**CS60 Chapter 02. File Systems and Databases**[[1]](#footnote-2)

**Contents**

[Introduction 31](#_Toc269218264)

[File Systems 32](#_Toc269218265)

[Database Systems 36](#_Toc269218266)

[Database Models 39](#_Toc269218267)

[Hierarchical Database Model 40](#_Toc269218268)

[Network Database Model 44](#_Toc269218269)

[Relational Database Model 46](#_Toc269218270)

[The Entity Relationship Diagram (ERD) 49](#_Toc269218271)

[Relational Schema 50](#_Toc269218272)

[Object Oriented Database Model (OODM) 52](#_Toc269218273)

[Summary 54](#_Toc269218274)

# Introduction

A **database** is a collection of **raw data** (e.g., names of people, addresses, graphics, videos, audio files) and **metadata** (data about data, such astable and column names, data types, constraints such as primary and foreign keys, and other elements of the framework used to store the raw data).

A **database management system** (DBMS) is the collection of utilities which allow tables and other objects to be created, altered, and dropped, and raw data to be inserted, deleted, updated, and retrieved. The DBMS is the interface between users and the raw data. Commercial DBMSs include graphical interfaces to help design the structure (the framework) that stores the data and also help to retrieve and change the raw data.

The raw data and information derived from raw data is vital to the success of all organizations. Raw data and information are pervasive in all walks of life. Their reliability depends primarily on the design of the database. A well-designed database will increase input efficiency by eliminating or reducing redundancy. **Redundancy occurs when (1) the same information is stored more than once in the system, and (2) eliminating the repeated data causes no loss of information.** Sometimes data is stored repeatedly, but the repetition is essential. In that case, the repeated data is not redundant.

**Redundancy degrades reliability of the data** because redundancy opens the door to inconsistent data or data that is wrong or obsolete. If the address of a person is stored multiple times, especially by different people at different times based on different sources, the redundant entries are more likely to be inconsistent. Redundancy requires additional labor to input the data. If the address changes, then additional labor is necessary to update the redundant entries and a greater likelihood exists that not all entries will get updated. Thus some data would be obsolete. The duplicate data requires additional storage on disk and additional time to transfer across a network.

# File Systems

The need to improve upon manual data handling on paper gave rise to **computerized data processing**, which was an electronic version of the manual system. The improvement came from data manipulation and report generation. Instead of handwritten spreadsheets that took days or weeks to generate, the computer-generated reports took days or weeks to program initially, but could be generated in minutes.

Data processing consisted of **flat files** whose structure was dictated by hardware and third generational language (3GL) access routines. The file was a stream of alphanumeric characters[[2]](#footnote-3) that used some method to separate the fields and records. Each field was either fixed length or variable length.

Figure 1 is a simple flat file of ASCII characters with **fixed-length records** using blanks to fill out the fields. It uses a <carriage return> or <carriage return><line feed> to separate the lines. This is a **text file**. The designer had to estimate what the greatest length of any field could be at the time of design. Otherwise, the program(s) to read, write, display, and print data would have to be rewritten in order to accommodate a field that exceeded his/her estimate. A change in the field width would require a special program to read the file in its old format and write it to the disk or tape in the new format.

Alfred A. Ramas 615-844-2573 218 Fork Rd., Babs, TN 36123

Leona K. Dunne 713-894-1238 Box 12A, Fox, KY 25246

Kathy W. Smith 615-894-2285 125 Oak Ln, Babs, TN 36123

Paul F. Olowski 615-894-2180 217 Lee Ln, Babs, TN 36123

Myron Orlando 615-222-1672 Box 111, New, TN 36155

Amy B. O'Brian 713-442-3381 387 Troll Dr., Fox, KY 25246

James G. Brown 615-297-1228 21 Tye Rd., Nash, TN 37118

George Williams 615-290-2556 155 Maple, Nash, TN 37119

Anne G. Farriss 713-382-7185 2119 Elm, Crew, KY 25432

Olette K. Smith 615-297-3809 2782 Main, Nash, TN 37118

**Figure 1**. The file system by Rob & Coronel in a table on page 9 simplified to reflect

a more accurate picture of a **flat file** with fixed-length fields and records of text. For most purposes, the first and third columns above are poorly designed because each cell stores more than one value (e.g., first name, middle initial, and last name are mixed together, or the street number and street name and parts of the address are stored together).

If the database has **variable-length records**, the fields were separated by a <tab>, comma, semi-colon, quotation marks, or some other unique character(s) used as a delimiter to distinguish between fields or between records. Figure 2 shows how a <tab> can separate fields.

**field delimiter** to separate fields here is a **<tab>**, but could have been

a colon (:), semi-colon (;), pound sign (#) or another unique character. Here, it could not be a comma because commas appear within the addresses.

**record delimiter** to separate records could be a <carriage return> or <carriage return><line feed> or some other unique characters. Although I show in this figure each new line being a new record, in the computer memory or as stored in secondary storage (disk), the data is one long string of data. The end of the 1st line and the beginning of the 2nd line would be stored as 36123<CR> Leona K. Dunne

if a <carriage return> character separates the records or

36123<CR><LF> Leona K. Dunne

if <carriage return><line feed> characters separate the records.

