DATABASE ARTICLE 1

How Does My Data Get Stored?

How the data is stored in a database is probably much simpler than you might think. Databases use a series of **Tables** to store the data. A table simply refers to a two dimensional representation of your data using columns and rows. For example:

John	Smith	jsmith@huh.com
Paul	McCartney	paul@beatles.com
Bill	Murray	gopher@caddyshack.com

So then, how does the database keep things straight? Well, first each database table is given a unique name. Without a unique name the **DBMS** (DataBase Management System) would get very confused.

Next, each column in the table is given a unique name. In our example above it would be something like first_name, last_name, email. This doesn't mean each column that you name has to be unique within the entire database. It only has to be unique within the table you have created. Also, notice that the names don't use any spaces. When naming tables and columns be sure to keep it simple with letters and numbers. Spaces and symbols can be illegal characters that will mess up the whole works, so if you need to clarify a name use the "_" instead of spaces.
Let's update our table now:

Table name: contacts

first_name	last_name	email
John	Smith	jsmith@huh.com
Paul	McCartney	paul@beatles.com
Bill	Murray	gopher@caddyshack.com

How Do I Organize My Data?

The next thing to understand about your table is the **Primary Key**. The Primary Key simply refers to a column in your table that is guaranteed unique. The Primary Key is then used for the purposes of indexing your table which makes it much more efficient to search, sort, link, etc.

So what is the Primary Key in our example? Good question. There is none. In our example, there is nothing that is going to be guaranteed unique. Obviously there are many people that share the same last name and/or first name that may be added to the database in the future. The email address is much more likely to be unique but what if two people shared the same email?

To avoid the uncertainty of using a data column as a primary key, many developers will create their own column which contains a computer generated unique number, an ID number of sorts. This way you

don't ever have to worry about its uniqueness since the database knows not to ever use the same number twice.

Table name: contacts

contact_id	first_name	last_name	email
1	John	Smith	jsmith@huh.com
2	Paul	McCartney	paul@beatles.com
3	Bill	Murray	gopher@caddyshack.com

How many tables should I use?

That depends on how you can logically break down your data. There is no limit to the number of tables, or columns for that matter, you can create. Keep in mind, though, that one huge table will be very inefficient while a bunch of little tables can be nearly impossible to keep straight. The best solution usually lies somewhere in the middle.

Here's an example: Let's say our contact table stores contact information for a subscription database for HTML Goodies. Now we need to store what newsletter(s) each person wants to subscribe to. We could simply add another column in our contact table that would store the name of the newsletter. This would allow us to save the information we need but cause names and email addresses to be duplicated, once for each different newsletter a person subscribes to. That would be highly inefficient.

What about making a second table for the names of the newsletters. This way each newsletter name and description would be stored only once.

Table name: newsletters

newsletter_id	name	description
1	Goodies to Go	A newsletter for HTML fans.
2	Design Goodies	A newsletter for web & graphic designers.

How Does It All Relate?

That's great. I have the people in one table and the newsletters in another table. How the heck am I supposed to know who is subscribed to what?

This is the best part. This is where the **Relational Database** gets its name. Relational Database? You never mentioned that.

So, far you have learned some of the basic elements of a database. Now you will learn how to take those basic elements and make them relate to one another so that the information you are storing remains logically linked together. Hence the new and improved Relational Database.

Now, using our example above we have a table of subscribers (contacts) and a table of newsletter information (newsletters). So, how do we know who subscribed to what? We make another table that links the two tables we already have together.

Table Name: contact_newsletter_link

contact_id	newsletter_id
1	2
2	1
2	2
3	1

This table then places a link between the contact table and the newsletters table using each tables unique ID number. So, by referring to the table above you can see that John Smith has subscribed to Goodies to Go, Paul McCartney subscribed to Goodies to Go & Design Goodies and Bill Murray subscribed to Design Goodies.

But what about that Primary Key thing you mentioned earlier? The ID numbers are used more than once. Nothing is unique.

