



1954

R 93

DIRECTORATE OF
DISTANCE EDUCATION

**M
A
T
H
E
M
A
T
I
C
S**

B.Sc., THIRD YEAR

PAPER - XII

ANCILLARY

DATA BASE MANAGEMENT SYSTEM

MADURAI KAMARAJ UNIVERSITY
PALKALAI NAGAR, MADURAI - 625 021.

Syllabus
Ancillary Paper - XII
Data Base Management System

Unit I Traditional approach to information processing - Data base approach, Data base definition, Use of database - DBMS-Data definition languages, data manipulation language,

Unit II Database models, relational, hierarchical, network database - comparison of models, database design problems of DBMS environment.

Unit III Building a database - creating, opening, database, entering data, Retrieving database records-searching the database - EXACT searching-sorting the database.

Unit IV Editing and modifying databases-creating and printing formatted reports-designing custom screen displays-managing number and dates, multiple data files.

Unit V File maintenance and performance-memory variables command file creation designing and writing programs.

Unit VI Oracle-The right tool for menu jobs-what a Relational Database, Management System Does-how Oracle organizes data-how Oracle accesses data the Oracle environment-why choose Oracle?

Unit VII Designing a database, creating tables, and adding, data relational database-understanding table components-creating a table in SQL *PLUS - creating tables from other tables- creating, synonyms-documenting table structures-Changing table structure-manipulating the data in a table-SQL Forms-The easy way to input COMMIT, ROLLBACK and AUTOCOMMIT - Using USDATW to change data. -- Querying a single table-ordering results:ORDER BY - grouping the results, accessing related data from several tables- Join-using subscribes-Connecting queries with conjunction clauses using editing commands and buffers. --- Defining and using views do - A program management applications- definition and querying one-tables-Inserting, updating, and deleting data through views dropping views.

Unit VII Operators and functions-format models. -- A New concept in application development entering Sql *forms customizing with the full-screen editor-programming with tiggers-generation a form-documenting a form.

Unit IX Types of reports Logging on to SQL *Report writer- The Main menu Moving around the menu system - help-action-entering the query- formatting your report-running a report. -- Understanding SQL *Menu -Managing Menu security - creating a menu - managing Libraries and applications - documenting your menus.

Unit X An Employee database application - Protecting provided - authenticating users-establishing user system - establishing user object - level privileges. - - Indexing - clustering.

Test Book

1. Computer Today - Suresh K. Basandra, Galgotia publishing Pvt, Ltd, New Delhi- 11002 - 1995 Chapter : 9
2. Understanding and using dbase III plus - Alan Simpson, BPB publishing, B/4, Connaught place, New Delhi - 110001.
3. Understanding ORACLE - James III plus -Rob Krumm, IEEE, PHI, Pvt, Ltd, New Delhi - 11000 - Chapter 1 - 12

Reference

1. Understanding and using dbase III plus. Krumm IEEE, PHI, Pvt. Ltd., New Delhi - 1994.
2. Principles of database management system-James Martin IEEE, PHI, Pvt, Ltd, New Delhi

Scheme Of Lessons

Data Base Management System

Pages
(1 to 17)

UNIT - 1

1. Traditional approach to informing processing.
2. Data Base approach to information processing.
3. Data Base.
4. Data Base Management Systems (DBMS)

Unit - 2

1. Data Base Models.
2. Relational Data Base.
3. Hierarchical Data Base.
4. Network Data Base.
5. Comparison of Models.
6. Data Base Design.
7. Problems in a DBMS Environment.

Unit - 3

1. Building a Data Base.
2. Navigating the menu screen.
3. Opening a Data Base.
4. Entering Data.
5. More Sample Records.
6. Retrieving Data Base Records.
7. Building search conditions.
8. Searching with locate command.
9. Exact searching.
10. Sorting the Data Base.

Unit - 4

1. Editing and modifying Data Base.
2. Creating and Printing Formated Reports.
3. Designing custom screen displays.
4. Managing Numbers and Dates.
5. Managing Multiple Data Files.

(74-154)

Unit - 5

(155-190)

1. File Maintenance and performance.
2. Understanding memory variables.
3. Creating command files with modify.

Unit - 6

(191-203)

1. Resisting Abstract Data types.
2. Security for Abstract Data types.
3. Indexing Abstract Data types.
4. How oracle works.

Unit - 7

(204-262)

1. Creating and Maintaining Tables.
2. Creating a table with the contents of another table.
3. Index - organised Tables
4. SQL * PLUS
5. Views

Unit - 8

(263-292)

1. Expressions and Operator1.
2. Triggers.

Unit - 9

(293-321)

1. Table Replication.
2. Numeric Formatting.

Unit - 10

(322-350)

1. The create user command.
2. The Grant Command.
3. Roles.
4. Indexes.

Model Question Paper
B.Sc., Degree Examination
Ancillary - Paper XII

Data Base Management System

Time : Three hours

Maximum : 100 Marks

Section - A

Answer any eight ($8 \times 5 = 40$)

1. Briefly write about four types of data base system users.
2. Write down the functions of DBA.
3. Write a short note on Query language.
4. What are the uses of Data Definition Language.
5. What do you mean by Data Model?
6. Write a short note on "Get Hold Class of statements" in hierarchical model.
7. Write a note on Modify Statement for Network Model.
8. How does the concept of an object in the object - oriented model differ from the concept of an entity in the entity - relationship model?
9. Write down the performance measures of disks.
10. Define key and primary key.
11. Write briefly about Deadlock.
12. Explain the difference between the three storage types - volatile, non-volatile and stable - in terms of cost.

Section - B

Answer any Six ($6 \times 10 = 60$)

13. What are the disadvantages of database processing.
14. Discuss about the major categories of programs involved in data base processing.
15. Write the explain with suitable example, the following relational algebra operation.
 - (i) UNION
 - (ii) JOIN
 - (iii) INTERSECTION
16. Discuss on IMS hierarchy.
17. What do you mean by Data base security write a short note on access control?
18. Discuss briefly about how to recover the database when it destroyed.
19. What is DBMS and explain the advantages of data base processing system?
20. Discuss on DBTG Network Model.
21. Discuss on Relational Calculus.
22. Discuss on (i) Concurrency Control. (ii) Lock.

Unit - 1

Introduction

Information, nodoubt, is a keyword in modern management. Anyone with a cluttered office knows having a large quantity of information on hand does not guarantee ready access to any particular piece of information perhaps the most important challenge facing information systems is to provide users with timely and versatile access to data stored in computer files. In a dynamic business environment there are many unanticipated needs for information. Often the basic underlying data to satisfy there information needs are contained in computer files but cannot be accessed and output in a suitable format on a timely basis. For effective decision - making one should have access to information whenever needed and what is more it should be up-to-date and cost effective.

Data base management systems have the potential to meet this challenge. A Data base Management system is an effective data management tool. Provides invaluable help in coping with data organisation and access problems and improves the quality of information available to the management for decision-making. In the past four decades, there has been a rapid increase in the capacity of electronic machines to store information and an equally rapid decrease in the capacity of electronic machines to store information and an equally rapid decrease in the cost of storage. Data base technology has become one of the most rapidly growing areas in computer and information science and data base systems are a major topic of current interest.

4.1 Traditional Approach to information processing

The traditional approach to information processing is file oriented prior to the advent of data base management systems (DBMS) each application maintained its own master file and generally had its own set of transaction files. **Fig. 1.1** illustrates this traditional approach. Files are custom-designed for each application and generally there is little sharing of data among the various applications programs are dependent on the files and vice versa that is when the physical format of the file is changed, the program also has to be changed. The traditional approach is file oriented because the primary purpose of many applications is to maintain on the master file the data required to produce management information. Therefore the master file is the centre piece of each application. Although the traditional, file - oriented approach to information processing its skill widely used, it does have some very important disadvantages. Among them are the following.

1. Data Redundancy

Often identical data are stored in two or more files. Notice that in fig. 1-2. each employee's social security number, name and department are stored in both the payroll and personnel files. A payroll file stores data concerning employees wages and salaries. A personnel file contains data about employees work and education histories as well as occupational skills. Many, large firms have separate payroll and personnel departments. A data base management system would allow the two department to share data. Obviously such data redundancy

increases data editing maintenance and storage costs. In addition data stored on two master files (which should in theory be identical) are often different for good reasons but such differences inevitably create confusion.

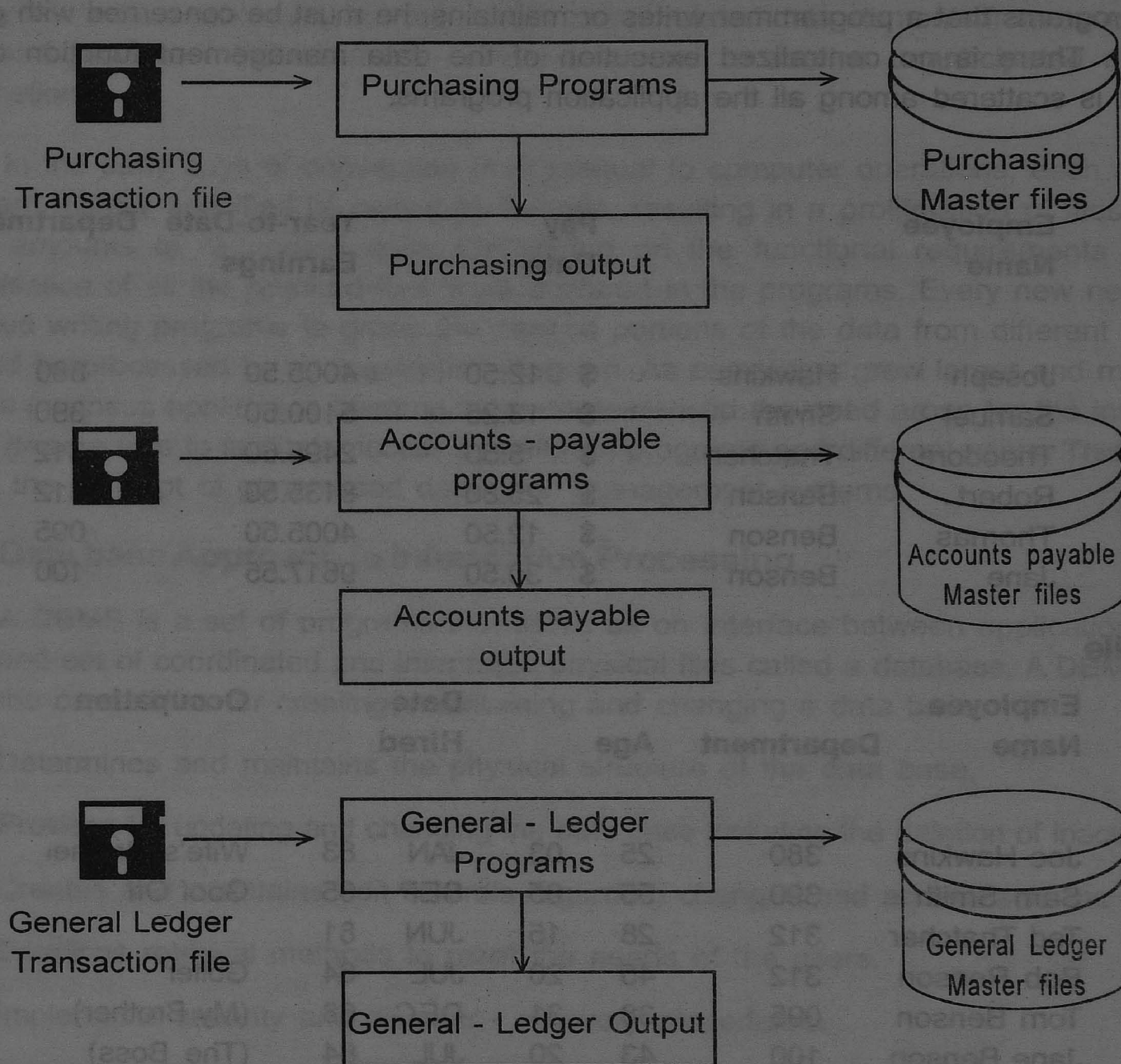


Fig. 1-1 Traditional Approach to information processing

2. Lack of Data integration

Data on different master files may be related as in the case of payroll and personnel master files. For example in **fig. 1-2**, management may want to report displaying employee's name department pay rate and occupation. However, the application approach does not have the mechanisms for associating these data in a logical way to make them useful for management's needs.

3. Program / Data Dependence

Under the traditional file oriented approach, programs are tied to master files and vice versa changes in the physical format of the master file such as the addition of data field require changes in all programs that access the master file. Consequently, for each of the application programs that a programmer writes or maintains, he must be concerned with data management. There is no centralized execution of the data management function data management is scattered among all the application programs.

Payroll file

Social Security Number	Employee Name		Pay Rate	Year-to-Date Earnings	Department
385686293	Joseph Hawkins		\$ 12.50	4005.50	380
390328453	Samuel Smith		\$ 13.25	5100.60	390
410686392	Theodore Thatcher		\$ 5.50	2495.60	312
425786495	Robert Benson		\$ 25.80	8135.50	312
510933492	Thomas Benson		\$ 12.50	4005.50	095
511945893	Jane Benson		\$ 30.50	9617.55	100

Personal File

Social Security Number	Employee Name	Department	Age	Date Hired	Occupation
385686293	Joe Hawkins	380	25 03	JAN 83	Wife's Brother
390328453	Sam Smith	390	55 05	SEP 65	Goof Off
410686392	Ted Thatcher	312	28 15	JUN 81	
425786495	Bob Renson	312	45 20	JUL 64	Golfer
510933492	Tom Benson	095	38 31	DEC 68	(My Brother)
511945893	Jane Benson	100	43 20	JUL 64	(The Boss)

Fig. 1.2 Data Redundancy and Lack of Integration among files.

A payroll file stores data concerning employees wages and salaries. A personnel file contains data about employees work and education histories as well as occupational skills. Many large firms have separate payroll and personal departments. A data-base management system would allow the two departments to share data.

Lack of Flexibility

The information - retrieval capabilities of most traditional systems are limited to predetermined requests for data. Therefore the system produces information in the form of scheduled reports and queries which it has been previously programmed to handle. If

management needs unanticipated data the information can perhaps be provided if it is in the files of the system. Extensive programming is often involved. Thus by the time of programming is completed, the information may no longer be required or useful. This problem has long plagued information systems. Management knows that a particular piece of information be produced on a one-time basis but the expense and time involved are generally prohibitive. Ideally information processing should be able to mix related data elements from several different files and produce information with a fast turnaround to service unanticipated requests for information.

In the early days of conversion from manual to computer operations, each applications program usually specified its own data formats, resulting in a proliferation of files containing large amounts of redundant data. Depending on the functional requirements the record organisation of all the required files were declared in the programs. Every new need for data required writing programs to group the desired portions of the data from different files before it could be processed by an application program. As computers grew larger and more flexible and as business application became more computerised the need arose for the integration of these diverse files to facilitate access by different programs and different users. This integration led to the concept of generalised data base management systems.

1.2 Data base Approach to Information Processing

1. A DBMS is a set of programs that serve as an interface between application programs and set of coordinated and integrated physical files called a database. A DBMS provides the capabilities for creating, maintaining and changing a data base.
2. Determines and maintains the physical structure of the data base.
3. Provides for updating and changing the data base including the deletion of inactive records.
4. Creates and maintains edit controls regarding changes and additions to the database.
5. Develops retrieval methods to meet the needs of the users.
6. Implements security and disaster - recovery procedures.
7. Maintains configuration control of the data base. Configuration control means that changes requested by one user must be approved by the other users of the database. One person cannot indiscriminately change the data base to the detriment of other users.
8. Assigns user access codes in order to prevent unauthorized use of data.

A data base administrator is extremely important, working very closely with users in order to create, maintain and safeguard the data base. In effect, DBA is the liaison between the data base and its users and therefore, must be familiar with their information requirements. The administrator must also be technically competent in the areas of DBMS and data storage and processing. Data base administration is becoming an attractive career option to individuals with programming, system and business backgrounds.

Why Data base?

An enterprise choose to store its operational data in an integrated database, because broadly a data base system provides the enterprise with centralised control of its operational data as data is one of its most valuable assets. This is in sharp contrast to the situation when each application has its own private files, quite often its own private magnetic tapes and disk packs too, so that the operational data is widely disposed and is therefore, probably difficult to control.

The foregoing implies that in an enterprise with a data base system there will be one identifiable person who has this central responsibility for the operational data. This person is the data base administrator whose job involves both a high degree of technical enterprise and the ability to understand and interpret management requirements at a senior level. It is important to realise that the position of the DBA within the enterprise is a very senior one. Some of the advantages that accrue from having centralised control of the data are discussed further.

1. Redundancy can be reduced

In non data base system each application has it us ours private files which can often lead to considerable redundancy in stored data with resultant waste in stored space. For example, a personnel application and on education records application may each own a file containing department information for employees. These two files can be integrated and the redundancy eliminated, if the DBA is aware of the data requirements for both applications, that is, if the DBA has the necessary overall control. It is not suggested that all redundancy, should necessarily be eliminated. Sometimes there are sound business or technical reasons for maintaining multiple copies of the same data. In a data base system, however, redundancy should be controlled, that is, the system should be aware of the redundancy and should assume responsibility for propagating updates.

2. Inconsistency can be avoided

This is related to the previous point. Suppose that a given fact about the real world, say, the fact that employee EMP3 works in department DEP9, is represented by two distinct entries in the data base, and that the system is not aware of this duplication in other words, the redundancy is not controlled. Then, there will be some occasions on which the two entries will not agree, that is, when one and only one has been updated. At such times the data base is said to be inconsistent. A data base that is the inconsistent state is capable of supplying incorrect or conflicting information. It is clear that if the given fact is represented by a single entry, that is, if the redundancy is removed, such an inconsistency cannot occur. Alternatively, if the redundancy is not removed but is controlled by making it known to the system, then the system could guarantee that the data base is never inconsistent as seen by the user, by ensuring that any change made to either of the two entries is automatically made to the other. This process is known as propagating updates. Here the term 'update' is used to cover all the operations of creation, deletion and modification.

3. Data can be shared

It means not only that existing applications, can share the data in the data base, but also that new applications can be developed to operate against the same stored data. In other words, the data requirements of new applications may be satisfied without having to create any new stored files.

4. Standards can be enforced

With central control of the data base, the DBA can ensure that all applicable standards are followed in the representation of the data. The applicable standards may include any or all of the following company, installation, departmental, industry, national and international standards. Standardizing stored data formats is particularly desirable as an aid to data interchange or migration between systems.

5. Security restriction can be applied

Having complete jurisdiction over the operational data, the DBA

- (a) can ensure that the only means of access to the data base is through the proper channels and hence
- (b) can define authorization checks to be carried out whenever access to sensitive data is attempted. Different checks can be established for each type of access, that is, retrieve, modify, delete, etc., to each piece of information in the database.

6. Integrity can be maintained

The problem of Integrity is the problem of ensuring that the data in the data base is accurate. In consistency between two entries representing the same 'fact' is an example of lack of Integrity which of course can occur only if redundancy exists in the stored data. Even if redundancy is eliminated, however, the data base may still contain incorrect data. For example, an employee may be shown as having worked 200 hrs in a week, or a list of employee numbers for a given department may include the number of a non existent employee. Centralised control of the data base helps in avoiding these situations, in so far as they can be avoided, by permitting the DBA to define validation procedures to be carried out whenever any update operation is attempted. Data integrity is even more important in a data base system than in a file-oriented environment, precisely because the database is shared for without appropriate validation procedures it is possible for a program with errors in it to generate bad data and thus, to 'infect' other innocent programs using the data.

7. Conflicting requirements can be balanced

Knowing the overall requirements of the enterprise, as opposed to the requirements of any individual user, the DBA can structure the data base system to provide an overall service that is 'best for the enterprise'. For example, a representation can be chosen for the data in storage that gives fast access for the most important application at the cost of poor performance in some applications.

8. Synchronization

Often many users running programs that access the data base at the same time DBMS provides protection against inconsistencies that result from two approximately simultaneous operations on a data item. For example, suppose that at about the same time, Two reservation clerks issue requests to reserve a seat on flight 757. Each request results in the execution of a program that might examine the number of seats available (say one seat is left), subtracts one and stores the resulting number of seats in the data base. If the DBMS does not sequence these two transaction (the two invocations of the reservation programme) properly, two passengers might wind up sitting on the same seat.

9. Data Independence

Data independence may be most easily understood by first considering its opposite. Most present day applications are data dependent. This means that the way in which the data is organised in secondary storage and the way in which it is accessed are both dictated by the requirements of the application and moreover, that knowledge of the data organisation and accessed technique is built into the application logic.

For example, it may be decided (for performance reasons) that a particular file is to be stored in indexed sequential form. The application, then must know that the index exists and must know the file sequence (as defined by the index) and the internal structure of the application will be built around this knowledge. In addition, the precise form of the various accessing and exception - checking procedures within the application will depend very heavily on details of the interface presented by the indexed sequential software.

Such an application is called data dependent because it is impossible to change the storage structure (how the data is physically recorded) or access strategy (how it is accessed) without affecting the application, probably drastically. For example, it would not be possible to replace the indexed sequential file above by a hash addressed File without making major modifications to the application. The portions of the application requiring alteration in such a case are the portions that communicate with the file handling software and the difficulties, involved are quite irrelevant to the problem the application was written to solve. They are difficulties introduced by the structure of the file handling s/w interface. (s/w = software)

In a database system, however, it would be extremely undesirable to allow applications to be data dependent, for at least the following two reasons.

1. Different applications will need different views of the same data. For example suppose that before the enterprise introduces its integrated data base, these are two applications A and B, each owning a file containing the field Logical Versus Physical Views of Data Shortage.

With most traditional data storage techniques, the programmer needs to be aware of the physical layout of data records on storage devices and thus, needs to understand the technical characteristics of many kinds of hardware. The problem tests even more complex in a multi-user environment where one programmer may have to use data files designed by

another programmer. Often a lot of time is wasted just trying to figure out what a particular data field is supposed to represent. A DBMs overcomes this problem by providing two views of data physical and logical.

Physical view of data is similar to traditional File Systems. It deals with the actual Location of bits and bytes on memory devices. Some Ms personnel need this information to be able to make efficient use of storage and processing resources. However, knowledge of all these details would serve no useful purpose for the application programmer who is interested only in using the information, no matter how it is stored.

The Logical view represents data in a format that is meaningful to the user and the application programmer. The emphasis here is on interrelating data fields and records such that they represent the underlying business reality.

Figure 1.3 shows how the DBMs insulates the user from physical storage details. The user or application programmer can refer to data items by using meaningful names, such as CUSTOMER NAME and TOTAL PURCHASE.

He no longer has to worry about specifying things like the number of bytes in a field. The ability to establish different logical views of the same data while insulating the user from concerns about how the data are physically stored has been a major contribution to the user friendliness of information systems.

Table 1.1 : Logical View of Sales Data

Sales person		Region	Year-to-date Sales		
1.D.#	Name		Product A	Product B	Product C
223	Smith	S.W.	6,395	4,328	5,875
227	O'Neill	S.W.	4,326	898	1,587
241	Maxwell	S.W.	12,331	8,976	7,215
256	Ware	East	8,232	6,554	7,321
257	Charles	East	2,111	4,573	5,321
258	Scholar	Midwest	5,221	6,632	6,331
276	Williams	Midwest	11,213	10,709	9,318
283	Mufti	Midwest	2,124	5,335	6,326
285	Cadd	Midwest	7,224	5,019	2,020
300	Harris	N.E.	3,423	3,302	8,824
307	Bentley	N.E.	8,635	5,661	3,624
310	Curtis	N.E.	10,728	7,187	8,721
322	May	N.E.	7,583	5,354	6,332

Basically, a data base system is nothing more than a computer based record keeping system, that is, a system whose overall purpose is to record and maintain information. The information concerned can be anything that is deemed to be of significance to the organisation the system is serving. In other words, anything that may be necessary to the decision-making processes involved in the management of that organisation. A data base system or environment consists of three major components : data base, software and users. Each of these is briefly discussed in the following paragraphs. Figure 1.4 illustrates the interconnection of these components.

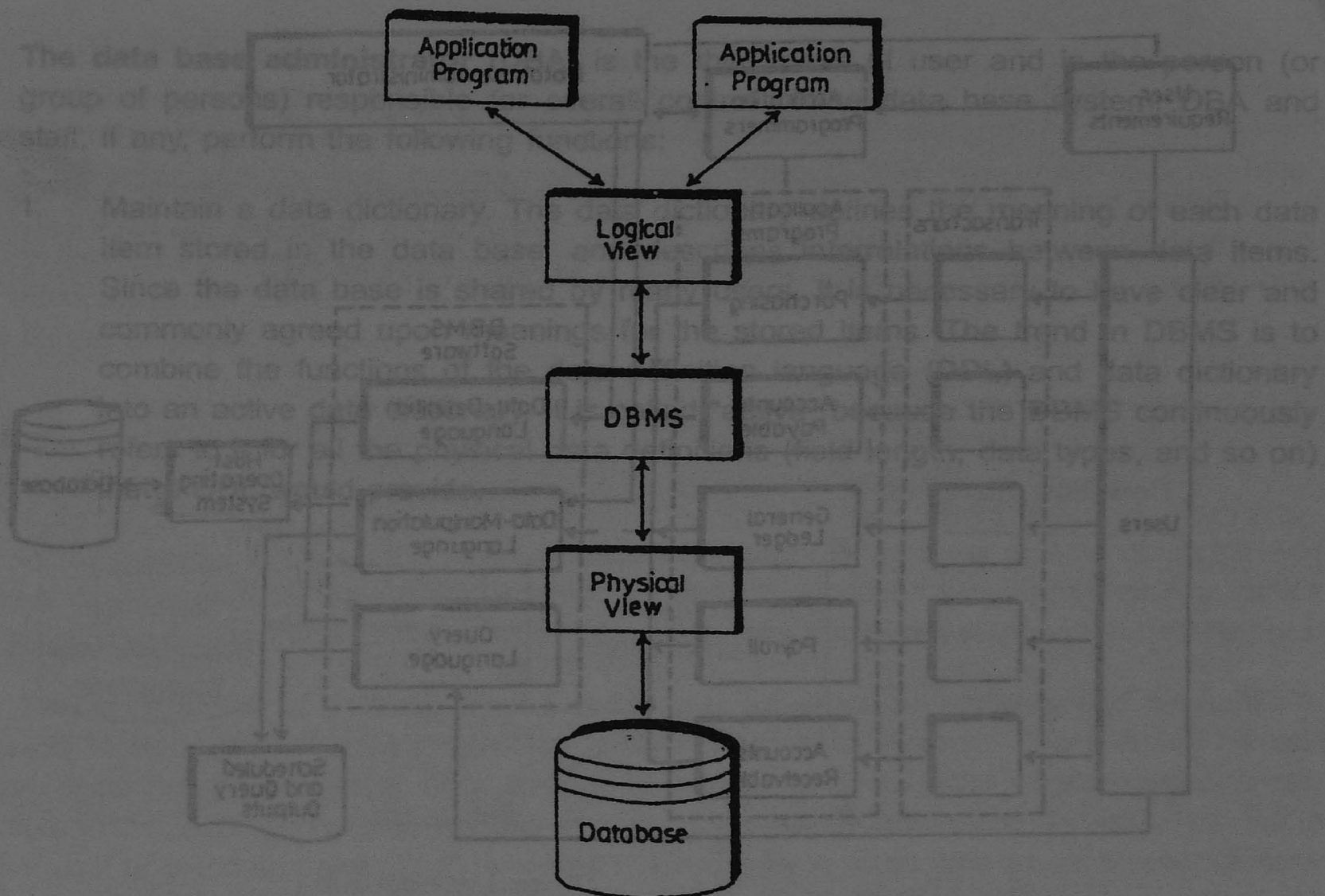
1.3 Data Base

Data base is the physical collection of data. The data must be stored on direct-access devices like magnetic disks. However, well-managed installations create backup copies of the data base on offline storage media such as magnetic tapes. The hardware, then, consists of the secondary storage volumes such as disks, drums, tapes, etc., on which the data base resides, together with the associated devices, control units, channels and so forth. The security measures are extremely important in a data base environment, since many departments and application programs may be dependent on a single, centralized data base.

Data base management systems are designed with a view towards optimizing the use of physical storage and CPU processing time. The logical view may contain redundant data items in order to make them more understandable to users. But the physical implementation of the DBMS attempt to make the physical storage nonredundant. This not only saves space, but also precludes the possibility of different values existing for the same data item at one time. The DBMS also uses other techniques to optimize resource utilization. Data records that are seldom used may be placed on inexpensive, slow-memory devices; whereas frequently used data may be put on the faster, but more expensive media.

1.4 Data Base Management System

Data base systems are rapidly gaining popularity among business users. Since data base operations tend to be high-volume processes, they often consume a large portion of the time and memory resources of the computer. Some vendors have developed computers that are dedicated entirely to data base operations. These computers, frequently referred to as data base machines, are special-purpose units. Certain capabilities are built into the hardware which make retrieval, sorting, updating, and other operations more efficient than with software programs.



The ability to establish different logical views of the same data while insulating the user from concerns about how the data are physically stored has been a major contribution to the user-friendliness of information systems.

Figure 1.3 : Logical versus Physical Views of Data

Data stored in the system is partitioned into one or more databases. A data base, then, is a repository for stored data. In general, it is both integrated and shared. By "integrated" we mean that the data base may be thought of as a unification of several otherwise distinct data files, with any redundancy among those files partially or wholly eliminated. By "shared" we mean that individual pieces of data in the data base may be shared among several different users in the sense that each of those users may have access to the same piece of data and may use it for different purposes. Such sharing is really a consequence of the fact that the data base is integrated.

Software

Between the physical data base itself, i.e., the data actually stored and the users of the system is a layer of software, usually called the data base management systems. All requests from users for access to the data base are handled by the DBMS. One general function provided by the DBMS is, thus, the shielding of data base users from hardware level detail.

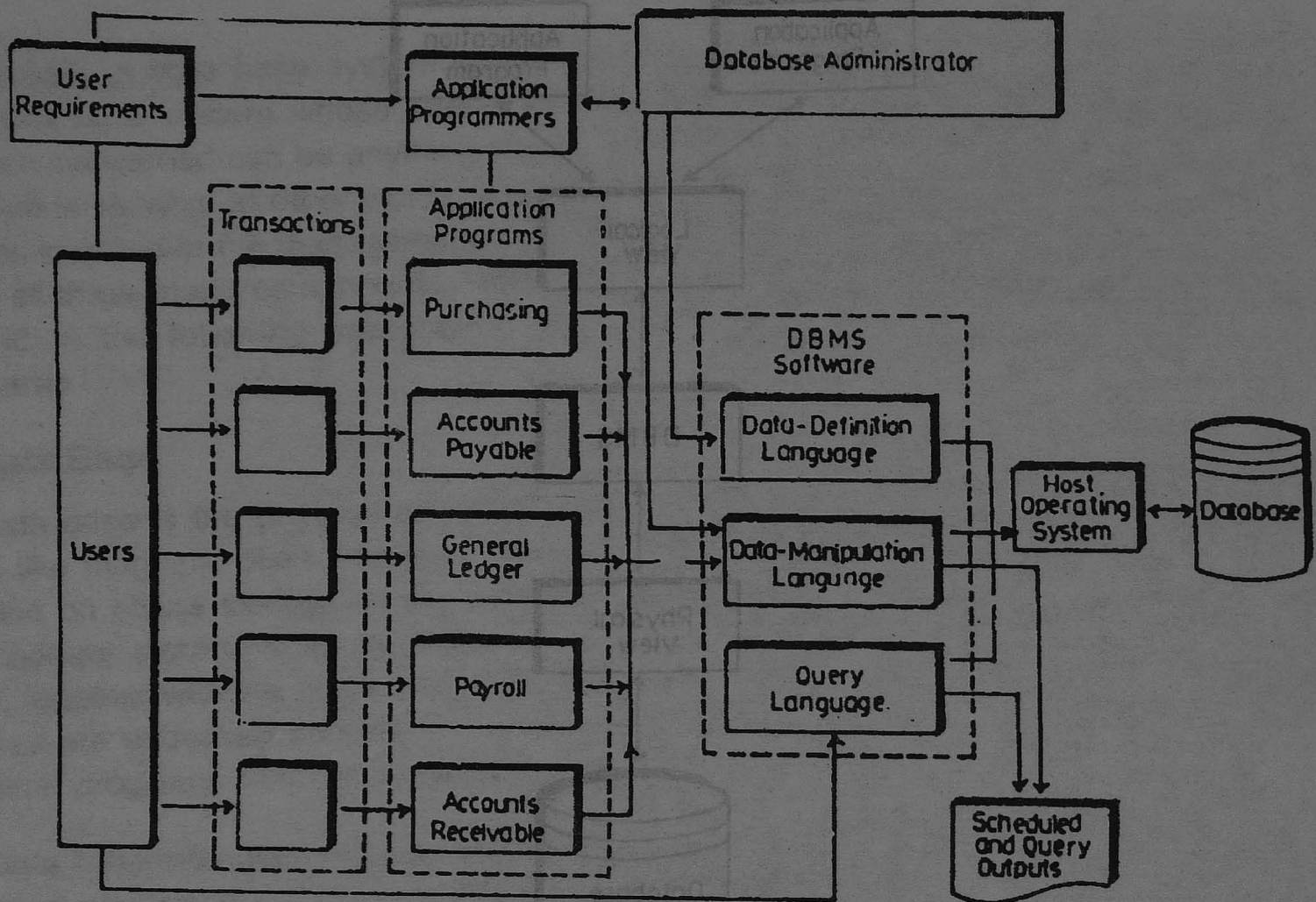


Figure 1.4 : Interaction among DBMS Components

Users

Three broad classes of users are application programmer, end-user or traditional users (such as management) and data base administrator. Users interact with the DBMS indirectly via application programs or directly via simple query language. The user's interactions with the DBMS also include the definition of the logical relationships in the data base (the logical view), and the input, alteration, deletion and manipulation of data.