Alfred A. Ramas**<tab>**615-844-2573**<tab>**218 Fork Rd., Babs, TN**<tab>**36123

Leona K. Dunne**<tab>**713-894-1238**<tab>**Box 12A, Fox, KY**<tab>**25246

Kathy W. Smith**<tab>**615-894-2285**<tab>**125 Oak Ln, Babs, TN**<tab>**36123

Paul F. Olowski**<tab>**615-894-2180**<tab>**217 Lee Ln, Babs, TN**<tab>**36123

Myron Orlando**<tab>**615-222-1672**<tab>**Box 111, New, TN**<tab>**36155

Amy B. O'Brian**<tab>**713-442-3381**<tab>**387 Troll Dr., Fox, KY**<tab>**25246

James G. Brown**<tab>**615-297-1228**<tab>**21 Tye Rd., Nash, TN**<tab>**37118

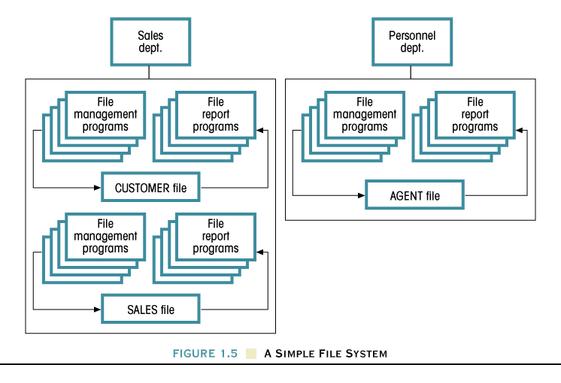
George Williams**<tab>**615-290-2556**<tab>**155 Maple, Nash, TN**<tab>**37119

Anne G. Farriss**<tab>**713-382-7185**<tab>**2119 Elm, Crew, KY**<tab>**25432

Olette K. Smith**<tab>**615-297-3809**<tab>**2782 Main, Nash, TN**<tab>**37118

**Figure 2.** A text file with variable-length fields with fields separated by a field delimiter and records separated by a record delimiter. Many of these fields (and also those in Figure 1) are not atomic; they are composite. For many purposes, for example, names such as Alfred A. Ramas should be separated into FirstName, MiddleInitial, and LastName. The address should be broken into the Street\_Address, City, State, Zip.

Third Generational languages (3GLs) such as COBOL simplified the writing of programs that accessed, manipulated the data, stored the text files, and generated reports. 3GLs eliminated having to code the programs in machine language or assembly language, both which require the programmer to know the details of the CPU's instruction set. 3GLs and 4GLs do not require that the programmer know the details of the CPU's instruction set. These details are handled by the compiler or interpreter that translates the source code (say in COBOL) into machine language (the pattern of bits in exactly the form that the CPU can "understand" without further translation).



**Rob Figure 1.5** Files and the many computer programs necessary to read,

write, manipulate, and print the data as reports. The file management programs would accept new data, delete data, modify data, and store the data. Special programs would change the format of how data is stored (e.g., change datatypes to allow longer names or add columns). File report programs would read the data and print reports.

The file system was programming intensive. Each program had to identify the **file structure:**

● whether fixed-length or variable-length fields (and its delimiters) are used to separate fields and separate records,

● how a number was stored (as separate digits using the ASCII character set or as a number in the binary number system or a number in floating point form) and how other data such as dates are stored,

● the order or arrangement of the fields,

● location of individual records,

and so on.

The coding of the file system was so rigid that any change to the file structure triggered an avalanche of programming changes to convert to the new system and thereafter use the new format. For example, if the width of the field for a name were changed, then a special program would read the data in its old format and write the data with its new format, and all other programs that read the data in its new format would also have to be changed.

As the newly discovered benefits of the computerized system proliferated, **third-generational languages (3GL) such as COBOL were developed** to simplify the programming necessary to access and manage the data. The disadvantages of the file system magnified:

* **Redundancy propagated**. As each department of an organization computerized its record-keeping, duplication of the same data multiplied.
* **Planning ahead became extremely important** because a change in file structure (e.g., number of columns and their arrangement) or a change in data type (e.g., is the last name stored as 20 or 25 characters?) would have a domino effect on all programs accessing the file.
* **Structural dependence**: the filenames, number of columns, and the order of the columns (e.g., name, address, phone, …) and **data dependence** (the datatypes and lengths of each field). Even using 3rd Generational computer languages such as COBOL, the programmer had to know the file structure and data structure in detail to access the data.
* Hardware and software costs escalated. The demand for timely information throughout the organization required larger storage capacity and more computer programs adapted to each department’s need. In addition to **data redundancy**, the file system lead to **application redundancy** (redundant programs to read, write, manipulate, display, and print data) since each department created programs to meet its specific demands.

The individualized needs of different departments spawned unique files with redundant data. When departments began to computerize their data, new file systems were easier to create than to adapt the old files to their needs. The files were differentiated by adding new fields or new formats (e.g., different delimiters or different lengths of fields) making pooling of data difficult. Thus were born the anomalies of data redundancy:

* ***Data inconsistency***. Because it was difficult to transfer data from one department to another, different departments often collected and stored their own data. New paper forms filled out by hand or typewritten and additional staffing were required to keep the computerized system current. A flaw in the manual system lead to inconsistent data between departments. One department might access obsolete data from another department because the other department had not updated the data.
* ***Data anomalies***. The dictionary defines *anomaly* as “inconsistent, contradictory.” Thus an anomaly arises from failure to systematically maintain current the same data in all locations. For example, the purchases by one client must be the same for all departments: accounting, shipping, accounts receivable, and payroll (for commission sales). Failure to share the data by using the same files meant that all departments had to separately update the data.

To emphasize the source of the anomaly, data anomalies are sometimes divided into three types:

* ***Insertion Anomalies***. A new customer would require data to be duplicated in every file within the company that referred to the customer. Insertion anomalies would occur if some insertions were omitted, the data that was inserted was inconsistent, or the data could not be inserted.
* ***Updating anomalies***. The file system lead to the same information stored in different files in different departments (or sometimes several files in the same department). Changing a customer representative would require changing records in several files which might be supervised and updated by different people. Updates may be omitted or inconsistent.
* ***Deletion Anomalies***. The last copy of some data might be lost if other data were deleted, or maybe not all copies of redundant data are removed.

