is Normalization? It is the process of decomposing large, inefficiently structured tables into smaller, more efficiently structured tables without losing any data in the process. Normalization supports the proposition that a well-defined database contains no duplicate data and keeps redundant data to an absolute minimum. This, in turn, guarantees data integrity and ensures that the information retrieved from the database will be accurate and reliable.

Figure 1 illustrates the Normalization process.

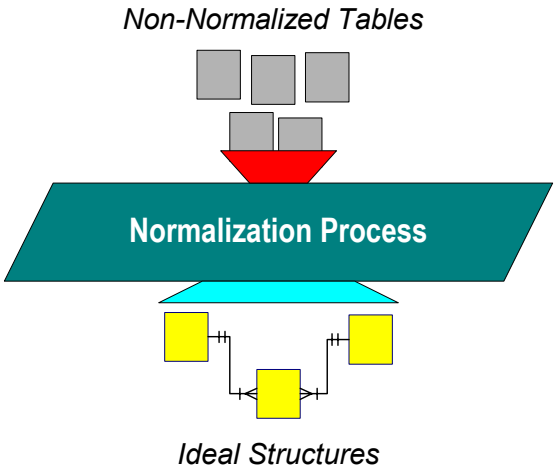

Figure 1. A graphical representation of the Normalization Process

Decomposing improperly structured tables is not an arbitrary process. It is instead a methodical process you perform by testing each table against a set of **Normal Forms**.

A Normal Form is an algorithm you use to test the structure of a table. It helps to eliminate possible table or field anomalies and helps to ensure efficient table and field structures. There are seven Normal Forms, and each one was created to deal with specific types of problems. Here are the names of the Normal Forms, including the general issue upon which each is based.

First Normal Form	Based on Functional Dependency
Second Normal Form	Based on Functional Dependency
Third Normal Form	Based on Functional Dependency
Fourth Normal Form	Based on Multi-Valued Dependency
Fifth Normal Form	Based on Join Dependency
Boyce/Codd Normal Form	Based on Functional Dependency
Domain/Key Normal Form	Based on the definition of Domains and Keys

Now that you have a basic idea of what the Normalization process is about, lets move on to the second fundamental concept: modification anomalies.

Modification Anomalies

We've just discussed the concept behind the Normalization *process*, but we didn't discuss *why* you should even go through the process in the first place. (You would have thought I'd mention it before this moment, but you'll soon see why it makes more sense to discuss it here.)

The main reason you put your tables through the Normalization process is to ensure sound, efficient table structures. Improperly designed tables typically exhibit poor data integrity and are subject to modification anomalies and data dependency problems. If you fail to address these problems, you'll find that the information you retrieve from the database will be inconsistent, inaccurate, and in some extreme cases, totally invalid.

In order for you to understand why you might consider a table to be improperly designed, you must understand the problems it exhibits. (And rather obscurely, as you might think at the moment.) We'll begin by discussing **modification anomalies**.

A constraint placed upon the ability to modify data in a table that is imposed by the table's structure is known as a Modification Anomaly. There are three types of Modification Anomalies that a table can exhibit: Insert, Delete and Update.

Insert Anomaly

An Insert anomaly exists in a table when there is an unnecessary or unreasonable constraint placed upon the task of adding a new record, or when adding a new record will cause unnecessary or unreasonable data redundancy

Figure 2-a illustrates an example of the first type of Insert anomaly. Because data on employees and departments is being stored in the same table, you cannot enter data for a new department until you have at least one employee assigned to the department. Conversely, you cannot add a new employee unless you're ready to assign him or her to a particular department. This type of anomaly can be hard to spot at first glance because it is somewhat subtle.

EmployeesAndDepartments

EmployeeID	EmpFirstName	EmpLastName	EmpPhoneNumber	Department	MeetingRoom	MainPhoneExtension
701	Ann	Patterson	555-2591	Administration	Conference A	112
702	Mary	Thompson	555-2516	Human Resources	26-B	201
703	Thomas	Fuller	555-2581	Human Resources	26-B	201
704	Janet	Leverling	555-2576	Accounting	10-C	116
705	John	Callahan	555-2561	Information Services	12-D	303
706	David	Viescas	555-2661	Administration	Conference A	112
707	Peter	Davolio	555-2601	Accounting	10-C	116
708	Susan	McLain	555-2301	Human Resources	26-B	201
709	Julia	Black	555-7859	Admin	Conference A	112
710	Gary	Holcomb	555-7751	Human Resources	26-B	201

Figure 2-a. Example of an Insert Anomaly imposing an unnecessary constraint.

The second type of Insert anomaly, shown in Figure 2-b, is easier to spot because of the redundant data in the table. In this case, all of the data for a given customer must be repeated when another sales representative is assigned to that customer. This is borne out in the records for Kenneth Peacock.

SalesRepAccounts

CustomerID	CustFirstName	CustLastName	CustCity	CustPhoneNumber	SalesRepFirstName	SalesRepLastName
1001	Suzanne	Viescas	Redmond	555-2686	Paul	Litwin
1002	Will	Thompson	Duvall	555-2681	Katherine	Ehrlich
1003	Gary	Hallmark	Auburn	555-2676	Joe	Rosales
1004	Michael	Davolio	Houston	555-2491	Kendra	Smith
1005	Kenneth	Peacock	Redmond	555-2506	Mike	Hernandez
1005	Kenneth	Peacock	Redmond	555-2506	Mindy	Martin

Figure 2-b. Example of an Insert Anomaly causing redundant data.

Delete Anomaly

A Delete anomaly exists when deleting a record would remove data not intended for deletion.

Figure 3-a shows the table used in Figure 2-a to illustrate an Insert anomaly. The same premise holds – data on both employees and departments is being stored in the table. However, this table also has a Delete anomaly that is just the flip side of the Insert anomaly. In this case, you have the possibility of deleting the only data you have for a particular department if you delete the wrong employee. Such is the case with the "Information Services" department; if you delete the record for John Callahan, you'll also delete the only data you have on that department.

EmployeesAndDepartments

EmployeeID	EmpFirstName	EmpLastName	EmpPhoneNumber	Department	MeetingRoom	MainPhoneExtension
701	Ann	Patterson	555-2591	Administration	Conference A	112
702	Mary	Thompson	555-2516	Human Resources	26-B	201
703	Thomas	Fuller	555-2581	Human Resources	26-B	201
704	Janet	Leverling	555-2576	Accounting	10-C	116
705	John	Callahan	555-2561	Information Services	12-D	303
706	David	Viescas	555-2661	Administration	Conference A	112
707	Peter	Davolio	555-2601	Accounting	10-C	116
708	Susan	McLain	555-2301	Human Resources	26-B	201
709	Julia	Black	555-7859	Admin	Conference A	112
710	Gary	Holcomb	555-7751	Human Resources	26-B	201

Figure 3-a. Example of a Delete Anomaly affecting a single record.