1. The **application programmer** is responsible for writing application programs that use the data base, typically in a language such as Pascal, COBOL or PL/1. These applications programs operate on the data in all the usual ways : retrieving information, creating new information, deleting or changing existing information.
2. The **end-user** accesses the data base from a terminal and he may employ a query language provided as an integral part of the system, or may invoke a user-written application program that accepts commands from the terminal and in turn issues requests to the DBMS on the end-user's behalf. Either way the user may again, in general, perform all the functions of retrieval, creation, deletion and modification, although retrieval is the most common function used by this class of user.

3. The **data base administrator** (DBA) is the third class of user and is the person (or group of persons) responsible for overall control of the data base system. DBA and staff, if any, perform the following functions:

1. Maintain a data dictionary. The data dictionary defines the meaning of each data item stored in the data base, and describes interrelations between data items. Since the data base is shared by many users, it is necessary to have clear and commonly agreed upon meanings for the stored items. The trend in DBMS is to combine the functions of the data definition language (DDL) and data dictionary into an active data dictionary. It is called "active" because the DBMS continuously refers to it for all the physical data definitions (field length, data types, and so on) that a DDL would provide.

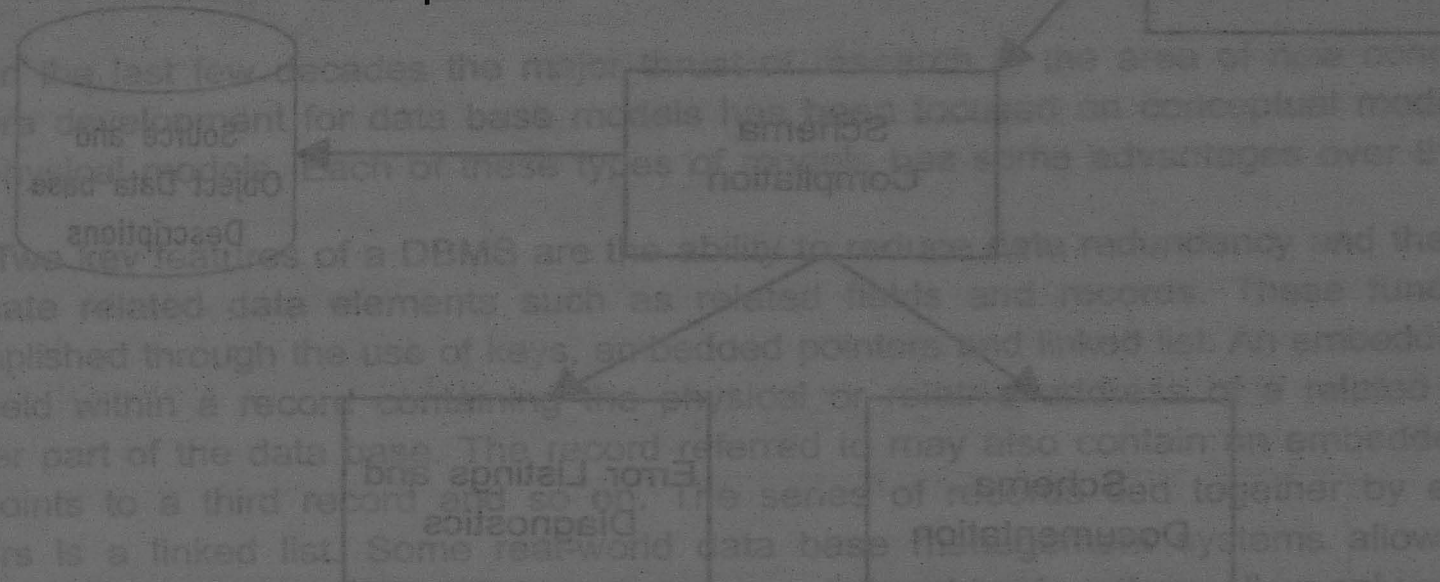


Fig. 2.1: Schema Compilation

Data manipulation facilities are provided by procedural commands that are usually used in conjunction with a conventional programming language (COBOL or PL/I for example) known as the "host language". The DBMS supplier then provides a pre-processor to translate the DML commands into the host language so that the whole program can then be compiled, using the standard compiler or assembler. The rest of the program logic for processing transactions lies and carrying out mathematical or logical function is thus written in the normal way. The preprocessor run typically operates as shown in Figure 2.2.

A data base program is normally executed in two stages. In the first stage, the data base is designed and created. A data base is designed and created in the second stage. In the second stage, the data base is accessed and linked to the program. This allows data base changes without having to re-compile programs.

The execute stage in which the DBMS routines are accessed by "CALL" statements to actually carry out the DML Commands. Generalized functions such as logging and privacy checking are carried out by the DBMS routines (see fig. 2.3).

How does a DBMS work?

DBMS has a compiler or translator program that accepts source definitions of the data base and produces a machine readable data definition which is almost always stored on a disk Library file. A typical scheme compilation is shown in Figure 2-1.

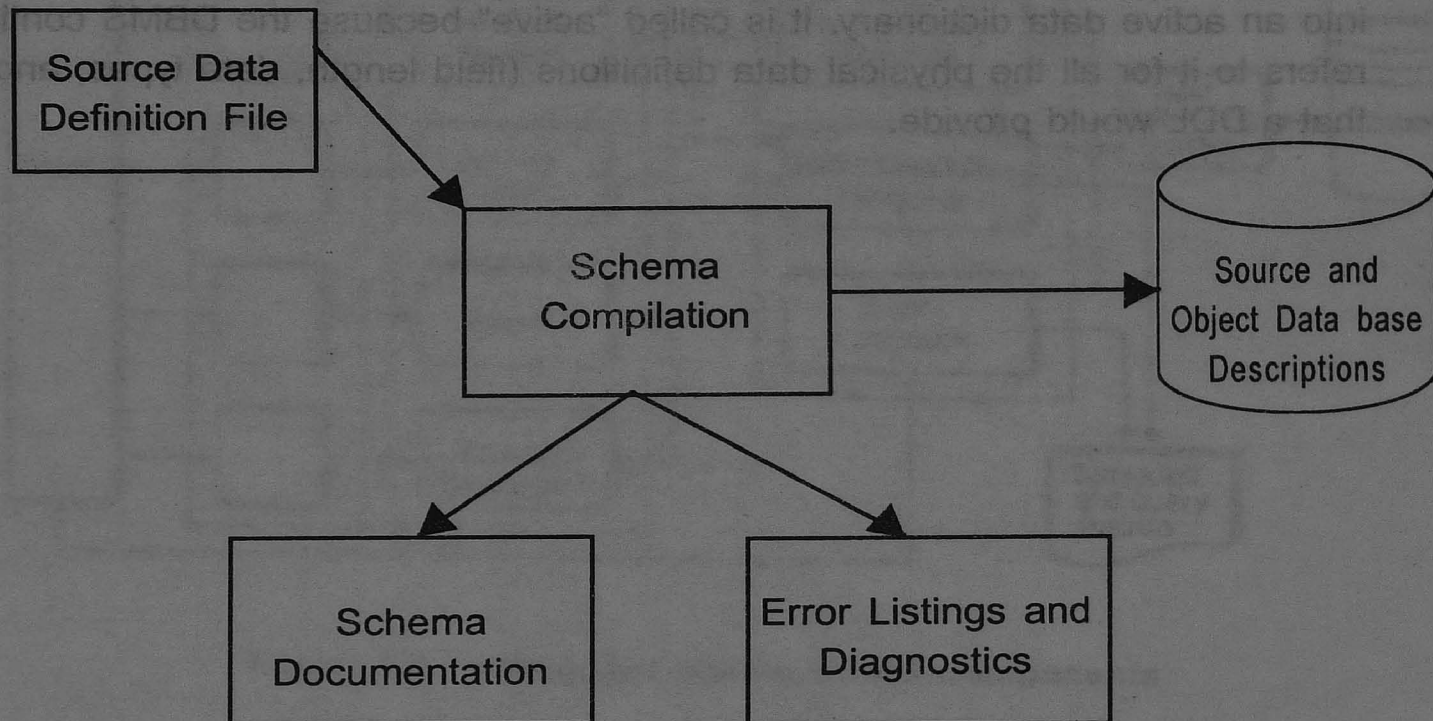


Fig. 2.1 : Schema Compilation

Data manipulation facilities are provided by procedural commands that are usually used in conjunction with a conventional programming language (COBOL or PL/1 for example) known as the "Host Language". The DBMS supplier then provides a pre-processor to translate the DML commands into the Host Language so that the whole program can then be compiled, using the standard compiler or assembler. The rest of the program logic for processing transaction files and carrying out mathematical or logical function is thus, written in the normal way. The preprocessor run typically operates as shown in Figure 2.2.

A data base program is normally executed in two stages :

1. The load stage in which the data descriptions are accessed and linked to the program definitions. This allows data base changes without having to re-compile programs.
2. The execute stage in which the DBMS routines are accessed by "CALL" statements to actually carry out the DML Commands. Generalized functions such as logging and privacy checking are carried out by the DBMS routines (see fig. 2.3)

2.1 Data Base Models

Data bases from the real world are frequently described within the framework of data models. Usually there are two classes of data models associated with data base management system software : one is referred to as the conceptual or semantic data model and the other is called the physical data model. The conceptual data model provides an environment to the user for interacting with the data base. The user can formulate his queries pertinent to the data base within the frame of the conceptual data model. Thus, conceptual models are the basis for development of high-level query languages.

The physical data model helps the data base administrator (i.e. users authorized to change the physical data) to organize the data and store them in the computer so that subsets of the data relevant to the user's request can be processed efficiently.

In the last few decades the major thrust of research in the area of new concepts and software development for data base models has been focused on conceptual models rather than physical models. Each of these types of models has some advantages over the others.

Two key features of a DBMS are the ability to reduce data redundancy and the ability to associate related data elements such as related fields and records. These functions are accomplished through the use of keys, embedded pointers and linked list. An embedded pointer is a field within a record containing the physical or relative address of a related record in another part of the data base. The record referred to may also contain an embedded pointer that points to a third record and so on. The series of records tied together by embedded pointers is a linked list. Some real-world data base management systems allow users to model and implement data on a tree, network or relational basis; others allow only one model, such as relational. Tree and network structured DBMSs usually tie related data together through linked list. Relational DBMSs relate data through information contained in the data, as we will see later.

In order to select a particular data model the designer of a large data base has to consider many potential users, each with a distinct perception of what the data base should look like. The way you organise information for a particular application is usually dictated by your information retrieval requirements. Information should be stored in a format that allows the data base management system to answer questions quickly and produce reports efficiently.

Briefly, in non relational systems, a cross-file query can be answered only if the link between the files has been described to the system such relationships must be defined when the data base is designed and created. As data base grows they can acquire new files with new relationships. With a relational data base system, the query language can use any relationship one of the major reasons for this is to accommodate the ever growing number of application programs directed at the data bases supported by data base management systems.

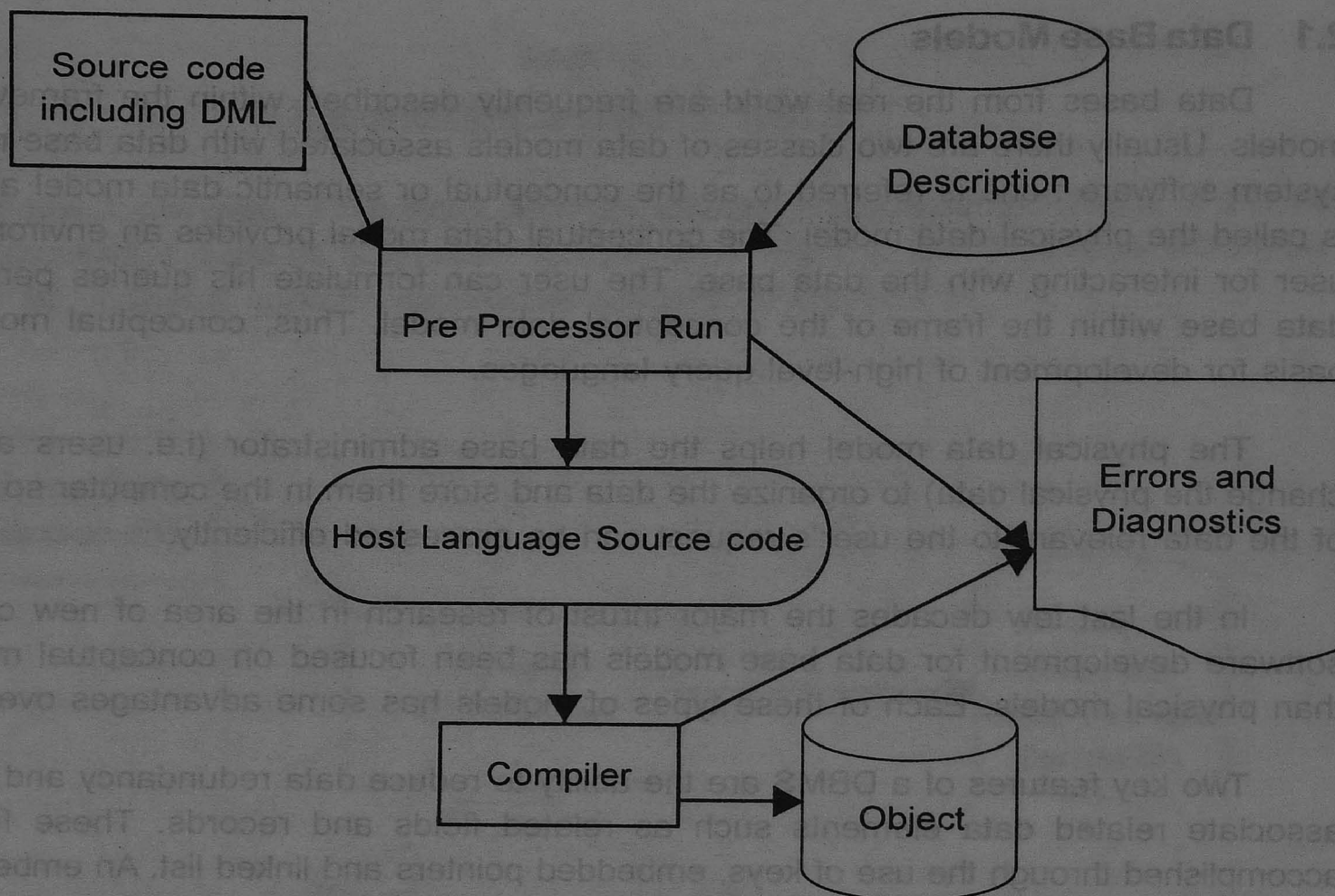


Figure 2.2 DML Pre-processor

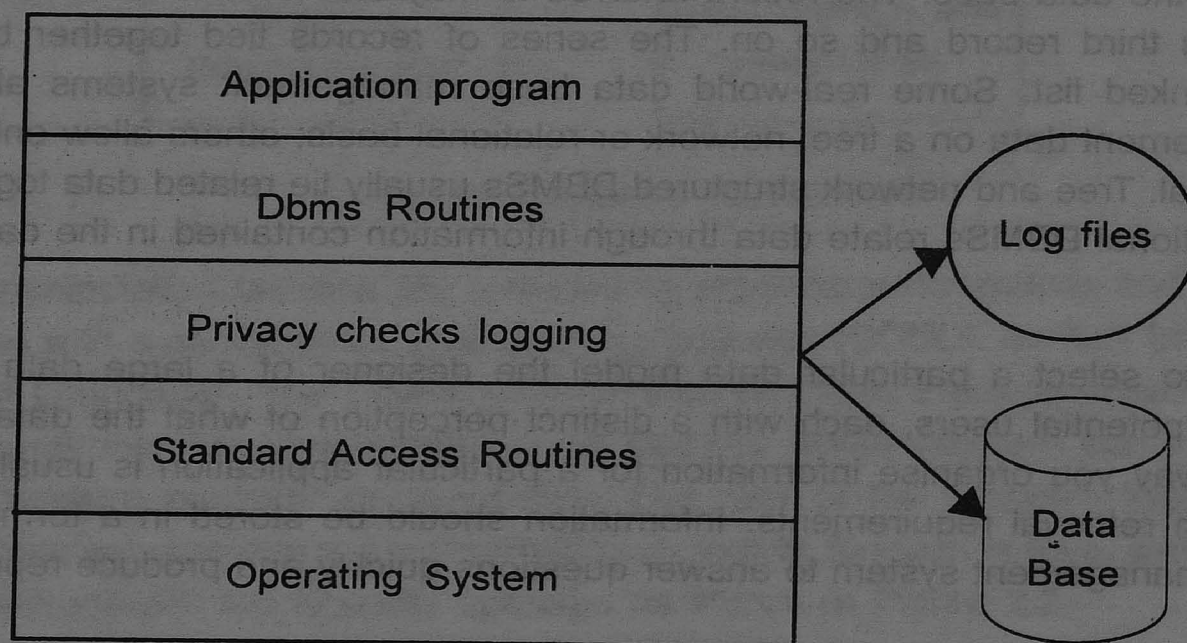


Figure - 2.3

DBMS in operation

Three major types of conceptual models are widely used in data base management systems. These are the relational model, hierarchical model and network model. The relational model is based on the concepts of relational theory and the hierarchical and network models are based on the concepts of graph theory that exists between files at any time. You do not have to predict the requirement for a relationship when you design the data base.

2.2 Relational Data Base

The basic concepts of relational models were introduced by code in the early nineteen seventies. Since then, these concepts have been extended and augmented by many researchers of data base management systems; today both experimental and commercial systems have been developed. In the relational model, the data are viewed as a collection of non hierarchical time-varying relations. These are special constructs of mathematical relations and are different from the traditional data processing files and tables. This view of the data enables the user to apply the powerful operations and expressions of relational algebra to data manipulations. This model provides a means of describing data with its natural structure only that is without superimposing any additional structure for machine representation purposes. It provides a basis for a higher - level data - language that yields maximal independence between the machine representation and organization of data. It also provides a sound basis for treating derivability, redundancy and consistency of relations. In this model, relational algebra has been used to decompose a complex logical structure into a collection of simple relations so that complex relations of the real world can be expressed also accessing and updating of data can be made simple and efficient.

A relational data base is composed of one or more relations. Each can be visualised as a table of data or file. Each row (tuple) in the relation represents one entity (such as a student in a student data base or an employee in a personnel data base) Each column (attribute) 'name' is drawn from the domain of names in the data base, or the attribute 'age' is drawn from the domain of possible ages. The value of an attribute is called an item.

In any relation no two rows can be identical and the ordering of the rows should not be significant. An attribute or set of attributes whose values uniquely identify a row of a relation is called a key. If a relation has more than one key, it is preferred to designate one of them as the primary key.

The beauty of relational database systems is that you can enter data without too much regard for how you are going to use it that is, you do not have to know how records will be combined when you enter the data. However if you do not plan how you are going to use data and enter it according to that plan, you have to spend some time getting the information out as the data base program searches through your data base files making the proper combinations and records the results. If you knew ahead of time how you wanted your information served you could have recorded it accordingly in a single file and saved yourself a lot of time when you retrieved it.

Education Data Base Example

This example gives some of the concepts mentioned in each of the data models. A company maintains an education and training department whose function is to run various training courses. Each course is offered at different locations within the company. The details of this education data base is as follows :

1. For each course; course number (unique). Course title course description, details of prerequisite course and details of all offerings (past and planned).
2. For each prerequisite course; course number and title.
3. For each offering of a given course; date, location, format, details about teacher's and students.
4. For each teacher of a given offering : Employee number and name.
5. For each student of a given offering : Employee number name and grade.

Figure 2.4 gives a relational view of the education data base example. The data are represented in tables; each table contains a key that identifies each record uniquely. The keys are also used to retrieve data from different tables.

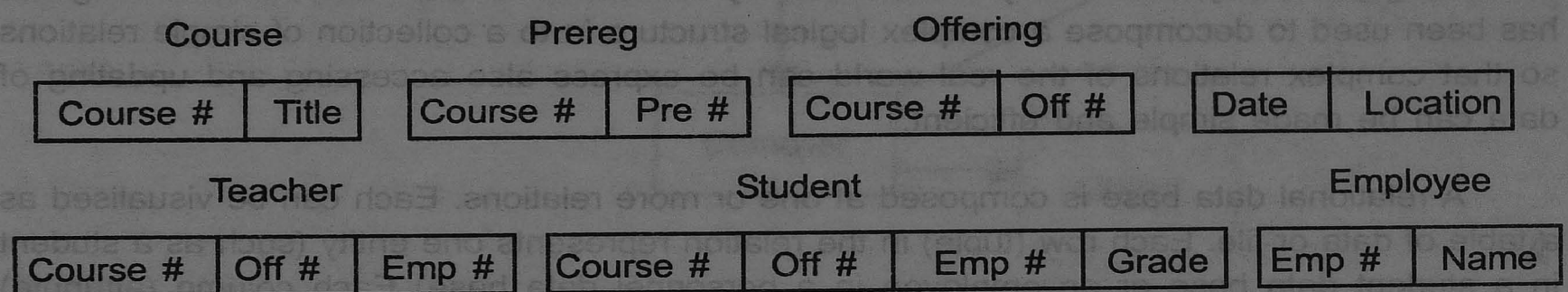


Figure 2.4 Relational view of the education data base

In relational data base, one can combine records from different files have one field in common. Under this system one can, theoretically, combine and retrieve selected records using a common field from an unlimited number of files with commands as SELECT, PROJECT and JOIN, SELECT operation selects a record or set of records corresponding to the value of a field. PROJECT creates subsets recards from existing records whereas JOIN Command combines records into new records

2.3 Hierarchical Data Base

Data is represented by a tree - structure type of organisation. The hierarchical tree specifies what record types are allowed to be included in the data base and the permissible relations between record types. Any record other than the root of the tree must be connected with a superior record (the parent) Thus, to insert a record, the user must select the parent record first when a record is deleted, all the descendants of the record are also deleted.

The hierarchical data base is made up of records, as is the relational data base, except that the records in a hierarchical data base do not have to be broken up in to fields. The records can look like relational records with several fields per record like a word processor file or like a one - dimensional array of data items. They can look like anything you want because the hierarchical system does not look at the contents of a record the way a relational system does. In a relational data base system, the connections between files depend on the data itself. For instance, although two files (or recordset) may have a common field, you do not join the files if the data in the two common fields is not alike.

In a hierarchical system, however, the connections between files do not depend on the data in the files. The connections are defined at the start and are fixed for the life of the data base. In this system, file A is always linked to file B. No matter what the contents of these two files are. And naturally enough, the connection between record type is hierarchical.

These connections are called the structure of the data and a diagram of them looks like an upside - down tree Figure 2.6 shows the data structure of the hierarchical data base. The file at the top is called the parent or owner file, while the files at the bottom are called the child or member files. The identifying characteristic of this system is the one-to-many connection between the owner and the member files. In other words, the member files have only one owner file.

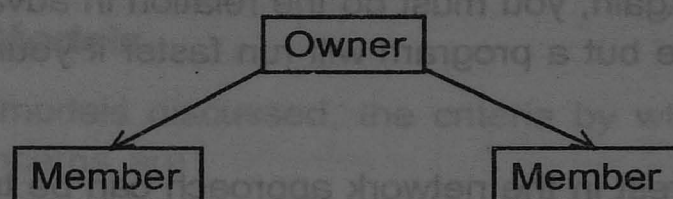


Fig 2.6 The Structure of a Simple Hierarchical Data Base

To implement a hierarchical data base, the designer must first specify how the records are related to one another. One common method of implementation is the use of pointers to maintain the relationship between files whose contents have similar characteristics Figure 2.7 shows a hierarchical view of the education data base example. COURSe is a root segment type the other segments are dependent segment types; Each dependent has a parent (segment type) e.g. the parent of TEACHER is OFFERING. Each parent has atleast one child (segment type). Retrieval of information from any segment type requires specifying the hierarchical path from the root to the required segment.

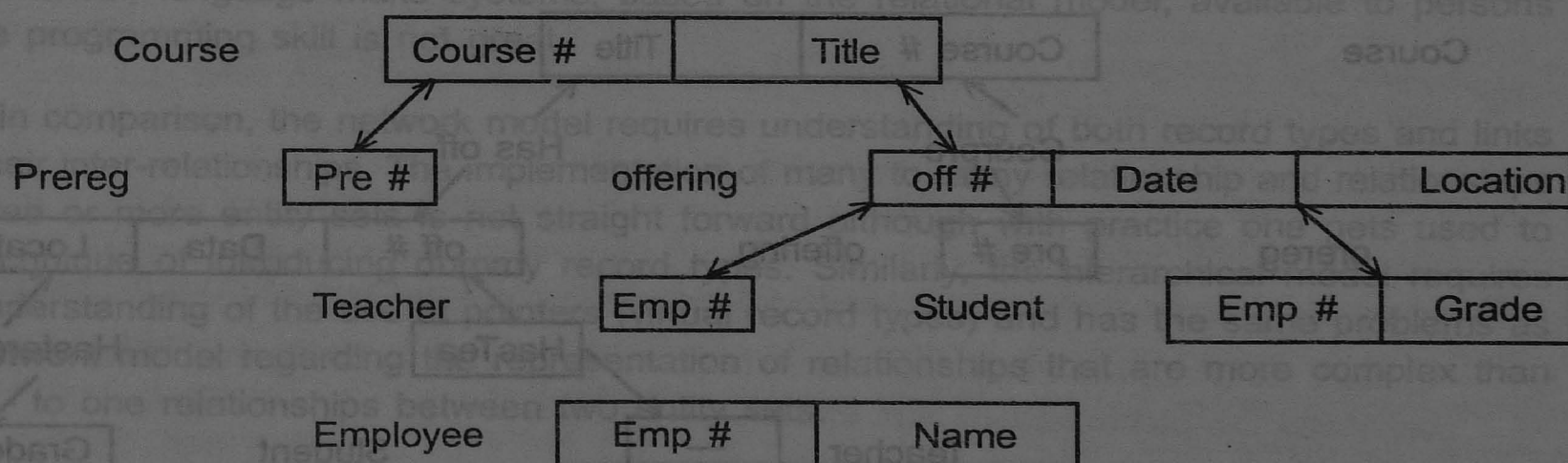


Figure 2.7 Hierarchical view of the Education Data base

2.4 Network Data Base

Network model is the generalization of the hierarchical model. In the network model, a segment can have multiple parent segments. In general, the segments are grouped as levels

but logical associations can exist between segments belonging to any level. Although, the logical associations are directional in nature any two levels can have both types of directional associations. In general, many to many logical associations can exist between any two segments and also their instances. The diagrammatic representation of the different logical associations between the segments resembles a graph. In General, these logical associations between the instances are implemented by pointers in the data set or by creating connecting relations that reflect the many - to many associations between the instances.

The net work data base is like a hierarchical data base but with one modification - you can relate any record types or files with any other record types or files that is, the data structure is many to many instead of one-to-many. A hierarchical data base is a degenerate - case of a network structure. Again, you must do the relation in advance. This is more restrictive than the relational data - base but a program will run faster if your define the data structure in advance.

Most of the current interest in the network approach can be traced back to the publication of the final report of the CODASYL Data Base Task Group (DBTG) in April 1971. The report noted that one of the most important aspects of the DBTG logical model is the set concept. Each occurrence of a set, represents a hierarchical relationship between the owner record in the set and the corresponding members connections are made by a chain of pointers that start from the over, pass through all the members and finally return to the owner record. Set types are also represented using the Bachman data structure diagram technique.

Figure 2.8 shows a network view of the education database example. The names on the arrows are set names, each set has a type of record as its owner, another record as its members. Notice that there can be many owner files for a single member file. In the set called COURPRE, the record COURSE is the owner and the record PREREQ is the member. The chain of pointers in COURPRE set is illustrated by an example in Figure 2.9.

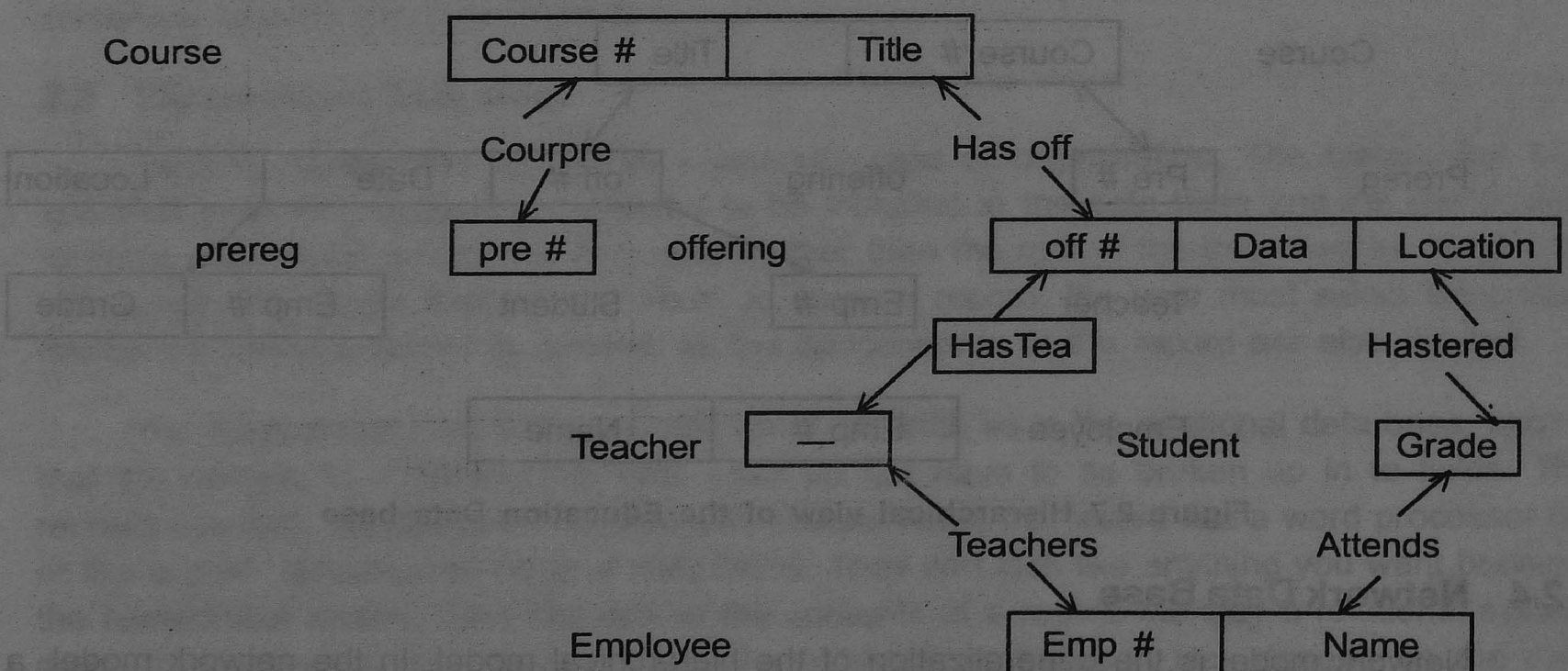


Figure 2.8 Network view of the Education Data Base

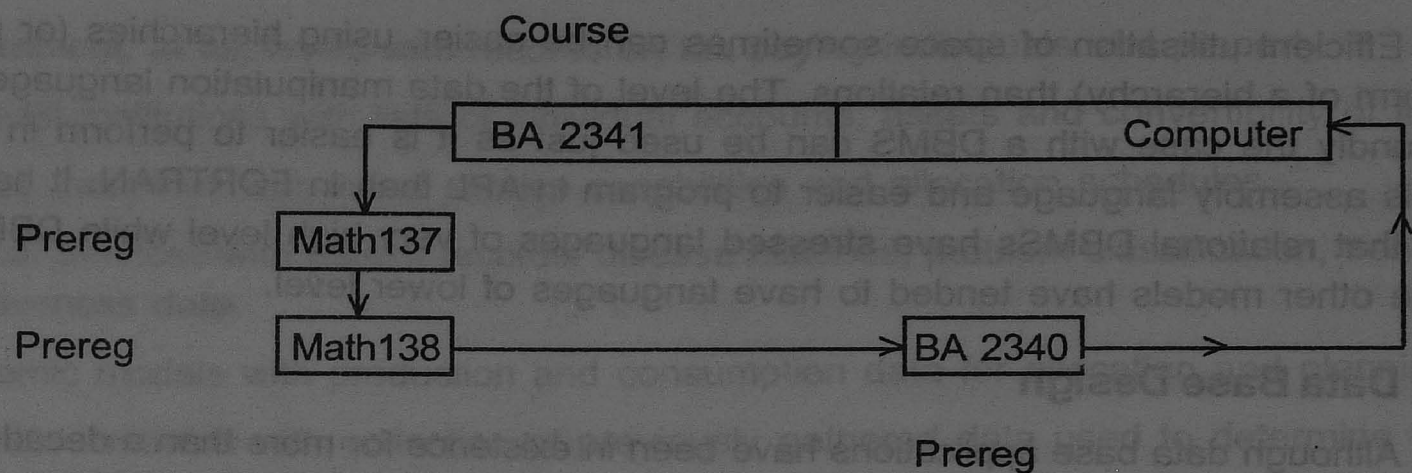


Figure 2.9 Chain Pointers in COURPRE Set

2.5 Comparison of the Models

To evaluate the three models discussed, the criteria by which they could be judged is stated first. Two primary concerns are

1. **Ease of use** : Especially in a small data base, of the order of thousands or tens of thousands of records the principal cost may be the time spent by the programmer writing applications programs and by the user posing queries. A data model is needed that makes accurate programming and the phrasing of queries easy.
2. **Efficiency of implementation** : When data bases are large, the cost of storage space and computer time dominate the total cost of implementing a data base. A data model is needed which is space efficient and in which queries can be answered efficiently.

By the criterion of easy use, there is no doubt that the relational model is superior. It provides only one construct that the programmer or user must understand. Moreover there are rich, high level languages for expressing queries on data represented by the relational model. These language make systems, based on the relational model, available to persons whose programming skill is not great.

In comparison, the network model requires understanding of both record types and links and their inter-relationships. The implementation of many to many relationship and relationships on three or more entity sets is not straight forward although with practice one gets used to the technique of introducing dummy record types. Similarly, the hierarchical model requires the understanding of the use of pointers (virtual record types) and has the same problems as the network model regarding the representation of relationships that are more complex than many - to one relationships between two entity sets.

Network and hierarchical models are better in terms of potential for efficient implementation Data structures such as the multi - list and the pointer - based implementation of variable length records do not generalise really to many-to-many mappings since relations can and often do, represents many to many mappings, efficient implementation can be more difficult for relations than for networks or hierarchical but some specialised data structures can be used to implement relations as well.