# Database Systems

The most significant advantage of the database system is the principle of a **single data repository***—*one place to store the raw data and the metadata. Current database systems centralize the metadata (which includes data structure, relationship between components, and access paths to all the components) and the raw data itself. This single reservoir of data is shared by all users.[[3]](#footnote-4)

Database systems range in complexity based on the number of users, data location, and frequency of access.

## Based on the number of users

* **Single-user database system**. Only one user may access the data at one time. It is also sometimes called a **desktop database** or a **personal database**. Microsoft Access is a popular single-user DBMS. Oracle and many other DBMSs offer both single- and multi-user versions.
* **Multi-user database system**. Many users can access the same data at the same time. A multi-user system will lock records being changed to maintain integrity. When used within a small environment, it is call a **workgroup database**. If it is used by more than 50 in an organization in many departments, it is called an **enterprise database**. Microsoft SQL Server and Oracle have multi-user versions. At Santa Monica College, the multi-user DBMS’s are client-server systems (with clients connected to the server usually by a network). But some high-end systems such as IBM DB2 still offer systems stored on one computer (a mainframe) that’s accessed by terminals (or computers running terminal emulation programs) connected serially to the mainframe. A mainframe may be a powerful computer, but it also could be a rather conventional desktop with lots of serial connections, an appropriate operating system, and an application program suited for that distributed processing.

## Based on data location

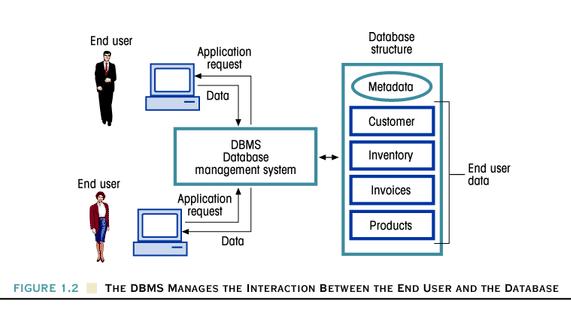
* **Centralized DBMS**. A database system operating from one location. This often is a mainframe computer that stores the database program (the Database Management System, DBMS) and the database (the reservoir of raw data, metadata, and even programs such as procedures, functions, and triggers, i.e., even source code and executable code may be stored in this reservoir).
* **Distributed DBMS**. A database system distributed across several sites. As parts of a distributed database, a table could be stored at a site in Los Angeles and another site in London.

## Based on frequency of access

* **Transactional DBMS** (or production DBMS): a database system with large number of transactions requiring constant up-to-date information and rapid access. The DBMSs for airline and hotel reservations, banking, ticket sales, and employee and student records are transactional DBMSs. These are usually multi-user systems.

A **transaction** is a series of steps, all which must be completed to complete the transaction, or all steps must be undone to undo the transaction. An example of a transaction is a transfer of money from a saving account to a checking account. The amount must be deleted from the savings and credited to the checking account to complete that transaction. The DBMS must not allow, even if the hardware fails in the middle of the transfer, that the data be permanently stored as an incomplete transaction. For example, funds were deleted from savings but not credited to the checking account or not credited in another way such as cash or a check would mean the transaction was incomplete.

* **Decision Support System (DSS)**. Typically relying on historical data, a DSS is a database used for strategic planning. Rather than hundreds or thousands of users wanting nearly instantaneous access to relatively current data (like the availability of tickets or enrollment data for this semester), the number of users of a DSS might be one person or a few people analyzing maybe terabytes of historical data from all over the world for the last 10 years. The result may be recognizing patterns useful to top management. Assembling data and recognizing patterns useful to support management decisions is why these are called Decision Support Systems. Providing nearly instantaneous access isn’t essential.



**Rob Figure 1.2** The DBMS manages the interaction between the end user and the database.

A DBMS includes the following functions, which were missing from the file system or were confronted with great effort, sometimes unsuccessfully, by the DP manager:

1. **Data dictionary (or system catalog) that stores metadata**—the metadata about every object in the database (such as a table), its name, column names, datatypes, and constraints such as primary and foreign keys and others.
2. **Data storage management**. The DBMS, by using the data dictionary, manages the physical storage task, thus relieving the DP manager of the burden of arranging the fields, field length, delimiters, etc.
3. **Data abstraction.** A term we will see with Object Oriented Databases, data abstraction relieves users from the constraints of database structure and datatype. It alleviates the problem of directly facing every last detail in every action involving the database. The DBMS generalizes the datatypes and handles the conversion to machine-recognizable format. Access to the data is not invalidated by a change in structure either.
4. **Security management**. The DBMS offers several layers of security to control who can access the raw data, who can change (insert, delete, modify) the raw data, who can change the metadata (for example, who can add a table or add or remove a column). This security is on top of the security associated with the operating system and the network security through user accounts, passwords, roles, privileges, and profiles.
5. **Multi-user access control** or **concurrency control**. The DBMS acts as a traffic cop to assure that the data is consistent throughout a user’s session, but allowing multiple people to access (read), write (add new records), and change data (update). Changes need to be coordinated.
6. **Data integrity management**. Related to record locking, the DBMS tracks the data on a transaction basis in order to manage concurrency and integrity. When more than one person is handling the same record, it recognizes whether the person is reading or writing the record, and then acts to yield consistent data.
7. **Saving the data: Backup and recovery management**. A DBMS can save data (write the data to disk), backup, and recover in different ways. A system could write each transaction immediately, thereby exposing to loss only the current transaction. Or a system could temporarily store transactions in a buffer file, and not commit or save the data until instructed to do so.

