Preferred Customer Dave Lieberman 6 197 2001-02-20T18:26:00Z 2001-01-20T18:08:00Z 2001-02-20T22:07:00Z 9 3651 20816 Extensity, Inc. 173 41 25563 9.2720 1

AWD san francisco database integration

WHAT IS A DATABASE, ANYWAY?

By Guest Author,
Aaron Cheeseman, MSDBA

Database Terms

A database is simply a container of information. To find an example of a database that most people use everyday, you need look no further than your address book or the yellow pages. Websites use databases to keep track of information, mostly for Internet commerce or recorded user information and usage. In fact, before we bring computers into this, lets cover some basic database terms that are universal.

Item

Lets consider the type of information thats in an address book. Youve got peoples names, peoples addresses, peoples phone number, maybe even peoples birthdays. Youll notice that the common element here is "people". In this example, each person is considered an ***item***. In other words, the item is the very thing or things that the database is storing information about. A database can certainly store information about multiple sets of items (customers, orders, products, and so on) but our address book example is a simple one, and well keep it that way for the moment.

Field

Now, what kinds of questions are we asking about those people whose information we wish to store in our database? Whats your address? Whats your first name? Whens your birthday? Each question that we ask about our items is a ***field***.

Record

If you met someone, and you wished to add them to your address book, you would ask all of your questions, adding them into your book, thus creating a new ***record***. All of the answers about one particular item is known as a record. Careful, there is room here for confusion about the distinction between items and records, so lets put it this way: the *item* is the very thing that were storing information about. The item exists, whether or not theres a database. A *record* is the set of information stored in your database about one of the items.

Value

Finally, a ***value*** is the actual text (or numerical amount or date) that is used to fill in the blank when adding information to your database. Put it all together, and it looks something like this:

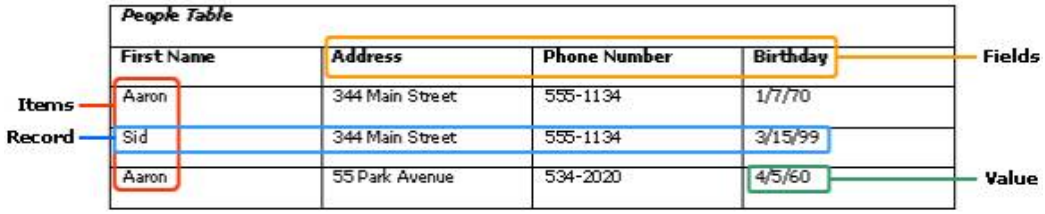
Putting it all Together in a Table

People Table			
First Name	**Address**	**Phone Number**	**Birthday**
Aaron	344 Main Street	555-1134	1/7/70
Sid	344 Main Street	555-1134	3/15/99
Aaron	55 Park Avenue	534-2020	4/5/60

Table 1A basic table

When we put the information all together in a grid like this, that object is called a ***table***. A database can exist with just one table, or a database can have lots of tables. The main idea here is that each *table* deals with one sort of *item*. A company would have a separate table for their customers, a table of their products, a table of their suppliers, a table of their orders, and so on. *The database is really just a container that holds tables and manages how they can be used.* Well examine the relationship between a database and the tables it contains in more detail soon.

For now, lets make some observations about our People Table. Each person, that is, each *item*, gets a *record*. In this little table of ours, weve got three records, so apparently, weve got information on three people. Weve asked four questions (name, address, phone number and birthday) so wed say there are four *fields*. Fields are almost always shown in columns, with the records being in the rows, and so the terms *field* and *column* are sometimes used interchangeably, as are *records* and *rows*. And an example of an actual *value* would be, say, *555-1134* for the first records phone number.



Keeping Things Straight

An important thing to note here is that two of the items have the same First Name (Aaron), but they clearly arent the same person, since we can see they have different birthdays. Also, two people have the same value for address, so they must be housemates. When creating a database, an important feature is ***record uniqueness***. In any given database, some items may have the same value for a particular field. That is, two people might have any number of coincidences such as the same name or the same address. Imagine if a son, Jeff Spicoli, whos named after his father Jeff Spicoli, registers on a website. The database software would see two records for Jeff Spicoli for the same address. Clearly, it is important to be able to distinguish between separate items having duplicate values from accidental duplication of records caused by user or computer error. Usually, this distinction is accomplished using some sort of number that uniquely identifies a record. Sometimes real data, such as a social security number, is used for this identification. However, there is often no such pre-existing number or name known to us that is 100% unique to an item. In this case, almost every database system has a feature that allows the database designer to create a special additional field in a table in which it automatically assigns unique sequential numbers for each new entry. (Stand by for another database vocabulary word.) Generally speaking, every table has a ***key field***, which is simply a designated field that stores 100% unique values throughout the database. As I said before, the key field may contain real data