Efficient utilisation of space sometimes can be easier, using hierarchies (or networks in the form of a hierarchy) than relations. The level of the data manipulation language can affect profoundly the case with a DBMS can be used just as it is easier to perform in FORTRAN than in assembly language and easier to program in APL than in FORTRAN. It has been the ease that relational DBMSs have stressed languages of very high level while DBMS's based on the other models have tended to have languages of lower level.

2.6 Data Base Design

Although data base applications have been in existence for more than a decade, advances in techniques or data - base design have not kept pace with the advances in DBMS development and their applications. As data base applications become more prevalent and more complex, problems in designing a data base to support the applications become more difficult. Furthermore, as end - users of the data bases become more sophisticated, they place additional demands on design issues like integrity and consistency, which in earlier days were ignored, but now are an integral part of the design requirements.

Traditionally data base design has been separated into logical and physical design. Logical design covers that part of the design process wherein one designs a data base without the constraint of a DBMS and its storage and access techniques. Physical design is that part of the process that deals with the dependency of a DBMS. Unfortunately, although such a separation is desirable, in reality it is not quite achievable. One can easily see that it becomes difficult to evaluate the design without any reference to any DBMS. Logical design will therefore be reduced to merely the specification of applications and the checking of the specifications for completeness and in consistency. Nevertheless, the separation of the process into logical and physical design is useful in that the levels of the design will correspond to the levels of the DBMS.

Generally speaking, the steps in a data design process involve :

1. Collecting information, which includes defining fields, assigning and resolving fields, defining usage and so forth.
2. Formulating the requirements, including performance.
3. Forming records and files.
4. Analysing structure with respect to the need in (2) and repetition of (3) if necessary and
5. Forming an overall structure for the data base that includes the specification of access paths and the placement of files in storage if needed.

Applications

Application areas that can employ data base systems include :

1. Manufacturing with inventory management, Bills of Material Processing and production equipment scheduling.

2. Government, at all, levels with records on tax paying individuals and property.
3. Financial institutions with lists of individual accounts, assets and convertibility of funds.
4. Service industries with list of service capabilities and allocation schedules.
5. Medical Services with patient records, disease histories, problem classification, treatment effectiveness data.
6. Economic models with production and consumption data for allocation and planning.
7. Scientific research with collection of previously gathered data used to determine future research directions.
8. Officers who are automating their document management.
9. Libraries cataloging their abstracts and indexes of their holdings.
10. Airline reservations.
11. Ports Management.
12. Railway information system.
13. Statistics, used for survey and statistical data.
14. Others : (i.e) Electronic Mail

2.7 Problems in a DBMS Environment

Theoretically a DBMS promises a lot, yet, despite inherent advantages, the fact remains that there exist, even today, many limitations. Since the basic objective of a data base system is data sharing there is inevitably a prevailing sense of ownership, resulting in data being easily lost. In accurate data is often not detected. Unless extensive data integrity measures are adopted, there will be numerous problems like lack of security and loss of privacy. Also, when data is scattered in a number of data files, the availability of combined data from these files is some what restrictive. There is a difficulty in achieving 100 per cent data independence even with the best data base design. In cases where relations were not originally defined retrieval becomes very difficult.

It must be kept in mind that the data base approach cannot be adopted overnight. This requires not only the creation of a meaningful data base (containing all significant data) it also calls for a thorough preparatory job. How does one go about attaining the promise of the DBMS? Data base architecture must be planned at a high level, much before implementation. It has been observed that the barriers to the effective use of a DBMS are really more political and cultural than technical. Priorities ought to be defined, support from upper management essential. User awareness is another important requisite adequate training could well be dubbed mandatory.

The disadvantages of data base management system are relatively few, and in the long run are outweighed by the advantages. One disadvantage is that DBMS software is complex; the concepts used are often new to users. Therefore, a DBMS requires sophisticated data

processing personnel and sometimes a re-education of users. The current trend is to produce data base management systems that are easier for users to understand. Another disadvantage is that DBMS software creates additional overheads because it requires computer time to execute, disk space for software storage, and so on. However this is a disadvantage of all systems software. It is also not likely to be a major problem in the future as the cost of computer hardware declines.

Selecting Data Base Software

There is an increasing interest in the data base approach to data processing and the use of general purpose Computer programs - data base management systems DBMS - to aid the creation, updating and retrieval of information in data bases.

The selection of suitable software will obviously depend upon the situation with regard to hardware. In the simplest case, the hardware installed limits the choice to either implementing the package provided by the manufacturer or producing in house software. If the manufacturer of the hardware has a sufficiently large number of installations a number of independent software vendors may have produced packages which will have to be considered. The most complex case arises where both hardware and DBMS software are being selected and it may be that the choice of hardware will be influenced by the selection of the DBMS package.

The selection of a particular DBMS is likely to be for the long-term because of the expense a change would involve. It is therefore, important to consider all the alternatives.

Experience has shown that it is possible to perform a fairly superficial matching of requirements facilities to produce a shortlist of no more than two or three packages before going on to a more detailed evaluation.

These guidelines provide a list of questions which need to be asked when selecting data base software. Many of these questions are not concerned with the software techniques but are of vital importance in any evaluation.

Unit - III

3.1 Building a Data base

In this chapter you will learn the basic steps of getting dBASE II PLUS "Up-and-running" on your computer. This chapter will also teach you how to create a data base and add information to it. Finally, you will learn how to save your new data, retrieve it and how to properly exit dBASE.

Starting DBASE III PLUS

To try the examples in this section, you will read to get dBASE III PLUS UP and running on your Computer Before you can start dBASE you must install it according to the instructions in the Getting started manual that came with your dBASE III PLUS package. Then follow these starting instructions for the type of computer you have :

Computers with Two Floppy - Disk Drives

If you are using a computer with two floppy disk drives, place the dBASE III PLUS System Disk # 1 in Drive A and a blank, formatted disk in Drive B. Next to the DOS A > Prompt, type the command.

```
DBASE ↵
```

(Note : The symbol ↵ means "Press the Return or Enter Key")

Insert system Disk 2 an Press ENTER, or Press

Ctrol - C to abort.

Remove the disk from Drive A. Place your dBASE III PLUS System Disk ≠ 2 in Drive A, and press the Return Key. You will see the dBASE menu screen appear.

Computers with Hard Disks

If you are using a hard disk system and have already installed dBASE III PLUS. Simply log on to the appropriate directory on your hard disk (using the DOS CHDIR or CD Command) At the

```
C >
```

Prompt, enter the command

```
DBASE ↵
```

When the copy right notice appears, press, Return to move into the dBASE III PLUS menu screen.

Net worked Computer Systems

For techniques in starting dBASE III PLUS on a networked system of computers, See the "Getting Started" manual that came with your dBASE III PLUS package. Procedures will vary depending upon your particular network configuration.

3.2 Navigating the Menu Screen

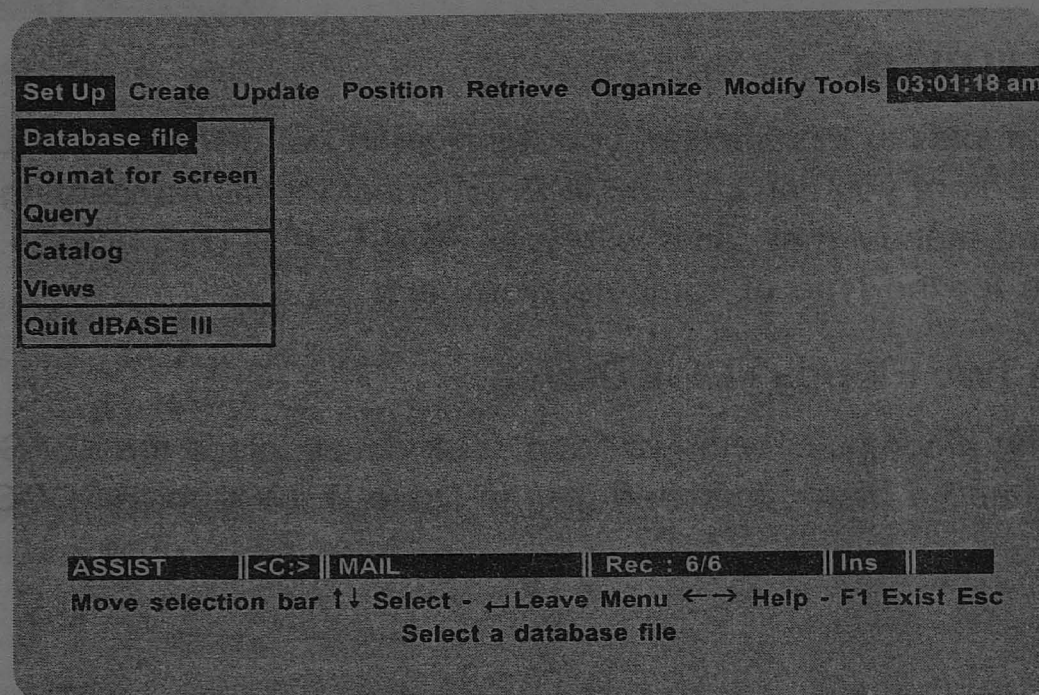


Figure 3.1

The dBASE III Plus Assistant Menu Screen.

Fig 3.1 shows the on-screen dBASE III PLUS menu screen. This menu is also called the Assistant, because its job is to assist you in building commands that dBASE can interpret and execute. As you will see later in the book, you can build your own commands without the Assistant Menu.

This screen consists of eight main menu options listed in a bar across the top of the screen, eight sub menus that pull down from the main menu bar as you highlight one of its options, a status bar and three message lines at the bottom of the screen.

If you press ← or → keys on the numeric keypad you will notice that the highlighting moves across the main menu bar, and "pull - down menus" representing sub - options appear beneath the highlighted option. (Note : If your arrow keys don't work and the "NUM" Message appears at the bottom of your screen, press the NUM LOCK Key and try again). You can also highlight items from the main menu bar by typing the first letter of the option. For example, typing the letter "T" brings the pull-down menu for the Tools option.

Notice that some items on the pull-down menus are brighter than others. The brighter options are those that are available for selection. The dimmer options can only be selected after certain pre-conditions are met. Most of the options will be available to you the moment that you select a data base.

Now type **S** to move the highlighting to the **Setup** option. You will select 'Data base file', is highlighted in reverse video. You can use the and keys to move this highlighting up and down within the pull - down menu. Pressing the Return Key selects the currently highlighted option.

The Dot Prompt

Pressing the Escape (Esc) Key "unselects" a pull-down menu option. If you press ESC before selecting a sub-option the menu screen will disappear and the dBASE III PLUS dot prompt will appear (which we will discuss in a later chapter). While the dot prompt is familiar to those with previous dBASE II or dBASE III experience, it may not mean a lot to the novice. If you should find that the menu disappears and only a period appears near the bottom of the screen, type the command

ASSIST ←

to bring back the Assistant Menu.

If you prefer entering commands at the dot prompt there is a way to completely bypass the dBASE III PLUS menu screen. However, most of these same commands are available as Assistant menu options. For more details, see "Entering commands at the Dot prompt later in this chapter.

The Status Bar

At the bottom of your screen (see Fig 3.1) you will see the status bar is reverse video. The status Bar keeps you informed of the current status of various optional settings. Just above the status Bar, the "ActionLine" will display the dBASE commands that you build from menu options. The action line is empty right now, but when you select menu, options later, it will display.

Beneath the status bar are the "Navigation Line" and Message Line. These two lines provide instructions and explain the currently highlighted option. When you are first learning remember to read these lines beneath the status bar to see what your options are and to get instructions for your next selection.

Getting Help

Before we go any further, let's take a moment to discuss a way to get help on-line (on the screen) while working with dBASE. Any time a menu option is highlighted, you can press the Help (F1) key to get some additional information about the option. For example, if you press F1 while the Create option is highlighted, you'll see the brief description of the option shown in Figure 3.2.

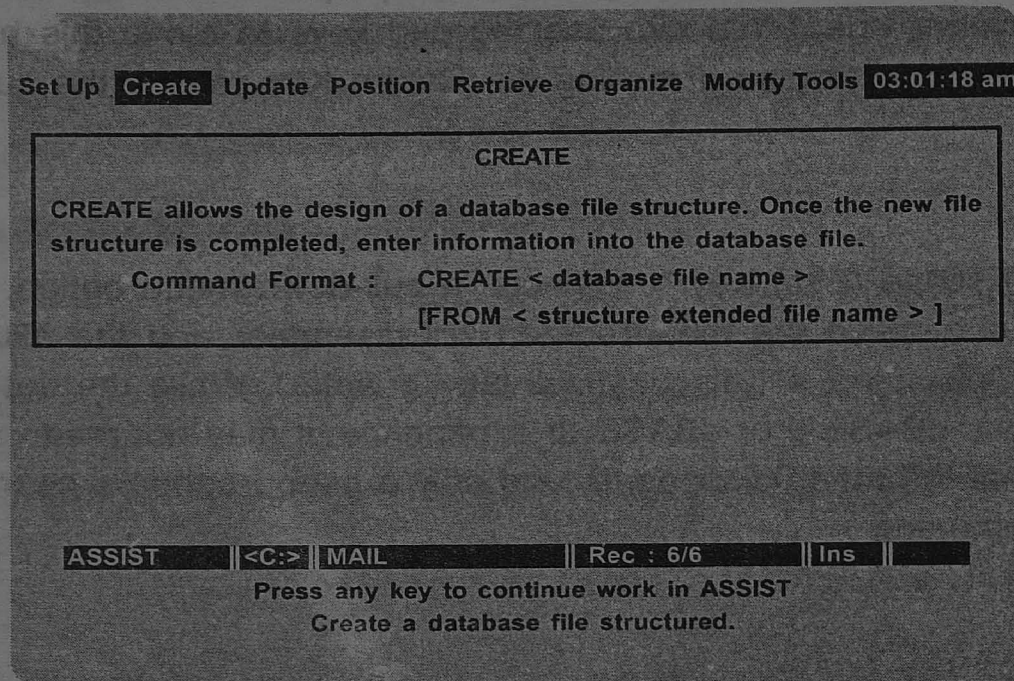


Figure 3.2 Help Screen for the Create Option

Some Help screens may seem a bit advanced at first because they explain techniques for working with dBASE III PLUS from the dot prompt (which we'll discuss later). Nonetheless, the F1 key is a quick way to get information without looking through the manuals.

When you finish viewing a help screen, press any key to resume your work. Now, let's build a database for managing a basic mailing list to put dBASE to work and start learning about the many capabilities dBASE III PLUS offers.

Creating a Data Base

To create a data base, highlight the "create" option from the main menu either by moving the highlighting to the word or by typing the letter **C**. The pull-down menu will present these options.

Data base file

Format

View

Query

Report

Label

To create a data base, select the Data base file option by pressing the Return Key. The screen will then display these options.

A :

B :

C :

If you are using a computer with two floppy-disk drives, select the B : option to create the data base on the disk in Drive B. If you are using a hard disk system the C : option

Next, dBASE will ask you to

Enter the name of the file

The name you enter can be up to eight characters long, but it may not contain spaces or punctuation marks. For this example, type the file name

MAIL ↵

If you make a mistake typing the name, you can use the Back space key to back up and make corrections dBASE will automatically add the extension DBF (for Data Base File) to the file name you assign. Therefore, when you enter MAIL as the file name, dBASE will actually store the data base under MAIL.DBF.

After you name your data base, dBASE provides some help at the top of a new screen. Note : Some keys have a ^ symbol in front of them, which means that you must press the ctrl key as you type the letter.

At the center of the screen, dBASE displays the blank form shown in Figure 3.3. You must fill in this form to create a data base structure.

Bytes remaining : 4000

CURSOR <-- -->	INSERT	DELETE	Up a field : ↑
Char : ← →	Char : Ins	Char : Del	Down a field : ↓
Word : Home End	Field : ^N	Word : ^Y	Exit/Save : ^End
Pan : ^↑ ^↓	Help : F1	Field : ^U	Abort : Esc

Field Name	Type	Width	Dec	Field Name	Type	Width	Dec
1	Character						

ASSIST | <C:> | MAIL | Rec : 6/6 | Ins |

Enter the field name.

Field name begin with a letter and may contain letters, digits and underscores

Figure 3.3 : Form for Creating a Database Structure

Let's discuss what all of this means. Notice that dBASE is asking for information about the fields in each record. For each field, it needs to know the name, type, width, and number of decimal places. The name of the field can be up to ten characters long, but no spaces are allowed, and the only punctuation allowed is the underscore character (_). dBASE also needs

to know the type of data being stored in the field. Data can be one of five types : 1) Character, for non-numeric data such as name, address, etc. 2) Numeric, for numbers that we want to do some math with, like dollar amounts or inventory quantity; 3) Date, for dates in MM/DD/YY format; 4) Memo, for long passages of text; 5) Logical.

Where the field is either true or false. Then dBASE needs to know the width of each field, the maximum number of characters that the field will contain. Finally, it needs to know how many decimal places are to be stored for numbers. For, instance in a dollar amount, we typically store two decimal places (\$999.99).

To fill in information on the screen, simply type in the field name, followed by a press on the Return Key. The cursor will move to the "type" prompt. Type the first letter of the type of data : Character data, Numeric data, Dates, Memos, or Logical data. In your first example just use all character types to keep things simple. The cursor will move to the "width" column. Type the maximum width for the field followed by a press on the Return Key. If you make errors while typing data, you can use the arrow keys on the numeric key pad (on the right side of the key board) to position the cursor for making changes.

Structure the MAIL.DBF data base like this :

Field	Field Name	Type	Width	Dec
1	LNAME	Character	15	
2	FNAME	Character	10	
3.	ADDRESS	Character	25	
4.	CITY	Character	15	
5.	STATE	Character	5	
6.	ZIP	Character	10	

Rather than typing in a seventh field, just press the Return Key. dBASE will display the message.

Press ENTER to confirm. Any other key to resume. Press the Return Key.

Notice that we've broken the first and last names into separate fields. The last-name field (LNAME) can take a last name fifteen letters long. The first-name field (FNAME) can hold up to ten letters.

Why are there two separate fields? Because in the future, you might want to sort the data alphabetically by last name. If you just had one field for both first and last names, such as Joe Smith, when you did your sort, dBASE would sort by first name. You and I can look at Joe Smith and immediately see that smith is the last name since the computer doesn't understand this you have established the difference between first and last name by providing a separate field for each. You have also put address, city, state and zip into separate fields. Note that 'C' indicates that each of these is in a character data field.

Now you will probably ask "why is zip code a character string"? Is n't 92/22 a number?" Yes, it is, but hyphenated zip codes like 92038-2802 could cause problems. In dBASE, the hyphen means "subtract" when dealing with numbers. So at some point 92038-2802 might become 89236 if stored as a number (92038 minus 2802 = 89236).

Another problem is that some foreign zip codes have letters in them, like A132-09. In dBASE III PLUS, letters are not allowed in Numeric data fields you will avoid a lot of trouble by making ZIP a character field. The only time you must make a field Numeric is when you need to do Math. Certainly, you will never need to total up ZIP codes!

So we now have a structured data base. All meaningful pieces of information are broken out into separate fields. Again, avoid the temptation to combine several pieces of information into one field.

(CITY : STATE : ZIP). dBASE III PLUS allows a maximum of 128 fields in each record, with their combined widths totalling up to a maximum of 4000 characters. There is plenty of room.

On your screen you will now see that the computer is asking if you want to

Input data records now? (Y / N)

Type N to answer No and return to the maximum there you have it - a data base called MAIL.DBF. Now, let's discuss how to open the data base and how to add some data to it.

3.3 Opening a Data Base

Imagine that you have a number of data bases, like shoe boxes filled with index cards. Before you can add cards (records) to one of these databases, you must open its box. So let's discuss the easy technique for opening your MAIL.DBF data base now on the main menu bar. dBASE displays these options on the pull-down menu:

Data base file

Format for screen

Query

Catalog

View

Quit dBASE III

Press Return to select the "Data base file" option. The screen displays

A :

B :

C :

for specifying a disk drive. As before, Select B : if you are using a computer with two floppy-disk drives, or C : if you are using a hard disk. dBASE will display the names of all existing data base files on the drive. Move the highlighting to MAIL.DBF and press return. dBASE will ask

Is the file indexed? [Y / N]

We won't be creating index files until a later chapter, so just type 'N' to answer No. At the bottom of the screen, on the highlighted status bar, you will see

< B : > MAIL

or

< C > MAIL

indicating that the MAIL.DBF data base is open and ready for work on either Drive B or C. Now let's add some data.

3.4 Entering Data

To add data to the file, highlight the "Update" option on the main menu, and select the "Append" option from the pull-down menu.

A blank form for entering new data will appear, as in Figure 3.4.

CURSOR <-- -->	UP DOWN	DELETE	Insert Mode : Ins
Char : <-- -->	Field : 1 1	Char : Del	Exit/Save : ^End
Word : Home End	Page : PgUp Pg Dn	Field : ^Y	Abort : Esc
	Help : F1	Record : ^U	Memo : ^Home

LNAME	[REDACTED]
FNAME	
ADDRESS	
CITY	
STATE	
ZIP	

ASSIST	<C:> .MAIL	Rec : None	Ins
--------	------------	------------	-----

Figure 3.4 : Blank Data Form

Notice the keys that appear at the top of the screen. You used some of the same keys creating the MAIL data base structure. Now you'll have a chance to try some of them out as you add new records to the data base.

A Sample Database Record

The following information will become the first record for your database:

John Q. Smith
123 A St.
San Diego, CA 92123

Because the order of the fields is slightly different than this Rolodex-card style, follow these steps to enter the first record.

Smith ↵
John Q. ↵
123 A St. ↵
San Diego ↵
CA ↵
92123

When you finish, your screen should look like Figure 3.5.

Record No. 1

CURSOR <-- -->	UP DOWN	DELETE	Insert Mode : Ins
Char : <-- -->	Field : ↑ ↓	Char : Del	Exit/Save : ^End
Word : Home End	Page : PgUp Pg Dn	Field : ^Y	Abort : Esc
	Help : F1	Record : ^U	Memo : ^Home

LNAME
FNAME
ADDRESS
CITY
STATE
ZIP

ASSIST || <C:> || MAIL || Rec : None || Ins ||

Figure 3.5 : First MAIL Record

Press Return after entering the zip code, and a blank form for filling in the second record will appear.

The Numeric Keypad and Cursor-Movement Keys

When you are entering a number of new records, it's easy to make mistakes. With dBASE it's also easy to correct mistakes as you enter data.

Before adding the second record to the database, take a moment to review the basic data-entry commands shown in Table 3.1.

(Note : The Symbol means, "Hold down the ctrl key while typing the letter that follows").

Key		Alternate Key	Effect
↑	or	^ E	Move cursor up one field
↓	or	^ X	Move cursor down one field
←	or	^ S	Move cursor left one space
→	or	^ D	Move cursor right one space
Del	or	^ G	Delete character over cursor
Ins	or	^ V	Turn the insert mode on/off
PgUp	or	^ R	Move back to previous record
PgDn	or	^ C	Move forward to next record
End	or	^ W	Save all newly added data and return to the menu
Home			Move to first letter of word
End			Move to end of word
^ Y			Delete contents to the right of the cursor
^ U			Delete/Undelete entire record

Table 3.1 Basic Data-Entry Control Keys

All of the control-key commands are in the left-hand portion of the keyboard, near the ctrl key. This is so that you can hold down the ctrl key with your little finger while pressing the appropriate key. The position of the keys suggests the direction that the cursor moves, as shown in Fig 3.6. The arrow keys and Ins and Del keys on the numeric keypad are shown in Fig. 3.7.

The arrow and control-key commands hold true for most dBASE forms that appear on the screen.

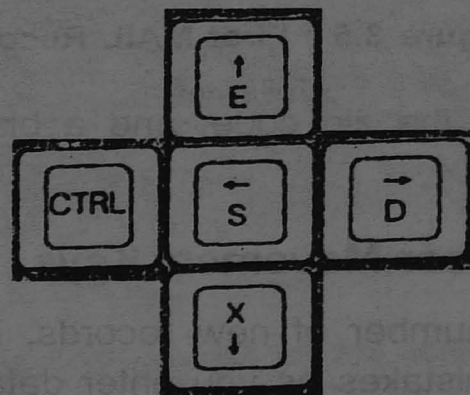


Figure 3.6 Cursor-Movement Keys

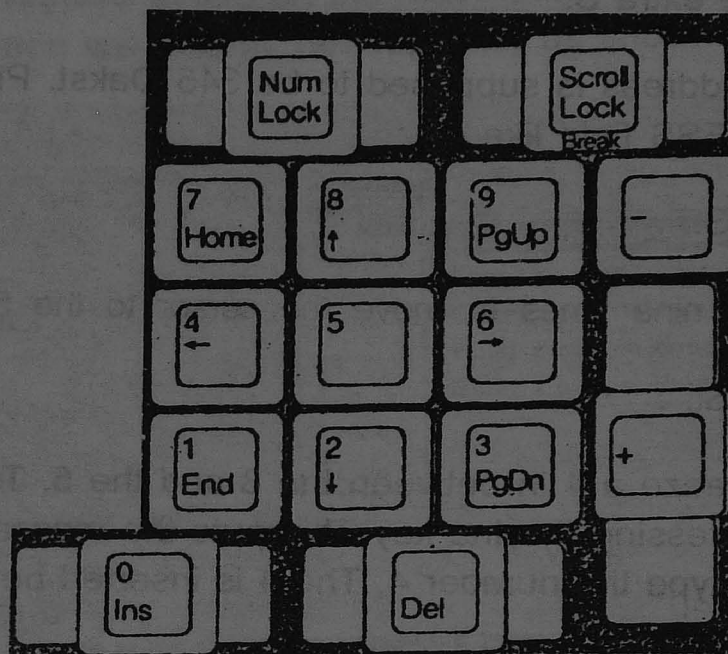


Figure 3.7 : Numeric Keypad

Let's add a second record. Suppose that while adding the second record, you type the following data :

```
L NAME      : Appleby      :
FNAME       : Andy         :
ADDRESS     : 35 Oakst     :
CITY        : Los Angeless :
STATE       : CA           :
ZIP         : -            :
```

Before typing the Zip Code, notice a couple of errors in the field above. To fix these, first, you can move the cursor up to the CITY field by pressing the ↑ key twice or by holding down the ctrl key and tapping the E key twice. The cursor moves to the beginning to the CITY field like so.

CITY : Los Angeless

Now to move the cursor to the right, press the End Key twice and then ← twice. The cursor will move to the Firsts' in Angeless, like this :

CITY : LOS Angeless :

Delete the character above the cursor using a ^G or by pressing the Del key. Now the city field like this :

CITY : Los Angeles :

You 've eliminated the extra S.

Next notice that the address is supposed to be 345 Oakst. Press the ↑ key to move the cursor up to the ADDRESS field like so :

ADDRESS : 35 Oak St. _ :

Then press the ← key nine times to move the cursor to the 5 in the ADDRESS field :

ADDRESS : 35 Oak St. :

Now you want to squeeze a 4 in between the 3 and the 5. To do so, your first have to join to an Insert mode by pressing the Ins key This puts the message "INS" at the bottom of the screen. Now if you just type the number 4. The 4 is inserted between the 3 and the 5 like this :

ADDRESS : 34 5 Oak St. :

Now move back down to the ZIP field by pressing the ↓ key three times, then pressing the ← key twice to move the cursor to the left of the entry box. Then type a Zip code (92123). Pressing Return after the zip code will then move the cursor onto record 3 for appending.

3.5 More Sample Records

So far so good. You have created a data base and added two records to it. At this point, we suggest that you put in more data. If the appeared form is still showing on the screen, just start keying in the following names and address (otherwise, use the Set Up, Update and "Append" options to return to APPEND Mode). Here are some data (records 3-6) for you to type. You will use them for future examples :

LNAME	FNAME	ADDRESS	CITY	STATE	ZIP
Smith	Dave	619E/mst	SanDiego	CA	92/22
Smith	Betsy	222 Lemon Dr.	Newyork	NY	1/234
Smithsonian	Lucy	461 Adans St.	San Diego	CA	92122-1234
Doe	Ruth	1142 J.St	Los Angeless	CA	91234

After you type the data for the last person on the list, dBASE will still ask for more data. To stop adding names, just press the Return key rather than typing a last name. Once you get all the data entered, you need to learn ways to retrieve the information.

3.6 Retrieving Database Records

Data stored in a data base doesn't do any, you much good until you can get it back out. We'll discuss many ways to organize and retrieve data throughout the next couple of chapters, but let's take a moment now to try a simple retrieval to verify that the data you have typed are stored on the disk.

First, highlight the Retrieve option on the main menu bar. Then, select the List option by pressing Return. The screen will display new options, as shown in Figure 3.8.

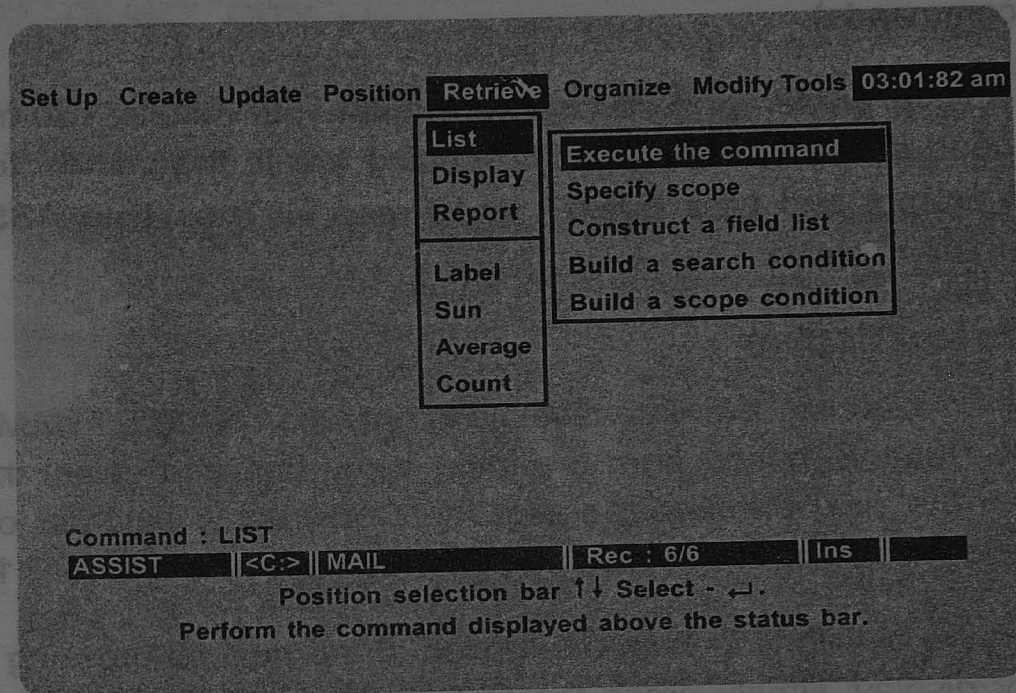


Figure 3.8 : Menu Options for Data Display

Next, select the Executive the command option from the submenu by pressing Return once again. dBASE will ask

Direct the output to the printer? [Y/N]

If you answer Y to this question and if your printer is ready, the data will be printed. For now, simply type N to display the data on the screen. You'll see the six stored database records appear, as in Figure 3.9.

Notice that the records were too wide for the screen, so they wrapped around, placing the zip code over on the left side of the screen. You can clean up this display by asking that

The screenshot shows the dBASE screen displaying the MAIL database records. The records are wrapped, with the zip code appearing on the left side of the screen. The status bar shows: ASSIST, <C:>, MAIL, Rec : 6/6, Ins. Below the status bar, it says: Press any key to continue work in ASSIST.

Record #	LNAME	FNAME	ADDRESS	CITY	ST
1	Smith	John Q.	123 A St.	San Diego	CA
2	Appleby	Andy	345 Oak St.	Los Angeles	CA
3	Smith	Dave	619 Elm St.	San Diego	CA
4	SMITH	Betsy	222 Lemon Dr.	New York	NY
5	Smithsonian	Lucy 461	Adams St.	San Diego	CA
6	Doe	Ruth	1142 J. St.	Los Angeles	CA

Figure 3.9 : On-Screen Display of MAIL Records

only certain fields be displayed. First, as the instructions at the bottom of the screen indicate, press any key to continue working in the Assist mode (working from the menu options rather than dot-prompt commands).

If you have a laser printer, and do choose to print your results, you may not see results right away. You'll need to eject the page after sending text to the printer. To do so, press Esc to access the dot prompt and type **EJECT** and Press **↵** Then type **ASSIST** and Press **↵** to return to the Assistant menu.

Specifying Fields for Display

Let's try another exercise, this time displaying only the LNAME, FNAME and CITY fields. Again, highlight the Retrieve option from the main menu bar, and select the List option from the submenu. From the next submenu, select the Construct a field list option. The screen displays a list of field names in the MAIL database, as shown in Figure 3.10.

You can move the highlighting from one option to the next by pressing the **↑** and **↓** keys. To select a field for display, simply press the Return key as the option is highlighted. In this

The screenshot shows a menu interface for the MAIL database. At the top is a menu bar with options: Set Up, Create, Update, Position, Retrieve, Organize, Modify Tools, and a timestamp 03:01:57 am. The 'Retrieve' option is highlighted. Below the menu bar, there are two columns of options. The left column lists fields: LNAME, FNAME, ADDRESS, CITY, STATE, and ZIP. The right column lists actions: List, Display, Report, Label, Sun, Average, and Count. A third column on the right contains a list of commands: Execute the command, Specify scope, Construct a field list (highlighted), Build a search condition, and Build a scope condition. Below these columns is a table with four columns: Field Name, Type, Width, and Decimal. The table contains one row: MAIL -> LNAME, Character, 15. At the bottom of the screen, there is a command line showing 'Command : LIST' and 'ASSIST'. Below the command line is a status bar with text: 'Position selection bar ↑↓ Select - ↵. Leave Menu ←→' and 'Specify which fields to include in this retrieval.'

Field Name	Type	Width	Decimal
MAIL -> LNAME	Character	15	

Command : LIST
 ASSIST <C> MAIL Rec : EOF/6 Ins
 Position selection bar ↑↓ Select - ↵. Leave Menu ←→
 Specify which fields to include in this retrieval.