DBMSs also include **recovery utilities** (**restore utilities**) used in case of hardware failure, attacks by computer viruses, and other causes of program and data loss. Using the backups, these utilities restore the operating system, the DBMS program, and the data. Part of a Database Administrator's (DBA's) job is to set up and use a backup system, and when necessary, use the backups to restore the programs and data.

1. **Data access**. The standard for accessing a relational database is standard query language (SQL), but SQL goes much further than merely retrieving data. The series of SQL commands called **data definition language (DDL)** provides a standard language for creating and defining a database. The series of SQL commands called **data manipulation language (DML)** are used for inserting, deleting, updating, and retrieving data from the database. Graphical interfaces may also provide access to the data. Types of DBMSs other than relational also have access to the data, of course, although different from SQL.

The DBMS incorporated functions that were missing in the file system to improve the reliability and timeliness of information.

# Database Models

Data independence and fewer hardware constraints allow database designers to think more abstractly about what they are trying to represent or *model* by their systems. Variations exist in how abstract a database system can be. The *model* allows us to shift our focus from what the machine will allow us to do to what we can do with the machine. The model is defined by two characteristics: (1) the nature of each piece of datum, and (2) the relationship among the pieces.

* The **conceptual model** replicates **the logical nature** of the data. The entity relationship model (E-R) and object-oriented model are examples of conceptual models*.*
* The **implementation model** focuses on the way **the data can be represented** in the database. Examples of implementation models are hierarchical, network, relational, and hybrid relational/object-oriented database models.

Database models define the **relationship between data** in three types of **connectivities**:

***1. One-to-many relationship,*** represented by 1:M. One row of data can be associated with many other rows of data.

● A painter can paint many paintings, but one painting has only one painter.

● A father can have many children, but each child has only one biological father.

● A salesman can have many accounts, but each account has only one salesman.

***2. Many-to-many relationship***, represented by M:N. Many data can be associated with yet another multiple of data.

● A student can take several classes at once, and each class can have many students.

● In real estate, sales representatives may work as a team, and each team may have many clients. Thus, one client can have more than one representative familiar with his/her transaction.

In a relational database, tables with a M:N relationship will be converted in two 1:M relationships with an intermediate bridge table.

***3. One-to-one relationship***, represented by 1:1.

● In the United States, a husband can have only one wife and, conversely, a wife can have only one husband.

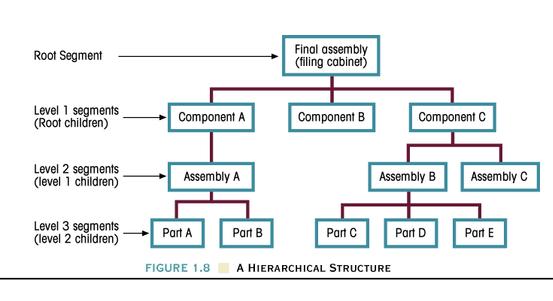
● A store can have only one manager, and a manager can manage only one store.

● A school can have only one principal, and a principal can head only one school.

# Hierarchical Database Model

The **hierarchical model**, born from the manufacturing-assembly environment evolving from generalized update access method (GUAM) and joint development between IBM and North American Rockwell, was marketed by IBM as an information management system (IMS).

The hierarchical model is based on the premise that a complete thing (such as an airplane) is made up of numerous subassemblies and each subassembly is made up of others until finally they can be traced down to individual parts. The hierarchical model has a tree-like structure made up of nodes and branches. A **node** is a collection of data attributes (characteristics) describing the entity at that point. The highest node of the hierarchical tree structure is called a **root**. The nodes at succeeding lower levels are called **children**.



**Rob Figure 1.8** Example of a hierarchical database structure.

The hierarchical path to Part D begins at the top-left and proceeds as

Final assembly → Component A → Assembly A → Part A → Part B →

Component B → Component C → Assembly B → Part C → Part D

The hierarchical model breaks down each segment into children segments, starting at the top with a parent and progressing through children segments. The hierarchical model is based on the 1:M relationship. It made heavy use of **pointers**, a referencing scheme that identifies the location of the data in the system.

Another example of the hierarchical model:

**Parent segment** stores a description of a customer, with customer name, address, phone numbers.This avoids redundancy by listing the customer description only once in the parent segment.

**Child segments** store what the customer has ordered in several invoices



Another example of a hierarchical representation is Rob Figure P1.15, page 52.

This has two segments, PAINTER and PAINTING.

The PAINTER segment has the attributes PT\_NUMBER, PT\_NAME, PT\_PHONE.

The PAINTING segment has the attributes PTG\_NUMBER and PTG\_TITLE

**Segment Attribute Values of**

**name name the attributes**



The hierarchical path to reach the data in the third PAINTING segment is

(1) The DBMS must access the PAINTER segment

10014, Josephine G. Artiste, 615‑999‑8963;

(2, 3) the two PAINTING segments are accessed:

21003, Database Sunshine,

11987, Hierarchical Paths;

(4) the third PAINTING segment is accessed:

25108, File Systems Folly

The hierarchical tree structure must satisfy the following conditions:

* It always starts with a root segment.
* Every segment consists of one or more attributes describing the entity at that point.
* Dependent segments can follow the succeeding levels. The segment in the preceding level becomes the parent segment of the new dependent segments.
* Every segment occurring at level 2 has to be connected with one and only one segment occurring at level 1.
* A parent segment can have one or several children segments.
* Every segment except the root has to be accessed through its parent segment.
* There can be a number of occurrences of each segment at each level.

As it evolved and was adapted to other industries, it developed to 1:1 relationships. It really never could handle M:N relationships. It was dominated by the *hierarchic sequence* of top-down traversal because the lower segments were identified by the parent segment and, ultimately the topmost segment. Theoretically no segments were accessible except through the parent. Some hierarchical applications developed a bottom-to-top view of the data.