Figure 3-b shows a table that stores data on both sales representatives and orders. This table also has a Delete anomaly, but the results of deleting a record are more serious. If you delete a sales representative, you have the possibility of deleting a large number of

records. Deleting Mike Hernandez, for example, will also delete data on orders 2, 4 and 5!

SalesRepOrders

SalesRepFirstName	SalesRepLastName	OrderID	CustomerID	OrderDate	ShipDate	ShipVia
Paul	Litwin	1	1	5/1/2001	5/4/2001	FedEx
Mike	Hernandez	2	3	5/9/2001	5/12/2001	UPS
Katherine	Ehrlich	3	1	7/4/2001	7/7/2001	UPS
Mike	Hernandez	4	2	8/1/2001	8/4/2001	Postal Express
Mike	Hernandez	5	1	8/2/2001	8/5/2001	FedEx
Katherine	Ehrlich	6	2	8/2/2001	8/5/2001	FedEx

Figure 3-b. Example of a Delete Anomaly affecting multiple records.

Update Anomaly

An update anomaly exists when modifying a specific value necessitates the same modification in other records or tables.

In Figure 4, you would have to make changes to three records if Lone Star Distributors decided to change their name. On the surface, this doesn't seem like a big deal. However, it is a significant problem when you're dealing with a large number of records. You could write programming code to deal with the problem, but then you're writing code that you really shouldn't have to write in the first place. "But I could just update the data using an SQL statement," you say. Unfortunately, that will work *only if all the entries are spelled exactly the same*.

CustomerOrders

OrderID	OrderDate	ShipDate	CompanyName	CustPhoneNumber
1	5/1/2001	5/4/2001	Lone Star Distributors	555-2686
2	5/9/2001	5/12/2001	ABC Manufacturing	555-2676
3	7/4/2001	7/7/2001	Lone Star Distributors	555-2686
4	8/1/2001	8/4/2001	Puget Sound Wholesale	555-2681
5	8/2/2001	8/5/2001	Lone Star Distributors	555-2686
6	8/2/2001	8/5/2001	Puget Sound Wholesale	555-2681

Figure 4. Example of an Update Anomaly imposing unnecessary modifications.

You've probably figured out by now that you want to do everything possible to avoid modification anomalies. You can and will avoid these anomalies by putting each of your tables through the Normalization process.

Let's now take a look at the third and final fundamental concept: **dependencies**.

Dependency: The Good, the Bad & the Ugly

The notion of dependencies (and modification anomalies, for that matter) falls under the umbrella of **Dependency Theory**. Dependency Theory is the field of study comprising Normalization Theory, dependency principles, and other related topics. You learned earlier that most Normal Forms are based on various types of dependencies, and it is for this reason that you must study them. Once you understand dependencies in general, Normal Forms are much easier to learn and understand.

Note This is by no means an exhaustive study of dependencies. The idea here is to provide you with a solid idea of what dependencies are about and how they fit into Normalization.

There are four types of dependencies we'll discuss in this section: functional dependencies, transitive dependencies, multi-valued dependencies, and join dependencies.

Functional Dependency – the Good

A functional dependency (**FD**) exists between two fields, **A** and **B**, when a distinct value of **A** is directly associated with a distinct value of **B**. Given a value in **A** for a specific record in a table, you can always retrieve the associated value in **B** for that record.

A FD is diagrammed as **A→B** and can be read equivalently as either of the following statements.

> "The value of **A** *determines* the value of **B**."
>
> "The value of **B** is *functionally dependent* on the value of **A**."

Note In any dependency diagram you encounter, the field on the left hand side is called the *determinant* and the field on the right hand side is called the *dependent*.

Properly designed tables always contain well-defined functional dependencies. An excellent example of a functional dependency is a Primary Key. The Primary Key functionally determines all non-key fields in the table – given a Primary Key value for a specific record in a table, you can retrieve the values of the remaining non-key fields in that record. (This is true of Candidate Keys as well. But, as you know, only one Candidate Key will serve as the Primary Key of the table.)

Figure 5 illustrates this example quite well. In this case, CustomerID determines the values of the other fields in the table.

Customers

CustomerID	CustFirstName	CustLastName	CustStreetAddress	CustCity	CustState	CustZipCode	CustAreaCode	CustPhoneNumber
1001	Suzanne	Viescas	15127 NE 24th, #383	Redmond	WA	98052	425	555-2686
1002	Will	Thompson	122 Spring River Drive	Duvall	WA	98019	425	555-2681
1003	Gary	Hallmark	Route 2, Box 203B	Auburn	WA	98002	253	555-2676
1004	Michael	Davolio	672 Lamont Ave	Houston	TX	77201	713	555-2491
1005	Kenneth	Peacock	4110 Old Redmond Rd.	Redmond	WA	98052	425	555-2506
1006	John	Viescas	15127 NE 24th, #383	Redmond	WA	98052	425	555-2511
1007	Laura	Callahan	901 Pine Avenue	Portland	OR	97208	503	555-2526
1008	Neil	Patterson	233 West Valley Hwy	San Diego	CA	92199	619	555-2541
1009	Jeffrey	Davolio	507 - 20th Ave. E.	Seattle	WA	98115	206	555-2586
1010	Margaret	Peacock	667 Red River Road	Austin	TX	78710	512	555-2571

Figure 5. Example of a Functional Dependency.

Transitive Dependency – the Bad

Assume three fields, **A**, **B** and **C**, have the following functional dependencies:

A→B

B→C

A transitive dependency (**TD**) exists between **A** and **C** because a distinct value of **A** is *indirectly* associated with distinct value of **C** by way of **B**. Here's the logic behind this statement:

> **A** determines the value of **B**.
>
> **B** determines the value of **C**.
>
> Therefore, **A** transitively determines the value of **C**.

A TD is diagrammed as **A⇒C** and can be read equivalently as either of the following statements.

> "The value of **A** *transitively determines* the value of **C** (via **B**)."
>
> "The value of **C** is *transitively dependent* on the value of **A** (via **B**)."

As you know, a properly designed table represents one and only one subject. However, a table that contains transitive dependencies will describe two or more subjects, depending on the number of transitive dependencies present. (For example, a table with one transitive dependency will describe two subjects and a table with two transitive dependencies will describe three subjects.) A table in this state is improperly designed and is subject to modification anomalies.