Figure 3.10 : On-Screen Display of Field Names

example, press Return to select the LNAME field. Then press Return again to select the FNAME field. Press **↓** to skip over the ADDRESS field, and then press Return to select the CITY FIELD. When you finish, the LNAME, FNAME, and CITY FIELDS will be dimmer than the others. (Because they have already been selected, they are no longer available for selection, and are shown in dimmer type). On the Action Line near the bottom of the screen, you'll see this command message :

Command : LIST LNAME, FNAME, CITY

Your menu selections have created a command that could have been entered at the dot prompt :

LIST LNAME, FNAME, CITY ↵

dBASE can understand this command in its own language. Now, press → to remove the submenu of field names. Then press ↑ twice to highlight the Execute the command option,

Record #	LNAME	FNAME	CITY
1	Smith	John Q.	San Diego
2	Appleby	Andy	Los Angeles
3	Smith	Dave	San Diego
4	SMITH	Betsy	New York
5	Smithsonian	Lucy 461	San Diego
6	Doe	Ruth	Los Angeles

ASSIST || <C> || MAIL || Rec : EOF/6 || Ins ||

Press any key to continue work in ASSIST

Figure 3.11 : On-Screen Display of Selected MAIL Fields

and press Return. When dBASE asks if you want to display the data on the printer, answer N. dBASE will display the LNAME, FNAME, and CITY fields from the MAIL database, as shown in Figure 3.11.

In the next chapter, we'll look at more advanced techniques for displaying data on the screen and printer. Let's first take a moment to discuss the dBASE dot prompt, which is an alternative method for accessing your data.

Entering Commands at the Dot Prompt

If you press the Esc key from the main menu, the menu will disappear and a period will appear at the lower-left corner of the screen.

From here, you can enter commands using the dBASE language. Entering commands from the dot prompt is somewhat faster and less tedious for more advanced users, although it does require familiarity with the language.

Recall that when you selected menu options to display only the LNAME, FNAME, and CITY fields from the MAIL database, the screen displayed the common LIST LNAME, FNAME CITY. You can type this command at the dot prompt to achieve the same result. Type this command :

LIST LNAME, FNAME, CITY ↵

As before dBASE will display the three fields when the dot prompt reappears, type this command :

```
LIST FNAME, LNAME STATE
```

Notice that dBASE displays the fields you requested :

```
LIST FNAME, LNAME, STATE
```

Record #	FNAME	LNAME	STATE
1.	John	Smith	CA
2.	Andy	Appleby	CA
3.	Dave	Smith	CA
4.	Betsy	Smith	NY
5.	Lucy	Smithsonian	CA
6.	Ruth	Doe	CA

To open a database when working at the dot prompt you use the USE command. For example, to open the MAIL.DBF database on a hard-disk system, you could enter this command:

```
USE MAIL ↵
```

For a floppy-disk system you would need to specify Drive B. In this case, you would first enter the command

```
SET DEFAULT TO B ↵
```

to tell dBASE that the data are stored on the disk in Drive B. Then you can enter the command:

```
USE MAIL ↵
```

to open the MAIL.DBF database.

For the novice or occasional user, the Assistant menu will probably be the preferred method for accessing your data for a while. Later, you may want to learn about the dBASE language for more speed or for more flexibility in entering your commands. If you plan on learning to program in dBASE III PLUS. You will need to learn all the commands.

Saving your work

Before we proceed any further, there is one very important item of information that you need to learn. Always exit dBASE before turning off your computer. Failure to do so might cause you to lose some data that you have entered in a database.

To exit dBASE from the menu screen, highlighting the Set Up option from the main menu bar and select the Quit dBASE III PLUS option from the pull-down menu (If the dot prompt is displayed, type the command ASSIST to bring the menu back on to the screen) you can also exit dBASE, by directly typing the command

QUIT ←

at the dot prompt

Quitting dBASE will ensure that all your data are safely stored on disk and return you to the DOS

A >

or

C >

prompt. To run dBASE again in the future, follow the steps we discussed at the beginning of this chapter.

For now, you might want to take a break and review the basic techniques we've discussed in this chapter for creating a database and adding data to it.

Reviewing Database Basics

In this chapter we've discussed the following basic techniques for creating a database, storing, records, and retrieving the data :

1. To start dBASE on a single-user Computer, enter the Command dBASE at the DOS A > or C > prompt, and press Return.
2. To select items from the main menu bar, use the ← and → keys to move the highlighting. (Optionally, you can select items from the main menu bar by typing the first letter of the option).
3. To select items from a pull-down menu, use the and arrows to move the highlight bar, and press return to select the highlighted option.
4. To create a new database, select the "create" and "Database file" options from the menus.
5. To get help on the screen at any time just press the F1 Key. After reading the help message, press any key to resume your work.
6. To work with a particular database, select the "Set Up" and "Database file" options from the menus.

7. To add new data to a database, select the "Update" and "Append" options.
8. To See data stored in a database, select the Retriever and List options from the menus, Then, selected the "Execute the Command" option from the sub menu.
9. To work with dBASE from the dot prompt, press Esc to leave the Assistant Menu. To return to the Assistant menu, enter the Command ASSIST at the dot prompt.
10. To exit and save your work, Select the Set Up and "Quit" dBASE III PLUS options from the menus.

Searching the Database

In this Section we will discuss basic techniques for searching or querying a database. In otherwords we will be looking at techniques which allow you to view records that meet some criteria.

If you 've exited dBASE since the last section be, sure to run dBASE again and use the Set up and "Database file" options from the menu to open the MAIL.DBF database. The examples in this section, assume that the MAIL.DBF database is open.

Building Search Commands with List

The easiest and most common technique for searching a data-base for records that meet some search criteria is with the list option. A simple example will demonstrate. Suppose that you want to retrieve records of all individuals who live in the 92123 Zip Code area. To do so, follow these steps from the Assistant Menu Screen

- ✓ Highlight the Retrieve option on the menu.
- ✓ Select List
- ✓ Select Build a Search condition.

Notice that this menu of field names appears on the screen :

LNAME

FNAME

ADDRESS

CITY

STATE

ZIP

Because you want to view records from the 92123 ZIP code area, you need to search on the ZIP field. So the next step is to select the ZIP field. Next, a menu of operators will appear on the screen :

- = Equal to
- < = Less than or equal to
- < Less than
- > Greater than
- > = Greater than or equal to
- < > Not equal to

you are looking for zip codes that are equal to 92123. So select the "Equal" to option.

Finally, you have to specify what ZIP should be equal to 92123 & dBASE displays the prompt.

Enter a Character String (without quotes) which translates into English as "Enter the thing that you are looking for, and don't use quotation marks". In this example, you are looking for records with Zip codes of 92/23, type

92/23 ↵

At this point, you have built the command that appears on the Action. Line at the bottom of your screen :

Command : **LIST FOR ZIP = '92123'**

you have built this command from your menu selections when dBASE executes this command it will display all records that contain the 92/23 zip code. Before you execute the command, you will need to select the No more conditions option, which tells dBASE that you have completed your request and are now ready to execute the command on the Action Line. The last step now is to select the "Execute the Command" option.

When dBASE asks : The display should be printed, answer No. dBASE will then display records in which the zip code is 92/23 :

1.	Smith	John Q.	123 A St	San Diego	CA	92123
2.	Appleby	Andy	345 Oak St	Los Angeles	CA	92123

Press any key to return to the Assistant Menu Screen

3.7 Building Search Conditions

For example, suppose that you want to view only Los Angeles residents and Furthermore, you only want to see the name of each individual. You need to use the construct a field list

and the Build a Search Condition options from the List sub-menus, as in these steps from the Assistant menu screen :

- ✓ Highlight the Retrieve option
- ✓ Select List
- ✓ Select construct a field list.
- ✓ Select the CITY, LNAME and FNAME fields
- ✓ Press → to leave the field-names menu
- ✓ Select Build a search condition
- ✓ Select CITY
- ✓ Select =
- ✓ Type Los Angeles
- ✓ Press the ←key

At this point, you 've built the command that appears on the Action Line at the bottom of your screen :

Command : LIST CITY, LNAME, FNAME FOR CITY = 'Los Angeles'

To execute this command, follow these steps :

- ✓ Select No more conditions.
- ✓ Select Execute the Command.
- ✓ Enter No in response to printer question.

The result will be a listing of the City, last name, and first name of Los Angeles residents :

2	Los Angeles	Appleby	Andy
6	Los Angeles	Doe	Ruth

Let's try a couple of other searches using operators other than the equal sign. First press any key to return to the menu.

Suppose that you want to view all residents whose zip codes are greater than, or equal to 90000. You'd need to use the > = operator, rather than the = operator.

Here are the steps :

- ✓ Highlight the Retrieve option.
- ✓ Select List.

- ✓ Select construct a field list.
- ✓ Select the ZIP, LNAME and FNAME fields.
- ✓ Press → to leave the field-name menu.
- ✓ Select Build a Search condition.
- ✓ Select ZIP
- ✓ Select > =.
- ✓ Type 90000
- ✓ Press ←
- ✓ Select No more conditions.

At this point, your on-screen command looks like this :

Command : LIST ZIP, LNAME, FNAME FOR = IP > = '90000' when you select the Execute the command option, and respond to the printer question, you will see records with zip codes greater than or equal to 90000 :

1	92123	Smith	John Q
2	92123	Appleby	Andy
3	92122	Smith	Dave
5	92122-1234	Smithsonian	Lucy
6	91234	Doe	Ruth

The less than (<) and less-than / equal-to (<=) operators can be used in a fashion similar to the > and > = operators. For example, to list records that have zip codes that are less than 90000, you would follow these steps :

- ✓ Highlight the Retrieve option.
- ✓ Select List.
- ✓ Select Construct a field list.
- ✓ Select the ZIP, LNAME and FNAME fields.
- ✓ Press → to leave the field-name menu.
- ✓ Select Build a Search Condition.
- ✓ Select STATE
- ✓ Type CA
- ✓ Press ←
- ✓ Select No more conditions.

The command on the Action Line looks like this :

Command : LIST FOR STATE <> "CA"

when the command is executed, the only non california resident will be displayed :

4 SMITH Betsy 222 Lemon Dr. Newyork NY 01234.

Combining Search Conditions

You can combine search conditions with AND and OR options to be more specific about the information you wish to pull out of the database. For example, suppose that you want to view all the smiths in the state of California. In computer argot, you want to view records that have the name Smith in the LNAME field, AND the word CA in the STATE field. To perform such a query, displaying the STATE, LNAME and FNAME fields in the results, follow these steps :

- ✓ Highlight Retrieve
- ✓ Select List
- ✓ Select Construct a field List.
- ✓ Select STATE
- ✓ Select LNAME
- ✓ Select FNAME
- ✓ Press →
- ✓ Select Build a Search Condition
- ✓ Select LNAME
- ✓ Select =
- ✓ Type Smith
- ✓ Press ←

At this point, this command appears at the bottom of your screen :

Command : LIST STATE, LNAME, FNAME FOR LNAME = 'Smith' The screen also displays these options :

No more conditions

Combine with .AND

Combine with .OR

You need to add an AND condition, so continue by following these steps :

- ✓ Select combine with .AND
- ✓ Select STATE
- ✓ Select =
- ✓ Type CA
- ✓ Press ←
- ✓ Select No more Conditions.

Now this command is displayed at the bottom of your screen :

Command LIST STATE, LNAME, FNAME FOR LNAME = 'Smith'

.AND.STATE = 'CA'

In English, this Command that you 've constructed through menu selections means, "show me records that have 'Smith' in the LNAME field, AND CA in the STATE field." When you select the 'Execute the Command' option, you 'll see these results :

Record #	STATE	LNAME	FNAME
1	CA	Smith	John Q
3	CA	Smith	Dave
5	CA	Smith S Onian	Lucy

You 're probably wondering why Smithsonian was included in the list. Well, unless you specify otherwise, 'Smith' is the same as Smithsonian because the first five letters match.

Searching from the Dot Prompt

By now you have probably noticed that as you select menu items, dBASE builds a command with a very specific syntax

< Command > < field list > < search condition >

For example, the command

LIST LNAME, FNAME, CITY FOR CITY = "SanDiego" can be built first by Selecting List from the Retrieve Menu. Then Select LNAME, FNAME and CITY from the select a field List menu option, and select CITY and = from the Build a Search condition sub menus. Eventually, you may find it easier just to key in commands for searches at the dot prompt. Also, you 'll find that the dot prompt offers more flexibility.

Searching with the LIST Command

Let's try searching with dot-prompt commands rather than from menu options. Press the Esc Key from the main menu and you 'll see the dot prompt. Then type the command

LIST LNAME, FNAME, CITY FOR CITY = "San Diego" ↵

You 'll see the San Diego residents displayed on the screen :

Record #	LNAME	FNAME	CITY
1	Smith	John Q	San Diego
3	Smith	Dave	San Diego
5	Smith Sonian	Lucy	San Diego

Now, let's take a look at some other searches typed at the dot prompt, and some of the unexpected results a search in dBASE might produce. Suppose that you type a simple command to view everyone with the last name Smith in the database :

LIST LNAME, FNAME FOR LNAME = 'Smith' ↵

The somewhat surprising result is

Record #	LNAME	FNAME
1	Smith	John Q
3	Smith	Dave
5	Smithsonian	Lucy

There are a couple of problems here. First, Betsy SMITH is missing why? Because Somebody typed her last name as SMITH, and Smith is not the same as SMITH from the computer's point of view.

Second, what is Smithsonian doing in there? you wanted Smiths, not everyone with Smith as the first five letters in their last name. Let's start solving these problems.

You can get rid of Smithsonian easily. Recall that when you created the MAIL.DBF structure, you allowed fifteen spaces for the last name field. So all the smiths are actually

"Smith....." as far as dBASE is concerned you can omit Smithsonian in your display by specifying that only smith, followed by a blankspace be displayed :

LIST LNAME, FNAME FOR LNAME = 'Smith' ↵

Now only the name Smith is listed :

Record #	LNAME	FNAME
1	Smith	John Q
3	Smith	Dave

Smithsonian did n't make it this time, because the first six letters of her last name are Smiths, not smith But you still have to deal with the absence of SMITH.

The uppercase function UPPER () displays all lowercase letters in a character field in uppercase. You can list this out by typing the Command.

```
LIST UPPER (LNAME) ↵
```

to get this list :

Record #	UPPER (LNAME)
1.	SMITH
2.	APPLEBY
3	SMITH
4	SMITH
5	SMITHSONIAN
6	DOE

Here every last name on the list is displayed in uppercase. Now you can get dBASE to list all Smiths, ignoring upper and lowercase by asking it to list all the people whose uppercase equivalent last name is SMITH. In dBASE, that looks like this :

```
LIST FOR UPPER (LNAME) = 'SMITH' ↵
```

In English, this statement reads, "List all the people whose last name, when translated to uppercase, is SMITH."

Now dBASE displays the list you want :

Record #	LNAME	FNAME	ADDRESS	CITY	STATE	ZIP
1.	Smith	John Q	123A St.	SanDiego	CA	92123
3	Smith	Dave	619 Elmst.	San Diego	CA	92122
4	Smith	Betsy	222 Lemon Dr.	Newyork	CA	01234

You really have to spell it out for these machines. They 're so literal. They hardly ever do what you mean; they always do what you say. No imagination! you got rid of Smithsonian by listing for Smith followed by a blank space and you got SMITH in by checking to see if the uppercase equivalent (UPPER) of the last name was SMITH.

3.8 Searching with the LOCATE Command

The LOCATE Command is used for locating the position of a record based upon a desired characteristic. Since LOCATE does not display its data like LIST, you 've to use the

DISPLAY Command along with it to see what dBASE has located. Like the LIST command, you use the FOR statement to indicate the characteristic you wish to find.

Let's assume that you want dBASE to locate information on Dave Smith. You could ask dBASE to search for Dave Smith by last name :

```
LOCATE FOR LNAME = 'Smith' ↵
```

This would give you the dBASE display

Record = 1

This number doesn't do much good, but you can see the contents of the record by typing

```
DISPLAY ↵
```

This record is displayed on the screen

Record #	LNAME	FNAME	ADDRESS	CITY	STATE	ZIP
1.	Smith	JohnQ	123 A St.	San Diego	CA	92123

whoops, this is n't Dave Smith. You can continue the search for Dave with the CONTINUE Command. So type

```
CONTINUE ↵
```

which gives you

Record = 3

If you display this record by typing

```
DISPLAY ↵
```

You have found Dave

Record #	LNAME	FNAME	ADDRESS	CITY	STATE	ZIP
3	Smith	Dave	619Elmst.	San Diego	CA	92122

Not too bad. However, if you had 10,000 names on your mailing list, this process could take a long time. A quicker approach would be to ask for the desired record none specifically. To do this use the AND operator.

```
LOCATE FOR LNAME = 'Smith'. AND FNAME = 'Dave' ↵
```

Now, two statements must be true for LOCATE to find the correct record, That is, the last name must be Smith and the first name must be Dave. The result of this Command is

Record = 3
on the first shot. If you type

DISPLAY ↵

You see the following record

Record #	LNAME	FNAME	ADDRESS	CITY	STATE	ZIP
3	Smith	Dave	619ElmSt.	San Diego	CA	92122

Got it in one try. If you type CONTINUE, you get

End of LOCATE scope

because dBASE has checked all other records and there is not another Dave Smith to be found.

Now, last but certainly not least, we will discuss the type of search where you need to know if a field roughly matches something you are looking for. For example, Suppose you want to search for people living on a street named lemon, no matter what the address number is or whether or not they live on Lemonst., Lemon Ave., or Lemon Blvd. If you LIST or LOCATE FOR ADDRESS = 'Lemon', no match will be found, because the word Lemon is embedded in the middle of the address field (222 Lemon Dr.) You need some way to say, "Display all records that have the word Lemon embedded in the address field." That's quite a mouthful, but not in dBASE, because the \$ function will find the embedded word. So, to locate an individual living on Lemon, use the Command.

LOCATE FOR 'Lemon' \$ ADDRESS ↵

Notice that the grammar is reversed from what you've used before. This is because \$ means embedded in. The Syntax makes sense because the above Command Says, "Find a record with the word Lemon embedded in the address field." when you type the above Command, you see on your screen

Record = 4

If you then type

DISP.LAY ↵

You See

Record #	LNAME	FNAME	ADDRESS	CITY	STATE	ZIP
4	SMITH	Betsy	222 Lemon Dr.	Newyork	CA	01234

Pretty good! It found a person living on Lemon. Keep in Mind that any of the search examples we've shown with the LOCATE Command work as well with the LIST Command, and vice versa. That is you could also LIST FOR 'Lemon' \$ ADDRESS. With this Command, all individuals who live on Lemon would be displayed on the screen.

To continue, you need to execute the command by following, these steps :

- ✓ Select 'No more conditions'.
- ✓ Select 'Execute the Command.'

The Screen displays the message

Record = 6

indicating that dBASE is now 'Pointing' at Record 6. To verify this, select the 'Display' option from the menu (which displays only a single record) by following these steps :

- ✓ Highlight the 'Retrieve' option
- ✓ Select 'Display'
- ✓ Select 'Execute the Command'

You will see Ruth Doe's record appear on the screen, along with the instructions to press any key to return to the menu.

3.9 Exact Searching

Recall that earlier in the Section, a search for people with the last name Smith displayed Smithsonian as well as the Smiths. You were able to leave Smithsonian out by performing a search for Smith (with a blankspace after the h). Another way to accomplish the same goal is to change the way in which dBASE makes search comparisons. To do so, press Esc until the menu disappears and the dot prompt appears. Then type this Command :

```
SET EXACT ON ↵
```

If you now enter the command

```
LIST FOR LNAME = "SMITH" ↵
```

or the Command

```
LIST FOR UPPER (LNAME) = "SMITH" ↵
```

You will see the Smiths, but not Smithsonian, because Smithsonian is not an exact match to Smith,

To return to the original EXACT setting, enter the command

SET EXACT OFF ↵
at the dot prompt.

To check the status of the EXACT parameter at anytime just enter the Command

SET ↵

at the dot prompt you will see a whole menu of SET Parameters as in Fig

Options	Screen	Keys	Disk	Files	Margin	Decimals	04:20:54 am
Alternate	Off						
Bell	ON						
Carry	OFF						
Catalog							
Century	OFF						
Config	OFF						
Deleted	OFF						
Delimiters	OFF						
Device	SCREEN						
Dohistory	OFF						
Escape	ON						
Exact	OFF						
Fields	OFF						
Fixed	OFF						
Heading	ON						
Help	ON						
History	ON						
Intensity	ON						

SET MAIL OPT : 1825

Position selection bar ↑↓ Change - ↵ Leave Menu ←→ Exist Esc

Specify the dBASE III environment options

Menu of Set Parameters

You can also combine search conditions to your heart's content. For example, if you want a listing of everyone who lives on either Elm or Oak streets, you could

LIST FOR 'Elm' \$ADDRESS .OR. 'Oak' \$ADDRESS ↵

which would give you

Record #	LNAME	FNAME	ADDRESS	CITY	STATE	ZIP
2	Appleby	Andy	345 Oak St.	Los Angeles	CA	92123
3	Smith	Dave	619 Elm St.	San Diego	CA	92122

The result is a listing of the individuals who live on either Oak or Elm Streets. Be careful to distinguish 'and' and 'or'. If you had asked for a LIST FOR 'Elm' \$ADDRESS .AND. 'Oak' \$ADDRESS, you would end up with nothing, because a given individual in your data base can't possibly live on both Elm and Oak streets at the same time (unless he did happen to live at 3421 Oak Elm St.)

In Summary, the OR.Operator requires that only one of the conditions has to be true to get a listing. The.AND.command requires that both search conditions be true. For instance, the command

```
LIST FOR 'Elm' $ADDRESS.AND.'San Diego' $CITY ←
```

would tell dBASE to display all of the individuals who live on Elm St. 'and' in San Diego (San Diego residents who live on Elm). The Command

```
LIST FOR 'Elm' $ADDRESS .OR. 'San Diego' $CITY ←
```

would display all individuals living on Elm St., regardless of what city, and all people living in San Diego, regardless of what street. The .OR.Command generally broadens a Search, Since only one condition out of two must be met for dBASE to bring the data to the screen. On the other hand, the .AND.function narrows the search since both search conditions must be met to find the correct data.

We can combine .AND. and .OR. Search conditions. Try the following command:

```
LIST FOR STATE = 'CA' .AND. ('Oak' $ ADDRESS.OR.; 'Elm' $ADDRESS) ←
```

Note : Command lines that are too wide to fit on the page in this book are broken with a semi color (;) you must type the command as one long line without the semicolon.

This command would first require that the individual live in california. Furthermore, the individual must live on either oak or elm to make it to the list. In otherwords, this command lists all california residents who live on either Oak or Elm.

Incidentally a LOCATE command can be built from the menu just as a LIST command can. For example, to view data for Ruth Doe, you would first follow these steps from the Assistant menu screen to create the LOCATE command line :

✓ Highlight the 'Position' option

✓ Select 'Locate'

✓ Select 'Build a Search Condition'

✓ Select LNAME

✓ Select =

✓ Type Doe ←

✓ Select 'Combine with .AND.'

✓ Select FNAME

✓ Select =

✓ Type Ruth ←

At this point, your menu selections have built this command displayed on the Action Line:

Command : LOCATE FOR LNAME = 'Doe'.AND.FNAME = 'Ruth'

You'll see the Exact option in its alphabetical position and either ON or OFF next to it. You can change the EXACT parameter while this menu is displayed by using ↑ and ↓ to move the highlighting to the 'Exact' option and pressing the Return key to change settings on the highlighted parameter. Press Esc to return to the dot prompt. Type

ASSIST ←

to return to the Assistant menu screen.

If you have a color monitor, you can use the screen option on this menu to change the colors displayed on your screen.

Reviewing Search Techniques

Take a moment to review these topics :

1. To search, or query, a database means to view records that match some criterion.
2. To search for records that meet some criterion, select the 'Build a search condition option from the List sub menu.
3. You can use the = (equal), < (less than), < = (less than or equal), > greater than, > = (greater than equal), and < > (not equal) operators to perform your searches from the menu.
4. To display only certain fields in a search, select the construct a field 'list' option from the sub menu.
5. When performing searches from the dot prompt with the LIST and LOCATE Command, you can also use the # (embedded in) operator, and the UPPER function to manage upper and lower case.
6. The SET EXACT Command allows you to change the way in which searches are performed.

3.10 Sorting the database

In most situations you add new records to a database as they become available. The, at some point, you need to rearrange the records in to some meaningful order. Such as by Zip code for bulk mailing or by last name for a directory. dBASE provides two options for sorting databases, 'Sort' and Index. We'll discuss each option in this section.

Sorting with the Sort Option

When you use the sort option to sort a database file, dBASE requires that you create a new database to store the sorted records. Once the sorting is done, you can 'copy' the contents of the sorted database back into the original file. Let's try an alphabetical sort with the MAIL file. When you sort, store the newly arranged data to a file called TEMP (for temporary). After dBASE sorts the data, you will copy the contents of TEMP back over to MAIL, so that the records in MAIL will be properly sorted.

First, make sure that the MAIL.DBF database is in use (The database name appears on the status bar). If not use the Set Up and database file options from the Assistant menu screen to open your database. Next, take a quick look at the current contents of the database by highlighting the 'Retrieve' option and selecting the List and 'Execute the Command' options. You 'll see the contents of the MAIL.DBF database.

1.	Smith	JohnQ	123 Ast	San Diego	CA	92123
2	Appleby	Andy	345 Oak St	Los Angeles	CA	92123
3	Smith	Dave	619 Elmst	San Diego	CA	92122
4	SMITH	Betsy	222 Lemon Dr.	Newyork	NY	01234
5	Smithsonian	Lucy	461 AdamsSt	San Diego	CA	92122-1234
6	Doe	Ruth	1142J St	Los Angeles	CA	91234

Now let's say that your database is not indexed and you want to put these records into alphabetical order by last name. To do so, follow these steps from the Assistant Menu Screen:

- ✓ Highlight the 'organize' option
- ✓ Select 'Sort'
- ✓ Select LNAME
- ✓ Leave the Submenu by pressing →
- ✓ Select a drive for the sorted file
- ✓ Type TEMP for the sorted file name.

After a few seconds, the screen displays this message

100% sorted 6 records sorted

Next highlight the 'Retrieve' option, and select the list and 'Execute the Command' options to view the database again. You 'll see these records on the screen along with a message to press any key to continue.

1	Smith	John Q	123A St	San Diego	CA	92123
2	Appleby	Andy	345 Oak St	Los Angeles	CA	92123
3	Smith	Dave	619 Elm St	San Diego	CA	92122
4	SMITH	Betsy	222 Lemon Dr.	Newyork	NY	01234
5	Smithsonian	Lucy	461 Adams St	San Diego	CA	92122-1234
6	Doe	Ruth	1142 J St	Los Angeles	CA	91234

Whoops! It doesn't look like anything happened here. That's because you 're looking at the MAIL.DBF database, but it is TEMP.DBF that actually contains the sorted records. To verify this, use the 'Set Up' and 'Database file' options from the menu to open TEMP.DBF. Then, use the 'Retrieve', 'List' and 'Execute the Command' options to view the contents of the database. You'll see that the records are indeed in alphabetical order

1	Appleby	Andy	345 Oak St	Los Angeles	CA	92123
2	Doe	Ruth	1142 J St.	Los Angeles	CA	91234
3	SMITH	Betsy	222 Lemon Dr.	Newyork	NY	01234
4	Smith	John Q	123 A St	San Diego	CA	92123
5	Smith	Dave	619 Elm St	San Diego	CA	92122
6	Smithsonian	Lucy	461 Adams St	San Diego	CA	92122-1234

So TEMP.DBF has the sorted records on it. But the MAIL database is still in random order. How do you act the sorted contents of TEMP into MAIL? Simple. Since the TEMP DBF database is open, you can just copy it to the MAIL.DBF database, overwriting the existing contents of MAIL.DBF. To do so, follow these steps :

- ✓ Highlighting the 'organize' option
- ✓ Select 'copy'
- ✓ Select a drive
- ✓ Type MAIL as the name of the database to copy to
- ✓ Select 'Execute the Command'

To make sure that you don't accidentally over write an important file d BASE III PLUS will double check before copying the file. You'll see this message :

MAIL.DBF already exists, overwrite it? (Y/N) If you answer yes, the existing contents of the mail database will be replaced by the data in TEMP.DBF.

Answer yes by typing 'Y'.

Now you need to open the MAIL.DBF database and see what's in it. As usual, select the 'setup' and 'Database file' options to open MAIL.DBF. Then use the 'Retrieve', 'List' and 'Execute the Command' options to view the contents of the database. You'll see the records displayed like this :

1	Apple by	Andy	345 Oak St	Los Angeles	CA	92123
2	Doe	Ruth	1142 J St.	Los Angeles	LA	91234
3	SMITH	Betsy	222 Lemon Dr.	Newyork	NY	01234
4	Smith	John Q	123 A St	San Diego	CA	92123
5	Smith	Dave	619 Elm St	San Diego	CA	92122
6	Smithsonian	Lucy	461 Adams St	San Diego	CA	92122-1234

Everything is in alphabetical order now.

Let's illustrate what took place on the disk with some pictures. To start with the disk had a database called MAIL.DBF on it, with the records in random order. (Fig. Shows the contents of the disk.

dBASE requires that when you 'sort' the database you must sort to another file. In this example, you sorted a database called TEMP.DBF. After the sort was complete, you had two databases on the disk MAIL.DBF, still in random order and TEMP.DBF which had the same contents as MAIL.DBF, but in sorted order, as in Figure.

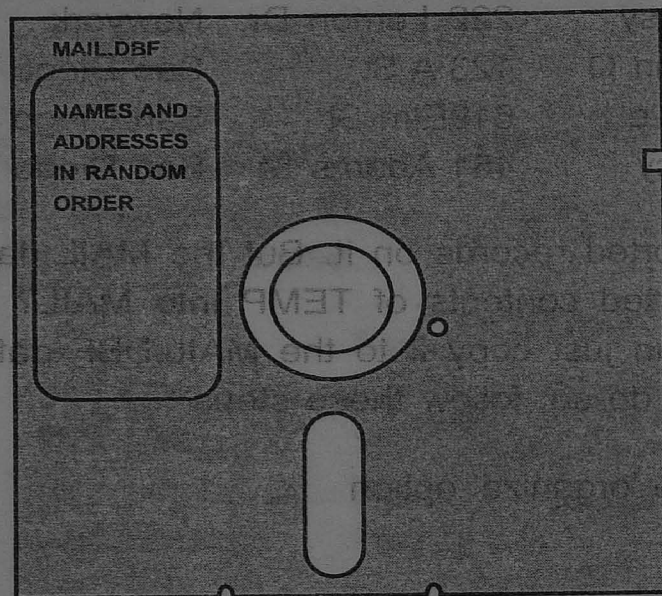


Figure : Disk Contents before Sorting

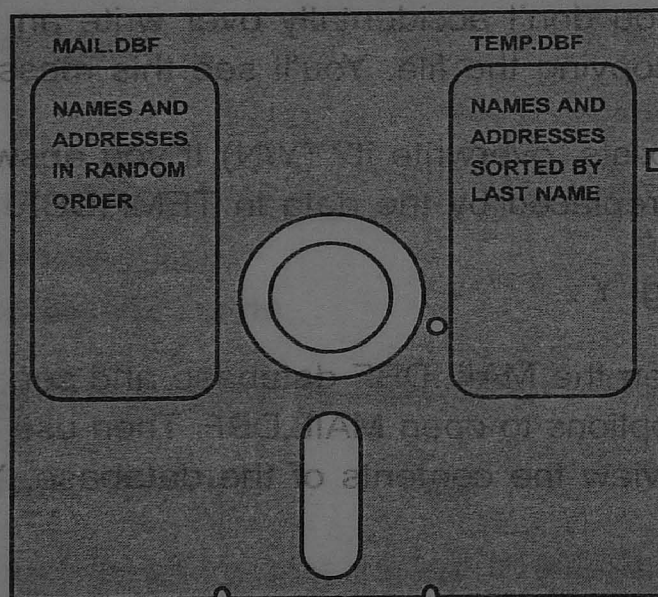


Figure : Disk Contents after Sorting to TEMP

To see the names and addresses in sorted order, you needed to open TEMP, then LIST the records. However, you wanted the data on MAIL to be sorted too, so you used the COPY command to copy the sorted contents of TEMP over to MAIL. After the copy was complete, you had two identical databases, as shown in Figure 4.3.

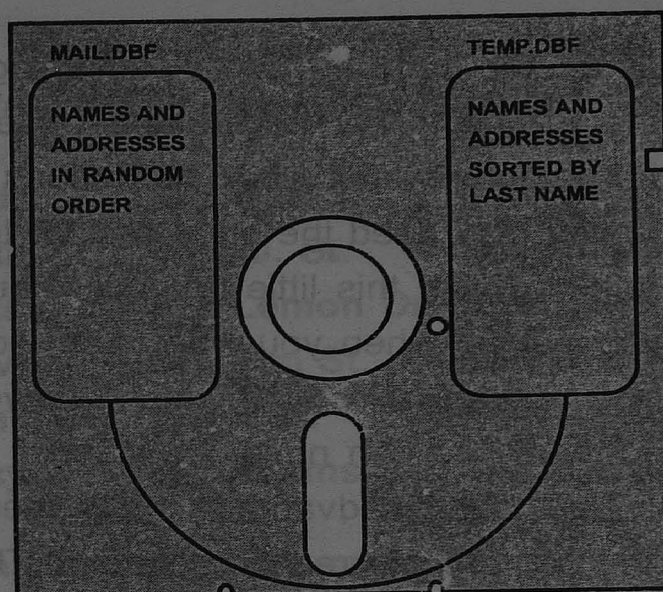


Figure 4.3: Copying a Sorted TEMP File

You just needed TEMP to temporarily hold the sorted records. We'll discuss techniques for deleting unnecessary data file in the next chapter.

Now suppose that you want to do a bulk mailing, and you need these records in zip code order. What do you do? I bet you can guess.