Non-manufacturing applications included banks and others that made heavy use of 1:M relationship.

## Advantages of hierarchical DBMS

* **Conceptual simplicity.** The fixed 1:M relationship is easy to understand and, therefore, easy to design.
* **Database security.** The one-entry configuration made security straightforward. You must have access to the root. By design, the hierarchical system is centralized and cannot be otherwise.
* **Data independence.** The data types were fixed by the system; therefore, they did not have to be addressed by the user interface.
* **Database integrity**. The centralized system eliminated data redundancy.
* **Efficiency.** The hierarchical model was made efficient by the segment configuration and the use of pointers.

## Disadvantages of hierarchical DBMS:

* **Complex implementation.** While the database could be remodeled, **pointers and hardware constraints were not totally eliminated** and had to be handled by the programmer.
* **Difficult to manage.** Rearrangement of the database structure required change in all applications. A designer had to model the environment with some forethought to potential changes. Otherwise, the system would have to be re-created from the ground up. Efficiency was affected inversely by the depth of the tree.
* **Lacks structural independence.**  The hierarchical model mocked-up the assembly of a product that was expected to persist for a long period of time. It had a built-in obsolescence. For each new major product, a new file would have to be created. The programmer had to know the structure and take it into account in the programs written to access the data. The fixed structure of the model had to be incorporated in the user interface. Any change in structure required a change in programming.
* **Application programming and use complexity.**  Familiarity with the pointer system required in-depth knowledge of its characteristics.
* **Implementation limitations.** The 1:M relationship was a built-in limitation. Although 1:1 could be handled easily, M:N or M:1 could not be handled well by the hierarchical model. The hierarchical model was designed without regard to the universe of the enterprise. Cross referencing inventory, for instance, was an afterthought. The system could not fathom the “multiple parent” mock-up[[4]](#footnote-5).
* **Lack of standards.** Because the hierarchical model was developed for mainframe application, there was never a need to establish standards. There were very few companies marketing mainframe computers. When competitive companies marketed their database products, there was no portability. As a result, a programmer familiar with one system would have to become familiar with whatever the client was using, or the client would have to change systems.

**Optional Note:** Coronel illustrates the hierarchical model for a purchase order tracking and related inventory reference system. Figure 1.9 on Coronel page 27 could be redesigned as follows:

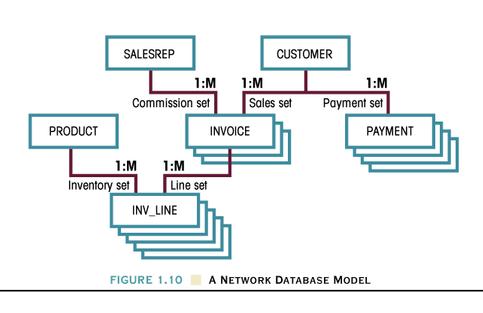


Coronel Figure 1.9 redesigned to model the relationship between inventory and order\_line.

Several of the commercially available hierarchical database management systems:

|  |  |
| --- | --- |
| **VENDOR** | **PRODUCT** |
| IBM | IMS |
| SAS | SYSTEM 2000 |
| INFORMATION BUILDERS | FOCUS |

# Network Database Model



**Rob Figure 1.10** A network database model with a collection of records in 1:M relationships which allows a record to have more than one parent. In this diagram, INVOICE has two parents (SALESREP and CUSTOMER), and INV\_LINE (one line of an invoice) has two parents (PRODUCT and INVOICE). But like a hierarchical database, a record in a network database also might have only one parent. For example, payments are only made by customers, although one customer may make many payments.

The network database model evolved from the hierarchical model. As such it attempted to eliminate the deficiencies of the hierarchical model: poor performance, lack of portability, and poor representation of real world models. The network model resulted from the efforts of the Conference on Data Systems Languages (CODASYL) group which established the DataBase Task Group (DBTG) to develop standard specifications for database applications. The DBTG produced **standards** as follows:

1. A **data definition language** (**DDL**) which outlined for the database administrator the components and structure of the database. This is the network schema (the blueprint for the database).
2. A subschema DDL which outlined the components and structure of the database for the user interface that accessed the database.
3. A **data manipulation language** (**DML**) which defines the methods by which the data could be inserted, deleted, updated.

The network model is an *implementation model* viewing the real world from a how-to approach. As such it focuses more on the relationship, called a set, between records. The model interconnects different areas of an enterprise via a network. Each area contains records of two types:

**● owner** (in hierarchical, parent) and

**● member** (in hierarchical, child) records.

The **set** describes the type of relationship between owner and member, which may be 1:M or M:N.

In Problem 19, page 53, the PYRAID company wants to track each PART used in one piece of EQUIPMENT. Each PART is bought from one SUPPLIER. A piece of equipment is composed of many parts but each part is used in only one specific piece of equipment. A supplier can supply many parts, but each part has been supplied by only one supplier.

The network structure and the sets for this database is shown below:



**Owner**

**(parent)**

# 

**Member**

**(clild)**

## Advantages of the network database model:

In the evolution of DBMSs, the network database model inherits many of the hierarchical attributes, but it refines the weaker ones.

1. **Conceptual simplicity.** The database is viewed from one viewpoint, thereby, theoretically, making it easier to design.
2. **Handles more relationship types.** It handles the M:N more effectively than the hierarchical database model without the redundancy.
3. **Data access flexibility.** Navigation is handled by way of the set, giving access to either owner or member records.
4. **Promotes database integrity.** In order for a set to be defined, there must be an associated member for each owner.
5. **Data independence.** Access to data is not compromised by changes in data type.
6. **Conformance to standards.** The network database model is fashioned after the DDL and DML standards, which permit uniform development of applications.