that is unique for each item, such as a social security number, or, it may be a special field where no manual data entry takes place, but instead, the system automatically assigns a new number for each newly added record. In our modified *Table 2* below, the field called "User_name" would certainly be designated as the key field since no two users are supposed to have the same user name. If two records display the same user name, we can be sure that the database programmer goofed. If the database designer had properly set the user_name field as the key field, then when the user tried to add a fourth record starting with, say, Burning Elk, he would have gotten an error message and the record would not be added.

User_ID	FirstName	Address	PhoneNumber	Birthday
Cool_Dude71	Aaron	344 Main Street	555-1134	1/7/70
Burning Elk	Sid	344 Main Street	555-1134	3/15/99
AirMan	Aaron	55 Park Avenue	534-2020	4/5/60
~~Burning Elk~~	~~Sid~~	~~344 Main Street~~	~~555-1134~~	~~3/15/99~~

Table 2--With User_ID # used as a key field, no duplicate records will be allowed into the table

Tables and Relational databases

The address book is a useful but limited example. It only makes use of one table. Most databases are going to be a bit more complex. Lets suppose that youre an online crack dealer and you want to record all data pertaining to your sales over the past month, including the name, address and phone number of the customer, when they purchased their crack, and how much they bought. Sounds easy. Why not just make a table called Sales, and store all the information there? The problem is that, though it could certainly be done, it would unnecessarily (and greatly) increase the amount of computational time spent performing data entry, of storage space, and of the number of errors introduced into the system. Why? Lets take a look at such a table, and determine why its a poor design:

OrderID	***Client***	***ClientAddress***	***Phone***	***SaleDate***	***#RocksBought***
10000	Sally	96 Fremont Ave	555-1526	12/5/00	2
10001	Chadwick	101 Haley Street	465-9877	12/5/00	3
10002	Chadwick	101 Haley Street	465-9877	12/6/00	1
10003	Chadwick	101 Haley Street	465-9877	12/7/00	2
10004	Gwen	41 Lacey Lane	987-5678	12/10/00	1
10005	Sally	96 Fremont Ave	555-1526	12/11/00	2
10006	Gwen	41 Lacey Lane	987-5678	12/19/00	9
10007	Sally	96 Fremont Ave	555-1526	12/13/00	5

Table 3A poor design

As you can see, we have small but loyal clientele. But with the way the table is set up, *every* time well sell to Chadwick, well have to store his name, address, and phone number once again. This is pointless. We already have his contact information, so theres no need to add it again. Moreover, sticking to this design is likely to introduce errors and make Chadwick mad. Every time Chad logs on to buy a fat rock, he has to type:

Chadwick	101 Haley Street	465-9877

At some point, it is likely that some of Chadwicks information will be entered incorrectly. And so if we ever query the database to ask how many purchases "Chadwick" has made, sales where his name was typed incorrectly wouldnt appear in the result.

So instead, it would make much more sense to have two tablesone for Client information, and one for Sales. We would need to add a key field in the Sales table (in this case, *ClientID*) so that, when we make a new sale, we can provide a reference which "points" to the client in the Client table that made that particular purchase. It would look something like this:

Table 4--We take the poorly designed Table 3, and break it into a Clients table and...

Clients			
ClientID	***ClientName***	***ClientAddress***	***Phone***
100	Sally	96 Fremont Ave	555-1526
101	Chadwick	101 Haley Street	465-9877
102	Gwen	41 Lacey Lane	987-5678

5--An Orders table

Orders			
OrderID	***ClientID***	***SaleDate***	***#RocksBought***
10000	100	12/5/00	2
10001	**101**	12/5/00	3
10002	**101**	12/6/00	1
10003	**101**	12/7/00	2
10004	102	12/10/00	1
10005	100	12/11/00	2
10006	102	12/19/00	9
10007	100	12/13/00	5

Things are much easier to manage now. If we get a new client, they are added once to the Clients table. If we get a new order, to indicate whos placed the order, we can simply require the client to type their ClientID, and then fill in the order information.