Figure 6 shows an Employee table with a transitive dependency between the EmployeeID and Department fields. Here's the logic:

> EmployeeID determines the value of DepartmentID.
>
> DepartmentID determines the value of Department.
>
> Therefore, EmployeeID transitively determines the value of Department.

Employees

EmployeeID	EmpFirstName	EmpLastName	EmpCity	EmpPhoneNumber	DepartmentID	Department
701	Ann	Patterson	Auburn	555-2591	1	Administration
702	Mary	Thompson	Duvall	555-2516	3	Human Resources
703	Thomas	Fuller	Tacoma	555-2581	3	Human Resources
704	Janet	Leverling	Kirkland	555-2576	2	Accounting
705	John	Callahan	Seattle	555-2561	4	Information Services
706	David	Viescas	Redmond	555-2661	1	Administration
707	Peter	Davolio	Seattle	555-2601	2	Accounting
708	Susan	McLain	Bellevue	555-2301	3	Human Resources

Figure 6. Example of Transitive Dependency.

Based on what you've just learned, you can see that the transitive dependency causes the table to describe two distinct subjects: employees and departments. You'll have to put the table through the Normalization process to remove the transitive dependency and keep the table free from all modification anomalies.

Multi-Valued Dependency – the Ugly

A multi-valued dependency (**MVD**) exists between two fields, **A** and **B**, when a distinct value of **A** is directly associated with *two or more* values of **B**.

An MVD is diagrammed as **A→→B** and can be read equivalently as either of the following statements.

> "The value of A *determines* multiple values of B."
>
> "Multiple values of B are *functionally dependent* on the value of A."

A multi-valued dependency can exist at the field level or record level, and two or more distinct, independent multi-valued dependencies can appear in a table simultaneously.

Multi-valued dependencies are similar to transitive dependencies in that their presence in a table indicates that the table describes two or more subjects. Not surprisingly, a table in this state is improperly designed and is subject to modification anomalies.

Figure 7-a shows a table with a *field-level* multi-valued dependency. The transitive dependency exists between the EmployeeID and Committees fields – a single EmployeeID value is associated with one or more Committee values. Although it's not obvious at this point, the table does describe two subjects: Employees and *CommitteeMembers*.

EmployeeCommittees

EmployeeID	EmpFirstName	EmpLastName	EmpPhoneNumber	Committees
701	Ann	Patterson	555-2591	Steering
702	Mary	Thompson	555-2516	Y2K Conformance, Safety
703	Thomas	Fuller	555-2581	Safety, Y2K Conformance, Steering
704	Janet	Leverling	555-2576	
705	John	Callahan	555-2561	Y2K Conformance
706	David	Viescas	555-2661	Steering, Safety, Y2K Conformance
707	Peter	Davolio	555-2601	
708	Susan	McLain	555-2301	Y2K Conformance, Safety

Figure 7-a. Example of a Field-Level Multi-Valued Dependency.

In the table shown in Figure 7-b, the multi-valued dependency is at the *record-level* and exists once again between EmployeeID and Committee. In this case, however, the employee data is repeated for each committee in which the employee participates. And this table, like its counterpart in the previous example, also describes the same two subjects: Employees and CommitteeMembers.

EmployeeCommittees

EmployeeID	EmpFirstName	EmpLastName	EmpPhoneNumber	Committee
701	Ann	Patterson	555-2591	Steering
702	Mary	Thompson	555-2516	Y2K Conformance
702	Mary	Thompson	555-2516	Safety
703	Thomas	Fuller	555-2581	Steering
703	Thomas	Fuller	555-2581	Y2K Conformance
703	Thomas	Fuller	555-2581	Safety
704	Janet	Leverling	555-2576	
705	John	Callahan	555-2561	Y2K Conformance
706	David	Viescas	555-2661	Y2K Conformance
706	David	Viescas	555-2661	Safety
706	David	Viescas	555-2661	Steering
707	Peter	Davolio	555-2601	
708	Susan	McLain	555-2301	Safety
708	Susan	McLain	555-2301	Y2KConformance

Figure 7-b. Example of a Record-Level Multi-Valued Dependency.

Figure 7-c shows an example of a table with two *independent* multi-valued dependencies. One multi-valued dependency exists between EmployeeID and Language, and the other exists between EmployeeID and DeveloperCertification. Note how the employee data is repeated for every language spoken or certification acquired. For example, if Ann Patterson obtains a Visual Studio certification, you'll have to enter yet another record for her in the table.

These multi-valued dependencies are called "independent" because one has absolutely nothing to do with the other. Speaking Spanish is not a requirement for being a Visual InterDev developer, and being a SQL Server developer is not a requirement for speaking German. As you've probably already determined, this table describes three subjects because of the two multi-valued dependencies: Employees, EmployeeLanguages and EmployeeCertifications.

EmployeeInformation

EmployeeID	EmpFirstName	EmpLastName	Language	DeveloperCertification
701	Ann	Patterson	Spanish	
701	Ann	Patterson	German	
701	Ann	Patterson		Visual Basic 6
701	Ann	Patterson		SQL Server
702	Mary	Thompson		Visual InterDev
702	Mary	Thompson	French	
702	Mary	Thompson		SQL Server
703	Thomas	Fuller		SQL Server
704	Janet	Leverling		
705	John	Callahan		Visual InterDev
705	John	Callahan		Visual Basic 6
706	David	Viescas	Italian	
706	David	Viescas	Spanish	
706	David	Viescas	French	

Figure 7-c. Example of two independent Multi-Valued Dependencies.

Regardless of the type, all multi-valued dependencies must be resolved by Normalization so that the table will be free of any modification anomalies

Join Dependency – the Odd Couple

A join dependency (**JD**) exists in table `A` if every record in the table can be reconstructed by an SQL JOIN operation that reunites all tables created by its decomposition. This must hold true for all records existing in table `A` at the time of its decomposition and for any *valid* record that could have been entered prior to its decomposition. (Records added to the decomposed tables must be able to form a valid record for table A when they are united via the JOIN.) Additionally, no records should be lost and no spurious records should be added.