## Disadvantages of the network database model:

1. **System Complexity**. Although the schema (the plan or blueprint) sounds simple, in practice it is complex. Administrators and programmers must be familiar with the logical structure in order to access it, and the programmer must know the position of the set occurrences when moving through the database.
2. **Lack of structural independence.** The location of owners and members must be fixed in order to define a set. Because the database is defined by the set, any change in owner/member causes a new set definition, which requires rewriting programs.

Commercial network databases:

|  |  |
| --- | --- |
| **VENDOR** | **PRODUCT** |
| CULLINET | IDMS |
| HONEYWELL | IDS |
| UNIVAC | DMS 1100 |

# Relational Database Model

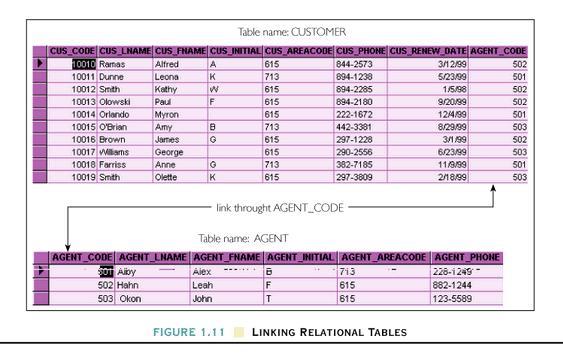
Dr. E. F. Codd of the IBM San Jose Research Laboratory established the foundation for the relational database model based on the mathematical principles of relational algebra.

The relational database model’s unit is a table (or more precisely, a table with special properties called a **relation)**. An entity, such as a customer, invoice, or part, is represented by a table with rows and columns. Each row represents one customer (record). Each column contains an attribute of the entity. The intersection of row and column represents a field.

Tables are linked by relationships with other tables. These relationships may be 1:1, 1:M, or M:N (implemented with the help of a bridge table). The relationship is declared by way of duplicating a unique identifying field from the 1 side of the relationship to a multiple occurrence of the same field on the many side of the referenced table.

The relational database model, in a sense, resembles the flat file in that the information is arranged by rows with the information for each entity contained on one line. Unlike a flat file, however, the data stored in a relation can more than merely text. The database administrator, programmer, and user don’t have to concern themselves with the actual coding for storing the information; it is built into the relational database model.

As shown in Coronel’s figure 1.11 below, a link exists between two tables: the AGENT\_CODE links the two tables. Each Customer has one agent (one salesperson to take orders and help the customer). Yet, each AGENT has several customers. In the Customer table, the AGENT\_CODE is repeated several times to indicate the agent’s customers. While an agent’s code can be repeated, it’s not redundant. This possible repetition is vital to the function of the relational database model, for it serves to identify the relationship between the two tables.



The two tables link through the AGENT\_CODE

## Advantages of a relational database:

1. **Structural independence.** Deleting any column except the identifying column (the primary key) to the table does not affect the ability to access the remaining data in a table. Neither does adding columns or rearranging columns affect the ability to access the data. As long as the foreign key and the referenced primary key remain intact, deleting other columns or adding columns don't affect the ability to access data from both tables simultaneously (in one query). Adding or deleting columns or changing column names could still require some programming changes, but the change would not require an entire recoding of all programs. In a relational database, the programming does not have to identify or address the database structure in order to access the data.
2. **Improved conceptual simplicity.** From a logical viewpoint, the relational database model is simple, as is the flat file system. The relational database model excels at implementing the logical model because implementation details are delegated to the DBMS system itself.
3. **Easier database design, implementation, management, and use**. By removing the burden of addressing constraints (using pointers) and thereby approaching structural and data independence, the relational database model is simpler than its predecessors.
4. **Ad hoc query capability**. By incorporating an intermediary compiler, the relational database model relieves the programmer from dealing with machine data formats and structure. This task is accomplished through the standard relational database language SQL.
5. **A powerful database management system.** In order to simplify the user’s end of the system, much is built into it to function as an interface with the operating system. This feature allows the users to extend the functionality and information relevance.

## Disadvantages of a relational database:

The power of the relational database model comes at a cost.

1. **Substantial hardware and system software overhead.** The power of the system could only exploited with an increased number of software modules and powerful computer hardware: increased processing speed, increased volatile memory (RAM), increased secondary (disk) storage. This overhead can be a problem today with large databases.
2. **Poor design and implementation is made easy.** A little bit of knowledge is dangerous. Simplifying the system can be a detriment. Relatively untrained people can construct a poorly designed database. Commercially available packages make it easy to access and manipulate data from tables without proper design. The risk lies in attempting to apply simplistic structure to more advanced use.
3. **May promote *islands of information* problems.** SQL is a universal language that can be implemented with a good user interface. Computer Aided System Engineering tools (CASE tools) also promotes access to information and adapting it to individualized use. Each individual may generate redundant data or isolated data that others might not have access to. This violates the principle of having a single repository for the data.

The costs of the relational database model are offset by decreasing hardware costs, the emphasis on a single repository of data, centralized software management, and the Internet.

# The Entity Relationship Diagram (ERD)

**An entity is a person, place, or thing about which data is to be collected and stored. An Entity Relationship Diagram is a graphical model of a relational database.** It models the entities and their relationships.

SKILL\_NAME

M

1

SKILL

EMPLOYEE

learns

SKILL\_

DESCRIPTION

EMPLOYEE\_

FIRSTNAME

EMPLOYEE\_

LASTNAME

SKILL\_NAME

EMPLOYEE\_ID

EMPLOYEE\_

CITY

EMPLOYEE\_

STREET

EMPLOYEE\_

ZIP

EMPLOYEE\_

STATE

An Entity Relationship Diagram (ERD) with two entities (EMPLOYEE and SKILL), a relationship between the entities (learns), and the connectivity (1:M). Attributes of an entity are attached ovals, which should include more attributes than shown here (but quickly an ERD becomes cluttered).

