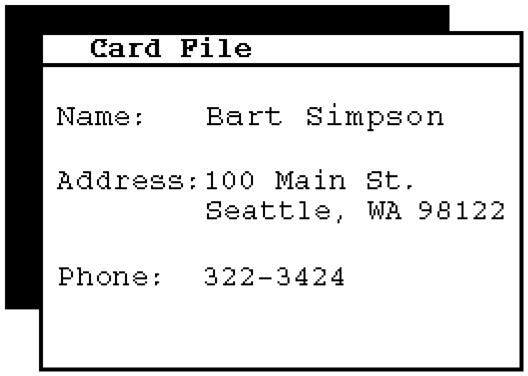
Introduction

The average American is somehow “processed” by a database upwards of 40 times a day. In most of today’s organizations, PC and mainframe databases form a range of applications from straightforward customer list management to mission-critical programs. Yet databases are not nearly as well understood as spreadsheets and word processors. Be it because of database programs’ size, conceptual complexity, or their interdependence with the fast-changing world of networking, many computer users steer a wide path around databases.

This document provides a brief introduction to the most important concepts of the PC database world. It begins with a working definition of a database, moves on to a short history of database management systems, sketches the current market and outlines its players, and ends with a glossary of database terminology.

What is a Database?

A database is a collection of information related to one topic, organized so that a user can easily look at it, add to it, and change it.



Name: Bart Simpson

Address: 100 Main St.
Seattle, WA 98122

Phone: 322-3424

Although most people think only of computerized databases, in fact we are surrounded by databases of all types. A dictionary, for instance, is a common database that allows users to determine a word’s spelling, part of speech, and definition quickly and easily. Similarly, a card file containing friends’ names and addresses, a phone book, a collection of recipes, and *TV Guide* are all common forms of databases. Computerized databases include customer files, employee rosters, equipment inventories, and sales transactions.

Database design requires some planning.

To use a database effectively, the database designer must organize the database in a logical way. Imagine how annoying it would be to use a dictionary not printed in alphabetical order! Database design is in some sense arbitrary: while for most people a dictionary is most useful when arranged alphabetically, some linguists might prefer one sorted by part of speech, with separate sections containing nouns, verbs, and adjectives. Likewise, a company’s customer database might simply contain a block of information called “customer address,” or might have several smaller pieces of data called “street name,” “city,” “state,” and “ZIP code”—the configuration will depend on how the designer intends to use the system. But while there is no one right way to design a database, there are ways that will make it much easier for users to get at relevant information. Indeed, researchers have spent years and written thousands of pages to guide efficient database design.

Databases have different modes of operation.

People use a database by putting data in, asking questions, and getting answers. Each of these modes gives rise to a different interface. For instance, data entry clerks often use screens designed to look like printed forms. These forms, in turn, send the data to the various tables where the data is stored. If a person wants to browse through many records, he would want to see a spreadsheet-like representation of the data. But to ask the database questions, a user might want a simple, uncluttered screen on which to formulate the request, rather than one chock full of data. And the report-generator part of a database system is likely to have a different look entirely, one that makes it easy to design groupings, summary lines, headers, and footers. The effect is that database systems can seem to be several programs in one—like spreadsheets with different modes for entering data and creating charts, databases have modes for designing tables, browsing, creating reports, programming applications, and so on.

The word "database" means different things to different people.

- Some people press spreadsheets into use as databases. While most spreadsheets have database functions, they also have severe limitations that a true database program overcomes. See Section 4.2, below, for a complete discussion of the limitations of using a spreadsheet as a database.
- In the dBASE® world, the word “database” refers to what database theorists define as a “table”; dBASE does not have the concept of a single, unified database. Microsoft® Access, by contrast, uses “database” to mean “a collection of information.”
- When some people refer to a database, they are really talking about a textbase—software that can locate and extract unstructured information (like newspaper articles), without reference to records and fields. None of the products discussed in this paper falls into the category of a textbase.

This paper should help cut through some of the confusion surrounding database terminology. For additional help, see the Glossary—Section 11 at the end of this document.

Who Uses a Database?