That's true. In this instance we are using **Foreign Keys**. A Foreign Key basically means that the number used in a Foreign Key column is not necessarily unique to the table it is in but is unique to the table it is referring to. In this case contact_id is unique to the contact table but not necessarily unique to the contact_newsletter_link table.

We now have the basics of a relational database.

What is a Data Type?

Well, before we get started making a table there is one thing you need to understand first, **Data Types**. Data Types are pretty straight forward. They indicate the type of data that you are storing in a given table column. Amazing, huh?

So, what are the different Data Types? Here is a list of some of the most common ones and what type of values they hold:

CHAR This will hold between 0 and 255 characters. You have to specify the maximum amount of characters that you will be putting here when you create your column.

LONG VARCHAR This will hold as many characters as you like up to 2 gigabytes of space.

BIT This can hold a value of 0 or 1. It is primarily used for yes/no and true/false issues. It is also referred to as a Boolean or Yes/No field.

FLOAT This type is used to store decimal numbers. It is primarily used for mathematical purposes.

INT	This type indicates that you are storing whole numbers here. You can store any whole number between -2147483648 and 2147483648.
SMALLINT	Same as above except you are limited to numbers between -32768 and 32768.
DATE	This stores a date. I know you're shocked.
DATETIME	This will store a date and time. It is also commonly referred to as a TIMESTAMP since it is primarily used to Time Stamp entries or updates to a row or table.

Granted, there are a whole lot more but these will get you started and are among the most common. You will also find that every DBMS has its own quirks and syntax. That means the same Data Type can be referenced by different names in different DBMS's.

You'll have to check your DBMS documentation for specifics.

How Do I Create a Table?

Creating a table is very easy. I'm going to show you 2 different ways to accomplish this task. The first will be to use a SQL statement. The second will be using a Graphical User Interface (GUI) tool. For our GUI example we are going to use OpenOffice Base since it is one of the most common databases out there today and, more importantly, it's already installed on my machine.

We'll use the example we started in Part 1.

First the SQL Statement:

```
CREATE TABLE contacts (
    contact_id INT IDENTITY (1, 1) NOT NULL ,
    first_name CHAR (30) NULL ,
    last_name CHAR (50) NULL ,
    email VARCHAR (75) NULL
);
```

At first glance this statement can be a little bit intimidating but it really makes sense once you break it down.

Alright, CREATE TABLE makes sense. This tells the DBMS to make a new table called "contacts".

Now for the part in the parentheses. First we have "contact_id". If you remember, "contact_id" will hold our unique index number that we call our Primary Key. That means we know that column will always be a whole number so we make it an INT (Integer). Next we use IDENTITY which tells the DBMS that this is our Primary Key. Now, the (1,1) means that we want to start with the number 1 and we want to increment each new ID number by 1. So, in our example, John Smith would have an ID of 1, Paul McCartney would have an ID of 2, etc. Lastly, there is the statement NOT NULL. This means that this column must have a value and can never be empty. This is a required element of any Primary Key.

Next is "first_name". Here we are stating that "first_name" is a character column (CHAR) that will store 30 or less characters in it. We are also saying the "first_name" can be empty if we want it to be. That is what the NULL statement is for.

The creating of "last_name" is exactly like "first_name" except we are allowing up to 50 characters in this column.

Lastly, we have "email". Just to be different I used VARCHAR instead of CHAR. Essentially, it is the same as the CHAR in the statement and serves the same purpose, storing a set of alpha-numeric characters (a string). The real difference between CHAR and VARCHAR is the VARCHAR is variable, hence the VAR. This means that when someone new is entered in the database the "first_name" and "last_name" columns will have a fixed size, 30 and 50 respectively, no matter how little information you put in them.

So, if you entered the name "John" in the "first_name" column, it would take up a 30 character block of storage instead of only the 4 characters it really needs. With VARCHAR it would only take up as much space is needed for storage. VARCHAR does come with a minimum storage requirement, so you will have to check your DBMS documentation to determine whether of not VARCHAR is going to be a benefit to you. VARCHAR has one other important feature, it can store a heck of a lot more than 255 characters if you want it to.

That's it. Alright, so it wasn't really easy but at least it is logical.