Vendors

VendorID	VendName	Discount	Status	VendCity	VendPhoneNumber	VendWebPage
1	Shinoman, Incorporated	15	Preferred	Bellevue	(425) 888-1234	#http://www.shinoman.com/#
2	Viscount	10	Preferred	St. Louis	(314) 777-1234	#http://www.viscountbikes.com/#
3	Nikoma of America	5	As Needed	Ballard	(206) 666-1234	#http://www.nikomabikes.com/#
4	ProFormance	5	As Needed	Albany	(518) 444-1234	#http://www.ProFormBikes.com/#
5	Kona, Incorporated	7	Regular	Redmond	(425) 333-1234	#http://www.konabikes.com/#
6	Big Sky Mountain Bikes	12	Preferred	Anchorage	(907) 222-1234	
7	Dog Ear	5	As Needed	New York	(212) 888-9876	
8	Sun Sports Suppliers	10	Preferred	Santa Monica	(310) 777-9876	
9	Lone Star Bike Supply	15	Preferred	El Paso	(915) 666-9876	
10	Armadillo Brand	7	Regular	Dallas	(214) 444-9876	#http://www.DilloBikes.com/#

Figure 8-a. Example of a table with a Join Dependency.

You could say that the table in Figure 8-a has a Join dependency because you can decompose it into smaller tables. (Just because you can decompose a table further doesn't necessarily mean you should.) For example, let's say that you wanted to keep sensitive information, such as a vendor's discount or status, from being accessed by everyone in the office. You could decompose this table into two smaller tables (VendorStatus and VendorInformation, respectively) as Figure 8-b shows.

VendorStatus

VendorID	Discount	Status
1	15	Preferred
2	10	Preferred
3	5	As Needed
4	5	As Needed
5	7	Regular
6	12	Preferred
7	5	As Needed
8	10	Preferred
9	15	Preferred
10	7	Regular

VendorInformation

VendorID	VendName	VendCity	VendPhoneNumber	VendWebPage
1	Shinoman, Incorporated	Bellevue	(425) 888-1234	#http://www.shinoman.com/#
2	Viscount	St. Louis	(314) 777-1234	#http://www.viscountbikes.com/#
3	Nikoma of America	Ballard	(206) 666-1234	#http://www.nikomabikes.com/#
4	ProFormance	Albany	(518) 444-1234	#http://www.ProFormBikes.com/#
5	Kona, Incorporated	Redmond	(425) 333-1234	#http://www.konabikes.com/#
6	Big Sky Mountain Bikes	Anchorage	(907) 222-1234	
7	Dog Ear	New York	(212) 888-9876	
8	Sun Sports Suppliers	Santa Monica	(310) 777-9876	
9	Lone Star Bike Supply	El Paso	(915) 666-9876	
10	Armadillo Brand	Dallas	(214) 444-9876	#http://www.DilloBikes.com/#

Figure 8-b. The result of decomposing the Vendors table.

Because of the Join dependency, you should be able to execute the following SQL statement and recreate the original Vendors table. Also, you should not lose any data or gain any bogus records in the process.

```
SELECT VendorInformation.VendorID, VendName, Discount,
       Status, VendCity, VendPhoneNumber, VendWebPage
FROM VendorInformation
INNER JOIN VendorStatus
  ON VendorInformation.VendorID = VendorStatus.VendorID
```

This SQL statement will, in fact, recreate the original Vendors table without any problem.

There is no requirement stating that every table must contain a Join dependency. In fact, the only tables that are candidates for Join dependencies are those that can be further decomposed into smaller tables. Once the decomposition has taken place, the rules stated above must hold for the Join dependency to be valid. Otherwise, you'll have to take the table through the Normalization process to determine whether it should indeed contain a Join dependency.

Well, we've covered all of the fundamental concepts you need to know before tackling Normal Forms. So let's get down to business and move to our discussion on Normal Forms.

Understanding Normal Forms

Before you begin…

So, you've identified your tables and populated them with the fields you believe are most appropriate at this time, and you've even gone so far as to identify relationships between some of the tables. Now you're ready to take them through the Normalization process. Or are you?

There are a couple of things you need to check before you start the Normalization process.

1. Each table must have a Primary Key.
2. A table cannot contain repeating groups of data.

Each of your tables must be in this state in order for the Normalization process to be effective. Otherwise, you could run into problems that could have easily been avoided.

Figure 9 shows a non-Normalized Orders table. Although it's in a sad state at the moment, it is ready for the Normalization process. It does have a Primary Key and it doesn't contain repeating groups of data, per se. (As you can see, there are repeating groups of values within the Items field.)

However, the Items field has two obvious problems: it is a multi-part field and a multi-valued field. It's a multi-part field because its value can be broken down into smaller, more distinct parts. It's a multi-valued field because a single OrderID value can be associated with one or more values within the Items field. (This is a field-level multi-valued dependency, isn't it?)

Orders

OrderId	CustomerId	OrderDate	Items
1	4	5/1/2001	5 32-hammer, 3 2-screwdriver, 6 76-monkey wrench
2	23	5/9/2001	1 32-hammer
3	15	7/4/2001	2 113-deluxe garden hose, 2 121-economy nozzle
4	2	8/1/2001	15 1024-hack saw
5	23	8/2/2001	1 2-screwdriver
6	2	8/2/2001	5 52-key

Figure 9. A Non-Normalized Orders table.

Because the table is in an acceptable state, you'll take it through the Normalization process. (We'll actually work with a few different tables throughout this discussion, but we'll start with this one first.)

Let's begin at the beginning: First Normal Form.

Note: As we examine each Normal Form, I'll start with its technical definition and then provide a layman's explanation and example.

The Normal Form definitions I use here are taken from C.J. Date's *An Introduction to Database Systems, 7th Edition* [Addison Wesley, 2000].

The term "relvar" (for *relational variable*) represents the term "table" for our purposes.

First Normal Form

A relvar is in 1NF if and only if, in every legal value of that relvar, every tuple contains exactly one value for each attribute.

The purpose of this Normal Form is to ensure that a table does not contain any multi-part or multi-valued fields, and that each field holds only a single value for any given record.

You can begin to normalize the Orders table by removing the *multi-valued* characteristics of the Items field.

Orders

OrderId	OrderDate	CustomerId	Item1	Qty1	Price1	Total1	Item2	Qty2	Price2	Total2	Item3	Qty3	Price3	Total3
1	5/1/2001	4	32-hammer	5	$12.99	$64.95	2-screwdriver	8	$7.99	$63.92	76-monkey wrench	6	$8.99	$53.94
2	5/9/2001	23	32-hammer	1	$12.99	$12.99								
3	7/4/2001	15	113-deluxe garden hose	2	$4.50	$9.00	121-economy nozzle	2	$3.85	$7.70				
4	8/1/2001	2	1024-hack saw	15	$17.00	$255.00								
5	8/2/2001	23	2-screwdriver	1	$7.99	$7.99								
6	8/2/2001	2	52-key	5	$1.75	$8.75								

Figure 10-a. Beginning to apply First Normal Form.