People who use databases tend to be spread over a very wide spectrum of sophistication—a significantly wider range than that of spreadsheets and word processors users. Because of the perceived complexity of database systems, and because DBMSs often require users to learn a computer language in order to access their full power, companies with information needs frequently hire developers to build custom applications around a database in order to insulate users from the system’s complexities. Thus, the sophistication spectrum ranges from data entry clerks who may not even realize which database they are using, to developers who write applications using DBMSs. More specifically,

- *Data entry clerks* and *novice users* accomplish tasks with their databases—tasks like maintaining customer files and performing mail merges to generate form letters. In many cases, these users work with applications written by others.
- *Power users* access information with their database, albeit with some degree of difficulty. These are the users who create relatively sophisticated reports like crosstabs, but typically have to turn to their manual to do so.
- *Developers* create database applications for other users—who may be Data Entry clerks—to use. They may not be great interactive users of the product, but have expertise in designing systems for others.

Why Use a Database?

People use databases to keep track of things.

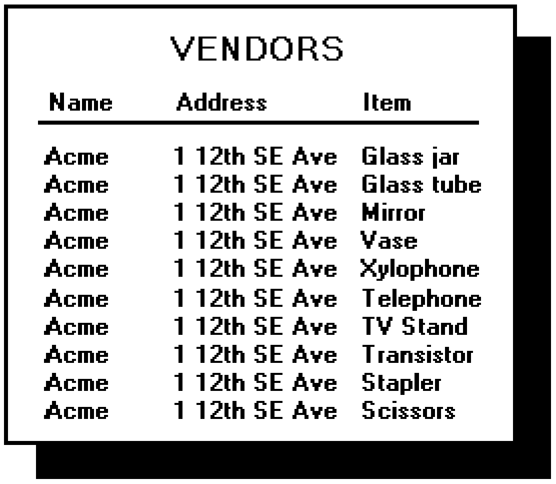
In broad terms, a database allows people to model and manage data. A personnel manager, for instance, would use a database to keep track of employees. Such a database might contain each person's name, ID number, an address, an evaluation, a salary, and perhaps even a picture. The manager might then ask the database to show all employees whose evaluation ranking falls within a specific range, like between 0 and 3. Or the manager might ask the database to increase by 10% the salary of all people with evaluation rankings greater than eight. Finally, such a user might ask for a report of all employees, grouped by ZIP code, so as to mail out the company newsletter.

The office manager, in turn, might use a database to manage the fixed assets of the firm. Each item might have a bar code attached to it, and the database would contain information like purchase price, age, depreciated value, and the employee to whom the equipment is assigned. As part of an office renovation, the office manager might ask the database to tell him the 100 oldest desks at a particular site. In order to track the performance of the purchasing department, reports would include a monthly report on the average new price of certain items.

The purchasing department might have a database of all the qualified office supply vendors, with their contact names, addresses, phone numbers, items stocked, and prices. In this way the department can quickly find out which vendors stock desks, and at what price.

Now let's think about the firm as a whole. A single listing of employees could help both the personnel manager and the office manager: links could be created between employee information and the fixed asset data to show which employee is responsible for what equipment. Of course, this gives rise to security issues, since the personnel manager would not want the office manager to have access the salary information. Moreover, managing multiple sets of data requires good data integrity rules—that is, if an employee leaves, the personnel manager should not be able to delete that employee's name if he still has equipment assigned to him. As will be seen below, the best relational database systems attend to these problems easily.

Why not just use a spreadsheet?



VENDORS

Name	Address	Item
Acme	1 12th SE Ave	Glass jar
Acme	1 12th SE Ave	Glass tube
Acme	1 12th SE Ave	Mirror
Acme	1 12th SE Ave	Vase
Acme	1 12th SE Ave	Xylophone
Acme	1 12th SE Ave	Telephone
Acme	1 12th SE Ave	TV Stand
Acme	1 12th SE Ave	Transistor
Acme	1 12th SE Ave	Stapler
Acme	1 12th SE Ave	Scissors

Let's think closely about this purchasing department alone. You can imagine a spreadsheet set up with columns called "Vendor Name," "Address," "Phone," "Contact," "Item," and "Price." Of course, most vendors stock more than one item, so you have several options. You could make "Item" a very wide field, and then simply type in every item the vendor stocks, perhaps along with the prices. Or you could re-type the info about each vendor for every item. Each of these approaches has problems: by typing all pieces of equipment into one field, you lose the flexibility of being able to see vendors that stock only a particular item.

