1 WHAT IS A DATABASE?

A database is an organized or structured collection of data. It contains the data necessary for some purpose or purposes. The data is arranged to allow a set of activities to occur, as well as ensuring the data can be accessed and altered in an efficient manner.

2 WHAT IS A DATABASE MANAGEMENT SYSTEM?

A database management system is software that manages data in an organized or structured way.
This definition follows very logically from the definition of a database given above. Common features offered by most database management systems include:

- A way to define containers (typically tables and columns) to hold data.
- Commands and other features allowing the user to search for and sort the data.
- Commands allowing the user to insert new data, update and delete existing data.
- A GUI builder, or other feature allowing the user to create forms to view the data
- A report generator allowing data to be printed
- Enforcement of relational integrity and business rule constraints
- Management of simultaneous data access among multiple users

3 WHAT IS A RELATIONAL DATABASE?

Early attempts to create computer databases were closely tied with the physical storage of data on disk. This resulted in onerous requirements on the database developer and user in order to get the desired results. For example, to search for records on disk, it was necessary to write code to sequentially load data using physical disk addresses.
The relational database model was developed to solve that problem. Fundamentally, the relational database model was created to divorce the design and use of databases from the storage of data on disk. This concept is known as data independence.
The relational database model views data in table form.

Part

Part Number	Description
27	Umbrella Stand
32	Spittoon
48	Buggy Whip

In relational terms, this collection of data is known as a table. Thus, the relational database model views data in terms of collections of data in tabular form. A relational database is simply a database in which the user perceives that all data contained within the system is in the form of tables. Notice that the physical storage of records is not relevant. The

only relevant view of the data is the user's view. How the data is stored on disk is immaterial for our purposes.

4 BASIC RELATIONAL DATABASE TERMINOLOGY

The foregoing discussion contained some terms which we will now take the time to define formally.

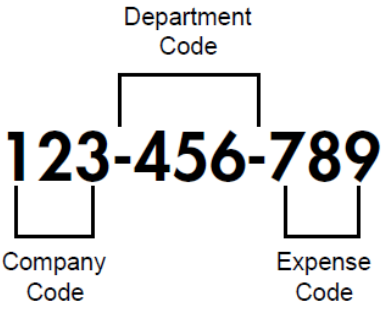
4.1 TABLE

A table is a two-dimensional collection of data. That is, it consists of columns and rows. Look again at the table shown previously. It looks a lot like a spreadsheet, doesn't it? But there is a unique feature which distinguishes a table from a spreadsheet.
A table is non-positional. This has two aspects. First, this means that the order in which the columns are presented can be switched around at will. Similarly, the order in which rows are displayed can be changed at any time. All of this can be done without changing the table at all. (While this can be done with a spreadsheet, it cannot be done without changing the spreadsheet.)

4.2 COLUMN

The definition of column follows from that of table given above. A column is simply an item of data which is stored in every row in a table. Columns are sometime called attributes since they contain something about the table which the database is intended to store.

Columns should contain only one item of data in each row. Stated another way, a column should not contain more than one item of data in a single row. Technically, this is referred to as a composite column. An example of this is a typical accounting code as shown on the diagram to the right for a column called Account Number.



Avoid this type of column in your databases whenever possible. It may make accountants smile, but it makes your databases difficult to work with. Either you must redundantly store the department code data in another column, or a search to retrieve accounts for a specific department will require a contains type search. Both alternatives are messy.

There are other problems which arise when one of the components of the composite column changes. For example, suppose that an expense code is transferred from one department to another. This would theoretically require that you change the value in the Account Number

column as well. That may not be possible, especially if this column is being used to identify each row in the table. The alternative is to leave the data in the composite column unchanged, and update other columns containing the components of the composite column. If you do that, your database is now inconsistent.

A better approach is to split the composite column up into three individual columns, as shown on the following example:

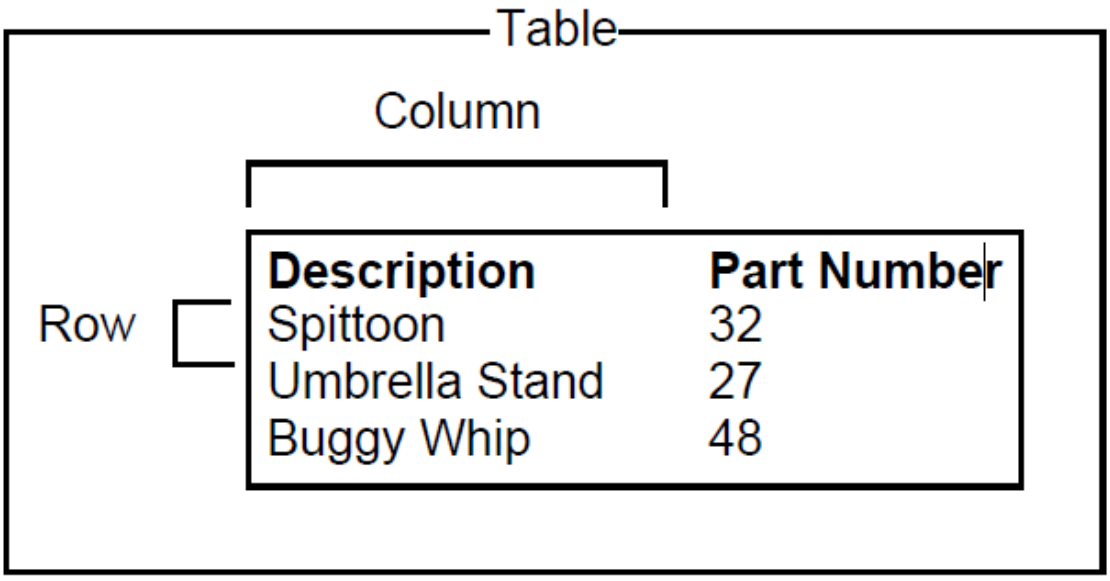
Company Code	Department Code	Expense Code
123	456	7890

Each of these three columns now contains one, and only one, item of data. The quality of a column which contains one, and only one, data value for each row is called atomic. When you design a database, make your columns as atomic as possible.

4.3 Row

Like columns, the definition of row follows from the definition of table given above. A row is simply a single instance of whatever is stored in a table. Every row contains every column in the table, even if the value in any given column is undefined.

The following diagram summarizes what we have discussed on tables, columns and rows:



4.4 Primary key

The concept of a primary key is basic to the relational database model. The relational database model states that every row in a table must be unique. Therefore, there must be

either a single column or a combination of columns which uniquely identifies each and every row in the table. This column or combination of columns is referred to as the primary key.

Thus, the definition of a primary key is the column or combination of columns which can be used to uniquely identify one row from any other row in the table.

Now consider three tables instead of just one:

Part

Description	Part Number
Spittoon	32
Umbrella Stand	27
Buggy Whip	48

Warehouse

Warehouse Code	Warehouse City
A	Buffalo
B	Hong Kong
C	New York

Stock

Part Number	Warehouse Code	Stock Quantity
27	A	10
27	B	15
27	C	7
32	B	25
48	A	8
48	B	12

The primary key of [Part] is [Part]Part Number. This column uniquely identifies each and every row in this table. Similarly, the primary key of [Warehouse] is [Warehouse]Warehouse Code.

When we get to [Stock], the issue is not so obvious. There is not any single column which uniquely identifies every row of this table.

The answer is that [Stock] has a multi-column primary key. The primary key of [Stock] consists of two columns: [Stock]Part Number and [Stock]Warehouse Code. A primary key, like [Part]Part Number, which contains only one column is referred to as a simple key. A primary key which contains more than one column, like the combination of [Stock]Part Number and [Stock]Warehouse Code, is referred to as a composite key.

Another aspect of a primary key is permanence. Permanence indicates that the value in

the primary key will never change over the life of a row. For example, examine the following table:

Person

Last Name	First Name	SSN
Williams	Doris	458-38-6214
Worth	Karla	155-78-1648
Vernon	Mark	788-45-3546
Jones	Marc	258-45-3577

While the combination of [Person]Last Name and [Person]First Name uniquely identifies each row in this table, there is a problem with the permanence of these columns. Assume that Mark Vernon and Karla Worth get married, and they decide to hyphenate their last names. Suddenly the values stored in the table become:

Person

Last Name	First Name SSN	
Williams	Doris	458-38-6214
Worth-Vernon	Karla	155-78-1648
Worth-Vernon	Mark	788-45-3546
Jones	Marc	258-45-3577

That might seem fine, just looking at this table alone, but what if you used these columns to relate to other tables? (See the section below on relations.) Now suddenly, you must change all the values stored in the related tables. But wait! That might affect other tables related to the related tables. This process can go on endlessly.
This points out another obvious fact. In a composite key, all columns must be unchanging. Any change to any of the columns in the primary key results in a change to the primary key, and that is not allowed.