The result is that you now have a repeating group of fields: Item1, Quant1, Price1, Item2, Quant2, Price2, etc. You take care of this problem by consolidating them into three distinct fields: Item, Quantity and Price. Additionally, you should remove the *multi-part* characteristics of the Item field by dividing it into two distinct fields: ProductID and Product. Figure 10-b shows the results of these modifications.

Orders

OrderId	OrderDate	CustomerId	ProductId	Product	Quantity	Price	Total
1	5/1/2001	4	32	hammer	5	$12.99	$64.95
1	5/1/2001	4	2	screwdriver	8	$7.99	$63.92
1	5/1/2001	4	76	monkey wrench	6	$8.99	$53.94
2	5/9/2001	23	32	hammer	1	$12.99	$12.99
3	7/4/2001	15	113	deluxe garden hose	2	$4.50	$9.00
3	7/4/2001	15	121	ecomomy nozzle	2	$3.85	$7.70
4	8/1/2001	2	1024	hack saw	15	$17.00	$255.00
5	8/2/2001	23	2	screwdriver	1	$7.99	$7.99
6	8/2/2001	2	52	key	5	$1.75	$8.75

Figure 10-b. The Orders table in First Normal Form.

Although the table is far from perfect, it is now in First Normal Form and is ready to be tested against Second Normal Form.

Second Normal Form

A relvar is in 2NF if and only if it is in 1NF and every non-key attribute is irreducibly dependent on the Primary Key.

Your table should already be in First Normal Form before you reach this point. Second Normal Form then ensures that each non-key field in the table is functionally dependent upon the Primary Key, and that the table does not contain calculated fields.

You can readily see that the Orders table does not conform to Second Normal Form because it has three distinct problems:

1. It actually describes two subjects: Orders and OrderDetails.
2. It contains a calculated field (Total)
3. It contains a transitive dependency between OrderID and Product.

Your first order of business is to decompose the table into two smaller tables called Orders and OrderDetails. This ensures that the Orders table describes a one and only one subject.

Orders

OrderId	CustomerId	OrderDate
1	1	5/1/2001
2	3	5/9/2001
3	1	7/4/2001
4	2	8/1/2001
5	1	8/2/2001
6	2	8/2/2001

OrderDetails

OrderId	ProductId	Product	Quantity	Price	Total
1	32	hammer	5	$12.99	$64.95
1	2	screwdriver	8	$7.99	$63.92
1	76	monkey wrench	6	$8.99	$53.94
2	32	hammer	1	$12.99	$12.99
3	113	deluxe garden hose	2	$4.50	$9.00
3	121	ecomomy nozzle	2	$3.85	$7.70
4	1024	hack saw	15	$17.00	$255.00
5	2	screwdriver	1	$7.99	$7.99
6	52	key	5	$1.75	$8.75

Figure 11-a. Applying Second Normal Form to the Orders table.

The newly revised Orders table is now in Second Normal Form, so you can turn our attention to the OrderDetails table.

Note that the transitive dependency and calculated field have migrated to this table during the decomposition process. Dealing with the calculated field is not a problem because all you have to do is remove it from the table; the transitive dependency is another matter. You learned earlier that the presence of a transitive dependency indicates that the table describes two subjects. In this case, the table actually describes OrderDetails and *Products*. You resolve the transitive dependency by removing the

Product field from the OrderDetails table and then creating a new Products table with ProductID and Products as its fields. It is important that you use the ProductID field as part of the new Products table because it is what will relate the Products table to the OrderDetails table. Once you're finished, the OrderDetails table is in Second Normal Form.

Figure 11-b shows the newly revised OrderDetails table and the new Products table. Pop quiz: Is the new Products table in at least Second Normal Form?

OrderDetails

OrderId	ProductId	Quantity	Price
1	32	5	$12.99
1	2	8	$7.99
1	76	6	$8.99
2	32	1	$12.99
3	113	2	$4.50
3	121	2	$3.85
4	1024	15	$17.00
5	2	1	$7.99
6	52	5	$1.75

Products

ProductId	Product	Price
2	screwdriver	$7.99
32	hammer	$12.99
52	key	$1.75
113	deluxe garden hose	$4.50
121	ecomomy nozzle	$3.85
1024	hack saw	$17.00
76	monkey wrench	$8.99

Figure 11-b. The OrderDetails and Products table in Second Normal Form.

With the OrderDetails table in Second Normal Form, you can now move on to Third Normal Form.

Third Normal Form

A relvar is in 3NF if and only if it is in 2NF and every non-key attribute is non-transitively dependent on the Primary Key.

As the definition states, a table must already be in Second Normal Form before you can apply Third Normal Form. If this is the case, you then apply Third Normal Form to ensure that the table has the following characteristics:

- Each field value is independently updateable; changing the value for one field in a given recrod does not adversely affect the value of any other field in that record.
- Each field identifies a specific characteristic of the table's subject.
- Each non-key field in the table is functionally dependent upon the entire Primary Key
- The table describes one and only one subject.

Because the Orders and OrdersDetail tables are already in Second Normal Form, you can now apply Third Normal Form to both tables.

When you apply Third Normal Form to the Orders table, you end up with the structure shown in Figure 12-a. Notice anything different about this structure and its Second Normal Form counterpart in Figure 11-a?

Orders

OrderId	CustomerId	OrderDate
1	1	5/1/2001
2	3	5/9/2001
3	1	7/4/2001
4	2	8/1/2001
5	1	8/2/2001
6	2	8/2/2001

Figure 12-a. The Orders table in Third Normal Form.

If your answer is "No", then you're absolutely correct. It just so happens that the Orders table already conforms to Third Normal Form, so you don't need to make any modifications to its structure.

If you take a look at the OrderDetails table back in Figure 11-b, you'll see that it has one minor problem: one of its fields does not describe the table's subject. Can you determine which field is the culprit? Here's a hint: the field in question is involved in a transitive dependency.

The offending field is the Price field. Price doesn't represent a specific characteristic of an order detail as much as it describes a specific characteristic of a particular product. Additionally, its value is actually determined by ProductID. If you consider that the OrderDetails table has a Composite Primary Key consisting of OrderID and ProductID, you can see that the value of Price is not dependent on the entire Primary Key, as required by Third Normal Form. You can solve this dilemma by removing Price from the table. (You won't lose anything in the process because Price is already in the Products table.) Figure 12-b shows the result of your modification.