Some RDBMs will convert the graphical ERDs or relational schemas into code that creates the tables and relationships (if you include column names, data types, and constraints). Or the ERD or relational schema (described on the next page) can serve as the plan for the relational database, which is implemented using SQL. The database components are represented by

**● entity instances** or **entity occurrences,** represented by rectangles with the name of the entity (a singular noun) in capital letters and centered in the rectangle**,**

**● attributes** (the characteristics of an entity) are attached to the entity in ovals (although often omitted to reduce the clutter), and

**●** the **relationship** between entities are shown in diamonds connected to the entities with lines. In the diamond is a verb that expresses how the entities relate. Sometimes to reduce clutter, the diamond is omitted and the relationship is written above the line (or omitted).

The classification (e.g., 1:M, 1:1, M:N), called the **connectivity**, sometimes is shown as those characters (1, M, N) beside the entities.

TABLEB

TABLEA

M

1

What does this 1 and M mean? The *M*  means that one row in TABLEA has *Many* rows in TABLEB (and you place the M for *Many* next to TABLEB), and the “1” means that one row in TABLEB has one row in TABLEA (and you place the 1 next to TABLEA).

Another style of an ERD shows a 1:M connectivity as a crow's foot on the *many* side.

TABLEB

TABLEA

# Relational Schema

A relational schema is another way to graphically design or document databases.

1

SKILL

EMPLOYEE

|  |
| --- |
| **SKILL\_NAME** |
| SKILL\_DESCRIPTION |

|  |
| --- |
| **EMPLOYEE\_ID** |
| EMPLOYEE\_LASTNAME |
| EMPLOYEE\_FIRSTNAME |
| EMPLOYEE\_STREET |
| EMPLOYEE\_CITY  ∞ |
| EMPLOYEE\_ZIP |
| SKILL\_NAME |

A **relational schema** lists the column names beneath the table name. Primary keys are bolded or underlined, and lines connect the foreign keys to primary keys. The connectivity for *Many* uses an infinity symbol (∞) rather than M.

## Advantages of entity relationship diagrams and relational schemas

1. **Exceptional conceptual simplicity**. The visual aid of the ERD improves the database design by simplifying the model. It focuses on entities, the relationships between entities, and attributes (and even attributes are sometimes omitted).
2. **Visual representation**. A picture is worth a thousand words. The visual representation of the model helps you understand the design, or at least the most important issues (what are the tables and how are they connected?).
3. **Effective communication tool.** Several or many people may be involved in the design of a database. Like structure diagrams used in top-down design of a computer program and like flowcharts used to show structures in programming, an ERD helps communication while evaluating and developing the design.
4. **Integrated with the relational database model**. Some database programs can generate programming code to create tables once the design is complete, making the design process less error prone and more efficient.

## Disadvantages of entity relationship diagrams

1. **Limited constraint representation**. Due to the inability to model non-connectivity constraints[[5]](#footnote-6), roll-up-the-sleeves programming is still required to implement the database.
2. **Limited relationship representation**. The extreme simplification of the model and the interest to maintain uncluttered models limits the ability to represent complex relationships between entities.
3. **No data manipulation language** (no DML). The ERM is limited to design, not to application programming and data manipulation. Therefore, many consider it incomplete.
4. **Loss of information content**. Because the tendency is to maintain simple models, usually the attributes are omitted from the model. This loss of information is a drawback.

# Object Oriented Database Model (OODM)

Our world is composed of objects with attributes, and the objects can act or behave in a certain manner.

The **attributes** are properties or characteristics of the object

The **methods** (behaviors, actions, or interactions) of the object with itself and other objects. Sometimes these are called the messages sent to the object. These programs are an intimate part of an object.

Some Object Oriented languages refer to **functions** as programs that can return or give back values of properties. Like methods, functions are also intimately bound to objects.

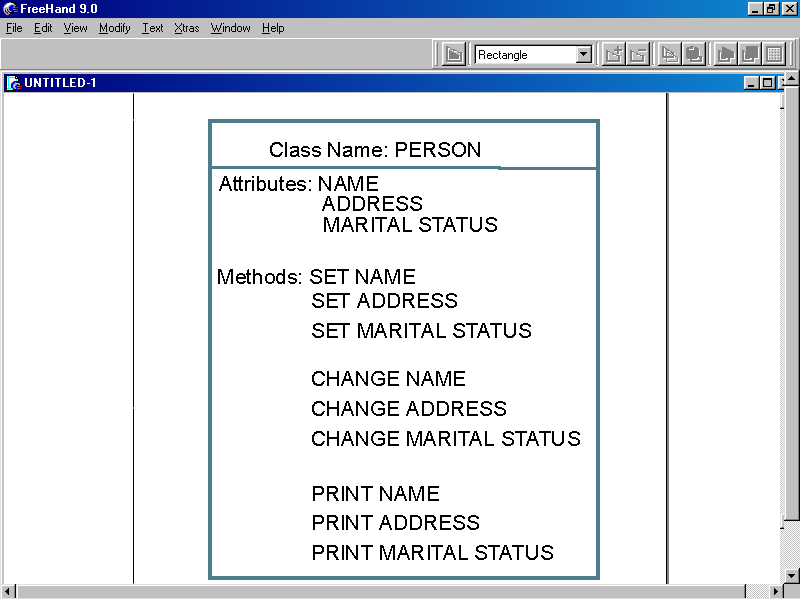


Figure . Object Oriented data model

In the **object-oriented data model**, objects are created to form a system. The objects can later be used to represent multiple real life facts that are not necessarily exactly the same. They can be adapted or extended to include new features or different features. Objects in OODM have the capacity to assume behaviors (called methods) to change or update its own characteristics and relate to other objects.