Select the *Organize* and *Sort* options from the menus. When the submenu of field names appears on the screen, use ↓ or ↑ to high light the field name **ZIP**, and press **Return** to select it. Press → to leave the submenu, select a drive from the next menu, and then type **TEMP** for a file name. Before overwriting the TEMP file with the new sorted file, dBASE will once again double check and ask for permission before proceeding with the copy. Answer **Y** to this prompt.

Now you can just quickly copy the contents of the TEMP.DBF file over to the MAIL.DBF file. To do so, open the TEMP.DBF database by selecting the *Set Up* and *Database* file options from the menus.

Next, select *Organize* and *Copy* from the menus to begin copying. Select a drive, enter MAIL as the database to copy to, and select *Execute the Command*. Once again dBASE will double check before overwriting the MAIL.DBF database. Just type **Y** in response.

Finally, use the *Set Up* and *Database* file options to open the MAIL.DBF database. Select the *Retrieve, List* and *Execute the command* options to view the records. You'll see that they are now in order by zip code:

1	SMITH	Betsy	222 Lemon Dr.	New York	NY	01234
2	Doe	Ruth	1142 J. St.	Los Angeles	CA	91234
3	Smith	Dava	619 Elm St.	San Diego	CA	92122
4	Smithsonlan	Lucy	461 Adams St.	San Diego	CA	92122-1234
5	Appleby	Andy	345 Oak St.	Los Angeles	CA	92123
6	Smith	John Q.	123 A St.	San Diego	CA	92123

Sorting in this fashion is useful, but there are disadvantages. First, sorting wastes disk space. Since dBASE does the sorting to another database file, you need at least as much empty space on a disk as the database itself fills. That means you can only use half a disk for your entire database since you may need the other half for the TEMP file. Second, sorting is quite slow. You may not think so with this little database, but you would if you had 5000 records. This can be especially painful when you want them sorted by name for a directory, sorted by zip code for mailings, and so forth. Also, since everyone's record number changes as the records become rearranged, you can never be sure of an individual's record number using the SORT command. This last disadvantage may seem trivial now, but with large databases, it's nice to have record numbers remain constant. The *Index* option is the solution to these problems.

Sorting with the *Index* Option

The *Index* option provides you with a much quicker and more efficient method of sorting records than does the Sort option.

Creating an Index File

Let's give it a whirl by putting the MAIL.DBF back into alphabetical order by name.

Highlight the *Organize* option from the main menu bar, and select the *Index* option from the submenu. dBASE will display these somewhat cryptic instructions:

The Index key can be any character, numeric, or data expression involving one or more fields in the database file. It is usually a single field. Enter an Index key expression:

For now, simply type the name of the field you want to sort. In this example, type

LNAME ↵

Next, you need to select a drive for storing the index file. dBASE then displays the message:

Enter a file name (consisting of up to 8 letters or digits) followed by a period and a file name extension (consisting of up to 3 letters or digits.)

Names for index files follow the same rules as database files; they can be up to eight characters long but may not contain spaces or punctuation. dBASE will automatically assign the extension.NDX to the file name you provide. For this example, type the file name

NAMES ↵

Now, to verify that the records are indeed back in alphabetical order, select the *Retrieve, List, and Execute the Command* options once again. You'll see these results:

5	Appleby	Andy	345 Oak St.	Los Angeles	CA	92123
2	Doe	Ruth	1142 J. St.	Los Angeles	CA	91234
1	SMITH	Betsy	222 Lemon Dr.	New York	NY	01234
3	Smith	Dave	619 Elm St.	San Diego	CA	92122
6	Smith	John Q.	123 A St.	San Diego	CA	92123
4	Smithsonian	Lucy	461 Adams St.	San Diego	CA	92122-1234

The records have changed to proper order, but the record numbers have remained the same. This is helpful, because that means Andy Appleby is still Record 5, Ruth Doe is still Record 2, etc. Also, you didn't have to go through the *Copy* rigamarole to see the records listed in proper order.

In the command you created, you asked dBASE to INDEX to NAMES. Isn't NAMES a data file? Yes, but it is not a database. It is a special file, an *index file* named NAMES.NDX. Its contents look very much like an index in a book. A book's index has a list of keywords in alphabetical order, and page numbers where the keywords appear in text. Your database index has a list of last names in alphabetical order, and the record numbers where they appear in the database, like so:

Appleby	5
Doe	2
SMITH	1
Smith	3
Smith	6
Smithsonian	4

After the index is complete, when you view the records, dBASE automatically uses information from the index file to determine the proper order to display the records in MAIL.DBF. The records in the MAIL.DBF database are still in their original order; the index file, however, tells dBASE the correct order in which to display the records.

After you do an index, the MAIL.DBF and NAME.NDX files exist on the disk, as in Figure 4.4.

Let's try another example. This time you'll index on the ZIP field. First, highlight the Organize option and select Index. Then, enter Zip as the field to index, and select a drive for the index file.

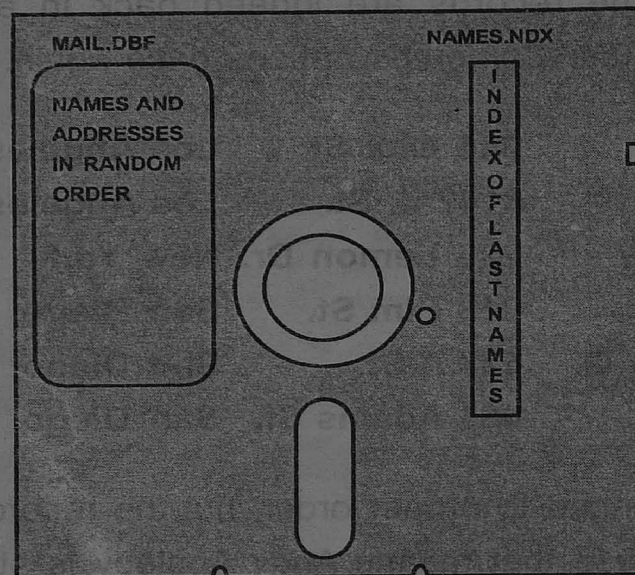


Figure 4.4 : Disk Contents with an Index File

Type ZIP as the file name. When the indexing is done, select the Retrieve and List options. Then use the Construct a field list option to place the zip codes in the leftmost column to better show the new order. Select the Execute option to view the records in the new sorted order:

Record#	ZIP	LNAME	FNAME	ADDRESS
1	01234	SMITH	Betsy	222 Lemon Dr.
2.	91234	Doe	Ruth	1142 J. St.
3	92122	Smith	Dave	619 Elm St.
4	92122-1234	Smithsonian	Lucy	461 Adams St.
5	92123	Appleby	Andy	345 Oak St.
6	92123	Smith	John Q.	123 A St.

The records are now displayed in zip code order. Furthermore you have a file called ZIP.NDX which tells dBASE the order in which to display the records, as in Figure 4.5.

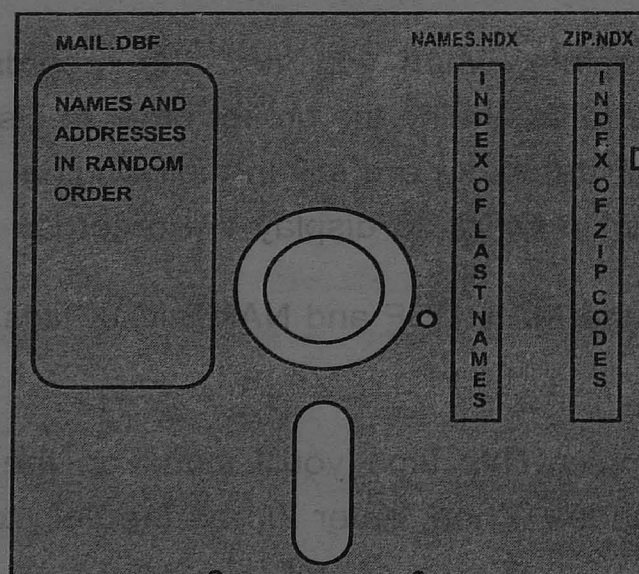


Figure 4.5 : Disk Contents with Two Index Files

If you want a quick view of your mailing list sorted alphabetically by name, you don't have to sort it again. Just use the Set Up option from the Assistant menu screen to open the database with the appropriate index file. Here's how:

- ✓ Highlight Set Up from the main menu bar.
- ✓ Select Database File.
- ✓ Specify a drive.
- ✓ Select MAIL.DBF.
- ✓ Answer Y to the "Is file indexed?" prompt.

At this point, a menu of all existing index file names will appear on the screen. As with most menu options, you can simply use ↑ and ↓ to move the highlighting through the options, and press Return to select items. The order in which you select items is important, though.

Because you want to see names in alphabetical order in this example, you need to select the NAMES.NDX index first. Then, you can select other index files that were created for this database, in any order you wish. (dBASE allows a maximum of seven index files per database.) We'll discuss reasons for selecting multiple index files in a moment.

When you select the NAMES.NDX and ZIP.NDX index files, you'll notice that the NAMES.NDX is called the Master index file:

NAMES.NDX	Master
TEMP.NDX	
ZIP.NDX	02

When you finish selecting index files, press – to leave the submenu. If you then select Retrieve, List, and Execute from the menus, you'll see the names in alphabetical order:

Record#	LNAME	FNAME	ADDRESS	CITY
5	Appleby	Andy	345 Oak St.	Los Angeles
2	Doe	Ruth	1142 J. St.	Los Angeles
1	SMITH	Betsy	222 Lemon Dr.	New York
3	Smith	Dave	619 Elm St.	San Diego
6	Smith	John Q.	123 A St.	San Diego
4	Smithsonian	Lucy	461 Adams St.	San Diego

When you want a quick look at your mailing list in zip code order, you don't have to re-sort anything because you've already indexed on ZIP.NDX still exists. Just use the Set Up and Database file options again to open the MAIL.DBF database. Answer Yes to the "Is the file indexed?" prompt. This time, when you select index files, be sure to select ZIP.NDX first, and NAMES.NDX second, so the screen looks like this:

NAMES.NDX	02
TEMP.NDX	
ZIP.NDX	Master

When you select the Retrieve and List options to view the database, you'll see that the records are back in zip code order:

Record#	ZIP	LNAME	FNAME	ADDRESS
1	01234	SMITH	Betsy	222 Lemon Dr.
2	91234	Doe	Ruth	1142 J. St.
3	92122	Smith	Dave	619 Elm St.
4	92122-1234	Smithsonian	Lucy	461 Adams St.
5	92123	Appleby	Andy	345 Oak St.
6	92123	Smith	John Q.	123 A St.

Don't Resort to Re-sorting

Now, you are probably wondering why I keep selecting more than one index file, when only the Master (first selected) index file affects the sorted order of the database. The reason for doing so is to make the index files active.

Once an index file is active, it is automatically updated anytime the database is altered. That means that as you add, change, or delete records in the MAIL.DBF database, both the NAMES.NDX and ZIP.NDX index files are also updated accordingly and instantly re-sorted! Therefore, once you have created index files, you need never use the Sort or Index options from the Organize menu again. With a large database, this will save you a great deal of time because you won't need to use the Index or Sort options to reorganize the database each time you add or change information.

You must remember to make the index files active before you add or change data on the database. If you don't, the index files will become corrupted. That is, they'll need to be recreated. If you only make one index file active before adding, changing, or deleting data, then the active index file will still be correct, but the unlisted index files will be corrupted.

You know when an index file has been corrupted because either

1. dBASE does not display newly added records when you enter the List option or command.
2. dBASE will display the error message "Record out of range" when you attempt to list the records.

In either case, the index files will have to be recreated from scratch. You'll have to select the Organize and Index options from the menus again, and create the index file just as though you were doing so for the first time.

dBASE III PLUS allows you to have up to seven index files active. You'll may want to wait until you have more experience before you try managing that many index files, but you should practice working with index files and experiment a bit.

Don't worry about damaging the database; it's impossible to do so by experimenting with index files.

Sorts within sorts

Although your MAIL.DBF database is too small to demonstrate the point, sometimes a single field is not sufficient for organizing a database. For example, suppose that there were 10,000 names on the MAIL.DBF database, and 100 people had the last name Smith. If you indexed on the LNAME field only, the Smiths might end up in a somewhat haphazard order:

Smith Rudolph
Smith Abigail
Smith Terri
Smith Barbara
Smith Prudence
Smith Zeke
Smith Oscar

This order won't do you much good when you're attempting to find a particular Smith.

To remedy this situation, you need to sort on more than one field (or in other words, do a sort-within-a-sort). If you indexed this hypothetical database on both the LNAME and FNAME fields, your Smiths would come out in alphabetical order:

Smith Abigail
Smith Barbara
Smith Oscar
Smith Prudence
Smith Rudolph
Smith Terri
Smith Zeke

To index on multiple fields like this, you merely need to Concatenate (stick together) the fields in the index expression with a plus (+) sign. Let's give it a whirl. Select the Organize and Index options.

When dBASE displays instructions

The index key can be any character, numeric, or data expression involving one or more fields in the database file. It is usually a single field. Enter an index key expression:

type this expression:

LNAME + FNAME ↵

Select a drive. Then enter a file name (BOTH).

When the indexing is complete, use the Retrieve, List and Execute the command options to view the records. You'll see the records in alphabetical order by last name and by first name within identical last names:

5	Appleby	Andy	345 Oak St.	Los Angeles	CA	92123
2	Doe	Ruth	1142 J. St.	Los Angeles	CA	91234
1	SMITH	Betsy	222 Lemon Dr.	New York	NY	01234
3	Smith	Dave	619 Elm St.	San Diego	CA	92122
6	Smith	John Q.	123 A St.	San Diego	CA	92123
4	Smithsonian	Lucy	461 Adams St.	San Diego	CA	92122-1234

Suppose now that you want all the records sorted by zip code, but within each zip code area, you want them sorted by last name. Furthermore, just in case there are several Smiths in the 92123 zip code area, you want the names sorted by first name within common last names. This would be simple enough to accomplish. You would just select the Organize and Index options once again, and enter this Index expression:

ZIP + LNAME + FNAME

Notice the order of the entries. Since ZIP is listed first, the records will be sorted into zip code mailing. However, within common zip codes, records will be sorted by last and first name, as in this small sample:

92111	Zeepers	Zach
92122	Adams	Archie
92122	Baker	Anne
92122	Miller	Arnie
92122	Miller	Millie
92122	Miller	Pia
92122	Muller	Wanda
92122	Peterson	Wayne
92122	Wilson	Arthur
92123	Ascii	Antoine

Indexing from the Dot Prompt

You can sort or index a database file using commands from the dot prompt. As we discussed earlier, the USE command opens a database file from the dot prompt. If you are using a computer with two floppy-drives, you can store the database and index files on Drive B simply by making B the default drive. To do so, enter the command:


```
SET DEFAULT TO B ↵
```

Now to open MAIL.DBF enter this command:

```
USE MAIL ↵
```

To create an index file from the dot prompt, you use the INDEX ON command, along with the field(s) to be used in the index file, and the name of the index file. For example, to create the BOTH.NDX index file of the last and first name fields, you would enter this command:

```
INDEX ON LNAME + FNAME TO BOTH ↵
```

To view the results of the index, enter this command:

```
LIST ↵
```

Type the field names next to the LIST command to view specific fields using the LIST command. For example, to view the LNAME and FNAME fields only, enter this command:

```
LIST LNAME, FNAME ↵
```

To use existing index files from the dot prompt, you can use the INDEX command directly with the USE command. For example, to open the MAIL.DBF database, with the NAMES.NDX index files as the Master, ZIP.NDX index files as secondary (02), you would enter the command:

```
USE MAIL INDEX NAMES, ZIP ↵
```

You can use the SET INDEX command as well. For example, to switch to zip code order, but still keep the NAMES.NDX index file active, you could simply enter this command:

```
SET INDEX TO ZIP,NAMES ↵
```

From this point on, records are displayed in order by zip code, but both ZIP.NDX and NAMES.NDX are updated automatically should any data be added to, deleted from, or changed in the MAIL.DBF database.

Searching for records with an Index File

Index files are certainly useful for creating and maintaining sort orders. , as we've seen in this chapter. Index files can also speed up searches for information on very large databases. For example, if you used LIST or LOCATE to look up information in a very large database, you might have to wait a couple of minutes to get the results. With an index file and the FIND option, the same search would probably take only a second or two.

Even though your MAIL.DBF database is 'co small to demonstrate the full speed of searching with an index file, you can use it to get some practice with genera' techniques. Suppose that you want to find Doe in the MAIL.DBF database using an index file. First of all,

you'll have to make sure that NAMES.NDX is the Master index, because searches only work with the Master index. Hence, from the dot prompt, enter this command:

```
USE MAIL INDEX NAMES, ZIP ↵
```

Next, to quickly locate Doe, just enter this command:

```
FIND Doe ↵
```

Enter this command to verify that the correct record was located:

```
DISPLAY ↵
```

To locate a particular zip code, first enter the command

```
USE MAIL INDEX ZIP, NAMES
```

and then the command

```
FIND 92122
```

Again, entering the command DISPLAY will show the record found, if any.

From the menus, you can assign indexes in the usual fashion through the Set Up and Database file options. Then, to look up data in the Master index, highlight the Position option on the menu and select Seek. dBASE will ask that you enter an expression. You need to type the item that you are looking for, enclosed in quotation marks (if it is the Character data type). For this example, type

```
"Doe" ↵
```

dBASE will give you the message to press any key to return to the menu. After doing so, highlight the Retrieve option, and select the Display and Execute the command options. This will display the single record located by the Seek option.

Reviewing Database Organization

In this chapter we've discussed numerous techniques and commands for organizing a database.

dBASE provides two options for organizing a database into some meaningful order, Sort and Index.

To Sort a database, highlight the Organize option and select Sort. You'll need to assign a file name for the sorted records, and then use the Copy option to copy the sorted records back to the original file.

To Index a database, highlight the Organize option and select the Index option. Enter names for the index file and fields to index.

The Index option is much faster and more efficient than the Sort option, and it allows you to define and maintain sort orders automatically.

For sorts-within-sorts, you can simply string together several fields in an Index expression, using a plus sign

ZIP + LNAME + FNAME

When several index files are open, the first-selected (Master) index file determines the sort order for displays of data, but any other active index files are also updated instantly and automatically.

Another advantage of index files is quick lookups with the Find or Seek options. Only the Master index file can be used in a search, though.

4.1 : Editing and Modifying Data Bases

When working with computers, the term edit means to change existing data on the database. For instance, if a certain individual who is already on our database moves to a new house, you would want to change his street address. That would be a database edit. Suppose you decide that you want to include phone numbers for each individual on our database, even though you did not originally designate a field for storing phone numbers. You would have to modify the structure of the database. The options you can use to perform such feats are discussed in this chapter.

Editing with Edit

Computer database need editing for a variety of reasons. People move and change their addresses, we make mistakes while entering data and have to fix them, and so forth. Editing with dBASE is a rather simple task if you know the number of the particular record you are looking for. You can't possibly remember all those numbers, but you can use the knowledge you've gained thus far to look up a record number quickly.

Getting Ready

Before you actually begin editing a database, make sure that the database is open, and that any index files that are used regularly with the database are also open. The index files must be open to be automatically updated as information is changed in the database.

In this example, select the Set Up and Database file options, and open the MAIL.DBF database. When dBASE asks if the file is indexed, answer Yes. Select NAMES.NDX and ZIP.NDX index files, so these will remain active throughout the exercises in this chapter. Press → to remove the index file menu.

Changing a Record

Now, let's suppose that Dave Smith moves, and you want to update his record with his new address. If this were a large database, your first task would be to find his record in the database file. There are a number of ways that you can do so, but the easiest in this example would be to use the Locate option, as in these steps from the Assistant menu screen:

- ✓ Highlight the Position option on the main menu bar.
- ✓ Select Locate.
- ✓ Select Build a search condition.
- ✓ Select LNAME.
- ✓ Select =
- ✓ Type Smith ↵
- ✓ Select Combine with .AND.
- ✓ Select FNAME
- ✓ Select =
- ✓ Type Dave ↵
- ✓ Select No more conditions.

Notice that your menu selections have created a command on the Action Line:

Command: LOCATE FOR LNAME = 'Smith' .AND. FNAME = 'Dave'

which tells dBASE, "Find the first record that has 'Smith' in the LNAME field, AND 'Dave' in the FNAME field." Select Execute the command.

Now, to edit this record, move the highlight to the Update option and select Edit. You'll see Dave Smith's data on the screen ready for editing, as in Figure 4.1.

Notice that the cursor is under the S in Smith. Now we can use the arrow keys and cursor-control keys to position the cursor to make changes. (Remember that the ^ symbol means, "Hold down the Ctrl key while pressing the next key.") Cursor commands for the EDIT mode are explained in Table 4.1.

CURSOR <-- --> Char : <-- --> Word : Home End	UP DOWN Field : ↑ ↓ Page : PgUp PgDn Help : F1	DELETE Char : Del Field : ^Y Record : ^U	Insert Mode : Ins Exit/Save : ^End Abort : Esc Memo : ^Home
--	--	--	---

LNAME	
FNAME	
ADDRESS	
CITY	
STATE	
ZIP	

ASSIST	<C:>	MAIL	Rec : None	Ins
--------	------	------	------------	-----

Figure 4.1 : Record Displayed in EDIT Mode

To change Dave's address, press the ↓ key twice to move down two lines. This positions the cursor to the beginning of the ADDRESS field. Press ^ Y to empty out the current address field, and type the new address, 123 B St. Press the ↓ key, then the ← key a few times to move the cursor to the beginning of the CITY field. Press ^ Y to erase the current city and type Los Angeles as the new city. Press the ↓ key twice, then the ← key a few times to move to the beginning of the ZIP field. Press ^ Y to delete the current zip code and type the new zip code, 90123. Now Dave Smith's data looks like this:

LNAME	:	Smith	:
FNAME	:	Dave	:
ADDRESS	:	213 B. St.	:
CITY	:	Los Angeles	:
STATE	:	CA	:
ZIP	:	90123_	:

At this point, check to see if the data looks ok. If it does, save the new data by pressing a ^W or ^End. This will bring back the Assistant menu.

Key	Alternate Key	Effect
↑	or ^E	Moves cursor up one line.
↓	or ^X	Moves cursor down one line.
←	or ^S	Moves cursor left one space.
→	or ^D	Moves cursor right one space.
Backspace		Moves cursor left and erases.
Del	or ^G	Deletes character over cursor.
^T		Erases one word to the right.
^Y		Erases all field contents to the right of the cursor.
^U		Deletes entire record.
Ins	or ^V	Turns Insert mode on/off.
PgUp	or ^R	Moves back one record.
PgDn	or ^C	Moves forward one record.
^End	or ^W	Saves changes and returns to Assistant or dot prompt.
Esc	or ^Q	Abandons changes and returns to Assistant or dot prompt.

Table 4.1: EDIT-Mode Control Keys.

If you practice using the EDIT mode, you will find it easy to use. It is a straightforward procedure; there is nothing particularly tricky about it. At this point, you can try editing a few records of your own.

Editing with Browse

The Browse option allows you to scroll through the database, horizontally and vertically, to edit or add records. As you pan, dBASE shows as much data as will fit on the screen. You can move the cursor to change whatever information you please. This is a very useful technique for locating easily-corrected errors, like misspellings.

You can use the Locate option to pinpoint a particular record to edit, as you did in the last example. But in this case, since the MAIL.DBF database is so small, you can just start editing from the top of the database.

To start at the top of the database with the first record, highlight the Position option on the main menu bar, and select the Goto Record and Top options. To enter the BROWSE mode now, highlight the Update option and select the Browse option. You'll see as much information as will fit on the screen, as shown in Figure 4.2.

CURSOR <-- -->	UP DOWN	DELETE	Insert Mode : Ins
Char : <-- -->	Field : 1	Char : Del	Exit/Save : ^End
Word : Home End	Page : PgUp Pg Dn	Field : ^Y	Abort : Esc
Pan : ^1 ^1	Help : F1	Record : ^U	Memo : ^Home

LNAME-----	FNAME-----	ADDRESS-----	CITY-----	STATE
Appleby	Andy	145 Oak St.	Los Angeles	CA
Doe	Ruth	1142 J. St.	Los Angeles	CA
SMITH	Betsy	222 Lemon Dr.	New York	MY
Smith	Dave	123 B St.	Los Angeles	CA
Smith	John Q.	123 A St.	San Diego	CA
Smithsonian	Lucy	461 Adams St.	San Diego	CA

BROWSE	<C:>	MAIL	Rec : 5/6	Ins
--------	------	------	-----------	-----

View and edit fields

Figure 4.2: Records on a BROWSE Screen

You can use arrow keys to move the large highlighting bar and the cursor within this bar. Then make whatever changes you want. The menu at the top of the screen summarizes the command keys. The control-key commands for the BROWSE mode are explained in more detail in Table 4.2.

Key	Alternate Key	Effect
↑	or ^E	Moves cursor up one line.
↓	or ^X	Moves cursor down one line.
←	or ^S	Moves cursor one space to the left.
→	or ^D	Moves cursor one space to the right.
Home	or ^A	Moves cursor one field to the left.
End	or ^F	Moves cursor one field to the right.
^→	or ^B	Pans one field to the right.
^←	or ^Z	Pans one field to the left.
Del	or ^G	Deletes character over cursor.
^Y		Deletes information to the right of the cursor in the field.
Ins	or ^V	Enters INSert mode, so that newly entered data is inserted into the field without overwriting existing data.
^Home		Displays/erases help menu on the top of the BROWSE screen.
^End	or ^W	Saves all newly edited data and returns to dot prompt.
Esc	or ^Q	Returns to dot prompt without saving changes made in the BROWSE mode.

Table 4.2: BROWSE-Mode Control Keys

To change the information of a field or record, just position the cursor where you want to make the change, and type the new data on top of the old data.

In the above example, the a ZIP field is not displayed because it can't fit on the screen with all the other information. To look at that field, you need to pan to the right. Press ^→ to pan to the right one field, so the zip code will be displayed and the last name will be invisible. Press ^← to pan the screen back to the left.

For more options that you can use in BROWSE mode, press ^Home. A menu of options will appear:

Bottom Top Lock Record No. Freeze Find

These options are described in more detail in Table 4.3.

Option	Purpose
Bottom:	Positions the highlighting to the last record in the database.
Top:	Places the highlighting on the first record in the database.
Lock:	Maintains the on-screen display of one or more fields to the left of the screen even as you pan to the right using the ^→ or ^B keys.
Record No:	Asks you to enter the number of a record to highlight.
Freeze:	Locks the highlighting into a single field (like ZIP) for easier editing of the same field in a number of records.
Find:	Looks up data in an indexed field. For example, with NAMES.NDX assigned as the Master index file, you could select Find and type Doe to quickly move the highlighting to Doe's record.

Table 4.3: BROWSE Menu Options

When you finish browsing, you can use a ^W or ^End to save all data and return to the Assistant menu.

Dot-prompt Commands for Editing

Like all Assistant menu options, you can edit a database directly from the dot prompt. For example, to browse through the database, just use the database (and any index files) and enter the BROWSE command at the dot prompt, as in these commands:

USE MAIL INDEX NAMES,ZIP ↵
BROWSE ↵

To edit an open database (and index files), use the EDIT command with the number of the record to edit:

USE MAIL INDEX NAMES,ZIP ↵
EDIT 2 ↵

Global Editing

The term global edit refers to a change to a database, that affects more than one record. Global edits are used quite often in database management to perform a large task quickly. For example, if you had an inventory system with several thousand parts on it, and one of your manufacturers raised the price of all his products by 10%, you could perform a global edit to increase the price of all items made by that manufacturer in a single step.

Let's give it a try using the MAIL.DBF database and a hypothetical example. Suppose that you have two different secretaries entering information into your MAIL.DBF database. One always spells out Los Angeles, while the other uses the abbreviation L.A. You decide to standardize the entry so that searches for Los Angeles residents do not require two separate searches for Los Angeles and L.A.

The least efficient way to make this change would be to work through the database, finding each occurrence of Los Angeles and retyping it as L.A. If there were 500 such from the Assistant menu screen:

- ✓ Highlight the update option.
- ✓ Select the Replace option.
- ✓ Select CITY.
- ✓ Type L.A. ↵
- ✓ Press →.

Notice the command you've created on the Action Line:

Command: REPLACE CITY WITH 'L.A.'

This will replace the CITY field with the abbreviation L.A. in the current record, but you want dBASE to do so for those records which currently contain Los Angeles. Therefore, you need to build a search condition for these records following these steps:

- ✓ Select Build a search condition.
- ✓ Select CITY.
- ✓ Select =.
- ✓ Type **Los Angeles.** ↵

Now the command is more specific:

Command: REPLACE CITY WITH 'L.A.' FOR CITY = 'Los Angeles'

In English, this translates to "Everywhere you see the words Los Angeles in the CITY field, put in the abbreviation L.A. instead." Select No more conditions to leave the submenu, and then select Execute the command. The screen will display this message:

3 records replaced

If you now use the Retrieve and List options, you'll see that the records which once contained Los Angeles now contain the abbreviation L.A.:

Record#	LNAME	FNAME	ADDRESS	CITY
5	Appleby	Andy	345 Oak St.	L.A.
2	Doe	Ruth	1142 J. St.	L.A.
1	Smith	Betsy	222 Lemon Dr.	New York
3	Smith	Dave	123 B St.	L.A.
6	Smith	John Q.	123 A St.	San Diego
4	Smithsonian	Lucy	461 Adams St.	San Diego

You can use the REPLACE command quite easily from the dot prompt as well, as long as you remember to use FOR condition to clearly define which records get changed. For example, exit the menu by pressing Esc. Then enter this command at the dot prompt:

REPLACE CITY WITH 'Los Angeles' FOR CITY = 'L.A.'

All the L.A. records will change back to Los Angeles.

The REPLACE command can be helpful with standardizing case in some situations. For example, in MAIL.DBF, you have a record with SMITH in uppercase. From the dot prompt, you can enter the command

to globally change all SMITHs to Smith. After trying these two REPLACE commands from the dot prompt, you can enter the command

LIST LNAME, CITY ↵

to quickly view the results.

Whether you make your global changes from the menu or from the dot prompt, you must be certain to enter the correct FOR condition to ensure that the command only acts upon the appropriate records. REPLACE has a high whoops factor, which means that by the time you realize that you've entered the wrong FOR condition, dBASE has already changed all the wrong records. So do be careful.

Now type

ASSIST ↵

to return to the Assistant menu screen.

Deleting Records

Deleting records from a dBASE database actually involves two steps. First, you mark the record for deletion. You can review these marked records prior to permanently deleting them, or you can temporarily delete them to perform a certain task. Second, whenever it's convenient, you can pack the database, which permanently deletes the records that have been marked for deletion.

Marking Records

The easiest way to mark a record for deletion is to press ^ U while the record is being edited via the Edit or Browse options. For example, suppose that you want to delete Dave Smith from the MAIL.DBF database. First, enter BROWSE mode from the Assistant menu screen:

- ✓ Select Update.
- ✓ Select Browse.

Next, move the highlighting to Dave Smith's record, as shown in Figure 4.3. Then press ^ U so that "Del" appears in the lower right corner of the screen (also shown in the figure). The ^U key acts as a toggle, which means that pressing it repeatedly marks and unmarks records for deletion.

After marking the record for deletion, save your work in BROWSE by typing ^ W or ^ End. If you now select the Retrieve and list options from the Assistant menu, you'll see that this record is marked for deletion with a leading asterisk, but it has not been deleted yet:

2	Doe	Ruth	1142 J. St.	Los Angeles	CA
3	*Smith	Dave	123 B St.	Los Angeles	CA
6	Smith	John Q.	123 A St.	San Diego	CA
1	Smith	Betsy	222 Lemon Dr.	New York	NY
4	Smithsonian	Lucy	461 Adams St.	San Diego	CA

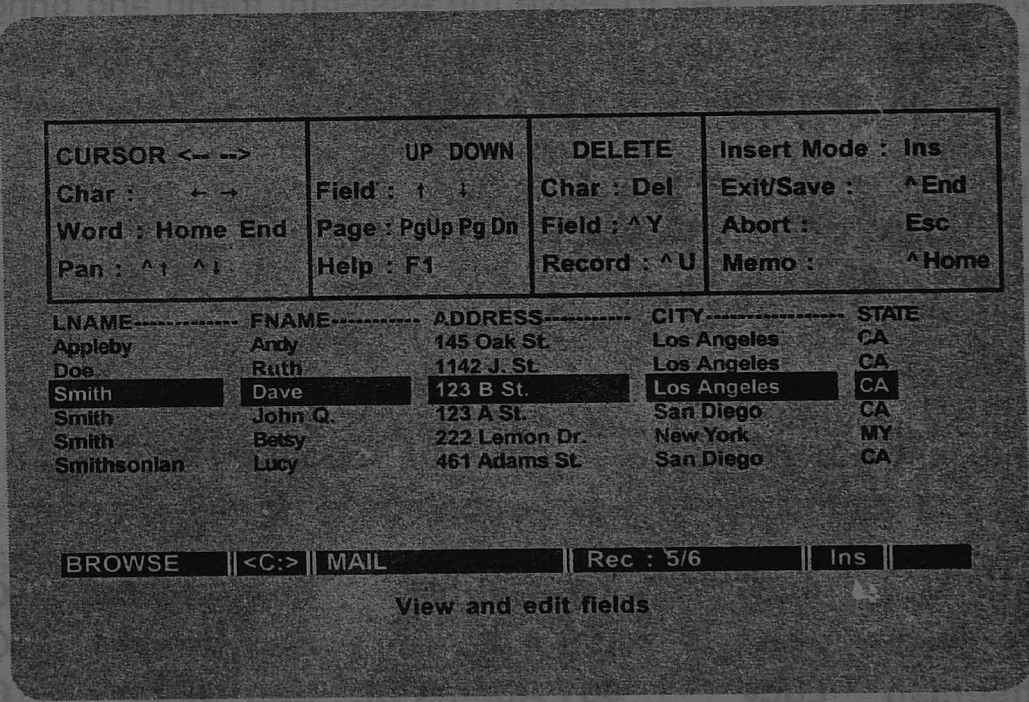


Figure 4.3: Record Highlighted on a BROWSE Screen

Packing the Database

Now, to permanently remove this record from the database, you must select the Pack option from the Assistant menu screen:

- ✓ Highlight the Update option.
- ✓ Select Pack.

dBASE will display the message

5 records copied

which means that there are now only five records on the database. You'll also see messages that the NAMES.NDX and ZIP.NDX index files are being updated. (When records are deleted from the MAIL.DBF database, they are automatically removed from NAMES.NDX and ZIP.NDX if you remembered to open your index files when you opened your database.)