OrderDetails

OrderId	ProductId	Quantity
1	32	5
1	2	8
1	76	6
2	32	1
3	113	2
3	121	2
4	1024	15
5	2	1
6	52	5

Figure 12-b. The OrderDetails table in Third Normal Form.

Let's now take a look at a slightly different way of arriving to the same structure by using Boyce/Codd Normal Form.

Boyce/Codd Normal Form

A relation is in Boyce/Codd Normal Form if and only if the only determinants are Candidate Keys.

Boyce/Codd Normal Form is a different version of Third Normal Form and, indeed, was meant to replace it. The purpose of Boyce/Codd Normal Form is twofold.

- It ensures that a field that determines the value of any or all non-key fields in a table must be a Candidate Key for that table.
- It ensures that a table that describes one and only one subject. (This is implied by enforcing Candidate Keys.)

Boyce/Codd Normal Form is slightly stronger than Third Normal Form in that it deals with the possibility of a table having more than one field that could act as the Primary Key. Provided that you can identify all the valid Candidate Keys in a table, you can ensure that the table is free of transitive dependencies, and by extension, free of modification anomalies.

Note Review time! A Candidate Key is a field or group of fields that has all the required characteristics of a Primary Key, the most important of which is that it determines the values of all non-key fields in the table. After you identify the Candidate Keys for a given table, you select one that will serve as the table's official Primary Key.

Figure 13-a shows an OrderDetails table with three determinants: OrderID and LineItemNumber (taken as a single unit), OrderID and ProductID (taken as a single unit) and ProductID. In order to apply Boyce/Codd Normal Form, you first need to identify if these determinants are Candidate Keys of the table. Would you say that all three are Candidate Keys?

OrderDetails

OrderId	LiineItemNumber	ProductId	Product	Quantity	Price	Total
1	1	32	hammer	5	$12.99	$64.95
1	2	2	screwdriver	8	$7.99	$63.92
1	3	76	monkey wrench	6	$8.99	$53.94
2	1	32	hammer	1	$12.99	$12.99
3	1	113	deluxe garden hose	2	$4.50	$9.00
3	2	121	ecomomy nozzle	2	$3.85	$7.70
4	1	1024	hack saw	15	$17.00	$255.00
5	1	2	screwdriver	1	$7.99	$7.99
6	1	52	key	5	$1.75	$8.75

Figure 13-a. A OrderDetails table containing three determinants.

You are correct if you answered "No" – OrderID\ProductID and OrderID\LineItemNumber are the only Candidate Keys. Although ProductID is not a Candidate Key, it does determine the value of Product and Price. As you may have already guessed, this means that the Product and Price fields are involved in transitive dependencies with both Candidate Keys. You will, therefore, have to remove the Product and Price fields from the table. You'll also have to remove the Total field because neither of the Candidate Keys determines its value. Instead, Quantity and Price determine the value of the Total field. Once you remove these three fields, the table will be in Boyce/Codd Normal Form. Figure 13-b shows the results of your modifications.

OrderDetails

OrderId	LiineItemNumber	ProductId	Quantity
1	1	32	5
1	2	2	8
1	3	76	6
2	1	32	1
3	1	113	2
3	2	121	2
4	1	1024	15
5	1	2	1
6	1	52	5

Figure 13-b. The OrderDetails table in Boyce/Codd Normal Form.

Most tables that are in Boyce/Codd Normal Form will require no further Normalization. However, you'll need to take the table through Fourth Normal Form if it contains any multi-valued dependencies.

Fourth Normal Form

Relvar R is in 4NF if and only if, whenever there exist subsets A and B of the attributes of R such that the (nontrivial) MVD A→→B is satisfied, then all attributes of R are also functionally dependent on A.

The purpose of Fourth Normal Form is to ensure that a table does not contain any multi-valued dependencies, and that it describes one and only one subject. (Have you noticed by now that the latter point is a recurring theme across the higher Normal Forms?)

You learned earlier that a table containing multi-valued dependencies describes two or more subjects, depending on the number of multi-valued dependencies present. You also learned that you must remove all multi-valued dependencies from the table. You'll accomplish this by applying Fourth Normal Form to the table.

Figure 14-a shows a table called EmployeeCommittees that contains a single multi-valued dependency. The first version has a field-level multi-valued dependency and the second version contains a record-level multi-valued dependency. The manner in which you apply Fourth Normal Form to each table is exactly the same and yields the same results.

EmployeeCommittees – Version 1

EmployeeID	EmpFirstName	EmpLastName	EmpPhoneNumber	Committees
701	Ann	Patterson	555-2591	Steering
702	Mary	Thompson	555-2516	Y2K Conformance, Safety
703	Thomas	Fuller	555-2581	Safety, Y2K Conformance, Steering
704	Janet	Leverling	555-2576	
705	John	Callahan	555-2561	Y2K Conformance
706	David	Viescas	555-2661	Steering, Safety, Y2K Conformance
707	Peter	Davolio	555-2601	
708	Susan	McLain	555-2301	Y2K Conformance, Safety

EmployeeCommittees – Version 2

EmployeeID	EmpFirstName	EmpLastName	EmpPhoneNumber	Committee
701	Ann	Patterson	555-2591	Steering
702	Mary	Thompson	555-2516	Y2K Conformance
702	Mary	Thompson	555-2516	Safety
703	Thomas	Fuller	555-2581	Steering
703	Thomas	Fuller	555-2581	Y2K Conformance
703	Thomas	Fuller	555-2581	Safety
704	Janet	Leverling	555-2576	
705	John	Callahan	555-2561	Y2K Conformance
706	David	Viescas	555-2661	Y2K Conformance
706	David	Viescas	555-2661	Safety
706	David	Viescas	555-2661	Steering
707	Peter	Davolio	555-2601	
708	Susan	McLain	555-2301	Safety
708	Susan	McLain	555-2301	Y2KConformance

Figure 14-a. Two versions of the EmployeeCommittees table.

In applying Fourth Normal Form to a table containing a single multi-valued dependency, you build a new table using a copy of the Primary Key and the field containing the multiple values. Using the Primary Key as part of the structure of the new table is important because it relates the new table to the original table. (Be sure to give the new table an appropriate name.) You then decompose the original table by removing the multi-valued field. And Voila! Both the newly revised table and the new table are now in Fourth Normal Form.

Figure 14-b shows the results of applying these steps to the EmployeeCommittes table.