Re-typing vendors' names is a waste of space and time, and causes problems if the vendor's name, address, or contact name changes.

In general, using a spreadsheet to handle complex data gives rise to a number of problems. While spreadsheets are conceptually straightforward (hence their popularity), they fall short when there are relationships between data, when more than one person needs access to data, or when the amount of data is large. Specifically, spreadsheets stumble with:

- **Referential Integrity**. Imagine a company that uses spreadsheets to keep track of its employees. If an employee leaves the company, there is no single point in a spreadsheet database where a user can go to change every reference to that employee from "current" to "former." A relational database would not allow a user to delete an employee's name without, say, reassigning all the furniture and other assets that belonged to him.
- **Data Integrity and Validity**. A spreadsheet has no means to limit acceptable values that go into a given cell. Any sort of database loses its value if the data cannot be trusted; databases give the designer mechanisms to ensure that values entered into the database are valid. Thus if a designer wanted to limit the data under "Amount of Sale" to positive numbers (and thus force returned merchandise to be handled correctly), she could.
- **Data Redundancy**. As noted above, a user of a spreadsheet database might be forced to re-type multiple instances of the same information, as is the case when one vendor stocks multiple items. Not only is this inefficient, but it also causes problems when items in the database change: if a vendor changes its name, users must then go and change every instance of that name, instead of accessing one central data point.
- **Limiting Data.** Spreadsheets do not allow a user to limit the working set of data—one must always work either with the entire data set or create a subset that users must then use to re-update the larger set.
- **Performance and Capacity**. Large amounts of data slow a spreadsheet to a crawl. And because spreadsheets lack speed-up features like indexes, searching through the data can be an excruciatingly lengthy process.
- **Data Entry**. Spreadsheets have no concept of forms, so only the simplest forms can be reflected on a spreadsheet-based data-entry screen.
- **Reporting**. Similarly, because spreadsheets lack tools to allow quick grouping and summarizing, all reporting functionality must be manually "hard-wired."
- **Programmability**. Spreadsheet macros are notoriously brittle and lack sophisticated control devices and error checking. Most databases, by contrast, are closely tied to full-featured programming languages.
- **Multiple users**. While usable multi-user spreadsheets are still a thing of the future, many databases have supported multiple users from their inception. With a database, multiple users can work with the same information at the same time, without worrying about concurrency problems—the database manages that.

A database solves the problems of the purchasing department, for instance, by allowing users to create multiple sets of data and link them. One might have one data set called "Vendors," with a complete list of vendors' names, addresses, and ID numbers, and a separate set called "Products," with product names and vendor IDs. By linking the two sets on a common characteristic like vendor ID, a user avoids the problems of redundancy and data integrity. Now vendors' names only get entered once, and products can be sorted individually. And multiple users can access up-to-date data simultaneously.

Where do Databases Come From?

The history of database management systems (DBMS) is closely tied to the history of the whole computer industry and has progressed through at least three states or epochs:

Hierarchical DBMS, Network DBMS, and Relational DBMS. Because DBMS software is complex and because of the large intellectual investment and time spent developing applications, once a particular system is adopted by a customer, that system can have a very long life. Systems from different epochs live well into the eras of later technology.

Hierarchical Databases.

In the 1960s the first modern mainframe computers were developed. This included the IBM® System/360 with OS/360, first delivered in 1963. These computers were extremely expensive, and both computing power and data storage consumed a large portion of a company's data processing budget. When storing and accessing data, therefore, the primary concern was to use space and computer time as efficiently as possible.

Hierarchical DBMSs became popular because of their excellent performance and storage efficiency. In such a database, data is organized like an inverted tree—a series of nodes with branches connecting the nodes. These systems were by no means easy to use: to get at data, a user had to navigate through a complex web of hard-wired relationships. But as long as data was accessed in a top-down fashion in a consistent manner, and as long as the relationships between the data did not change, this technology worked well. Unfortunately, this is hardly ever the case.

Network Databases.

During the 1970s, DEC and other companies developed the minicomputer. At the same time, hierarchical databases evolved to become network databases. These DBMSs allowed more complex data relationships to be "hard wired" into the system. Instead of allowing only top-down parent-child relationships, network DBMSs can support entire systems of pre-defined relationships, including lateral links. Of course, the user still had to understand these hard-wired relationships in order to access data.