If you select the Retrieve and List options now, you'll see that Dave Smith's record has indeed been deleted from the database:

4	Appleby	Andy	345 Oak St.	Los Angeles	CA
2	Doe	Ruth	1142 J. St.	Los Angeles	CA
1	Smith	Betsy	222 Lemon Dr.	New York	NY
5	Smith	John Q.	123 A St.	San Diego	CA
3	Smithsonian	Lucy	461 Adams St.	San Diego	CA

Deleting from the Dot Prompt

You can easily perform deletions from the dot prompt using the DELETE command. As usual, begin by pressing the Esc key to leave the Assistant menu and bring up the dot prompt. Now, suppose that you want to delete all California residents from the MAIL.DBF database. At the dot prompt, type this command:

```
DELETE ALL FOR STATE = 'CA' ←
```

dBASE would respond with

4 records deleted

The effect of this command would be to mark all records with CA in the STATE field for deletion. If you were to list the contents of the database now, you'd see

1	*Appleby	Andy	345 Oak St.	Los Angeles	CA	92123
2	*Smith	John Q.	123 A St.	San Diego	CA	92123
3	*Smithsonian	Lucy	461 Adams St.	San Diego	CA	92122-1234
4	*Doe	Ruth	1142 J. St.	Los Angeles	CA	91234
5	*Smith	Betsy	222 Lemon Dr.	New York	NY	01234

All individuals who live in California are marked for deletion. Don't PACK them now, or you'll end with only one record in your database, Betsy Smith, since she lives in New York. Rather than packing, let's

RECALL ALL ↵

so that you don't lose all your California residents permanently. (The earthquake hasn't hit yet, but the practice might be useful.)

If you LIST after recalling all the records you'll see

1	Appleby	Andy	345 Oak St.	Los Angeles	CA	92123
2	Smith	John Q.	123 A St.	San Diego	CA	92123
3	Smithsonian	Lucy	461 Adams St.	San Diego	CA	92122-1234
4	Doe	Ruth	1142 J. St.	Los Angeles	CA	91234
5	Smith	Betsy	222 Lemon Dr.	New York	NY	01234

Everyone is back in shape.

Global deletes are useful for getting the job done quickly, but there is an element of danger: you might accidentally delete records you wanted to keep. It's a good idea always to LIST the records that are marked for deletion prior to packing the database. For example, suppose that you decided to take a shortcut method for deleting John Smith from the database using the command

DELETE ALL FOR LNAME = 'Smith' ↵

You would end up with the response

2 records deleted

Whoops. You had actually only planned to delete one Smith, but ended up with two deletions. So to see who else you've accidentally deleted, you would ask dBASE to

LIST FOR DELETED() ↵

That is, list all the records that are marked for deletion. The result would be

2	*Smith	John Q.	123 A St.	San Diego	CA	92123
5	*Smith	Betsy	222 Lemon Dr.	New York	NY	02134

Apparently we've gotten a little carried away with our global delete here. We only meant to delete John Q. Smith, but unfortunately our global delete marked Betsy Smith for deletion also. You can bring back Betsy with the command to

RECALL RECORD 5 ↵

dBASE would release Record 5 from deletion. Global deletes are useful in cases where you want to delete records of a certain type, but be sure to LIST FOR DELETED() prior to PACKing to make sure you won't be deleting any innocents. Now type this command:

RECALL ALL ↵

so that you don't lose John Q. Smith permanently.

Modifying the Database Structure

It is not at all unusual to change your mind about what to store in a database after you have entered some records. For example, phone numbers might come in handy on your MAIL database, but you didn't make a field for them. dBASE III PLUS offers a very easy technique to change the structure of a database, no matter how much data you've already stored.

- Remember : to leave the dot prompt and return to the Assistant menu, type **ASSIST** and press ↵

For example, to add a new phone number field to the MAIL.DBF database, follow these simple steps from the Assistant menu screen:

- ✓ Highlight the Modify option.
- ✓ Select the Database file option.

The current structure of the database appears on the screen, ready for modification, as in Figure 4.4.

Bytes remaining : 3928

CURSOR <-- -->	INSERT	DELETE	Up a field : ↑
Char : ← →	Char : Ins	Char : Del	Down a field : ↓
Word : Home End	Field : ^N	Word : ^Y	Exit/Save : ^End
Pan : ^I ^J	Help : F1	Field : ^U	Abort : Esc

	Field Name	Type	Width	Dec
1	NAME	Character	15	
2	FNAME	Character	18	
3	ADDRESS	Character	15	
4	CITY	Character	15	
5	STATE	Character	5	
6	ZIP	Character	18	

MODIFY STRUCTURE	<C:>	MAIL	Field : 1/6	Ins
------------------	------	------	-------------	-----

Enter the field name.

Field name begin with a letter and may contain letters, digits and underscores

Figure 4.4: MODIFY STRUCTURE Screen for MAIL.DBF

Now, Pop quiz. What type of data will the phone number be? At first you might think it should be Numeric data, but this is incorrect. Phone numbers contain non-numeric characters, like parentheses and hyphens. Remember; the numeric data type is for real numbers only. Therefore, the phone number will be character data.

Most phone numbers look something like (123)456-7890, therefore we'll assign a width of 13 characters to this new field.

You can insert fields using the ^N key, but in this case field 7 seems as good a place as any for a phone number. Using the ↓ arrow key to move the cursor to the first empty field slot on the screen, type the field name PHONE, use the Character data type, and enter the length, 13. Your screen display should look something like this:

MAIL.DBF

	Field Name	Type	Width	Dec
1	LNAME	Character	15	
2	FNAME	Character	10	
3	ADDRESS	Character	25	
4	CITY	Character	15	
5	STATE	Character	5	
6	ZIP	Character	10	
7	PHONE	Character	13	
8		Character		

After adding the new field to the database, save the new structure by typing a ^ W or ^ end. dBASE displays this message:

Press ENTER to confirm—any other key to resume Database records will be APPENDED from backup fields of the same name only!!

This somewhat cryptic message is a little easier to understand if you know how the MODIFY STRUCTURE command works. When you ask dBASE to MODIFY STRUCTURE, it first copies all of the records from the database to a separate database. Then, it deletes all records in the current database and allows the user to make changes. When you are done changing the structure, it reads all the records from the backup database back into the current database. However, if you changed the name of any field, it does not read in data for the new field name. For example if you had changed the LNAME file to LAST_NAME, you would have

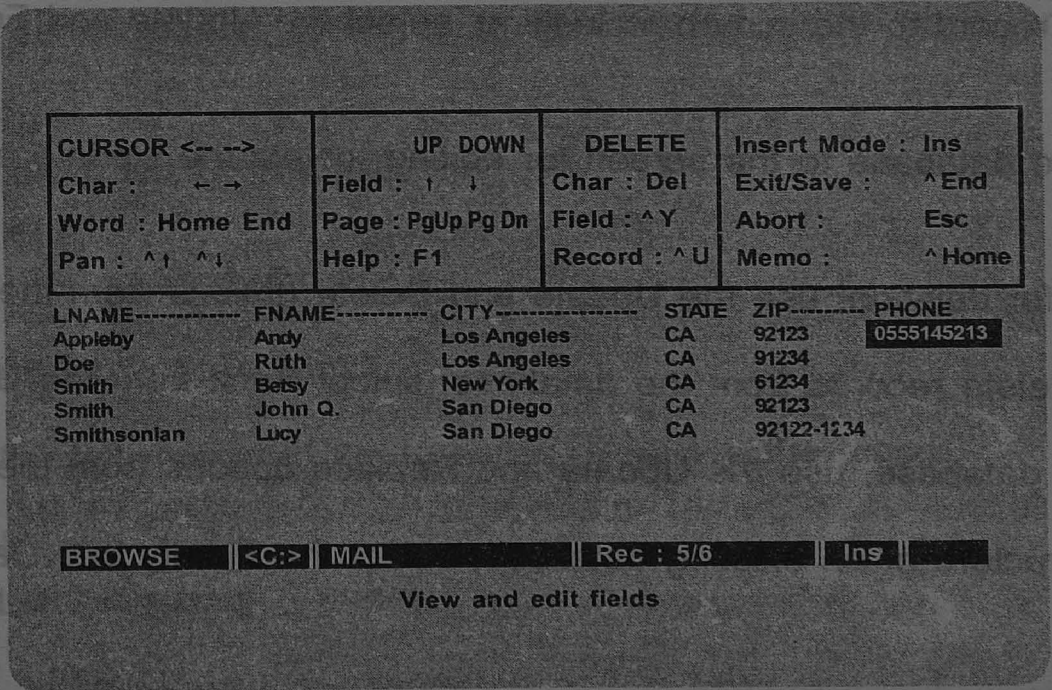


Figure 4.5: Modified BROWSE Screen

lost all of the last names in the database. This is a useful reminder, because if you have changed a field name, you can press any key and then change the field name back to its original name.

For this example, you didn't change any field names; you just added a new field, so it's safe to proceed. Press the Return key now to complete the modification of the database. You'll be returned to the assistant menus.

Unfortunately, there is no command to fill in the phone numbers automatically in the MAIL database. To add new phone numbers, you'll need to do so one record at a time.

The easiest way to do this would be to highlight the Update option and select Browse. Then press ^Home to bring up the menu, and select the Lock option. Type 2 and press Return to lock the two leftmost fields. Next, type ^Home again to bring up the menu, and select Freeze. Type the field name PHONE to keep the highlighting in the PHONE field.

Next, press ^ → a few times to bring the new phone number field onto the screen. Now you can easily move from record to record and add new phone numbers. Figure 5.5 shows the MAIL.DBF BROWSE screen with the new PHONE field.

Type ^End or ^W when done filling in phone numbers to save your work and return to the menu.

Reviewing Editing

In this chapter we've discussed many techniques for changing an existing database:

When editing (making changes to) a database, make sure to first open your database and the index files you use regularly.

To bring a single record to the screen for editing, select the Update and Edit options from the menu.

To bring the entire database to the screen for editing, select the Update and Browse options from the menu.

To mark records for deletion, use the Update and Delete options from the menu.

To permanently delete records from the database, select Pack from the Update menu.

To globally edit a database, use the Update and Replace options from the menu.

To change the structure of an existing database, select the Modify and Database file options from the menu.

Section 4.2

Creating and Printing Formatted Reports

So far you've been displaying data on the screen without any particular format. I've even cheated a little in displaying **LISTs** in this book so that they would fit the page. If you don't provide dBASE with an exact format displaying data dBASE will list records and fields in its own fashion. To print formatted reports with dBASE III PLUS, you use its built-in report generator.

Creating A Report

You can create as many report formats for a database as you wish. Like database and index files, you assign a name (maximum of eight characters long, no spaces or punctuation) to the report, and dBASE stores it on disk with the file name extension. **FRM.MAIL.DBF** database.

The first steps is to make sure that the **MAIL.DBF** database is in use. If the **MAIL** file name does not appear in the Status Bar at the bottom of the screen, use the **Set Up** and **Database** file options to open the **MAIL.DBF** database and the **NAMES.NDX** and **ZIP.NDX** index files. For this example, be sure to open **NAMES.NDX** before **ZIP.NDX** to create an alphabetized report.

Next, highlight the **Create** option on the main menu bar. Select **Report** from the submenu. dBASE will ask for the drive and a file name for the report format. As usual, select a drive and then type the file name

BYNAME ↵

dBASE first displays a menu of the format options for printing reports, as shown in Figure 4.6. The meaning of each of these options is summarized in Table 4.6.

For your first sample report, follow these steps to set up the page format:

- ✓ Select **Page Title**.
- ✓ Type **Mailing list by Name**.
- ✓ Press ↵
- ✓ Press ^ End to finish the title.

Figure 4.6: Format Options Menu

Next, adjust the left margin to zero spaces. Here are the steps:

- ✓ Press ↓ twice.
- ✓ Select **Left Margin**.
- ✓ Type **0**.
- ✓ Press ↵

Options
Groups
Columns
Locate
Exit
09:03:57 am

Page Title
Page width (positions)
Left margin
Right margin
Lines per page
Double space report
Page eject before printing
Page eject after printing
Plain page

90
4
8
58
No
Yes
No
No

CURSOR : (-- --)
Char : ← →
Word : Home End

Delete char : Del
Delete word : ^T
Delete row : ^U

Insert row : ^N
Toggle menu : F1
Abandon : Esc

Insert : Ins
Zoom in : ^PgDn
Zoom out : ^PgUp

MODIFY LABEL || <C:>|| TWOCOL.LBL || Opt : 1/5 || Ins ||

Position selection bar - ↑↓. Select - ←→. Leave Menu - ←→.
Enter upto four lines of text to be displayed at the top of each report page.

Figure 4.6 : Format Options Menu

The rest of the settings on the menu are sufficient for this report. Now let's define the contents of the report itself. You can skip the Group option on the main menu, because that is used primarily for subtotals—a topic we'll discuss in a later chapter. Here

On-Screen Prompt

Effect of Your Response

Page title:	Defines a title to be printed at the top of the report.
Page width:	Defines the maximum page width for the report (usually 80 columns, although can be extended to 500 columns for very wide printer paper).
Left margin:	Defines a left margin measured in number of spaces.
Right margin:	Defines a right margin number of spaces.
Lines per page:	Defines the number of lines to be printed on each page (usually 58 for 8 1/2 by 11 inch paper).
Double space report:	Prints the report without blank lines between rows unless you change this option to Yes.
Page eject before Printing:	Yes ensures that the printed report starts on a new page. No prints the report starting at the current printer position of the paper.
Page eject after printing:	Yes makes the printer move to the top of the next page after printing the last page of the report.
Plain page:	No prints the report title, page numbers, and current date.

Table 4.6 : Summary of Format Options

are the steps to put the last name in the first column of the report:

- ✓ Highlight Columns on the Report menu.
- ✓ Select Contents.
- ✓ Press F10 to display a menu of field names.
- ✓ Select LNAME
- ✓ Press ↵ to finish the entry.
- ✓ Select Heading.
- ✓ Type Last Name.
- ✓ Press ↵
- ✓ Press ^ End to finish the entry.

Now let's narrow the column just a bit to try to conserve space. Here are the steps:

- ✓ Select the Width option.
- ✓ Press ↓ until the width is 12.
- ✓ Press ↵

That defines the contents of the first column of the report. Notice that a template of the report appears in the bottom half of the screen with the Last Name heading displayed, and 12 X's to show the width of this column on the report, as in Figure 4.7

Contents	LNAME
Heading	Last Name
Width	12
Decimal Places	
Total this column	

Report Format

Last Name

xxxxxxxxxxxxxxxx

MODIFY REPORT | <C> SUBTOTAL.FRM | Opt : 8/8 | Ins

Position selection bar - ↑↓. Select - ← Prev/Next column - PgUp/PgDn.

Enter a field or expression to display in the indicated report column.

Figure 4.7: Template of BYNAME Report

Next, put the First Name in the next column of the report. The steps are outlined below.

- ✓ Press PgDn to move to the next column.
- ✓ Select Contents.
- ✓ Type FNAME.
- ✓ Press ← to enter the field name.
- ✓ Select Heading.
- ✓ Type **First Name**.
- ✓ Press ←
- ✓ Press ^ End to finish the entry.

The column should be defined like this:

Contents	FNAME
Heading	First Name
Width	10
Decimal Places	
Total this column	

Now press PgDn to define the next column. Place the ADDRESS field in this column by filling in these options:

Contents	ADDRESS
Heading	Address
Width	20
Decimal places	
Total this column	

Press PgDn to define the next column, and place the CITY field in this column by filling in these options:

Contents	CITY
Heading	City
Width	15
Decimal places	
Total this column	

Press PgDn to move to the next column, and fill in the STATE column:

Contents	STATE
Heading	St.
Width	2
Decimal places	
Total this column	

Finally, press PgDn to describe a column for the zip code. Your screen should look like Figure 4.8 when you have selected the last format option for BYNAME. Now you must save your report format.

Options Groups **Columns** Locate Exit 09:03:57 am

Contents	Zip
Heading	Zip Code
Width	18
Decimal Places	
Total this column	

Report Format

First Name	Address	City	St	Zip Code
XXXX XXXXXXXXXXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXX XX XXXXXXXX				

MODIFY REPORT || <C:~|| SUBTOTAL.FRM || Opt : 8/8 || Ins ||

Position selection bar - ↑↓. Select - ←→ Prev/Next column - PgUp/PgDn.
 Enter a field or expression to display in the indicated report column.

Figure 4.8: Report Format Selections

Saving a Report Format

when you've defined the basic format for your report, save your work by highlighting the Exit option on the menu at the top of your screen and selecting Save. dBASE will return to the main menu where you can now print your report.

Printing the Report

To print your report, highlight the Retrieve option, and select Report. Specify a drive and select BYNAME.FRM. For this first run, simply select the Execute the command option, and answer Yes to the question about printing the report. Your finished report will look like Figure 4.9.

Mailing List by Name

Last Name	First Name	Address	City	St	Zip Code
Appleby	Andy	345 Oak St.	Los Angeles	CA	92123
Doe	Ruth	1142 J.St.	Los Angeles	CA	91234
Smith	Betsy	222 Lemon Dr.	New York	NY	01234
Smith	John Q.	123 A St.	San Diego	CA	92123
Smithsonian	Lucy	461 Adams St.	San Diego	CA	92122-1234

Figure 4.9: Sample Directory Report From MAIL.DBF

Like the List and Locate options, you can specify that only certain records be printed on the report. For example, to list only California residents, follow these steps:

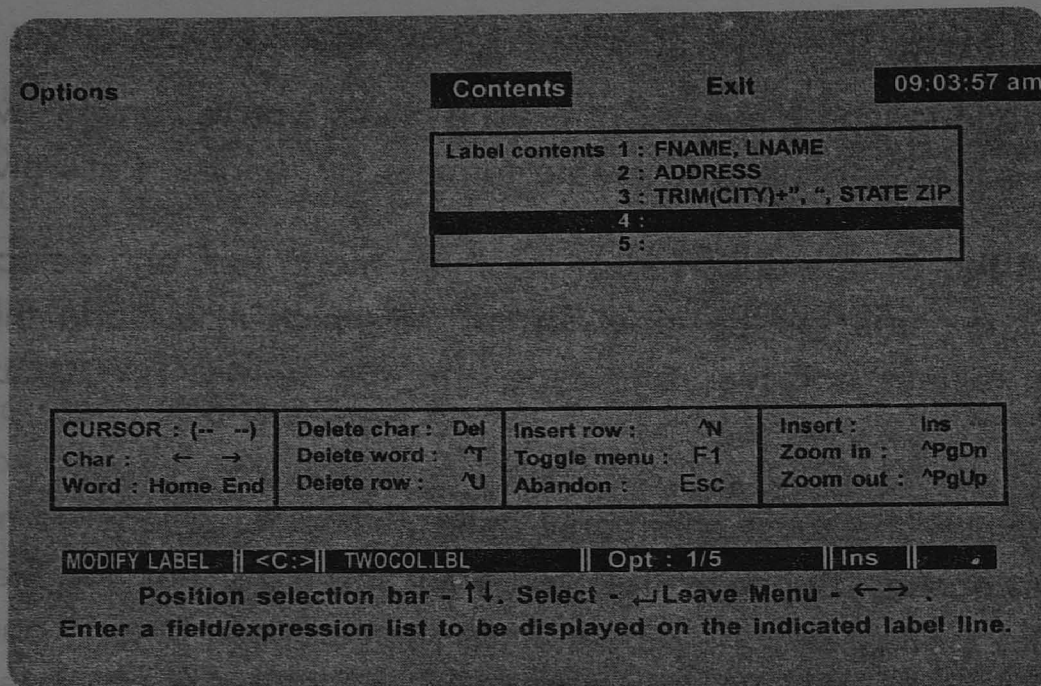


Figure 4.11: Contents for MAIL.DBF Labels

Notice that the first line contains

FNAME, LNAME

This command tells dBASE to print the FNAME and LNAME fields from each database record on the first line of the label, with a single space between fields. The next line contains only the ADDRESS field. The third line of the label contains.

TRIM(CITY) + ', ', STATE,ZIP

This line tells dBASE to place the CITY, STATE, and ZIP fields on the third row of the label. You've had to go to some lengths to get it just right, though. First, the TRIM function tells dBASE that if a city's name is shorter than the 15 spaces you defined for your database CITY field, leftover blank spaces should be trimmed, not printed. (These extra blank spaces are called trailing blanks.) The plus sign tells dBASE to concatenate (link) the following text or symbols (in this case, a comma) directly to the CITY field without inserting a blank space. The comma in quotation marks (",") tells dBASE to print a comma and a blank space. (Note: The quotation marks distinguish this comma from the others. Usually commas tell dBASE to leave a blank space between two fields.) The final portion of this line.

,STATE,ZIP

simply places the state and the zip code on the same line, with a blank space in front of each.

After defining the label format, highlight the Exit option and select Save.

Printing Labels

No doubt you'll want to print mailing labels in zip code order rather than alphabetical order, so you'll want to make the ZIP.NDX index file primary. To do so, select the Set Up and Database file options from the Assistant menu, and select MAIL.DBF. When you select index files, be sure to select ZIP.NDX first.

Then, to print labels, highlight the Retrieve option, and select Label. Specify a drive, and select the TWOCOL.LBL file. Select the Execute the command option, and answer Yes to the printer question. Your labels will look like Figure 4.11(a).

Betsy Smith
222 Lemon Dr.
New York, NY 01234

Lucy Smithsonian
461 Adams St.
San Diego, CA 92122-1234

John Q. Smith
123 A. St.
San Diego, CA 92123

Ruth Doe
1142 J. St.
Los Angeles CA 91234

Andy Appleby
345 Oak St.
Los Angeles CA 92123

Figure 4.11(a) : Two-Column MAIL.DBF Labels

Of course, as with most options, you can also specify search conditions for your mailing labels (such as ZIP > = 90000) to print only labels that meet some criterion.

Modifying Label Formats

To modify a label format file, highlight the Modify option from the main menu, and select Label. Specify the drive and file name. Then you can change the label format. (Use the arrow keys to move the cursor and the F1 to get help.)

Using Label and Report Commands

Like everything else in dBASE III PLUS, reports and mailing labels can be accessed directly from the dot prompt. To create a report format for a database, set the default drive, open the appropriate database, then enter the command MODIFY REPORT with the name of the report format file. For example, to modify the BYNAME.FRM report format the dot prompt, you would enter the commands: (On a hard-disk system, omit the SET DEFAULT TO B).

SET DEFAULT TO B ↵

USE MAIL ↵

MODIFY REPORT BYNAME ↵

To print a report, make sure the appropriate database is open, and any index files that you want to use are active. Then, enter the command REPORT FORM with the name of the report format file. In this example, the commands open the MAIL.DBF database with the NAMES.NDX index file primary (so the records are listed in alphabetical order) and print the data in the BYNAME.FRM report format:

```
SET DEFAULT TO B ↵  
USE MAIL INDEX NAMES, ZIP ↵  
REPORT FORM BYNAME TO PRINT ↵
```

Adding the command TO PRINT to the end of the REPORT FORM command prints the report rather than displaying it on screen.

You can use the FOR option to specify a search condition for printing reports. For example, this command prints data for people whose last names begin with the letters A through M:

```
REPORT FORM BYNAME FOR LNAME > + "A";  
.AND. LNAME < + "M" TO PRINT
```

To create or change a mailing label format for a database, use the command MODIFY LABEL along with the name of the format file. The appropriate database must be open:

```
SET DEFAULT TO B ↵  
USE MAIL ↵  
MODIFY LABEL TWOCOL ↵
```

To display labels, use the LABEL FORM command with the name of the label format file. If you want them in zip code order, use the appropriate index file, too:

```
USE MAIL INDEX ZIP, NAMES ↵  
LABEL FORM TWOCOL ↵
```

To print labels, add the TO PRINT command to the end of the LABEL FORM command:

```
LABEL FORM TWOCOL TO PRINT ↵
```

When printing labels, you can use the SAMPLE option to display a facsimile of the labels to be printed, which will help you to line up the labels properly in the printer. For example, this command allows you to check the alignment of the labels in the printer before printing labels from the TWOCOL. LBL format file:

```
LABEL FORM TWOCOL SAMPLE TO PRINT ↵
```

You can, of course, build a search condition with the FOR option when printing labels.

This command prints mailing labels for individuals in the 90000 to 95000 zip code areas, using the TWOCOL.LBL format file:

```
LABEL FORM TWOCOL FOR ZIP > = '90000';  
.AND. ZIP < = '95000' TO PRINT
```

Reviewing Reports and Labels

In this chapter, you have learned how to use formats to create reports and mailing labels.

To create a report format for a database, select the Create and Report options from the menus.

To print a report, select the Retrieve and Report options from the menus.

To modify an existing report format, select the Modify and Report options from the menu.

To create a mailing label format for a database, select the Create and Label options from the menu.

To print mailing labels, select the Retrieve and Label options from the menu.

To print a report or mailing labels in sorted order, use an existing index file, or optionally, create a new one.

Commands for creating report and label formats, as well as for printing reports and labels, can be entered at the dot prompt.

Section 4.3

Designing Custom Screen Displays

As you've seen in previous chapters, the Append and Edit options allow you to enter and edit data through a simple screen which displays the fields in a vertical row.

Through the use of the dBASE III PLUS Screen Painter, you can create your own custom for adding and editing data. Figure 4.12 shows such a custom screen for MAIL.DBF, which you'll develop in this chapter. However, you'll see that being creative is easy, so you might prefer to try your own screen as you go through the chapter.

Creating a Custom Screen

To get started creating a form, follow these steps:

- ✓ Highlight the Create option.
- ✓ Select Format.
- ✓ Select a drive.
- ✓ Type **ADDNAMES**, or create your own form name.

A screen for developing a form will appear, as well as a menu of options. First, you need to tell dBASE which database this form will be used for. So, follow these steps:

- ✓ Select *Select Database file*.
- ✓ Specify MAIL.DBF.

Now, as a shortcut to developing a form, we can have the screen painter put together a general purpose form, which we in turn will modify. (This is easier than trying to create the form completely from scratch.) Follow these steps:

Add/Change Mailing List Data			
First Name :	Betso	Last Name :	Smith
Address :	222 Lemon Dr.		
City :	New York	State :	NY Zip Code : 01234
Phone :	0999823542		
Cursor Move :	Page Move :	Delete :	Done :
Up	Next : PgDn	Letter : Del	
Left Right	Previous : PgUp	Field : ^Y	Save Work : ^End
Down		Record : ^U	Abandon : Esc
arrow keys			
EDIT <C> MAIL Rec : 1/5 Ins			

Figure 4.12 : Custom Data-Entry Screen

- ✓ Select Load fields.
- ✓ Highlight each field, and press ←

You'll notice that as you press ↓, the highlighting moves through each field name on the menu. If you press Return while a field name is highlighted, a triangle appears next to it indicating that the field has been selected for inclusion on the custom form. For this exercise, use this method to select all the field names for your screen:

LNAME
FNAME
ADDRESS
STATE
ZIP
PHONE

Press → after all field names have been selected. You'll see a simplified form appear on the screen, as in Figure 4.13. This screen is called the blackboard because, as you'll see, it is easy to move, add, and change information on the screen.

The screenshot shows a terminal window titled "Set Up" with a menu bar containing "Set Up", "Modify", "Options", and "Exit". The time "09:03:57 am" is displayed in the top right. The main area lists field definitions:

```
LNAME : 
FNAME : 
ADDRESS : 
CITY : 
STATE : 
ZIP : 
PHONE :
```

Each field name is followed by a series of 'x' characters indicating the field's width. Below the field definitions is a status bar with the text "MODIFY SCREEN || <C: C:\ADDNAMES.SCR || Rec : 1/5 || Ins ||". Below the status bar is a prompt: "Enter text. Drag field or box mader cursor with ← F19 for menu". At the bottom, it says "Screen field definition blackboard".

Figure 4.13: Loaded Fields on the Screen Painter Blackboard

Notice that there are field labels (such as LNAME, FNAME, etc). There are also field highlights which show where the reverse video portion of the screen will appear when the form is used for entering and editing data. The Xs show number of characters in each field.

Moving Information on the Blackboard

It's pretty easy to move information around on the blackboard after you've practiced a bit. Take a moment to review the basic editing keys listed in Table 4.7.

Key	Alternate Key	Effect
F10		Switches between menu and blackboard.
←		Moves cursor left one character.
→		Moves cursor right one character.
↑		Moves cursor up one line.
↓		Moves cursor down one line.
Ins	or ^V	Turns the INSert mode ON and OFF when cursor is not in a field, or extends the length of the highlighted field where the cursor rests.
^N		Adds a blank line between two lines.
End	or ^F	Moves cursor to beginning of next word.
Home	or ^A	Moves cursor to beginning of current to previous word.
↵	or ^M	Inserts a new line if INSert is On. Otherwise, moves cursor down one line. Also moves field highlighting and boxes.
Del	or ^G	Deletes character at the cursor or decreases the size of a highlighted field where the cursor rests.
Backspace		Deletes character to left of cursor.
^T		Deletes word to right of cursor.
^Y		Deletes an entire line from form.
^U		Deletes a field or box.
PgDn	or ^C	Scrolls down 18 lines on screen.
PgUp	or ^R	Scrolls up 18 lines on screen.

Table 4.7: Blackboard Editing Keys

Now let's start shifting some things around. First, press ^ N four times to put some blank lines at the top of the screen for a title (which you'll add later). Make sure that the cursor is in the upper-left corner of the screen and that the blackboard is still displayed on the screen. (Note: Press F10 if you want to switch from Blackboard to menu.) Next, let's move the LNAME field highlighting next to the FNAME field. Follow these steps to move a single field:

- ✓ Press ↓ four times to move the cursor to LNAME.
- ✓ Press the End key to put the cursor inside the field highlighting.
- ✓ Press ↵ to enter DRAG mode.
- ✓ Press → 12 times to move the cursor 12 spaces to the right.
- ✓ Press ↓.
- ✓ Complete the move by pressing ↵

The LNAME field highlighting will now be next to the FNAME field highlighting, as shown in Figure 4.14.

Figure 4.14: customized LNAME Location

Notice the general steps that you used to move the field highlighting. First, you put the cursor inside the field highlighting that you wanted to move. When the cursor was inside the highlighting, you pressed the Return key to begin the Dragging option. The bottom of the screen then read:

Move field with ↑↓ ←→. Complete with ↵

You used the → and ↓ keys to move the cursor to the new location, and pressed Return to complete the move. (You'll have a chance to practice this again in a moment.)

Now, let's add a few blank lines to this form. To move the ADDRESS field down a line, follow these steps:

- ✓ Press ↓.
- ✓ Press ^←.
- ✓ Press ↵

To move the CITY field down a line, follow these steps:

- ✓ Press ↓.
- ✓ Press ↵

Now, to move the STATE field next to the CITY field, follow these steps:

- ✓ Press ↓ and End to move the cursor into the STATE field.
- ✓ Press ↵ to start the move.
- ✓ Using → and ↑, move the cursor to row 09, column 36.
- ✓ Complete the move by pressing ↵

Now move the ZIP field over by the STATE field. Here are the steps:

- ✓ Press ↓ twice, and then press Home to move cursor inside the ZIP field.
- ✓ Press ← to start the move.
- ✓ Move the cursor to row 09, column 53.
- ✓ Complete the move by pressing ←

Now you can delete the STATE and ZIP field labels by following these steps:

- ✓ Press ↓.
- ✓ Press ^←.
- ✓ Press ^Y.
- ✓ Press ^T.

The form is taking shape now, as shown in Figure 4.15. Next let's create some more appropriate field to be displayed on a custom data-entry screen.

Adding Field Labels

Field labels like LNAME and FNAME may be easy for you to remember, but they are not the best labels for other people who will enter data through your custom screen. You can create field labels simply by typing them on the screen. Here are some simple steps to put a whole new set of field labels on the ADDNAMES form. Note: If pressing the Space bar moves text to the right, press the Ins key while the cursor is not inside a field.

- ✓ Press Ins if the "INS" message appears at the bottom of your screen.
- ✓ Move the cursor to the L in LNAME, and press ^T.
- ✓ Press ↓.
- ✓ Press the Space bar, and type **First Name:**.
- ✓ Move the cursor to row 05, column 28.
- ✓ Type **Last Name:**.
- ✓ Move the cursor to row 07, column 00.
- ✓ Press the Space bar, and type **Address:**.
- ✓ Move the cursor to row 09, column 00.
- ✓ Press the Space bar, and type **City:**.
- ✓ Move the cursor to row 09, column 30.
- ✓ Type **State:**.
- ✓ Move the cursor to row 09, column 44.
- ✓ Type **Zip Code:**.
- ✓ Move the cursor to row 11, column 00.
- ✓ Press the Space bar, and type **Phone:**.

Set Up Modify Options Exit 09:03:57 am

LNAME : xxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxx

FNAME : xxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxx

ADDRESS : xxxxxxxxxxxxxxxxxxxxxxxx

CITY : xxxxxxxx xx xxxxxxxx

PHONE : xxxxxxxx

MODIFY SCREEN <C: C:ADDNAMES.SCR Rec : 1/5 Ins

Enter text. Drag field or box mader cursor with F19 for space

Screen field definition blackboard

Figure 4.15 : Customized Field Locations

Notice that you pressed Space bar for cosmetic reasons—to move field names away from the left edge of the screen. When you’ve finished, your custom form will look like Figure 4.16.

Let’s customize this form a little more, by adding some boxes and titles.

Set Up Modify Options Exit 09:03:57 am

First Name : xxxxxxxxxxxxxxxx Last Name : xxxxxxxxxxxxxxxx

Address : xxxxxxxxxxxxxxxxxxxxxxxx

City : xxxxxxxx State : xx Zip Code : xxxxxxxx

Phone : xxxxxxxx

MODIFY SCREEN <C: C:ADDNAMES.SCR Rec : 1/5 Ins

Enter text. Drag field or box mader cursor with F19 for space

Screen field definition blackboard

Figure 4.16 Custom Form with Field Labels

Drawing Boxes

Let’s continue creating a custom data-entry screen by drawing a box for the title. First, call up the menu by pressing F10. Then, highlight Options. This will display these choices:

Generate text file image
Draw a Window or a Line
Single Bar
Double Bar

Select the Double Bar option. These instructions appear at the bottom of the screen:

Position cursor to box corner with ↑↓ ←→. Complete with ↵

In this exercise, use the arrow keys to move the cursor to row 01, column 01, and press Return to mark the starting position.