EmployeeCommittees

EmployeeID	Committee
701	Steering
702	Y2K Conformance
702	Safety
703	Steering
703	Y2K Conformance
703	Safety
705	Y2KConformance
706	Y2K Conformance
706	Safety
706	Steering
708	Safety
708	Y2K Conformance

Employees

EmployeeID	EmpFirstName	EmpLastName	EmpPhoneNumber
701	Ann	Patterson	555-2591
702	Mary	Thompson	555-2516
703	Thomas	Fuller	555-2581
704	Janet	Leverling	555-2576
705	John	Callahan	555-2561
706	David	Viescas	555-2661
707	Peter	Davolio	555-2601
708	Susan	McLain	555-2301

Figure 14-b. The Employees and EmployeeCommittees tables in Fourth Normal Form.

If you encounter a table with two or more multi-valued dependencies, you just repeat the same steps for each dependency. For example, Figure 14-c shows an employees table with two independent multi-valued dependencies.

EmployeeInformation

EmployeeID	EmpFirstName	EmpLastName	Language	DeveloperCertification
701	Ann	Patterson	Spanish	
701	Ann	Patterson	German	
701	Ann	Patterson		Visual Basic 6
701	Ann	Patterson		SQL Server
702	Mary	Thompson		Visual InterDev
702	Mary	Thompson	French	
702	Mary	Thompson		SQL Server
703	Thomas	Fuller		SQL Server
704	Janet	Leverling		
705	John	Callahan		Visual InterDev
705	John	Callahan		Visual Basic 6
706	David	Viescas	Italian	
706	David	Viescas	Spanish	
706	David	Viescas	French	

Figure 14-c. An Employees table with two independent multi-valued dependencies.

You'll deal with each multi-valued dependency as you did in the previous example.

1. Create a new table using the Primary Key (EmployeeID) and the first multi-valued field (Language). Give the new table an appropriate name.
2. Create another new table using the Primary Key(EmployeeID) and the second multi-valued field (DeveloperCertification). Give the new table an appropriate name.
3. Remove the two multi-valued fields from the original table.

The newly revised original table (EmployeeInformation) and the two new tables are now in Fourth Normal Form. Figure 14-d shows the results of applying Fourth Normal Form to the EmployeeInformation table.

EmployeeInformation

EmployeeID	EmpFirstName	EmpLastName
701	Ann	Patterson
702	Mary	Thompson
703	Thomas	Fuller
704	Janet	Leverling
705	John	Callahan
706	David	Viescas

EmployeeLanguages

EmployeeID	Language
701	Spanish
701	German
702	French
706	Italian
706	Spanish
706	French

EmployeeCertifications

EmployeeID	DeveloperCertification
701	Visual Basic 6
701	SQL Server
702	Visual InterDev
702	SQL Server
703	SQL Server
705	Visual InterDev
705	Visual Basic 6

Figure 14-d. The EmployeeInformation, EmployeeLanguages, and EmployeeCertifications tables in Fourth Normal Form.

Fifth Normal Form

A relvar R is in 5NF – also called Projection/Join Normal Form (PJ/NF) – if and only if every non-trivial join dependency that holds for R is implied by the Candidate Keys of R.

You learned earlier in this discussion that a join dependency exists for a given table if the table and all of its original records can be reconstructed by an SQL JOIN operation that reunites all tables created by its decomposition. You can test for this type of dependency by using Fifth Normal Form.

By the time a table has achieved Fourth Normal Form, it should be free of all transitive and multi-valued dependencies. In most cases, you shouldn't need to decompose the table any further. However, if you suspect that you can (or should) decompose the table once more, you must test whether there is a valid join dependency in the table. There are three key questions that you must answer before decomposing the table any further:

1. Can I create the new table(s) using the Primary Key or a Candidate Key as part of the new table structure? (Remember that this was a requirement for resolving transitive and multi-valued dependencies.)
2. Can I recreate the original table by using an SQL JOIN operation that reunites all of the tables recreated by the decomposition?
3. Will I lose any records in the process of decomposing the table?

If the answer to each question is "Yes", then the table is in Fifth Normal Form and you can confidently make the decomposition. However, just because you *can* decompose the table further, it doesn't necessarily mean that you *should*.

Figure 15-a shows an Employees table that could possibly be decomposed into smaller tables. Is it in Fifth Normal Form? Study the table for a moment and use the three questions stated above to make your determination.

Employees

EmployeeID	SocialSecurityNumber	EmpFirstName	EmpLastName	EmpCity	EmpState	EmpPhoneNumber	MotherMaidenName	BirthDate
701	111-11-1111	Ann	Patterson	Auburn	WA	555-2591	Peters	5/16/1955
702	222-22-2222	Mary	Thompson	Duvall	WA	555-2516	Jamison	7/1/1949
703	333-33-3333	Thomas	Fuller	Tacoma	WA	555-2581	Ward	6/30/1977
704	444-44-4444	Janet	Leverling	Kirkland	WA	555-2576	Rocca	8/15/1988
705	555-55-5555	John	Callahan	Seattle	WA	555-2561	McClain	4/4/1980
706	666-66-6666	David	Viescas	Redmond	WA	555-2661	Hernandez	7/31/1969
707	777-77-7777	Peter	Davolio	Seattle	WA	555-2601	Romano	5/1/1964
708	888-88-8888	Susan	McLain	Bellevue	WA	555-2301	Stevens	9/5/1956

Figure 15-a. Is this table in Fifth Normal Form?

The answer is "Yes." You *can* use the Primary Key (EmployeeID) or the Candidate Key (SocialSecurityNumber) as part of a new table structure, you *can* recreate the original table, and you *won't* lose any records in the decomposition process.

Why might you possibly want to decompose this table? One possible reason is that you want to separate sensitive employee data from general employee data. For example, you could decompose the original table into the two new tables shown in Figure 15-b.

EmployeeInformation

EmployeeID	EmpFirstName	EmpLastName	EmpCity	EmpState	EmpPhoneNumber
701	Ann	Patterson	Auburn	WA	555-2591
702	Mary	Thompson	Duvall	WA	555-2516
703	Thomas	Fuller	Tacoma	WA	555-2581
704	Janet	Leverling	Kirkland	WA	555-2576
705	John	Callahan	Seattle	WA	555-2561
706	David	Viescas	Redmond	WA	555-2661
707	Peter	Davolio	Seattle	WA	555-2601
708	Susan	McLain	Bellevue	WA	555-2301

EmployeeConfidential

EmployeeID	SocialSecurityNumber	MotherMaidenName	BirthDate
701	111-11-1111	Peters	5/16/1955
702	222-22-2222	Jamison	7/1/1949
703	333-33-3333	Ward	6/30/1977
704	444-44-4444	Rocca	8/15/1988
705	555-55-5555	McClain	4/4/1980
706	666-66-6666	Hernandez	7/31/1969
707	777-77-7777	Romano	5/1/1964
708	888-88-8888	Stevens	9/5/1956

Figure 15-b. New EmployeeInformation and EmployeeConfidential tables.