Desirable features of an object-oriented DBMS:

1. Their greatest strength comes from **reusability**, which allows developers to build bigger and more powerful applications in less time.
2. OODBMS are generally **more portable** across different platforms (different CPUs and other hardware and different operating systems). Therefore, code does not have be rewritten to execute on different platforms. However, some layer of software must handle the differences between platforms.
3. Increasing computing power allows more of the semantics of the business environment to be reflected by the system.
4. The full capability of OODBMS has not yet been identified as it has with other database systems. Therefore, as more is demanded from a system, object-oriented features become more desirable.

## Basic Structure of Object Oriented Databases

* An object can represent any real-world entity or event. An object has attributes or properties such as a PERSON’S name and address. Objects include programs called methods and functions.
* A **class** is the basic building block, a blueprint or recipe for creating objects..
* **Classes can be extended to fit individual situations.** For example, a general airplane class might be extended to include the special properties and methods for planes that take off vertically or amphibious planes that take off from either land or water. This connotes a class hierarchy, but it is not anything similar to the hierarchy or its limitations that we have seen in hierarchical database models.
* When a class is extended, it **inherits attributes** from the class that it descended. The newly defined class does not have to define all attributes or methods from its ancestor class; only the new features need be defined. All the other attributes and methods are acquired automatically.
* Classes are also boosted by **interfaces** to acquire more abilities or to act on other objects within the system.

## Advantages of Object Oriented Databases

1. **Adds semantic content.** The data model can be defined accord to the real-world situation that it is used in. For example: The airplane class can be extended to fly backwards, much the same as words add on new meanings based on usage and regional customs.
2. **Database integrity**. Class definition and interfaces assure database integrity, because once the class has been specified, each application of it (each instance or object) must conform to that structure.
3. **Both structural and data independence.** There is no constraint to changing the structure or the data type. The user interfaces and programming application are really only concerned with the object attributes and methods.
4. **Portability.**Object oriented applications are platform independent; therefore, they can be used on a variety of systems.
5. **Extensible.** OODBMS can capitalize on the fact that as it evolves, classes are being developed that can be reused. As new classes are developed, data access and complex models will be easier to configure.
6. **Leading companies recognition.** Object-oriented applications are being incorporated into the leading RDBMS, which legitimizes it as desirable system. Their assimilation will help it mature and expand its use.

## Disadvantages of Object Oriented Databases

1. **Lack of standards.** Lack of standards confines users to applications and somehow limits options, even though a system may be able to be used with different operating systems, unlike the early database management systems.
2. **Complex navigational data access.**Along with the flexibility comes complexity. It is almost a necessary evil. The flexibility requires a lot more understanding and more application work. This may be offset by its extensibility.
3. **Steep learning curve.** The transition from procedural programming to *object-oriented think* is a hurdle for programmers. *Object think* requires new ways of viewing the world. As new classes are being defined and developed, it gives the database administrator and programmers more to work with and more to think about.
4. **High system overhead slows transactions.** Implementing object-oriented systems requires powerful computers, particularly with graphical user interfaces or graphical object manipulation. Faster computers will help overcome this disadvantage.

# Summary

Computer programs have come a long way and they have a long way to go yet. The true revolution in programming and personal computer ownership began with graphical user interfaces.

With each advance in software and hardware, we find new ways of doing things. The file system gave way to the clumsy hierarchical system. But along the way **some of the burden of dealing directly with the hardware was relieved**. The new tools of the network database increased productivity and expanded the thinking to find more innovative ways to model the environment we are dealing with. Slowly, data and structural independence has been largely achieved, and the DDL and DML languages for the network model evolved into the DDL and DML statements of SQL that we use today with relational databases.

The relational model was a big breakthrough that is still evolving. It meant structural and data independence, even though we can still feel limitations from its application. The ERM graphical case tools to develop relational database design opened the door to Object-Oriented applications. It represents the next step in the evolution process. The inability of the ERM to advance to full system application meant the opportunity for object-oriented modeling.

Through all the evolution and the application of each system, one thing has stood out above all the other: That there cannot be enough emphasis on design. **The system can be only as good as the design. Great programming cannot make up for bad design.** Modeling the real world is really a function of design. Application and development is simply implementing the model we have to work with. A system will survive through generations to come if it is able to maintain pace with the semantics and demands of real-world computer modeling.

1. Drafted by Edward Estrada as part of CS88A, Independent Studies [↑](#footnote-ref-2)
2. Files also can mix datatypes. For example, a file could mix text, integers that store whole numbers according to the binary positional number system, dates, and others. Such a file cannot be opened by a text editor such as Notepad, but high level languages could write and read such files. [↑](#footnote-ref-3)
3. This idea also applies to one user storing one reservoir. This would not be the case when a person stores the data on his/her **office computer** and stores a separate set of data on a **home computer**, and actively changes both. [↑](#footnote-ref-4)
4. What Coronel illustrates as a multiple parent relationship is in actuality a M:N relationship. In the customer invoice model, there is one customer who persists, never changes. In the inventory-order\_line model, the inventory part is not one specific part but rather a reference to a type. When the part is included in the invoice and shipped to the invoice, it ceases to exist in the universe of the entity. From the part viewpoint, it really is a one-to-one relationship. The point Coronel is making is that the hierarchical model does not handle this kind of reference well. In order to track the volume of parts type sold, redundancy would have be built into the model. [↑](#footnote-ref-5)
5. Examples of non-connectivity constraints include that the value of a column lie in a certain range or be one of a set of values or whether the user must enter data in a column. [↑](#footnote-ref-6)