The typical end user will never have direct access to database tables. They only see what you want them to see, and they will select from the options you give them. But under the hood of the database, the efficiency with which you store information will be critical to both database speed and storage space.

By the way, remember the term ***key field*** that I threw at you a few paragraphs ago? It is a field in a table that stores a unique value such as social security number that uniquely identifies an item in the database. In the Clients table, which field do you think the designer is going to designate as the key field? Sure, its got to be the ClientID. Note that in this case, were not asking our clients for personal information like their social security number, so weve chose to have the system generate clandestine numbers in the key field.

Type once, Refer Many

Another thing I want you to notice about our two tables here is that even though the field ClientID also appears in both tables, ClientID values in the Clients table arent allowed to be duplicated. (If properly set up, an error would occur if you tried to add a duplicate ClientID number.) But in the *Orders* table, the ClientID field has plenty of duplicate values because they are identifying which of your customers placed the order. The term used to describe the relationship between such tables is ***parent* and *child***. Related tables like these will share a common field (actually, the name of the fields can be different, but the data must be common). To keep straight which is the parent in the relationship, look to see which table has duplicate values in the common field. Just as a mother can have many children but a child cannot have many (biological) mothers, the parent table will contain only unique values in the common field, and the child table will have lots of duplicate values in the common

field, which are used to make references to those unique values in the parent table. In other words, we *store data about people* once, but we *refer to those people* many times throughout the database. This will increase the speed at which the database can work, and will reduce storage space.

Remember that when we first defined tables, we said that a table contains information about *one set* of items? A close examination of Table 3 will show that the reason its design is poor is that its got information about *two* sets of items: Client items and Order items. Note that, though the Order table certainly needs to have some reference to a person (because *people* place *orders*), information like address and phone number is about the *client* and so it is sufficient to provide just a reference to the client in the Order table.

A very important rule in relational database design is that each table should contain information about one and only one set of items. So the term relational database essentially means that instead of storing every bit of information we have in one gigantic, completely unmanageable table, we have a bunch of separate tables with defined relationships joining them to each other. Now this is something on which we dont need to go into great detailafter all, thats the job of the database designerbut you do need to understand the nature of the relationships between the tables. Lets take another look at our new Clients and Orders. Do you think both tables will grow at the same rate? Probably not. Chances are pretty good that if were selling quality crack, the rate of new orders from customers will be much greater than the rate of finding new customers. Moreover, one customer may place many orders, but one particular order cannot be placed by many customers. There are a few different types of table relationships, but 99% of the relationships will be like that between our clients and their orders, and it is known as a *one-to-many relationship*. Hmm, I wonder how all the tables in a real database will look and relate to each other

Full-blown Relational Databases

Lets see how all the tables and relationships come together in a typical database.

In Figure 3, we are selling Groceries. WebVan, look out! Below, you can see eight boxes, which represent eight tables in our database. The names of the tables are at the top of each box and identify the type of information stored within it. The words in the box represent all the fields in each table. As you can see, weve got Suppliers, the Products they sell to us, and the Categories that the foods fall into (e.g., beverages, deserts, and so). Weve got Employees, some of which will be salespeople who will be associated with the Orders that are placed with them by our Customers. Well need Shippers to ship our ordered products, of course.

We have a Orders table and a Orders Details table. Why do we need two tabled for Orders? Consider that when one customer places one order, there is information that is pertinent to the *whole* order such as who placed the order, when they need it, when we actually shipped it. However, within that order, there will be probably be lots of different products purchased, and each product they purchase will have its own small set of information such as how many of that particular product was ordered. Therefore, when an actual order is placed for 3 boxes of apples, a box of toothpaste, and 10 boxes of crack, well add *one* row to the Orders table, with its own unique OrderID, indicating the who, when, and whatnot of the whole order. In the OrderDetails table *Three corresponding rows* with the same OrderID would be placed so we can record the quantity, price, and discount of each product ordered, without repeating the who, when, and whatnot of the whole order for each product ordered.