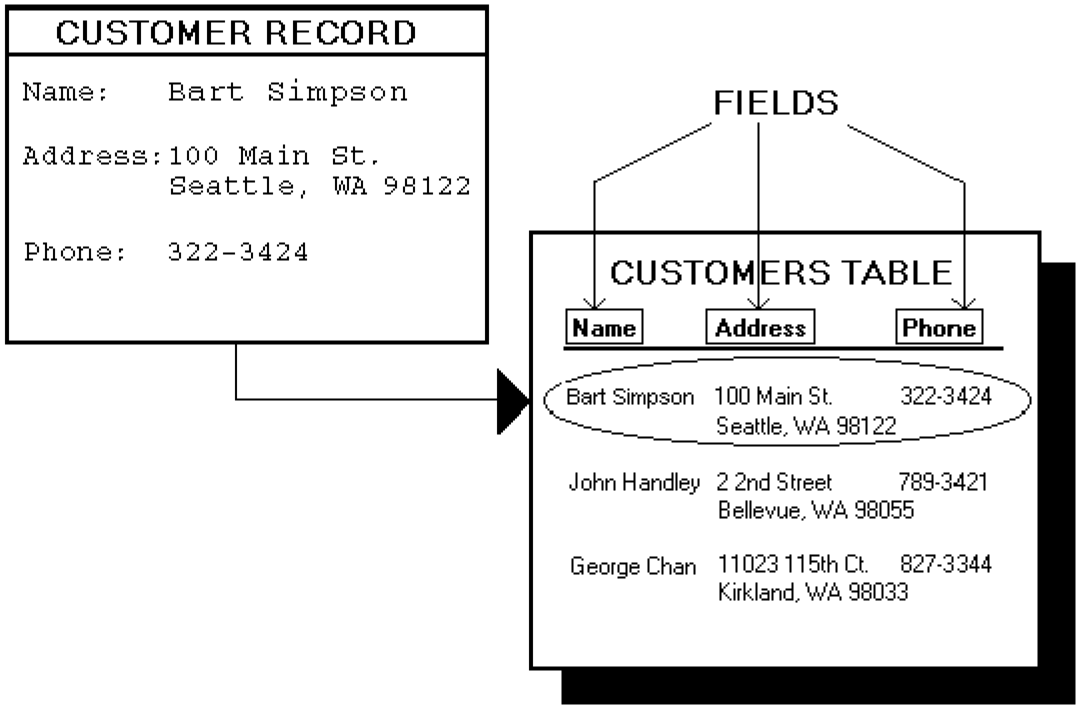
The move towards on-line transaction processing helped change the course of database history. Both hierarchical and network databases moved to support on-line, interactive users, instead of simply batch processing. However, because of the difficulties in changing DBMSs, most hierarchical systems lagged behind network systems in providing on-line transaction support. The network DBMS became the workhorse of the minicomputer-based on-line transaction processing world.

Relational Databases.

In the early 1970s, IBM's E.F. Codd published a paper on his research into a new form of DBMS structure based on a mathematical foundation, the relational database. IBM began work on SYSTEM/R, which later became SQL/DS and then DB2. Oracle® shipped the first commercial relational database system, Oracle, in 1978.

Codd developed the relational model to overcome the problem of database rigidity. Relational systems insulate the end-user from the physical links in the database and allow the relationships between data to be easily restructured, thus responding quickly to changing needs. That is, where before it was necessary to understand and navigate through a maze of relationships, a relational database theoretically made it possible to ignore these underlying complexities.

The central metaphor of a relational DBMS is the table. A table is a set of information concerning a given topic. A table called "Customers" would describe a company's customers by giving the name and related information about each one. By definition, a (correctly used) relational database contains multiple tables. A typical customer database might contain the tables "Customer Names" and "Customer Transactions." And a book publisher might have a database containing the tables "Titles," "Customers," "Authors," and "Commission Rates."