The obvious solution is to make sure that all columns in the primary key never change during the life of a row. Looking at the [Person] table you might choose the [Person]SSN column as the primary key. (This is very common.)
But there is a problem with this approach as well. Obviously, a primary key must be defined for every row in the table. (It must be mandatory.) Assume that Karla and Mark have a baby whom they name Jason. You now wish to enter a row for Jason, but during the first two years of his life, Jason doesn't have a value for [Person]SSN, leading to the following:

Person

Last Name	First Name SSN	
Williams	Doris	458-38-6214
Worth-Vernon	Karla	155-78-1648

Worth-Vernon	Mark	788-45-3546
Jones	Marc	258-45-3577
Worth-Vernon	Jason	

This is unacceptable. There is no way to uniquely identify Jason's row. (By the way, this also implies that every column in a composite key must be mandatory.)

For this reason, you decide to create a unique column to serve as the primary key, as shown:

Person

Last Name	First Name	SSN	Person Number
Williams	Doris	458-38-6214	1
Worth-Vernon	Karla	155-78-1648	2
Worth-Vernon	Mark	788-45-3546	3
Jones	Marc	258-45-3577	4
Worth-Vernon	Jason	5	

A primary key (like [Person]SSN if it works) which is already contained in the table, and is mandatory, permanent and uniquely identifies every row is referred to as a natural key.
A primary key which you manufacture yourself (such as [Person]Person Number, as shown above) is referred to as a synthetic key.
Natural keys are rare, but they do exist.

One other thing about primary keys. All columns in a table which are not in the primary key is called non-key columns. For example, in the above table, [Person]Last Name, [Person]First Name and [Person]SSN are non-key columns.

4.5 Relations

Closely related to the concept of primary key is that of a relation. Take another look at the [Stock] table:

Stock

Part Number	Warehouse Code	Stock Quantity
27	A	10
27	B	15
27	C	7
32	B	25
48	A	8
48	B	12

As indicated above, the combination of [Stock]Part Number and [Stock]Warehouse Code is the primary key for this table. Now assume that you wish to see the [Part]Description of each part which is contained in the [Stock] table. By creating a relation, you can now produce the following view:

Stock/Part Description View

Part	Number	Description from Part	Warehouse Code	Quantity
27	Umbrella Stand	A	10	
27	Umbrella Stand	B	15	
27	Umbrella Stand	C	7	
32	Spittoon	B	25	
48	Buggy Whip	A	8	
48	Buggy Whip	B	12	

Notice that the values for Description are being pulled from the [Part] table. That is to say, you are not redundantly storing the Description in every row in the [Stock] table. You can do this because the values stored in [Stock]Part Number match the values stored in [Part]Part Number. Notice, for example that the value for Umbrella Stand (Part Number 27) matches in both [Stock] and [Part].
This points out several things about the relational database model:

- The relation database model states that links between tables should be performed only by matching data values stored in columns. In the example above, [Stock]Part Number is being used to match to [Part]Part Number.
- This feature allows two tables to be related. A relation is simply the link between two tables both of which contain matching data in columns. (In this guide, we will use the terms related, relation and relationship interchangeably. They all refer to a link between two tables as described in this section.)
- A relation is two-way. You should be able to navigate to the matching row or rows of either table from a row of the other table. For example, you should be able to get the [Part] row for any given [Stock] row, and you should be able to get the matching [Stock] rows for any given [Part] row.
- Relations come in three flavors: many-to-many, one-to-many or one-to-one.
- A one-to-many relation is shown like this: 1:M.
- A many-to-many relation is shown like this: M:M.
- A one-to-one relation is shown like this: 1:1.
- A table on the one side of a 1:M relation is called the one table. A table on the many side of a 1:M relation is called the many table.

M:M relations can be shown during the design phase, but a M:M relation must eventually be resolved into two 1:M relations using a third linking table. (More on this later.)
A foreign key is a column or set of columns which is being used to match values in a relation

and is not the primary key of its table. In a correctly designed database, a foreign key will always be on the many side of a 1:M relation. Also, the matching set of columns of the one table will always be the primary key of the one table. If either of these statements is not true, your design is wrong.
Look again at our example using the [Stock] table. The relation between the [Part] table and the [Stock] table is 1:M. That a part can be in stock in more than one warehouse, but a [Stock] row can only refer to one part. In this situation, the matching column of the many table, i.e. [Stock]Part Number, is the foreign key of the relation.

Notice that the matching column of the one table, i.e. [Part]Part Number, is the primary key of [Part]. The reason why the matching set of columns on the one side of a 1:M relation is always the primary key of the one table is simple: The primary key is the only way to reliably identify the desired row in the one table. A primary key is guaranteed to be permanent, mandatory and unique. This means that the primary key is always the best way to identify a row in a table.

Also, notice that [Stock]Part Number is not the primary key of [Stock]. (While Stock]Part Number participates in the primary key of [Stock], it is not by itself the primary key of [Stock].)