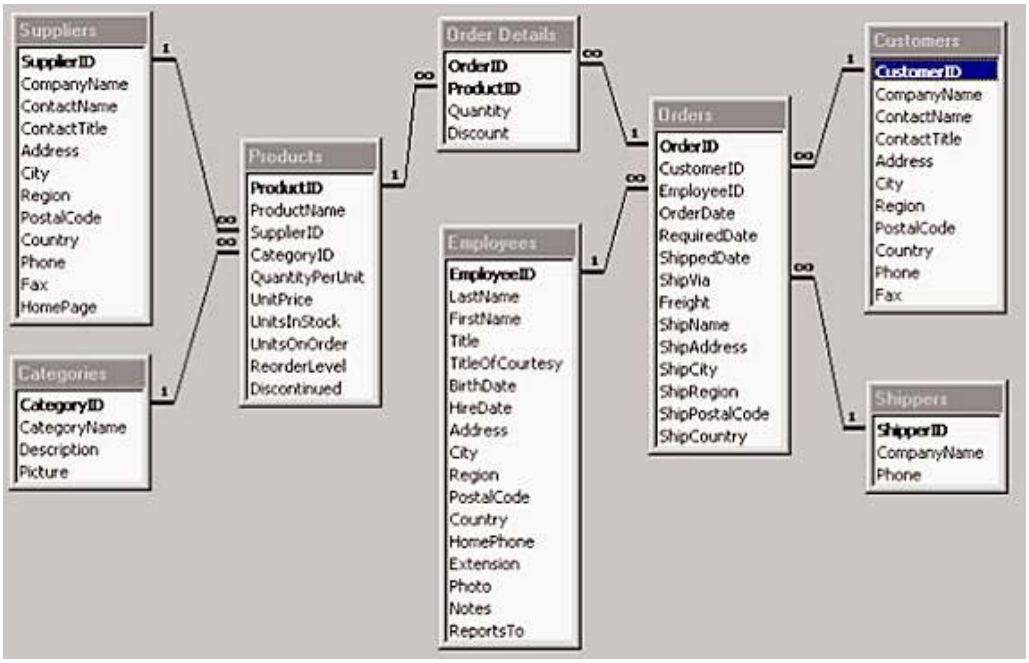


Figure 3 The relationships between tables of a database

Interpretation of Graphical Database Symbols

The lines joining the tables identify the relationships between the tables. Notice the little 1s and ∞s? (FYI, the little ∞ is the mathematical symbol for infinity, but in this case, is used to imply *many*.)Theyre there to show whos the parent of whom. It actually makes a good deal of sense when you look at it this way. Notice that Products is a parent of Order Details, and a child of Suppliers and Categories (unlike real life, a tables parents arent necessary related to each other except for their common child table). In other words, **one** supplier probably sells us **many** different products. And, **one** food category likely contains **many** products. However **one** product will certainly be purchased through **many** orders. So in the case of Products and Suppliers, both have a field called SupplierID. In the parent table (Suppliers), SupplierID is a number that uniquely identifies each supplier we do business with, but in the child table (Products) SupplierID makes a reference indicating which of our suppliers sold us that particular product.

Queries

At this point, you may ask yourself a number of questions. The key question, I suspect, has to do with unifying the data. That is, weve gone to great lengths to separate the data into tables. When we need to display our information all together, such as in an invoice, how does that happen? Thats where queries come in.

Just as the name suggests, a query is something we use to gather specific information from our vast pool of data. For example, suppose I want to produce a complete report of all orders placed during a particular month. There is no one table that will do the job. Although we do have a table called Orders, it only has numerical references to the customers, the shippers, and the employee involved with the sale. Even though humans dont have an easy time memorizing lots of numbers and associating them with real people, computers excel in it! Not only will a query help us join all that information together, it will also help us ***filter*** that information. We dont want to see every order weve ever had. We only want to see orders whose date falls within a particular month. So *filtering* just means the query is also going use criteria we give it to hide some data and present only what we want to see. Some databases, like MS Access, have graphical means to create queries, but most industrial

strength RDBMS (a common abbreviation that stands for Relational DataBase Management System) dont offer GUI tools. Thats where SQL, pronounced "sequel", comes in. It stands for Structured Query Language, and any database administrator worth his or her salt will know basic SQL commands to query and configure their database. In fact, the tools provided by programs such as Access and Dreamweaver UltraDev are actually graphical facades that create SQL statements on what you click and drag.