CUSTOMER RECORD
Name: Bart Simpson
Address: 100 Main St. Seattle, WA 98122
Phone: 322-3424
FIELDS
CUSTOMERS TABLE
Name
Address
Phone
Bart Simpson 100 Main St. Seattle, WA 98122 322-3424
John Handley 2 2nd Street Bellevue, WA 98055 789-3421
George Chan 11023 115th Ct. Kirkland, WA 98033 827-3344

Tables are in turn broken down into **records**, which give information about a certain item in a table, and **fields**, which contain discrete nuggets of information about each record. Records are like cards in a 3x5 card file; fields are like the spaces for “Name,” “Address,” and “Phone Number” on each card. In the case of a book publisher, the customer table would probably be organized into records (rows), with each record representing a customer, and fields (columns) named “name,” “address,” and so on. The Titles table would then have a record for each book and fields called “title,” “author,” and “retail price.”

Early relational DBMSs were significantly slower than established hierarchical and network systems. Recently, however, much has been done to improve their performance. Optimizers, for instance, have sped up queries significantly. Just as high-level programming languages relieve developers from managing all the machine details, optimizers adjust queries so that they are executed efficiently regardless of how they are worded. Thus comparing a 100-record table to a 1,000-record table (which requires 100 match attempts) can be made as fast as comparing a 1,000-record table to one with 100 records (which, if inefficiently performed, could comprise as many as 1,000 match attempts).

Enter the PC.

As personal computers became available in the late 1970s, two interesting things happened. First, a large number of users began to realize that the PC was a serious machine. One could build real-world applications on micros. To help support this development, various productivity tools were developed. In addition to spreadsheets and word processors, PC DBMSs became one of the most important tools in a user’s suite of applications.

Second, file management systems became popular. Very few people would use a mainframe to manage lists of personal information. Because of the dedicated nature of the PC, users were interested in sorting personal information, primarily lists. The two sets of needs—list management and application development—clearly segmented the PC DBMS market. Only in the last several years has the PC database market had anything other than two poles: a “high end,” consisting of programming environments, and a “low end” made up of flat-file databases, or “filers.” See Section 7 for a complete description of each of these types of database systems.

How Does One Use a Relational Database?

While user interfaces for relational database programs differ considerably, the underlying functionality is similar regardless of the product. This section describes how one might typically design and use a database. Note that it refers primarily to the programming-oriented relational databases like Paradox®, dBASE, and R:BASE® described above in Section 7.3.

Database design

The first step a user must take is that of designing the database—that is, thinking through the various tables, forms, and reports that will likely be needed. Currently, no product is particularly helpful at this stage: just as the best word processor cannot compose a letter, database products aren't very good at anticipating what structure would best suit a user's needs.

A large part of the challenge of database design results from the fact that the best way for a database to store and manipulate data is frequently not the best way for a user to access that data. As noted above, all databases have various modes in which they operate, e.g., browsing mode, reporting mode, forms creation mode, etc. Each of these modes puts different demands on the database system; what makes life complicated is that a user must think through each of these, considering both his information needs and those of the database itself.

Table design

The first step in which the program is actually used is table design. The user starts with a single table, giving it a name and determining the fields that will comprise it. He then must decide each field's type—that is, what kind of data will be entered into that field, be it alphabetical, numeric, logical, dates, or whatever. A given field can only contain a single type of data, and choosing that type can be of critical importance. Mis-assigning the field "Cost" as an alphanumeric field increases the difficulty of summing that field, for example.

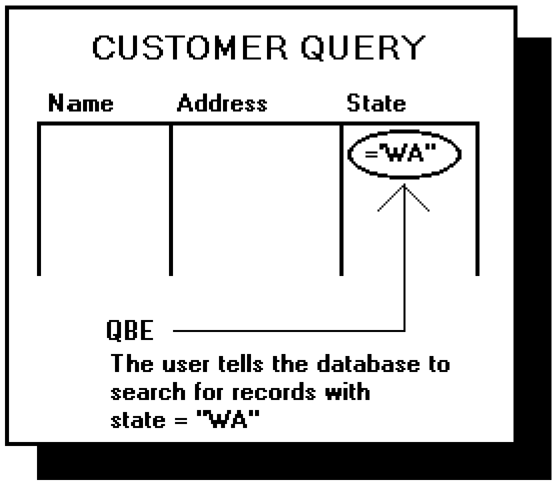
The user then assigns certain properties to each field. For alphabetic fields, for instance, a user might specify a maximum length of 20 characters. Another field might only accept positive numbers in it. A very important step is assigning one or more fields as the primary key which will uniquely identify each record, such as an employee number. This field will both speed up searches and allow users to link tables together based on that key. A user might assign an index to other fields, creating a behind-the-scenes series of pointers that allows the database application to find records quickly within that field.