The screen instructions then display:

Position cursor to other corner with ↑↓ ←→. Complete with ↵

Now follow these steps:

- ✓ Press ↓ twice.
- ✓ Press → until the cursor gets to row 03, column 68.
- ✓ Press ↵ to finish marking the corner.
- ✓ Move the cursor inside the box that appears on the screen.
- ✓ Type **Add/Change Mailing List Data**, or create your own title.

Now your screen has a boxed title.

If you are feeling particularly ambitious or if the people who will use your screen are novices, you may want to try adding your own Help Menu to your custom data-entry screen. Figure 7.6 shows some help information added to the bottom of the ADDNAMES form. Create the box with Double Bar and Single Bar lines. Type the rest of the text just as you see it in Figure 4.17.

Saving the Form

By this time you've put quite a bit of work into your form. So, you might want to save the form now, just to play it safe. To do so, move the highlighting to the Exit option. (If the menu is not showing, press F10 first.) Then select the Save option. You'll be returned to the main menu.

To resume working on a form, (or to make changes in the future), highlight the Modify option from the main menu, and select Format. Select a drive and the form name (ADDNAMES.SCR in this example). Then press F10 to view the blackboard.

Templates

Templates are used to reduce the likelihood of errors being typed into forms, as well as to simplify data entry. For example.

you can add a template like this to the PHONE field:

:() - :

Because the parentheses and hyphen are already in the field, the template encourages a standardized format for entering telephone numbers and simplifies their entry.

To add this template to the ADDNAMES form, first move the cursor to the first character in the PHONE field highlighting. Then call up the menu (F10) and highlight the Modify option. At the bottom of the pull-down menu, you'll see these options:

Picture Function
Picture Template
Range

Set Up Modify Options Exit 09:03:57 am

Add/Change Mailing List Data

First Name : xxxxxxxxxxxx Last Name : xxxxxxxxxxxx

Address : xxxxxxxxxxxxxxxxxxxx

City : xxxxxxxx State : xx Zip Code : xxxxxxxx

Phone : xxxxxxxx

Cursor Move : Up Left Right Down arrow keys	Page Move : Next : PgDn Previous : PgUp	Delete : Letter : Del Field : ^Y Record : ^U	Done : Save Work : ^End Abandan : Esc
---	---	---	---

MODIFY SCREEN <C: C:ADDNAMES.SCR Rec : 1/5 Ins

Enter text. Drag field or box mader cursor with , F19 for space

Screen field definition blackboard

Figure 4.17: ADDNAMES Form with Custom Help Menu

Select the Picture Template option. You'll see a menu of special characters that you can use in your picture template. These characters are explained in Table 4.8.

Special Character	Effect
A	Allows letters only to be entered.
L	Allows logical data only (T, F, Y, or N).
X	Allows entry of any character.
#	Allows numbers and + or - signs.
9	Allows numeric digits only.
!	Converts entry to uppercase.
other	Adds any other character to the entry.

Table 4.8: Special Template Character

dBASE will also ask you for the template. Type this template:

(999)999-9999

Notice that you've specified that only numbers can be entered into the field (by placing the special character 9 into the template). Therefore, the template also helps to prevent mistakes in entering data.

Let's discuss a few other tricks that you could do with picture templates. For example, if you put the template.

!AAAAAAAAAAAA

in the LNAME field highlighting, and the template

!AAAAAAA

in the FNAME field highlighting, both of these field would automatically switch the first letter of the entry uppercase. The special A characters will allow the other characters to remain as they were entered.

Placing the template

!!

into the STATE field automatically converts any state abbreviation to all uppercase. Placing the template

#####

into the ZIP field allows numbers, spaces, plus (+) and minus (-) signs to be entered for the zip code. (If your mailing list includes foreign zip codes, however, you'll want to stay with the XXXXXXXXXX template which allows any characters.) Figure 4.18 shows how the ADDNAMES screen form looks with all these new picture templates entered.

The screenshot shows the 'ADDNAMES' screen form with the following fields and templates:

- First Name : AAAAAAAAAAAAA
- Last Name : AAAAAAAAAAAAA
- Address : xxxxxxxxxxxxxxxxxxxxxx
- City : xxxxxxxx
- State : !!
- Zip Code : #####
- Phone : 0999823542

Below the fields is a control grid:

Cursor Move :	Page Move :	Delete :	Done :
Up	Next : PgDn	Letter : Del	Save Work : ^End
Left Right	Previous : PgUp	Field : ^Y	Abandon : Esc
Down		Record : ^L	
arrow keys			

At the bottom, a status bar displays:

MODIFY SCREEN || <C: || C:ADDNAMES.SCR || Rec : 1/5 || Ins ||

Enter text. Drag field or box mader cursor with ← F19 for space

Field : MAIL-> PHONE Type : Character Width : 13 Picture

Figure 4.18: ADDNAMES Form with Picture Templates

When you're done adding the picture templates to the ADDNAMES form, highlight the Exit option and select Save. Now let's look at ways to put the custom form to work.

- To use a custom form to add or edit database records, you must first remember to open the appropriate database (that is, the one you used to design the form in the first place).

Using a Custom Form

To use a custom form, you simply highlight the Set Up option from the main menu, and select the Format for screen option. dBASE will ask for the drive and file name. In this example, select the appropriate drive, and select ADDNAMES.FMT from the menu of form names.

The custom form will now be displayed whenever you add new records, or edit through the EDIT mode. For example, if you highlight the Update option and select Edit, you'll see the first record from your database displayed on your custom screen, as in Figure 4.19.

Add/Change Mailing List Data			
First Name :	Betso	Last Name :	Smith
Address :	222 Lemon Dr.		
City :	New York	State :	NY Zip Code :
Phone :	0999823542		
Cursor Move :	Page Move :	Delete :	Done :
Up	Next : PgDn	Letter : Del	Save Work : ^End Abandon : Esc
Left Right	Previous : PgUp	Field : ^Y	
Down		Record : ^U	
arrow keys			

EDIT	<C:>	MAIL	Rec : 1/5	Ins
------	------	------	-----------	-----

Figure 4.19: MAIL.DBF Record Display on Custom Screen

All keys work with the new form as they did in the normal APPEND and EDIT modes. That is, the arrow keys move the cursor, Del and ^ Y delete characters and fields, PgUp and PgDn scroll up and down through records, and ^ End saves your work and returns to the Assistant menu.

Using other screen painter options

There are several other options available in the screen painter which we'll discuss briefly now. Some of these features are quite advanced.

The Modify Menu Options

While working on the blackboard, you can place the cursor in any field, and press F10 to call up the Screen Painter menu. If you highlight the Modify option from the Screen Painter

menu, you'll see a pull-down menu that looks something like this:

Screen Field Definition

Action: Edit/GET

Source: MAIL

Content: FNAME

Type: Character

Width: 10

Decimal:

Picture Function:

Picture Template:

Range:

Let's briefly discuss what each of these options provides:

- Action:** Toggles between Edit/GET and Display/SAY. In the Edit/GET mode, the field can be modified in the EDIT and APPEND modes. In the Display/SAY mode, data are only displayed and cannot be changed in either APPEND or EDIT mode. The latter is most commonly used in advanced applications where a field value, such as an account number, is fixed and can not be changed.
- Source:** *Source* is the name of the file that the field is from and this option cannot be changed.
- Content:** Content is the name of the field associated with the field highlighting that contains the cursor.
- Type:** This is the data type, as defined in the database structure (Character, Numeric, Date, Logical, or Memo).
- Width:** The field width as defined in the database structure. The number that you enter will change the width of the field definition in the database structure, so use this option with caution.
- Decimals:** The number of decimal places used with Numeric data.
- Picture Function:** Like picture templates, but only requires that you enter a single character. Options are: ! (convert all letters to uppercase). A (allow only alphabetic characters to be entered into field), D (display dates in American mm/dd/yy format—for Date data type only), E (display dates in European dd/mm/yy format—for Date data type only). The S option limits the size of a field display on a form, but allows data to scroll horizontally within it. Hence, if you use the picture function S5 in the LNAME field, then the highlighting on the screen will only be five characters wide when entering or editing data. However, longer names will scroll through

the five-character "window." The R option, rarely used, displays literals in a date, even if a nod-Date data type is used. For example, picture function RE will request data in DATE mode, even if a field is numeric.

Picture

Template: Allows standardized entry formats and data-entry checking. See Table 4.8 for option.

Range: Used with Numeric and Date data only, this option allows you to define a highest and lowest acceptable range of entries.

Adding, Changing, Deleting Fields

The width option from the Modify pull-down menu allows you to change the width of a field. However, this option makes the same change to the database structure. More often than not, you'll simply want to change the size of a field on a form for aesthetic reasons.

To increase the size of a field highlighting on the data-entry screen without affecting the database structure, follow these steps:

- ✓ Call up ADDNAMES.SCR.
- ✓ Move the cursor to the field that you want to lengthen.
- ✓ Press the Ins key until the field looks wide enough.
- ✓ Press ^ End to save the form.

Similarly, to decrease the field width on the data-entry screen and not in the database structure, follow these steps but press the Del key instead of the Ins key.

To add a new field from your database to this form, follow these steps:

- ✓ Call up ADDNAMES.SCR.
- ✓ Move the cursor to the additional field location.
- ✓ Press F10.
- ✓ Highlight Modify on the Screen Painter menu.
- ✓ Select the Content option.
- ✓ Select the field name from the submenu that appears.
- ✓ Highlight the Width option, and enter a width.
- ✓ Finally, press F10 to return to the blackboard.

To delete a field from a form, move the cursor to the appropriate field highlighting and type ^ U. dBASE will ask if you want to delete the field from the database structure as well. If you answer Yes, all data in that field will be lost. Therefore, answer No unless you are absolutely certain that you will never need this information in your database.

Creating Screens from the Dot Prompt

If you begin working from the dot prompt, there are several simple techniques you can use to handle forms. First, the command **MODIFY SCREEN <file name>** will enter the Screen Painter. For example, entering the command

```
MODIFY SCREEN ADDNAMES ↵
```

will allow you to create a screen format named **ADDNAMES.SCR** (screens always have the extension **.SCR**).

The Screen Painter automatically creates a separate file with the same name but the extension **.FMT** (**ADDNAMES.FMT**). This is the file dBASE actually uses to display forms on the computer's screen. The **.SCR** file is used only while you are designing a screen through the Screen Painter.

From the dot prompt, you can activate a **.FMT** file with the **SET FORMAT** command. For example, the commands below open the **MAIL.DBF** database with the **NAMES.NDX** and **ZIP.NDX** index files. Then, the **SET FORMAT** command opens the **ADDNAMES.FMT** format file:

```
SET DEFAULT TO B ↵  
USE MAIL INDEX NAMES, ZIP ↵  
SET FORMAT TO ADDNAMES ↵
```

To add new records to the **MAIL.DBF** database using the custom screen, you merely need to enter the command

```
APPEND ↵
```

at the dot prompt.

To edit data on the **MAIL.DBF** database, simply enter this command at the dot prompt:

```
EDIT ↵
```

Optionally, you can specify a record number:

```
EDIT 4 ↵
```

You can use a **FOR** condition with **EDIT** as well. For example,

```
EDIT FOR LNAME = "Doe" .AND. FNAME = "Ruth" ↵
```

will edit all records with **Ruth Doe** as the name.

To stop using the custom form and return to the normal APPEND and EDIT screens, simply enter this command at the dot prompt:

CLOSE FORMAT ←

Reviewing Custom Data-entry Screens

In this chapter, you have learned many techniques for creating custom data-entry screens:

To create a custom form for a database, select the Create and Format options from the Assistant menu screen.

To move fields on the Screen Painter blackboard, place the cursor inside the field highlighting that you want to move, and press Return. Use the arrow keys to move the cursor to the new location, and press Return to complete the move.

To change field labels on a form, simply position the cursor and type over existing labels. Use the arrow, Ins, and Del keys to move the cursor and add/delete text.

To draw boxes on the form, select Draw a window or line from the Screen Painter options menu.

To save a custom form, highlight the Exit option from the Screen Painter menu and select Save.

To add templates to a field highlight, move the cursor to the first character in the field and call up the menu (F10). Select the *Picture Function* or *Picture Template* option.

To use a custom form, highlight the Set Up option from the main menu bar and select the Format for screen option. The custom form will be used in the APPEND and EDIT modes when selected from the menu.

To change the size of a field highlight on the screen, move the cursor to within the field and press Ins to increase its length, or Del to decrease it. Press ^ U to delete the field altogether.

From the dot prompt, the commands MODIFY SCREEN, SET FORMAT TO, APPEND, EDIT, and CLOSE FORMAT access a custom screen.

Section 4.4 Managing Numbers and Dates

Up to this point, our database has contained only Character data. Eventually, you'll probably want to include dates and numbers in a database. There are several techniques for managing dates and numbers, and for performing basic arithmetic on them. These are the topics of this chapter.

Creating a Sales Database

For the exercises in this chapter we'll create a database named SALES.DBF. From the Assistant menu, highlight the Create option and select Database file. Select the appropriate drive for your computer, and type the file name SALES. Define the database fields structure as shown in Figure 4.20.

Bytes remaining : 3955

CURSOR <-- -->	INSERT	DELETE	Up a field : ↑
Char : ← →	Char : Ins	Char : Del	Down a field : ↓
Word : Home End	Field : ^N	Word : ^Y	Exit/Save : ^End
Pan : ^I ^I	Help : F1	Field : ^U	Abort : Esc

Field Name	Type	Width	Dec
1 CODE	Character	5	
2 TITLE	Character	15	
3 QTY	Numeric	5	8
4 AMOUNT	Numeric	12	2
5 DATE	Date	8	
6 ZIP	Character	18	

MODIFY STRUCTURE || <C:>|| MAIL || Field : 6/6 || Ins ||

Enter the field name.

Field name begin with a letter and may contain letters, digits and underscores

Figure 4.20: SALES.DBF Database Structure

You have defined five fields for your SALES.DBF database. Notice that CODE and TITLE are Character data fields. Only Numeric and data with a maximum of five digits (99999) and no decimal places can be entered in your QTY (quantity) field. AMOUNT is also a Numeric data field, but its maximum width is twelve with two decimal places (for pennies). dBASE automatically assigns a width of eight characters (mm/dd/yy) to the DATE field.

Once you've entered the database structure, press Return rather than defining a sixth field, or press ^ End to save the structure. When dBASE asks you to confirm the new structure, press Return. When dBASE asks if you want to add new data now, answer Yes and fill in the following sample records. (Note: You need not enter the record numbers because dBASE keeps track of those automatically.)

Record#	CODE	TITLE	QTY	AMOUNT	DATE
1	AAA	Rakes	3	15.00	03/01/86
2	BBB	Hoes	2	12.50	03/01/86
3	CCC	Shovels	3	21.50	03/01/86
4	AAA	Rakes	2	10.00	03/01/86
5	CCC	Shovels	4	26.50	03/01/86
6	AAA	Rakes	2	11.00	03/02/86
7	CCC	Shovels	1	7.50	03/01/86
8	BBB	Hoes	2	12.50	03/02/86
9	AAA	Rakes	5	23.50	03/02/86

Summing Numbers

To sum a numeric field, highlight the Retrieve option from the Assistant menu, and select Sum. If you immediately select Execute the command after selecting Sum, dBASE will display the sum of all numeric fields for every record in the database:

9 records summed

QTY AMOUNT

24 140.00

You can use the Construct a field list option from the submenu to specify that only certain fields be summed. For example, suppose that you want to know the sum of sales to date. The steps that tell dBASE to add only the numbers in the AMOUNT field are out lined below.

- ✓ Highlight **Retrieve** from the main menu bar.
- ✓ Select **Sum**.
- ✓ Select **Construct a field list**.
- ✓ From the menu of field names, select **AMOUNT**.
- ✓ Press → to finish selecting fields.
- ✓ Select **Execute the command**.

Then dBASE will add of the numbers you've entered in the AMOUNT field and display the results:

9 records summed

AMOUNT

140.00

By using the Build a search condition option from the Assistant menu you can also tell dBASE that only certain records be included in the sum. For example, suppose that you want to know how many rakes you have sold. To find out, ask dBASE to sum the QTY field for records with Part Number AAA. Here are the steps:

- ✓ Highlight the Retrieve option from the main menu bar.
- ✓ Select Sum.
- ✓ Select Construct a field list.
- ✓ Select QTY.
- ✓ Press → to finish selecting fields to sum.
- ✓ Select Build a search condition.
- ✓ Select CODE.
- ✓ Select =.
- ✓ Type AAA.
- ✓ Press ←
- ✓ Select No more conditions.
- ✓ Select Execute the command.

With these menu selections, you have built

Command: SUM QTY FOR CODE "AAA"

dBASE will display the result

4 records summed

QTY

12

indicating that the sum of the QTY field for records with CODE AAA is twelve. In other words, you've sold twelve rakes.

If you are interested in sales on a certain date, follow these steps:

- ✓ Highlight the Retrieve option from the main menu bar.
- ✓ Select Sum.
- ✓ Select Build a search condition.
- ✓ Select DATE.
- ✓ Select =.
- ✓ Type 03/02/86.
- ✓ Select No more conditions.
- ✓ Select Execute the command.

dBASE will show the sum of the QTY and AMOUNT fields for wall records with dates of 03/02/86:

3 records	summed
QTY	AMOUNT
9	47.00

Of course, you can be even more specific in your sums by using .AND. and .OR. in your search condition. For example, if you want to know how many AAA rakes you sold on March 1, 1986, and how much these sales came to, follow these steps:

- ✓ Highlight Retrieve from the main menu bar.
- ✓ Select Sum.
- ✓ Select Build a search condition.
- ✓ Select CODE.
- ✓ Select =.
- ✓ Type AAA.
- ✓ Press ←!
- ✓ Select Combine with .AND.
- ✓ Select DATE.
- ✓ Select =.
- ✓ Type 03/01/86.
- ✓ Select No more conditions.
- ✓ Select Execute the command.

dBASE displays the sum of the QTY and AMOUNT fields for sales of part number AAA on March 1, 1986:

2 records summed	
QTY	AMOUNT
5	25.00

Averaging Numeric Amounts

The Average option from the Retrieve menu works in exactly the same way as the Sum option, but it calculates an average rather than a sum. For example, if you select Average and Execute the command, dBASE will display

9 records averaged	
QTY	AMOUNT
3	15.56

the average QTY and AMOUNT of all nine records.

You can use the Construct a field list option from the menu to specify numeric fields to average. Of course, you can use the build a search condition option to average only records, such as Part Number AAA or Date = 03/01/86, just as you did with the Sum option.

Counting Records

The Count option works on any data type because it simply counts the number of occurrences in records. For example, if you highlight the Retrieve option and select Count,

and then select Execute the command, dBASE will display

9 records

indicating that there are nine records on the database.

Suppose that you want to know how many sales records about BBB hopes have been entered into your database. If you select Count and Build a search condition from the menus, and specify CODE = BBB, dBASE will display

2 records

because only two records in the database have the code BBB.

Printing Totals and Subtotals in Reports

You can also total and subtotal columns of information in dBASE III PLUS reports. The most important point to keep in mind with subtotals, however, is that the database must either be sorted or indexed by the field which determines how the records are subtotals. For example, if you want your sales report to present quantity and price totals and subtotals for each product (the CODE field), you need to first index the SALES.DBF database by the CODE field. Here are the steps to create a CODE.NDX index file from your SALES.DBF database

- ✓ Highlight the Organize option.
- ✓ Select Index.
- ✓ Type **CODE**.
- ✓ Press ←
- ✓ Select a drive.
- ✓ Type **CODE** as the name for the index file.
- ✓ Press ←

When the indexing is complete, press any key to return to the Assistant menu screen. Now you can develop a report using the report generator. Let's build one now, just the practice. First, highlight the Create option, and select Report. Select a drive for storing the report format, and type the report file name SUBTOTS.

Fill in these options on the first report menu:

Page title	Sales by Product Code]
Page width (positions)	80
Left margin	0
Right margin	0
Lines per page	58
Double space report	No
Page eject before printing	Yes
Page eject after printing	No
Plain page	No

Next, highlight the Group, option and specify CODE as the field to group. Also, you can assign the group title Product Code.

Group on expression	CODE
Group heading	Product Code
Summary report only	No
Page eject after group	No
Sub-group on expression	
Sub-group heading	

Next you'll want to fill in individual columns. Begin by highlighting the Columns option and defining the first report column:

Contents	CODE
Heading	Part No.
Width	8
Decimal places	
Total this column	

Press PgDn, and define the second report column:

Contents	TITLE
Heading	Part Name
Width	15
Decimal places	
Total this column	

Press PgDn again, and fill in the specifications for the third report column:

Contents	QTY
Heading	Qty
Width	5
Decimal places	0
Total this column	Yes

Notice that the Total this column option is marked Yes, which means that this numeric field will be automatically totaled and subtotaled in the printed report.

Press PgDn. The Total this column option is marked Yes in the fourth column definition so that the AMOUNT field will be totaled and subtotaled.

Contents	AMOUNT
Heading	Amount
Width	12
Decimal places	2
Total this column	Yes

Finally, press PgDn and place DATE in the last column of the report by filling in these options:

Contents	DATE
Heading	Date
Width	8
Decimal places	
Total this column	

After the report format is defined, your screen should look like Figure 4.21. Highlight the Exit option and select Save to store the report format and return to the Assistant menu.

Figure 4.21 : Specifications for SUBTOTS.FRM

Now, to print the report, highlight the Retrieve option on the main menu bar and select Report. Select the appropriate drive and the SUBTOTS.FRM report. Select Execute the command. Your report will look like Figure 4.22 with subtotals and totals of sales for each product.

Sub-subtotals in a Report

You can subtotal on two separate fields in a report, so that you get subtotals of subtotals. For example, suppose that you want your report to subtotal each day's sales, but you also want subtotals for each product within each date. As with other subtotals, the database needs to be presorted or indexed on the two subtotalling fields. For this example, you need to create an index file of dates and product codes, and here is where data types tend to get a little tricky.

If you attempt to index on DATE + CODE, you'll see the error message

Date Type Mismatch

because the DATE field is the Date data type and CODE is the Character data type . dBASE won't let you combine mismatched data types. However, if you go through the usual process of creating an index file, but enter the expression

DTOC(DATE) + CODE ↵

when dBASE asks for the index expression, everything will be fine. The DTOC function changes the Date data type to the Character data type, but only in the index file. In other words, there is no longer a data type mismatch error, but your database still accepts

Sales by Product Code

Part No.	Part Name	Qty	Amount	Date
* * Product				
AAA	Rakes	3	15.00	03/01/86
AAA	Rakes	2	10.00	03/01/86
AAA	Rakes	2	11.00	03/02/86
AAA	Rakes	5	23.50	03/02/86
* * Subtotal * *		12	59.50	
* * Product				
BBB	Hoes	2	12.50	03/01/86
BBB	Hoes	2	12.50	03/02/86
* * Subtotal * *		4	25.00	
* * Product				
CCC	Shovels	3	21.50	03/01/86
CCC	Shovels	4	26.50	03/01/86
CCC	Shovels	1	7.50	03/01/86
* * Subtotal * *		8	55.50	
* * * Total * * *				
		24	140.00	

Figure 4.22: Subtotaled Report Organized by Code

only dates entered in this field. (We'll discuss DTOC and other functions in more detail later in the chapter.)

Next, just create the report format in the usual fashion using Create and Report from the Assistant menu screen. Name the report DATECODE. You can fill out the first menu of report options as shown here, or use your own formatting:

Page title	Sales by Date and Product Code
Page width (positions)	80
Left margin	0
Right margin	0
Lines per page	58
Double space report	No
Page eject before printing	Yes
Page eject after printing	No
Plain page	No

Next, highlight the Group option, and specify DATE as the field to group, and CODE as the Sub-group:

Group on expression	DATE
Group heading	Date
Summary report only	No
Page eject after group	No
Sub-group on expression	CODE
Sub-group heading Product	Code

(Note: If you change the *Page eject after group* option to Yes, each day's sales will be printed on a separate page.)

Finally, use this summary to define the columns for your DATECODE report:

Contents	Heading	Width	Decimal Places	Total?
DATE	Date	12	0	No
CODE	Part No.	5	0	No
TITLE	Part Name	15	0	No
QTY	Qty	5	0	Yes
AMOUNT	Amount	12	2	Yes

These column specifications appear at the bottom of your screen:

Report Format

Date	Part No.	Part Name	Qty	Amount
mm/dd/yy	XXXXX	XXXXXXXXXXXXXXXXXX	#####	#####.##

After you save your DATECODE report form, select the Retrieve and Report options, and specify DATECODE.FRM to display the report shown in Figure 4.23. The report presents each day's sales subtotals for each product.

Sales by Date and Product Code

Date	Part No.	Part Name	Qty	Amount
* * Date 03/01/86				
* Product Code	AAA			
03/01/86	AAA	Rakes	3	15.00
03/01/86	AAA	Rakes	2	10.00
* Subsubtotal *			5	25.00
* Product Code BBB				
03/01/86	BBB	Hoes	2	12.50
* Subsubtotal *			2	12.50
* Product Code CCC				
03/01/86	CCC	Shovels	3	21.50
03/01/86	CCC	Shovels	4	26.50
03/01/86	CCC	Shovels	1	7.50
* Subsubtotal *			8	55.50
* * Subtotal * *				
			15	93.00
* * Date 03/02/86				
* Product Code	AAA			
03/02/86	AAA	Rakes	2	11.00
03/02/86	AAA	Rakes	5	23.50
* Subsubtotal *			7	34.50
* Product Code BBB				
03/02/86	BBB	Hoes	2	12.50
* Subsubtotal *			2	12.50
* * Subtotal * *				
			9	47.00
* * * Total * * *				
			24	140.00

Figure 4.23: Report Subtotaled by Date and Product Code

Creating Summary Reports

Under the *Group* option, you may have noticed the Summary report only suboption on the report generator menu. Changing this option to Yes displays only the subtotals without the data used in the calculations. To test this option, slightly change the SUBTOTS.FRM report format.

Remember that the SUBTOTS.FRM report format required presorting the database by product CODE, and that you created an index file named CODE.NDX to perform the sort. Before you reprint the report, you'll need to reactivate that index file. Here are the steps:

- ✓ Highlight Set Up from the main menu bar.
- ✓ Select Database file.
- ✓ Select a drive.
- ✓ Select SALES.DBF.
- ✓ Answer Yes to the "Is the File Indexed?" prompt.
- ✓ Select CODE.NDX.
- ✓ Press → to finish selecting database and index files.

Now the SALES.DBF database is sorted by the CODE field. Here are the steps to change the SUBTOTS.FRM report to a summary report.

- ✓ Highlight the *Modify* option from the main menu bar.
- ✓ Select *Report*.
- ✓ Select a drive.
- ✓ Specify SUBTOTS.FRM.
- ✓ Highlight the *Groups* option from the report menu.
- ✓ Highlight the *Summary report only* option.
- ✓ Press ← to switch the option to Yes.

At this point, your menu selections look like this:

Group on expression	CODE
Group heading	Product Code
Summary report only	Yes
Page eject after group	No
Sub-group on expression	
Sub-group heading	

Now, save the report format with the Exit and Save options. From the Assistant menu screen, select the Retrieve and Report options, and specify SUBTOTS.FRM Select Execute the command. Your summary report of product sales will look like Figure 4.24.

Sales by Product Code

Part No.	Part Name	Qty	Amount	Date
* * Product Code AAA				
* * Subtotal * *				
		12	59.50	
* * Product Code BBB				
* * Subtotal * *				
		4	25.00	
* * Product Code CCC				
* * Subtotal * *				
		8	55.50	
* * * Total * * *				
		24	140.00	

Figure 4.24: Summary Product Sales Report

Using Dot-prompt Commands

As we've discussed before, working from the dot prompt does require some fluency with the dBASE language and may go beyond your requirements. However, to fully explore techniques for managing numbers and dates, you'll need to use some commands from the dot prompt for options that the Assistant menu cannot provide.

To practice these dot-prompt commands, press Esc to get rid of the Assistant menu. Then, to make sure that the SALES.DBF database is in use, enter the command:

USE SALES ↵

if you're using a hard-disk system, or

USE B:SALES ↵

if you're using a computer with two floppy-disk drives.

Note : If you do not see the results of a SUM, COUNT, or AVERAGE command at the dot prompt, type the command **SET TALK ON** and press ↵ Then press ↑ twice to re-access your command, and ↵ to execute it.

To sum a numeric field from the dot prompt, you use the SUM command with the names of the fields to sum. For example, if you ask dBASE to add the data in the QTY field of your database records with the command

```
SUM QTY ↵
```

this answer displays just as though data were summed from the menu options:

9 records summed

QTY

24

To specify certain records in the sum, use the FOR option in the command:

```
SUM QTY FOR CODE = "AAA" ↵
```

This command sums the QTY field for records that have AAA as the product CODE. dBASE displays this answer:

4 records summed

QTY

12

The COUNT command is used in a similar fashion, and performs the same task as the Count option from the Assistant menu. For example, the command

```
COUNT FOR CODE &LT> "AAA" ↵
```

displays the number of records in the database that do not have part number AAA.

5 records

Now let's look at a more advanced technique from the dot prompt, using the AVERAGE command.

If you want the average selling price of part AAA, you need to take into account the quantity sold for each transaction. That is, the average selling price is the average of the amounts divided by the quantities. So, tell dBASE to average the amount divided by the quantity for part number AAA with the following command:

```
AVERAGE (AMOUNT/QTY) FOR CODE = 'AAA' ↵
```

Note : The basic dBASE math operators are + (add), - (subtract), / (divide), and * (multiply). Chapter 11 provides additional information.

(The / symbol stands for divided by, as in the fraction 3/4.) dBASE displays

4 records averaged
(AMOUNT/QTY)
5.05

The average selling price for part number AAA is \$5.05. In some cases you might prefer to know how many records contain certain information, rather than knowing about sums of fields. For example, you might wish to know how many transactions involved part AAA. You use the COUNT command for this. To find out how many records on the database have AAA as the part number, ask dBASE to

```
COUNT FOR CODE = 'AAA' ↵
```

dBASE will tell us

4 records

There are four records with part AAA.

Using Commands to Manage Dates

In our SALES database, we've included a field named DATE, and assigned it the Date data type. dBASE III PLUS contains many functions for managing dates. The (DATE()) function displays today's date, as it was typed when you first booted up your system. For example, if you ask dBASE what the date is *using the ? command)

```
? DATE() ↵
```

you'll see today's date on the screen. To change that date, type this command:

```
RUN DATE ↵
```

The screen will display the current date and allow you to change it:

```
Current date is Tue 11-06-1986  
Enter new date: _
```

To try out some new exercises, fill in the new date as 03/01/86, and press Return:

```
Current date is Tue 11-06-1986  
Enter new date: 03/01/86 ↵
```

Now, let's try out some new exercises. First, with the SALES database still in use, ask dBASE to display all records that match today's date, using the command

```
LIST FOR DATE = DATE() ↵
```

dBASE displays

Record#	CODE	TITLE	QTY	AMOUNT	DATE
1	AAA	Rakes	3	15.00	03/01/86
2	BBB	Hoes	2	12.50	03/01/86
3	CCC	Shovels	3	21.50	03/01/86
4	AAA	Rakes	2	10.00	03/01/86
5	CCC	Shovels	4	26.50	03/01/86
7	CCC	Shovels	1	7.50	03/01/86

Next, ask dBASE to display all records that have dates that are later than today's date:

LIST FOR DATE > DATE() ↵

These records appear on your screen:

Record#	CODE	TITLE	QTY	AMOUNT	DATE
6	AAA	Rakes	2	11.00	03/02/86
8	BBB	Hoes	2	12.50	03/02/86
9	AAA	Rakes	5	23.50	03/02/86

The dBASE functions for managing dates are explained in Table 4.9.

Function	Purpose
CDOW	Day of week as a Character type (Monday).
CMONTH	Month as a Character type (January).
CTOD	Character-to-date conversion.
DAY	Day of month (3.1)
DOW	Day of week as number (Sunday = 1. Monday = 2. etc.).
DTOC	Date-to-character conversion.
MONTH	Month as number (1-12).
TIME	Time expressed as HH:MM:SS.
YEAR	Year of date (1985).