While it is not critical that you learn SQL, you very well may encounter SQL programmers in the future, so I will show you a very basic example of how it works. This example will also solidify some concepts from the previous section. Lets refer back to Figure 1. Suppose I want to see a list of our products and their respective prices, but I dont want to be distracted by all the other information in the table. Easy! *Where* we type the following SQL statement depends on the application were working with, but whatever it is, well provided a place to type and execute this command:

```
SELECT Products.ProductName, Products.UnitPrice
FROM Products
```

The SELECT keyword indicates the field or fields you wish to be displayed. Youll notice that each item following the SELECT statement has two parts. The syntax is just *tablename.fieldname*. So here, were asking to display the fields ProductName and UnitPrice from the Products table. The FROM statement lists all the tables that are being used in this operation. Here, its just Products. In Figure1, notice that the Products table has a field called UnitsInStock? Suppose that I want to display a list similar to the one above, but I only want to see such a list of products that are in stock. We would only need to add a line to our SQL code like this:

```
SELECT Products.ProductName, Products.UnitPrice
FROM Products
WHERE Products.UnitsInStock > 0
```

Okay, I wont kill you with code, but you get the idea. As computer languages go, SQL is awfully straightforward. A database designer will create a user interface that your client can use to query the database without typing code.

As an "extra credit example" (meaning, if it doesnt make sense to you, dont sweat it), Ill show you how a query would unite data that is spread across two tables. Suppose I still want to show a list of our in-stock products, but I want to add a third column to display the name of the supplier of that product. The Product table does have a field called SupplierID, but thats a *number*. We want to show the suppliers name instead. We can predict how the SELECT statement will change. Well just add one more item, which will be the SupplierName field from the Suppliers table (remembering to precede the field name with the name of the table its in). The trickier part is in the FROM statement. We not only need to include a second table in our statement, we need a simple way to describe how, given a particular product, which name from the Suppliers table we will associate with in. It all looks like this:

```
SELECT Products.ProductName, Products.UnitPrice, Suppliers.CompanyName
FROM Products INNER JOIN Suppliers ON Suppliers.SupplierID = Products.SupplierID
WHERE Products.UnitsInStock > 0
```

The ***TableA* INNER JOIN *TableB*** part of the FROM statement says these table have related information. The **ON TableA.FieldA =TableB.FieldB** part indicates what fields will match a record in one table to that of another. In this case, its the SupplierID field in both tables that will identify Suppliers in one table, and refer to them in the other.

Performance Issues

While it wont be your job to serve and protect the database itself, you very well may deal with DBAs (database administrators) who will throw strange new terms at you. If you find that youre waiting a surprisingly long time to view data, you may have to talk to them! There are lots of factors that could cause this common symptom. Lets look examine what some of them are, so that when you deal with those DBAs, youll understand what theyre talking about.

"The Network is Slow"

This is another world unto itself, and a slow network can be caused by a vast number of problems including insufficient server RAM or processing capability. For every person that simultaneously connects to a database, that requires the server to allocate a portion of its memory to that connection. Too many connections + not enough memory on the server = a frustrated user. If the server has all the greatest hardware available, that wont ensure that youll have quick easy access to the data you need to test your product. A critical feature of databases is the Index. The index is a component that speeds up table queries. To explain what an index does, heres a good analogy. Suppose you want to look up the word [AC1] in the dictionary. Would you start on page one in the A section, and start scanning down the pages until you hit the word you want? Not unless youve got a lot of time on your hands and you dont know how to alphabetize lists. Chances are, you would skip the first 2/3 of the book, perhaps finding yourself in the Q section. Would you begin scanning from there? Probably not. Youd probably hold your place in the Qs with one finger, jump ahead another half-inch in the book, find yourself in the U section, and deduce that the word you want is located in that half-inch of pages youve got in your hand. You would proceed in a similar manner until you find the page that ***sesquipedalian*** must be on. This is essentially the job of the index. It speeds up searches on a field by extrapolating the approximate location of the item or items youre looking for by jumping ahead to computer-calculated points in the list of records, so it doesnt have to scan the entire list. An index wont make much difference if you only have a couple of users and your database is relatively small. But imagine if you have a million records, and dozens of people were simultaneously searching for information from the same database. Then, an index will make the difference between a quick response and the agony of waiting five minutes for one short list of records.

Queries are also useful ways of speeding your construction and testing process. If you were building an interface to a very large database, you would not want to casually call an entire table. One table might contain gigabytes of raw data. Calling the whole set of information will not only take a long time, youre also going to tie up the network and spread frustration among others. A query can be used to limit the data you gain access to. For example, instead of calling all orders, you could test your product by calling just orders from the previous week.

Check out the web design course we offer!

-

|

PAGE /# "'Page: '#'

'" [AC1]

sesquipedalian /ses-kwi-pi-DAYL-yun/, *adjective*:

1. Given to the overuse of long words; "sesquipedalian orators"
2. (Of words) long and ponderous; having many syllables; as, "sesquipedalian technical terms"