Data Entry

Having designed one or more tables, a user might choose to enter data (either using the keyboard in a spreadsheet-type browse mode or through some data import facility) or create a form that would allow easier data entry. Most products have a forms creation mode that allows picking and choosing fields from a table and placing them on a form, and then painting the screen to resemble, say, an order form. More complex forms include those with lookup boxes that allow a user to pick from acceptable values from one table to enter into a given field in another table, or list boxes that permit more than one item to be associated with a given record, as is the case when one vendor stocks multiple items.

Queries

To ask the database questions, a user must enter query mode. Many products have several ways to perform queries. For years, the standard way to query a database was to write a line of code in interactive mode that said, for instance, "Sum all Payments for State='WA'." That is, in order to ask a question, the user had to know the language that the database spoke. While these languages are frequently fairly straightforward—they still require considerable time to learn the proper syntax and lexicon.


CUSTOMER QUERY
Name
Address
State
="WA"
QBE
The user tells the database to search for records with state = "WA"

Query By Example, or QBE, offers a means for users to query a database without learning a procedural language. Under a QBE system, a user would see a display of the field names from those tables he wanted to query. He would then move his cursor around the screen, checking off columns that he wanted to see, putting criteria like “WA” under fields like State, and so forth. The user then presses a key to execute the query and get the answer. Now, the implementation of this feature can differ considerably: dBASE and R:BASE use QBE to generate the appropriate line of code, display that code on the screen, and then execute the query. Paradox and DataEase (whose Query by Form is similar) generate no lines of code in the traditional sense; instead, the QBE itself serves as code.

Reports

Another step in using a database is creating reports. As in forms design, a screen painting routine helps the user to lay out the document, add field names, and draw boarders. In addition, a user must determine how to group and summarize the data: reports are usually grouped into meaningful divisions on which the user can subtotal and total. Formatting, headers and footers can then be added to round out the document.

Database Configurations

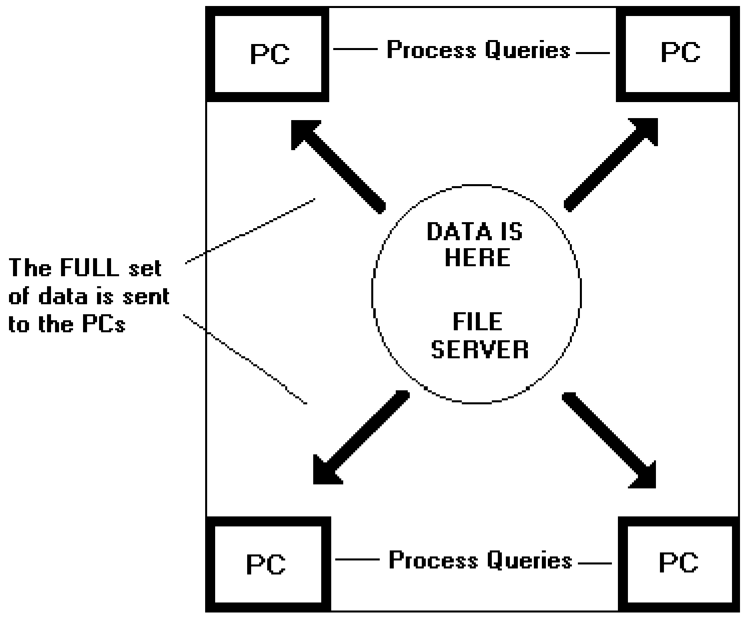
Most relational databases can be configured to work as either a standalone product or as a part of a file server LAN. Increasingly, RDBMSs can serve as the front end in a client-server environment as well, though with varying degrees of ease. This section outlines the differences between these three configurations.