When you want to see the information from the original table, you can use this SQL statement to do so.

```
SELECT EmployeeInformation.EmployeeID,
       SocialSecurityNumber, EmpFirstName, EmpLastName,
       EmpCity, EmpState, EmpPhoneNumber, MotherMaidenName,
       BirthDate
FROM EmployeeInformation
INNER JOIN EmployeesConfidential
  ON EmployeeInformation.EmployeeID =
     EmployeesConfidential.EmployeeID
```

You won't often run into situations where you need Fifth Normal Form, but it's good to know that it's available when you need it.

Domain/Key Normal Form

Relvar R is said to be in DKNF if and only if every constraint on R is a logical consequence of the domain constraints and key constraints that apply to R.

Domain/Key Normal Form is a newer Normal Form (as Normal Forms go) and is similar to Boyce/Codd Normal Form in that it is partially based on the enforcement of Primary Keys and Candidate Keys. So you already understand at least that much of this Normal Form.

But it is also based upon the idea of Domains. Most texts on database design state that a Domain is merely a set of acceptable values from which a specific field can draw its own values. That's only partially true; a Domain is much more than that. A Domain has two sides: a logical side and a physical side. The logical side deals with issues such as default values, range of values, whether the value is required and whether the value can be Null. The physical side deals with issues such as data type, length, decimal places

and allowable characters. Once you understand this idea, you can use this Normal Form.

In order for a table to be in Domain/Key Normal Form it must fulfill these requirements.

1. Each field must be fully and properly defined.
2. Each field must represent a characteristic of the table's subject.
3. Each non-key field in the table must be functionally dependent upon the entire Primary Key.
4. Each table should represent only a single subject. (Sound familiar?)

A table in Domain/Key Normal Form will be free of transitive dependencies, multi-valued dependencies and modification anomalies. In fact, a table in Domain/Key Normal Form is automatically in Fifth Normal Form.

Figure 16-a shows an Employees table that might be a candidate for Domain/Key Normal Form. Can you Normalize this table further by applying Domain/Key Normal Form?

Employees

EmployeeID	EmpFirstName	EmpLastName	EmpCity	EmpPhoneNumber	DepartmentID	Department
701	Ann	Patterson	Auburn	555-2591	1	Administration
702	Mary	Thompson	Duvall	555-2516	3	Human Resources
703	Thomas	Fuller	Tacoma	555-2581	3	Human Resources
704	Janet	Leverling	Kirkland	555-2576	2	Accounting
705	John	Callahan	Seattle	555-2561	4	Information Services
706	David	Viescas	Redmond	555-2661	1	Administration
707	Peter	Davolio	Seattle	555-2601	2	Accounting
708	Susan	McLain	Bellevue	555-2301	3	Human Resources

Figure 16-a. Can you Normalize this Employees table any further?

You can indeed. One of the requirements of Domain/Key Normal Form is that the Primary Key determines the value every non-key column in the table. This certainly isn't the case with the Department field. You'll have to remove the Department field in order to place the table in Domain/Key Normal Form. Figure 16-b shows the result of your modification.

Employees

EmployeeID	EmpFirstName	EmpLastName	EmpCity	EmpPhoneNumber	DepartmentID
701	Ann	Patterson	Auburn	555-2591	1
702	Mary	Thompson	Duvall	555-2516	3
703	Thomas	Fuller	Tacoma	555-2581	3
704	Janet	Leverling	Kirkland	555-2576	2
705	John	Callahan	Seattle	555-2561	4
706	David	Viescas	Redmond	555-2661	1
707	Peter	Davolio	Seattle	555-2601	2
708	Susan	McLain	Bellevue	555-2301	3

Figure 16-b. The Employees table in Domain/Key Normal Form.

Using Domain/Key Normal Form depends as much on intuition as anything else. You have to understand the concepts of Keys and Domains thoroughly before you can really benefit from this Normal Form. But it works pretty well once you get the hang of it.

De-Normalization

Now we tackle one of the most oft' asked questions: "What about de-Normalization?" The real answer is this: Ask a dozen database developers and you'll get a dozen opinions.

Many people who de-Normalize their database structures do so for “performance reasons”. Here are some of the problems they claim to encounter.

- Queries run slowly.
- Reports take too long to print.
- On-screen forms take time to populate.
- Web pages take too long to populate.

Speed and performance are really relative to a user's definition of the terms. What some people may consider painfully slow may be as fast as greased lightning for others. You should always take performance complaints with a grain of salt and try to take the time to investigate the true nature of a perceived problem.

Although you could justify some de-normalization from a strictly pragmatic sense, you should always de-Normalize as a last resort. Instead, try some of the following remedies before you embark upon any de-Normalization.

- *Update your computer equipment.* Prices are really low and you can purchase a powerful system quite inexpensively.
- *Optimize the operating environment.* Do what you can to optimize your network by working closely with your network administrator.
- *Optimize the RDBMS program.* Make certain you're loading only those pieces of the software that you really need, and carefully tweak any options and settings that are available to you.
- *Use indexes effectively.* Indexes can speed up query processing enormously, so using them judiciously can have a very positive effect on the time it takes to retrieve information from your database.
- *Write good, tight procedural code.* Make certain that you're using optimal code structures and that you're code offers that path of least resistance to your data.
- *Write well-structured SQL statements.* Although there are several ways to pose the same query, some statements are optimized better than others. Be sure to check your database documentation for more information on this subject.
- **START WITH A NORMALIZED STRUCTURE!** 'Nuff said.

The most important point for you to remember is that you will *always* re-introduce data integrity problems when you de-Normalize your structures! This means that it becomes incumbent upon you or the user to deal with this issue. Either way, it imposes an unnecessary burden upon the both of you. De-Normalization is one issue that you'll have to weigh and decide for yourself whether the perceived benefits are worth the extra effort it will take to maintain the database properly.

A Final Thought

In all the years that I've been involved in the database management profession, I've learned that database design is as much an art as it is a science. You learn the SCIENCE of database design through instruction and training, and the ***art*** of database design through personal experience. You'll find that studying database theory, design and technology is an on-going exercise and a rewarding experience.

This job *never* gets boring!