Standalone databases

A standalone database user manipulates data on a single machine. The data resides locally, usually on the machine’s hard disk, and is used by only the person logged on. In the past, the data was almost always homogeneous—that is, a given product could work only with data stored in that product’s format. dBASE users, for instance, could work directly only with dBASE data; using data import and export, they could translate files to and from formats like Lotus and ASCII. Recently, however, DBMSs have been given the ability to manipulate data in non-native formats—R:BASE, for example, can read and write dBASE data files directly, though with some loss of performance.

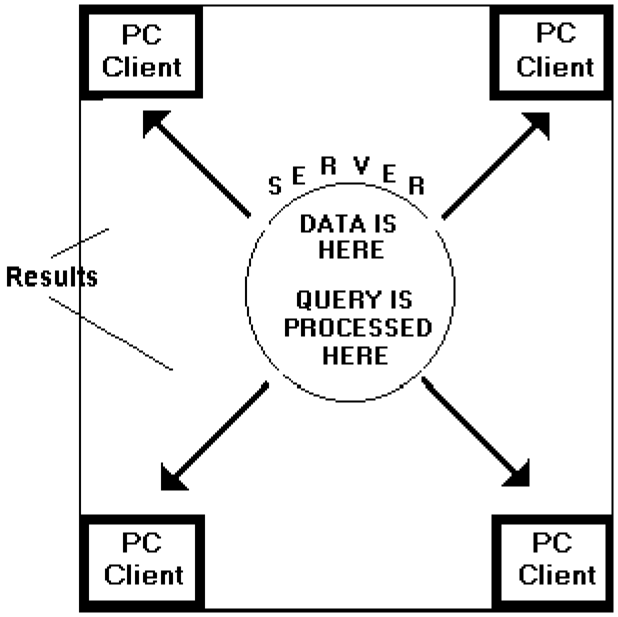
Using a database in standalone mode is completely adequate in many situations. By definition, however, it cannot allow different members of a workgroup to share information easily: no two users can access data simultaneously. Recognizing the many advantages that result from the sharing of enterprise-wide computer data by different functional areas, many companies are now trying to integrate their computing resources.

LAN File Servers



LAN File Servers give companies one way to do this. In this configuration, a single computer is dedicated to holding all database files, controlling access, and providing security. Users then can access this remote data over the network as if it resides on their own computers (though usually with some degradation in speed). In most systems, multiple users can access the same table simultaneously; the program updates the various local tables periodically.

LAN file server arrangements allow data sharing, then, but at some cost. Large amounts of data has to travel across network lines: because all processing is performed at the local level rather than at the server, entire tables must be sent between the server and the user. And because the server functions as little more than a storage device, data integrity and security are problematical.



Client/Server

Client/server architecture refers to a technology that combines the capabilities of large database management systems with the ease of use associated with PCs. As in the LAN file sharing system, the database server is a dedicated machine in a LAN environment that holds all the database files, controls access, and handles security. In addition, however, the server has an “engine” that can look after data integrity and processes requests for data from workstations in the network.

This system provides both centralized database control and generalized database access. The application’s user interface is stored at the individual workstation, while the database engine is installed on a network server. The engine (or back end) provides security and concurrency control. Separating an application into two functions (front end and back end) allows the workstation to be optimized for ease of use and the server to be optimized for maximum throughput.

When a workstation application, or client, requires data for its task, it requests the data from the database server. Rather than give the workstation a copy of the entire database (which may contain thousands of records), the server sends only the relevant subset, reducing both the message traffic across the network and the amount of processing performed at the workstation.

Structured Query Language (SQL, often pronounced “sequel”) is one common way to handle communications between workstations and servers. Developed by IBM for use with its DB2 mainframe relational database, SQL has since moved onto PCs as a way for workstations to gain access to remote data. It is a set-based language designed to perform relational database queries quickly with as few verbs and objects as possible.

The benefits of a standard data manipulation language like SQL are as clear as having a common language among people. As with Esperanto, however, problems arise in the implementation. First, the SQL standard isn’t quite standard. Due in part to the inadequacies of the ANSI standard, SQL’s various vendors have introduced proprietary extensions to customize the language. Second, integrating SQL into another language—say, dBASE—is a tricky affair that can cause almost as many problems as it solves. Finally, while SQL is a powerful data manipulation language, it is a disastrous end-user tool. The syntax can be confusing and queries are often difficult to decipher. Insulating the end-user from SQL’s complexities while allowing programmers to get at its power is a tricky prospect that DBMS makers are still fine tuning.

Glossary

Back-end	The server part of a client/server system that processes requests for data.
Break Level	A reporting function that allows a user to summarize information by category.
Browse Mode	A mode of operation in a database that gives users a spreadsheet-like view of multiple records. Contrast with **Forms Mode.**
Client/Server	Refers to the division of database management into two software components, with the client at the front end sending and requesting data and the server at the back end acting as a data store.
Commit	A statement issued by a user in a multi-user environment that indicates that the user is finished editing certain records and that the database should recognize those changes.
Concurrency	Multiple users accessing data simultaneously gives rise to concurrency issues. DBMS systems use locks to provide a means to ensure data integrity when more than one user is editing data at once.
Crosstab	A query or report that displays its answer in rows and columns. The user chooses items to be displayed horizontally (like months) and vertically (like salespeople), and the database generates an answer that, say, sums each salesperson’s sales by month.
Data Type	A characteristic of a particular field. Databases require that all data within a given field be of a single type. Common types include alphanumeric, numeric, logical, date, and time.
DBMS	Acronym for Database Management System
Engine	The part of the database where data is stored. Can be local (residing on the same PC as the interface) or remote.
Expression	A combination of operators (like < and >) and values (such as text or numbers) that a database evaluates.
Field	A category of information, such as a last name. Fields are usually represented as columns.
Filer	A simple database that deals with only one table at a time. Also called flat-file databases.
Form	An on-screen representation of a data-entry form. Forms usually look simple to the end-user, but often hide complexities under their surface: data entered into a single form may be sent to many different underlying tables, for instance.
Forms Mode	A mode of operation on a database that gives users a customized view of a single record. Often this form resembles a paper form. A single form can present data from multiple tables. Contrast with **Browse Mode.**

Front-end	The client part of a client/server system that comprises the user interface.
Index	A group of pointers that allows a DBMS to find values in a certain field quickly.
Join	A database operation that combines some or all records from two different tables.
Key	One or more fields (columns) that uniquely identify each record. Examples include employee numbers and stock numbers. Relational databases usually require that one field (column) of each table be designated a key field to facilitate linking tables to each other.
Locking	The process of temporarily shutting out users in a multi-user system so that they do not interfere with each other. Locks can be placed either on tables or rows (records), with row locking allowing greater flexibility.
Optimizer	An element of a DBMS that determines the fastest way to resolve queries. Sophisticated optimizers rearrange the sequence of data requests, build temporary indexes, and perform other magic to get the answer back to the user as quickly as possible.
QBE	Query By Example. A way of requesting information by specifying the criteria visually—filling in the blanks—rather than through a programming language.
QBF	Query by Form. A variation on Query By Example that allows users to fill in a data-entry form to limit the working set of data to those records that match the criteria.
Query	A question that a user asks a database. Queries can either request information (like a Select query) or request that the database perform an action (like a Delete query).
Record	All of the information about a person, place, event, or some other thing. A record is represented as a row in a table.
Relational DBMS	A database in which the user can create relationships between different sets of data called tables, allowing for more efficient and non-redundant data storage and manipulation.
Relationship	The way in which fields (columns) in two or more different tables are related. There are three fundamental relationships: one-to-one, one-to many, and many-to-many.
Report	A printed or on-screen display of specific information from a database, usually with some summarizing characteristics (like subtotals or averages).
Rollback	A statement issued by a user in a multi-user environment that indicates that the user is finished editing certain records but that the changes should be ignored.

SQL	Structured Query Language, a standard data manipulation language. SQL is an efficient language that many DBMS know how to speak, but is clumsy as an end-user tool. Thus many DBMSs have custom user interfaces that hide the complexities of SQL from users, but transmit SQL code to servers for processing.
SQL Server	A product developed by Sybase® and co-marketed by Ashton-Tate and Microsoft that acts as a server in client/server architectures. Many “front-end” clients can talk to SQL server with SQL.
Table	A term that refers to a set of data, usually about a particular category of things like employees or invoices. Consists of fields (columns) and records (rows).
View	A “virtual” representation of the information in a database, used for convenience in performing queries or analysis. For example, a view might involve a temporary table that is actually a join of two existing data tables.