Table 4.9 : Date Functions

Test these out with the SALES database. This command asks dBASE to list the date, day of week (DOW), month (CMONTH), day (DAY) and year (YEAR) of each date in the SALES database:

LIST DATE,CDOW(DATE),CMONTH(DATE),DAY(DATE),YEAR(DATE) ↵

dBASE displays

Record#	DATE:	CDOW(	CMONTH(	DAY(	YEAR(
		DATE)	DATE)	DATE)	DATE)
1	03/01/86	Saturday	March	1	1986
2	03/01/86	Saturday	March	1	1986
3	03/01/86	Saturday	March	1	1986
4	03/01/86	Saturday	March	1	1986
5	03/01/86	Saturday	March	1	1986
6	03/02/86	Sunday	March	2	1986
7	03/01/86	Saturday	March	1	1986
8	03/02/86	Sunday	March	2	1986
9	03/01/86	Sunday	March	2	1986

To view records that fall upon a certain day (as is often, useful when scheduling), you can use the CDOW (Character Day of Week) function:

LIST FOR CDOW(

dBASE displays records with dates that fall on a Saturday (March 1, 1986, is a Saturday):

Record#	CODE	TITLE	QTY	AMOUNT	DATE
1	AAA	Rakes	3	15.00	03/01/86
2	BBB	Hoes	2	12.50	03/01/86
3	CCC	Shovels	3	21.50	03/01/86
4	AAA	Rakes	2	10.00	03/01/86
5	CCC	Shovels	4	26.50	03/01/86
7	CCC	Shovels	1	7.50	03/01/86

To see which records have dates in March, use the CMONTH function:

LIST FOR CMONTH(

dBASE displays

Record#	CODE	TITLE	QTY	AMOUNT	DATE
1	AAA	Rakes	3	15.00	03/01/86
2	BBB	Hoes	2	12.50	03/01/86
3	CCC	Shovels	3	21.50	03/01/86
4	AAA	Rakes	2	10.00	03/01/86
5	CCC	Shovels	4	26.50	03/01/86
6	AAA	Rakes	2	21.00	03/02/86
7	CCC	Shovels	1	7.50	03/01/86
8	BBB	Hoes	2	12.50	03/02/86
9	AAA	Rakes	5	23.50	03/02/86

To display a specific date, you need to convert the date in the database to a Character type with the DTOC function. This command lists all records with the date March 2:

```
LIST FOR DTOC(DATE) = "03/02" ←
```

dBASE displays

Record#	CODE	TITLE	QTY	AMOUNT	DATE
6	AAA	Rakes	2	11.00	03/02/86
8	BBB	Hoes	2	12.50	03/02/86
9	AAA	Rakes	5	23.50	03/02/86

Incidentally, in case you should need to know the time at any given moment, you can use this TIME function:

```
? TIME() ←
```

dBASE will display the time in HH:MM:SS format:

14:26:18

(That's 2:26 pm, plus 18 seconds.) The time that dBASE displays is based upon the time you keyed in when you first booted up the system. To change the time, enter this command:

```
RUN TIME ←
```

Date Arithmetic

You can perform data arithmetic, too. For example, if you want to know at what date the amounts in the SALES file became ninety days overdue, just add 90 to each of the dates:

```
LIST DATE + 90 ←
```

dBASE displays

Record#	DATE + 90
1	05/30/86
2	05/30/86
3	05/30/86
4	05/30/86
5	05/30/86
6	05/31/86
7	05/30/86
8	05/31/86
9	05/31/86

Let's reverse the order of the operation a bit. Suppose that today is May 30, and you want to know which records on the SALES database are 90 or more days past due. You want to list for those dates in which the difference between the current date minus the database DATE was greater than or equal to ($> =$) 90:

```
LIST FOR CTOD)("05/30/86") - DATE > = 90 ↵
```

dBASE displays the appropriate records:

Record#	CODE	TITLE	QTY	AMOUNT	DATE
1	AAA	Rakes	3	15.00	03/01/86
2	BBB	Hoes	2	12.50	03/01/86
3	CCC	Shovels	3	21.50	03/01/86
4	AAA	Rakes	2	10.00	03/01/86
5	CCC	Shovels	4	26.50	03/01/86
7	CCC	Shovels	1	7.50	03/01/86

When you want to determine how many days have passed between two dates, just subtract the smaller, earlier date from the larger, later date. Make sure the dates are Date data types. If not, use the CTOD function to convert them. For example, suppose you were born on March 31, 1986, and today is November 6, 1986. To find out how many days you've been alive, subtract your birth date from today's date. If you forget to convert the data first and type the equation

```
? "11/06/86" - "03/31/56" ↵
```

you get a strange result:

```
11/06/8603/31/56
```

However, if you remember to use the CTOD conversion function

```
? CTOD("11/06/86") - CTOD("03/31/56") ↵
```

dBASE will tell you exactly how many days have passed between the two dates:

```
11177
```

Similarly, if you forget to do the date conversion when working with a database

```
LIST DATE - "01/01/86" ↵
```

dBASE will produce an error message because you are trying to subtract the character string "01/01/86" from a date (the DATE field). On the screen, you'll see

Date type mismatch

LIST DATE "01/01/86"

Do you want some help? (Y/N)

Just remember to use the CTOD function to convert the character string surrounded by quotation marks ("01/01/86") to date:

LIST DATE - CTOD("01/01/86") ↵

and everything will be fine.

Sorting by Date

If you want your records to be displayed in chronological order, you can index on the DATE field:

INDEX ON DATE TO DATES ↵

When you list the database or print a report, the records will be in order by date:

Record#	CODE	TITLE	QTY	AMOUNT	DATE
1	AAA	Rakes	3	15.00	03/01/86
2	BBB	Hoes	2	12.50	03/01/86
3	CCC	Shovels	3	21.50	03/01/86
4	AAA	Rakes	2	10.00	03/01/86
5	CCC	Shovels	4	26.50	03/01/86
7	CCC	Shovels	1	7.50	03/01/86
8	AAA	Rakes	2	11.00	03/02/86
8	BBB	Hoes	2	12.50	03/02/86
9	AAA	Rakes	5	23.50	03/02/86

In some cases, you might want to combine the date with another field for sorting. For example, you might want the SALES database to be sorted by date, and within common dates be further sorted by product code (CODE). In this case, you need to convert the date to a Character type, using the DTOC function, like you did with the DATECODE report. From the dot prompt, the indexing command looks like this:

INDEX ON DTOC(DATE) + CODE TO DATECODE ↵

If you want the records in descending order by date, that is, from the most-recent to least-recent date, index on the inverse of the date. Specify the DATE field subtracted from a large date, such as 12/31/99:

INDEX ON CTOD("12/31/99") - DATE TO INVDATE ↵

When you list the database, the records will be in descending chronological order:

Record#	CODE	TITLE	QTY	AMOUNT	DATE
6	AAA	Rakes	2	11.00	03/02/86
8	BBB	Hoes	2	12.50	03/02/86
9	AAA	Rakes	5	23.50	03/02/86
1	AAA	Rakes	3	15.00	03/01/86
2	BBB	Hoes	2	12.50	03/01/86
3	CCC	Shovels	3	21.50	03/01/86
4	AAA	Rakes	2	10.00	03/01/86
5	CCC	Shovels	4	26.50	03/01/86
7	CCC	Shovels	1	7.50	03/01/86

Note: The FIND or SEEK command will not work with this index file unless you first subtract the date you are trying to find from CTOD("12/31/99").

Reviewing Dates and Numbers

In this chapter you have learned a number of techniques for managing dates and numbers:

To sum values in a numeric field, select Sum from the **Retrieve** menu. The **Construct a field list** option allows you to sum only certain fields, and the **Build a search condition** option allows you to sum only certain records.

To average values in a numeric field, select the Average option from the **Retrieve** menu. As with Sum, you can use **Construct a field list** and **Build a search condition**.

To count how many records contain an item of information, select the **Count** option from the **Retrieve** menu.

To subtotal information in a report, highlight the **Groups** menu in the report generator. The database needs to be pre-sorted or indexed on the grouping field(s).

To print a summary report, select the **Summary report only** option under the report generator **Groups** menu.

The dot prompt allows many functions for managing dates, including CDOW (Character Day of Week), CMONTH (Character Month), CTOD (Character to Date Conversion), DAY (Numeric Day of Month), DOW (Numeric Day of Week), DTOC (Date to Character Conversion), MONTH (Numeric Month), and YEAR (Year of Date).

Section 4.5 Managing Multiple Date Files

The techniques we've discussed so far are useful for managing a simple database. However, many larger business applications require that information be divided into separate database. In this chapter, we'll look at reasons for dividing information into separate database, as well as techniques for managing information in separate files.

An Accounts Receivable Database Design

Some business applications require multiple databases because any given record in one database might have several items of information related to it. Furthermore, the number of related items might be totally unpredictable (for example, the number of items charged by a given customer during the course of a year). Let's take a look at an example of a database that should be broken into two separate databases.

Figure 4.25 shows a single, poorly designed database for storing accounts receivable. Each record stores a customer's name, address, charge amount, and billing date.

LNAME	FNAME	ADDRESS	AMOUNT	DATE
Adams	Andy	123 A St.	50.00	1/1/86
Adams	Andy	123 A St.	100.00	1/9/86
Adams	Andy	123 A St.	75.00	3/3/86
Miller	Michele	P.O. Box 2345	221.66	2/12/86
Watson	Wilbur	670 Baldy View	21.22	3/3/86
Adams	Andy	123 A St.	66.22	4/7/86
Miller	Michele	P.O.. Box 2345	888.90	4/1/86
Zeepers	Zeppo	261 Ocean View Dr.	99.99	4/1/86
Miller	Arnold	455 Madison Ave	61.55	4/1/86

Figure 4.25: Poor Design for Storing Accounts Receivable Data

The database shown in Figure 4.25 is a very poor design for the application because of the following problems:

1. The database wastes disk space by repeating name and address for each transaction. These repetitions are particularly space-consuming in a database with 10,000 or more records!
2. The database creates extra work for input operators who must type the repeated information each time a new record is added. (For example, each time Andy Adams orders another item, his name and address have to be re-entered into the database.)
3. Because names and addresses would be entered each time a new record is added, the likelihood of typographical errors is high, and therefore the likelihood of billing errors is also high.

A better way to store these data would be to break the information into two databases, as shown in Figure 4.26. Notice that names and addresses are stored only once in the CUSTLIST.DBF database (like the MAIL.DBF database you created earlier), and individual financial transactions are stored in a separate database named CHARGES.DBF.

Notice that both the CUSTLIST.DBF and the CHARGES.DBF databases have a field called CUSTNO (customer number). This field (often called a common field) is used to relate the two databases. Even though you don't see Andy Adams's name and address on any transaction in the CHARGES database, you can tell which transactions belong to him by simply looking for records that have 1001 as the customer number.

The common field that relates the two databases is an important one. For example, had you tried to relate the two databases by last name, then any charge to an individual named Miller would cause LIST.DBF database. Granted, you could relate the two databases by last and first names, but then what if there are two John Does? You could use the address as a tiebreaker, but by this point, you're back to the repetition problem in the CHARGES database.

First Database: CUSTLIST.DBF

CUSTNO	LNAME	FNAME	ADDRESS	CITY
1001	Adams	Andy	123 A St.	Los Angeles
1002	Miller	Michele	P.O. Box 2345.	Burbank
1003	Watson	Wilbur	670 Baldy View	Pomona
1004	Zeepers	Zeppo	261 Ocean View Dr.	Ramona
1005	Miller	Arnold	455 Madison Ave	Newark

Second Database: CHARGES.DBF

CUSTNO	AMOUNT	DATE
1001	50.00	1/186
1001	100.00	1/9/86
1001	75.00	3/3/86
1002	221.66	2/12/86
1003	21.22	3/3/86
1001	66.22	4/7/86
1002	888.90	4/1/86
1004	99.99	4/1/86
1005	61.55	4/1/86

Figure 4.26: Accounts Receivable Data Split into Two Databases

The customer number is the ideal common field because it does not take up much disk space. It is also very easy to assign each customer a unique number. It is important that each record has a unique customer number to relate it to the transactions in the CHARGES.DBF database.

When a database has a field such as CUSTNO (customer number) where each record has a unique value, the field is referred to as a key field. It is the key field that relates the CHARGES.DBF database to the CUSTOMER.DBF database. Let's try some exercises with a couple of databases that could be used to help manage an accounts receivable system. First, highlight Create on the Assistant menu screen, and select the Database file option. Name the database CUSTLIST. Define the structure shown below.

Structure for database: CUSTLIST.DBF

Field	Field Name	Type	Width	Dec
1	CUSTNO	Numeric	4	0
2	LNAME	Character	12	0
3	FNAME	Character	10	0
4	ADDRESS	Character	20	0
5	CITY	Character	15	0
6	STATE	Character	5	0
7	ZIP	Character	13	0
8	PHONE	Character	13	0

Next, add a few hypothetical customers to the database. Assign each customer a unique customer number, as in this sample database:

CUSTNO	LNAME	FNAME	ADDRESS	CITY	ST	ZIP	PHONE
1000	Adams	Andy	123 A St.	San Diego	CA	92122	(123)555-1000
1001	Schumack	Susita	1096 Crest Dr.	Encinitas	CA	92024	(123)555-8903
1002	Norris	Nancy	P.O. Box 1234	La Jolla	CA	91191	(123)555-9910
1003	Davies	David	432 Oaktree Ln.	Los Angeles	CA	92210	(123)555-2323
1004	Teasdale	Rhonda	P.O. Box 2802	Leucadia	CA	92211	(123)555-1212

Next, you'll want to create an index file of customer numbers to help manage the related files. Highlight the Organize option and select Index. Enter the field name CUSTNO as the expression when requested, and name the index file CUSTLIST.

Next, create a database file for storing individual billing transactions, using Create and Database file menu options. Name the file CHARGES, and structure it like this:

Structure for database: CHARGES.DBF

Field	Field Name	Type	Width	Dec
1	CUSTNO	Numeric	4	0
2	PRODUCT	Character	15	0
3	QTY	Numeric	4	0
4	UNIT_PRICE	Numeric	12	2
5	DATE	Date	8	0
6	BILLED	Logical	1	0

Notice that the CUSTNO field in both database has exactly the same name, data type, and width. Note: It is essential for the common field between two databases to have the identical name, type, and width; otherwise, there is no guarantee that the records from the two files will link correctly.

You might also notice that the CHARGES.DBF database has a field with the Logical data type. This data type is always one character wide, and can contain one of two possible conditions: True (.T.) or False (.F.). In this sample database, the Logical field is used to mark records as either having been billed (True) or not billed (False). (When entering data into these fields, you just type F or T without the dots.)

After you've created the CHARGES.DBF database, add a few sample records to it:

Record#	CUSTNO	PRODUCT	QTY	UNIT_PRICE	DATE	BILLED
1	1001	Floppy Disks	10	2.11	03/12/86	.F.
2	1002	8 Mhz Clock	2	16.39	03/15/86	.F.
3	1001	Color Card	1	101.00	04/31/86	.F.
4	1000	Turbo Board	1	550.00	04/01/86	.F.
5	1004	Video Cable	10	16.00	04/15/86	.F.
6	1002	Ram Disk	1	1100.00	04/11/86	.F.
7	1003	8 Mhz Clock	1	16.39	04/15/86	.F.
8	1000	Ram Disk	1	1100.00	04/18/86	.F.
9	1003	Tape Backup	1	1250.00	04/20/86	.F.
10	1003	40 Meg. Disk	1	450.00	04/20/86	.F.

To help set up the relationship between the two databases, and to keep the CHARGES.DBF records in customer number order, create an index file named CHARGES.NDX by highlighting the Organize option and selecting Index. Enter CUSTNO as the key expression for the index file and CHARGES as the index file name.

Now, keep in mind that for general file maintenance such as adding new records or editing existing data, you'll want to treat these as two separate databases. That is, you'll open either database and the appropriate index files by highlighting Set Up and selecting the Database file option from the Assistant menu screen. Then use the Append, Edit, Brows, Delete, and Pack menu options or dot-prompt commands to handle each database.

However, there will be times when you'll want to retrieve information simultaneously from both databases. In that case, you may want to set up a dBASE III PLUS View.

Setting Up a View of Two Files

A View allows you to relate two separate databases to one another through a common field. Then you can select particular fields to display on the screen or in a report. You can create many different Views for databases, and thereby view your data from different "angles."

Opening Multiple Databases

For this example, let's suppose that you want to view billings, but you want to see the name and address of the customer for each transaction. In this case, you'll be working primarily with the CHARGES.DBF database (because you're interested in billings), but will need to get related information (names and addresses) from the CUSTLIST.DBF database. Here is how to open the database and index files you'll need before setting up a View:

- ✓ Highlight the Create option.
- ✓ Select View.
- ✓ Select a drive.
- ✓ Type **ACCTREC** as the name for the view, press NCR.
- ✓ From the menu of database files, select CHARGES.DBF.
- ✓ From the menu of index files, select CHARGES.NDX.
- ✓ Press → to leave the index files submenu.
- ✓ Select CUSTLIST.DBF as the second file for this View.
- ✓ Select CUSTLIST.NDX from the index file submenu.

Triangles will appear on-screen as you select the files.

Specifying the Relationship

When you have selected the databases, you must tell dBASE which field relates the databases. To do so, first use → to highlight the Relate option that appears after your selections from the menu. dBASE shows the names of files that you've opened.

The order in which you open these databases is important. In this case, you are primarily interested in billings, but you also want to "pull" the name and address from the CUSTLIST.DBF database. So, you select CHARGES.DBF, the most important database, first.

dBASE will then show a list of that can be related to CHARGES.DBF (only CUSTLIST.DBF in this example). Press Return to select CUSTLIST.DBF. Next, at the bottom of the screen, dBASE will ask that you define the common field. Press F10 to display the fields submenu, and select CUSTNO. Press Return again. Your screen displays the names of your two databases, your selected file, and this relationship statement:

Relation Chain: CHARGES.DBF→CUSTLIST.DBF

Selecting Fields to View

Now you can select which fields from the two databases you want to see in this View. Use → to highlight the Set fields option. Select CHARGES.DBF to see a submenu of these field names:

- ▶ CUSTNO
- ▶ PRODUCT
- ▶ QTY
- ▶ UNIT_PRICE
- ▶ DATE
- ▶ BILLED

Notice that each field has a ► symbol next to it, indicating that it has already been selected. To use all of these fields, just press → to leave the submenu.

Next, select CUSTLIST.DBF from the menu, and you'll see a submenu of fields from CUSTLIST. You only need LNAME and FNAME for this example, so use ↓ and Return to "unselect" the other fields. The ► symbol is removed from the unselected fields. Figure 4.27 shows how the screen looks when only two fields remain selected from the CUSTLIST.DBF database.

Now you can use → to select the **Exit** option and select **Save**. The View which we've just created will be stored on disk under the file name ACCTREC.VUE. (dBASE automatically assigns the .VUE extension.)

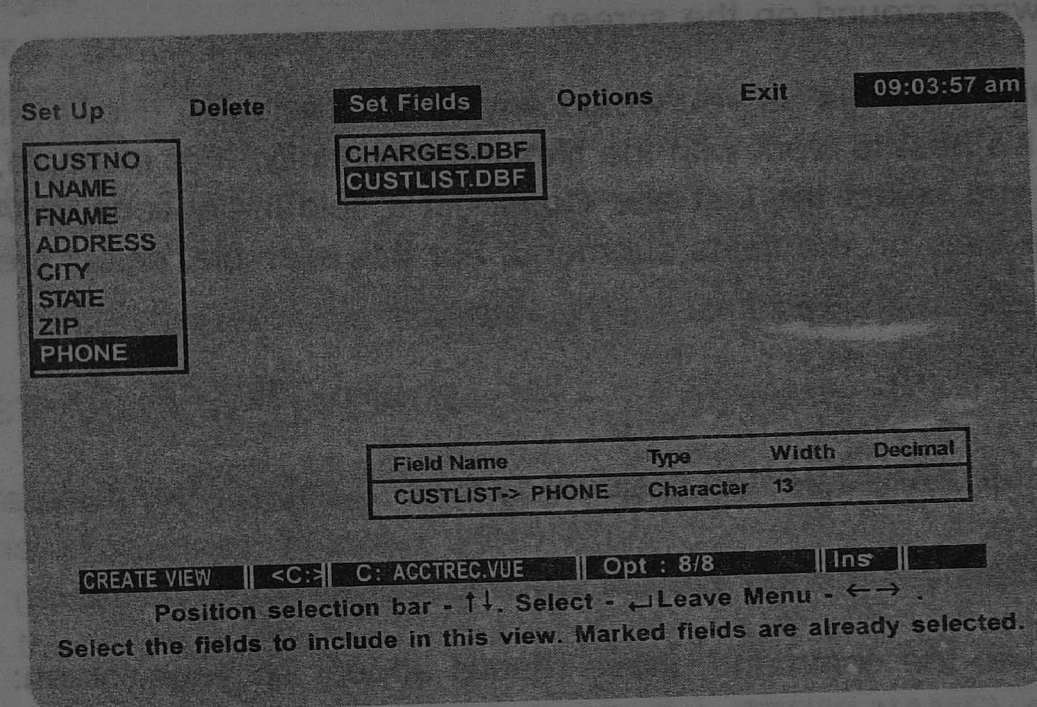


Figure 4.27: Two Fields Selected for Viewing

Opening a View

To use the View, follow these steps from the Assistant menu screen:

- ✓ Highlight the **Set Up** option.
- ✓ Select **View** from the submenu.
- ✓ Select the drive, then ACCTREC.VUE.

Nothing much seems to happen; only the Assistant menu appears. However, if you highlight **Retrieve** and select the list and **Execute** the command options, the contents of the database will appear on the screen just though they were from a single database:

CUSTNO	PRODUCT	QTY	UNIT_PRICE	DATE	BILLED	LNAME	FNAME
1000	Turbo Board	1	550.00	04/01/86	.F.	Adams	Andy
1000	RAM Disk	1	1100.00	04/18/86	.F.	Adams	Andy
1001	Floppy Disks	10	2.11	03/12/86	.F.	Schumack	Susita
1001	Color Card	1	101.00	03/31/86	.F.	Schumack	Susita
1002	8 Mhz Clock	2	16.39	03/15/86	.F.	Norris	Nancy
1002	RAM Disk	1	1100.00	04/11/86	.F.	Norris	Nancy
1003	8 Mhz Clock	1	16.39	04/15/86	.F.	Davies	David
1003	Tape Backup	1	1250.00	04/20/86	.F.	Davies	David
1003	40 Meg. Disk	1	450.00	04/20/86	.F.	Davies	David
1004	Video Cable	10	16.00	04/15/86	.F.	Teasdale	Rhonda

Note: Your data will warp around on the screen.

You can treat the View file as you would any single database. (However, for appending and editing records, it's best to work with the files independently.) For example, if you highlight the Retrieve option, and select the List and Construct a field list options, dBASE will display all the fields from the View on the fields submenu. For this example, select these fields in this order:

CUSTNO
LNAME
FNAME
QTY
UNIT_PRICE

When you execute the command, these results appear on your screen:

Record#	CUSTNO	LNAME	FNAME	QTY	UNIT_PRICE
4	1000	Adams	Andy	1	550.00
8	1000	Adams	Andy	1	1100.00
1	1001	Schumack	Susita	10	2.11
3	1001	Schumack	Susita	1	101.00
2	1002	Norris	Nancy	2	16.39
6	1002	Norris	Nancy	1	1100.00
7	1003	Davies	David	1	16.39
9	1003	Davies	David	1	1250.00
10	1003	Davies	David	1	450.00
5	1004	Teasdale	Rhonda	10	16.00

A much more interesting report can be developed using the report generator with two databases open in a View, as we'll discuss next.

Printing A Report From Two Databases

With the View open, you can design an interesting report using data from both database files. As usual, highlight the Create option and select Report. Select a disk drive, and assign ARREPT as the file name.

For the page formatting, you can simply assign this report name:

Page title	Accounts Receivable Summary Report
Page width (positions)	80
Left margin	8
Right margin	0
Lines per page	58
Double space report	No
Page eject before printing	Yes
Page eject after printing	No
Plain page	No

From the Groups menu, enter this Group on expression statement:

STR(CUSTNO,4) + ": " + TRIM(LNAME) + ", " + FNAME

The expression used to group is wider than the screen allows, but the screen will scroll as you type the expression. You can use ^ PgDn and ^ PgUp to zoom in and out of the highlighting to get more room. Type Customer as the Group heading. Your screen will look something like this:

Group on expression	STR(CUSTNO,4) + ": " + TRIM(LNAME) + ", " + FNAME
Group heading	Customer
Summary report only	No
Page eject after group	No
Sub-group on expression	
Sub-group heading	

You could have just used CUSTNO as the group expression, which would group (subtotal) records by customer number. However, the group heading would show only the customer number as well. Therefore, simultaneously add the customer number, last name, and first name to the group expression, which will make all three fields appear at the top of each subtotaled group.

Notice the use of the STR function in the grouping expression. CUSTNO is Numeric data type, and LNAME and FNAME are both Character data types. As we've discussed before, data types must be identical in a grouping expression which uses the plus (+) sign. Therefore, I've used the STR (STRing) function to change CUSTNO into Character data for the grouping expression. (Character data are also called character strings, or just strings for short. The STR function gets its name from the word STRing, because it converts a number to a character

string.) The number 4 tells dBASE that the converted number should be four characters long. To separate the customer number from the customer name, add a literal colon and a couple of spaces enclosed in quotation marks (" "). The TRIM function makes sure that the customer name does not come out looking like this in the report:

Adams Andy

Use a literal comma and space enclosed in quotation marks (" ") to separate the last and first names, so they look like this in the report:

Adams, Andy

Most of the individual columns for the report were defined using usual techniques from the Columns option. The arrow (→) portion of each column definition is supplied automatically by dBASE when you use the F10 menu to select fields for the Contents of the column. The CHARGES→ symbol tells dBASE to take the DATE field from the CHARGES database.

To define the first column, highlight the Columns option, select Contents, press F10, and select DATE from the field submenu.

Then enter the heading DATE:

Contents	CHARGES→ DATE
Heading	Date
Width	8
Decimal places	
Total this column	

Press PgDn and define the second column of the report:

Contents	CHARGES→ PRODUCT
Heading	Product
Width	15
Decimal places	
Total this column	

Press PgDn to define the third column:

Contents	CHARGES→ QTY
Heading	Qty
Width	4
Decimal places	0
Total this column	Yes

Define the fourth column:

Contents	CHARGES- > UNIT_PRICE
Heading	Unit Price
Width	12
Decimal places	2
Total this column	Yes

Finally, the fifth column tells us something new; the total price of the transaction. This expression includes the quantity times the unit price. Because this column contains an expression rather than a field name, it has to be typed rather than selected from the fields submenu (F10). Notice that the CHARGES- > database is defined for each field, and the asterisk (*) is used for multiplication. Type the field content exactly as shown below. (Note: The arrow consists of a hyphen followed by a greater-than sign.)

Contents	CHARGES- > QTY * CHARGES - > UNIT_PRICE
Heading	Total Sale
Width	14
Decimal places	2
Total this column	Yes

When you have defined the report format, these column specifications will appear at the bottom of your screen:

Report Format	Qty	Unit Price	Total Sale
> > Date Product	XXXXXXX	#####.##	#####.##
mm/dd/yy	XXXXXXXXXXXXXXXXXXXX	#####.##	#####.##

Finally, highlight the Exit option and select Save to save the ARREPT.FRM report format.

Note : If you create a report format with a view open, you must always remember to open that view before printing that report in the future.

To view the report, highlight Retrieve from the Assistant menu screen, and select Report. Select a drive and the ARREPT.FRM report format. Select Execute the command. The report will look like Figure 4.28 when it's printed.

Modifying a View

To change a View, highlight the Modify option from the main menu bar, and select View. Specify a drive a View file name, and you'll be back to the View menu. From there, just use arrow keys to highlight menu options and make changes. Use the Exit and Save options to save the changes, or Exit and Abandon to delete the current changes and keep the original View.

Note : If you forget to close a view, and attempt to reopen a single file that's already in the view, you'll see the error message "File is already open." Use the CLOSE DATABASES command to close the view; then you can open any single database file that you wish.

Closing a View

When you quit dBASE, the View will be closed automatically. You can also close a View from the dot prompt. While the Assistant menu is on-screen, press Esc to call up the dot prompt. The type this command:

CLOSE DATABASES ↵

Enter this command

ASSIST ↵

to bring back the Assistant menu, You'll need to use Set Up to open a new database or View, because no database, index, or View files will be open.

Summary Report for Accounts Receivable

Date	Product	Qty	Unit Price	Total Sale
* * Customer	1000: Adams, Andy			
04/01/86	Turbo Board	1	550.00	550.00
04/18/86	RAM Disk	1	1100.00	1100.00
* * Subtotal * *		2	1650.00	1650.00
* * Customer 1001: Schumack, Susita				
03/12/86	Floppy Disks	10	2.11	2.10
03/31/86	Color Card	1	101.00	101.00
* * Subtotal * *		11	103.11	122.10
* * Customer 1002: Norris, Nancy				
03/15/86	8 Mhz Clock	2	16.39	32.78
04/11/86	RAM Disk	1	1100.00	1100.00
* * Subtotal * *				
3	1116.39	1132.78		
* * Customer 1003: Davies, David				
04/15/86	8 Mhz Clock	1	16.39	16.39
04/20/86	Tape Backup	1	1250.00	1250.00
04/20/86	40 Meg. Disk	1	450.00	450.00
* * Subtotal * *		3	1716.39	1716.39
* * Customer 1004: Teasdale, Rhonda				
04/15/86	Video Cable	10	16.00	160.00
* * Subtotal * *		10	16.00	160.00
* * * Total * * *		29	4601.89	4781.27

Figure 4.28: Accounts Receivable Report (ARREPT) Printed from Two Databases

Entering Commands

You can manage multiple databases directly from the dot prompt as well. Here are a few examples using the CUSTLIST.DBF and CHARGES.DBF databases. First, to start with a clean slate, close all open databases and index files by entering this command:

```
CLOSE DATABASES ↵
```

Next, to open two database simultaneously, use the SELECT commands with the letters A through J, or the numbers 1 through 10. For example, to open both the CUSTLIST.DBF and CHARGES.DBF databases, with their respective index files, enter the commands:

```
SELECT A ↵
```

```
USE CHARGES INDEX CHARGES ↵
```

```
SELECT B ↵
```

```
USE CUSTLIST INDEX CUSTLIST ↵
```

With both files open, you can now set up a relationship between the A and B files based upon the common field CUSTNO. Use these commands:

```
SELECT A ↵
```

```
SET RELATION TO CUSTNO INTO CUSTLIST ↵
```

There are some differences between a View and simply opening two files simultaneously. For example, if you enter the list command

```
LIST ↵
```

you'll only see data from the currently selected database (CHARGES.DBF). (Entering the command SELECT B will make CUSTLIST.DBF the currently selected database, and LIST would show its data.)

Record#	CUSTNO	PRODUCT	QTY	UNIT_PRICE	DATE	BILLED
4	1000	Turbo Board	1	550.00	04/01/86	.F.
8	1000	RAM Disk	1	1100.00	04/18/86	.F.
1	1001	Floppy Disks	10	2.11	03/12/86	.F.
3	1001	Color Card	1	101.00	03/31/86	.F.
2	1002	8 Mhz Clock	2	16.39	03/15/86	.F.
6	1002	RAM Disk	1	1100.00	04/11/86	.F.
7	1003	8 Mhz Clock	1	16.39	04/15/86	.F.
9	1003	Tape Backup	1	1250.00	04/20/86	.F.
10	1003	40 Meg. Disk	1	450.00	04/20/86	.F.
5	1004	Video Cable	10	16.00	04/15/86	.F.

To display records from the unselected database, you need to use a letter and arrow before the field name. For example, when the A file is selected, use B- > in front of the field names from the B file:

```
SELECT A ↵
```

```
LIST CUSTNO, B- >LNAME, B- >FNAME, QTY, UNIT_PRICE, (QTY * UNIT_PRICE) ↵
```

Notice that this LIST command contains a calculated field (QTY * UNIT_PRICE), which displays the total of the sales transaction. The results look like this:

Record#	CUSTNO	B- >LNAME	B- >FNAME	QTY	UNIT_PRICE	(QTY * UNIT_PRICE)
4	1000	Adams	Andy	1	550.00	550.00
8	1000	Adams	Andy	1	1100.00	1100.00
1	1001	Schumack	Susita	10	2.11	21.10
3	1001	Schumack	Susita	1	101.00	101.00
2	1002	Norris	Nancy	2	16.39	32.78
6	1002	Norris	Nancy	1	1100.00	1100.00
7	1003	Davies	David	1	16.39	16.39
9	1003	Davies	David	1	1250.00	1250.00
10	1003	Davies	David	1	450.00	450.00
5	1004	Teasdale	Rhonda	10	16.00	160.00

To get a quick calculation of the column in the display, simply enter this command:

```
SUM QTY * UNIT_PRICE ↵
```

dBASE displays

10 records summed

QTY * UNIT_PRICE

4781.27

You can also print the report from the dot prompt, but you'll have to make a change using the MODIFY REPORT command.

Earlier in this chapter, you entered this Group on expression:

```
STR(CUSTNO,4) + ": " + TRIM(LNAME) + ", " + FNAME
```

This expression must be changed to

```
STR(CUSTNO,4) + ": " + TRIM(B- > LNAME) + ", " + B- > FNAME
```

because without the View in effect, LNAME and FNAME need to be defined as fields from the SELECT B database file. (Other reports created from a View might need similar modifications.)

Once a report format exists with fields from the two (or more) databases, you can use the REPORT FORM command from the dot prompt:

```
REPORT FORM ARREPT TO PRINT ↵
```


This command prints a copy of the ARREPT.FRM report.

If at any time when working from the dot prompt you become confused about what files are open, selected, and so forth, you can just enter this command:

DISPLAY STATUS ↵

dBASE displays a status report that looks something like this:

Currently Selected Database:

Select area: 1, Database In Use: C:CHARGES.DBF Alias: CHARGES

Index file: C: CHARGES.NDX Key: CUSTNO

Related Into: CUSTLIST

Relation: CUSTNO

Select area: 2, Database In Use: C:CUSTLIST.DBF Alias: CUSTLIST

Index file: C:CUSTLIST.NDX Key: CUSTNO

This status report tells you that the CHARGES.DBF and CHARGES.NDX files are open in Select area 1 (same as SELECT A). Furthermore, CHARGES.DBF is the currently selected database, and is related to CUSTLIST, based on the CUSTNO field.

In Select area 2 (B), the CUSTLIST.DBF database is open, with the CUSTLIST.NDX database.

To close all open databases and index files, enter this command at the dot prompt:

CLOSE DATABASES ↵

An Inventory Database Design

In this section, we'll discuss another design requiring multiple databases and new techniques for managing data. Some of the more advanced techniques in this section are not accessible from the Assistant menu, so we'll do this entire section directly from the dot prompt.

If you are using a computer with two floppy disks, you should make sure that you have a work disk in Drive B. Then enter this command at the dot prompt before you begin the following exercises:

SET DEFAULT TO B ↵

This will ensure that all files are stored and accessed on the disk in Drive B.

Master File and Transaction Files

Inventory systems usually involve multiple databases as well. The relationship among these database is often called a Master File Transaction File relationship. To simplify managing the data, inventory information is most easily divided into three separate databases: one for sales, one for purchases (or new stock), and a master inventory database. Envision the relationship among these databases, as shown in figure 4.29.

The Sales and New Stock files are called transaction files, because they record information about individual sales and purchase transactions. The inventory file is called the master file because it contains the current status of each item in stock. In this section, we'll see how to update the current status of the master file from the contents of the transaction files.

An Inventory System

First, let's create the master inventory file and give it the file name MASTER. Enter this command at the dot prompt:

CREATE MASTER ↵

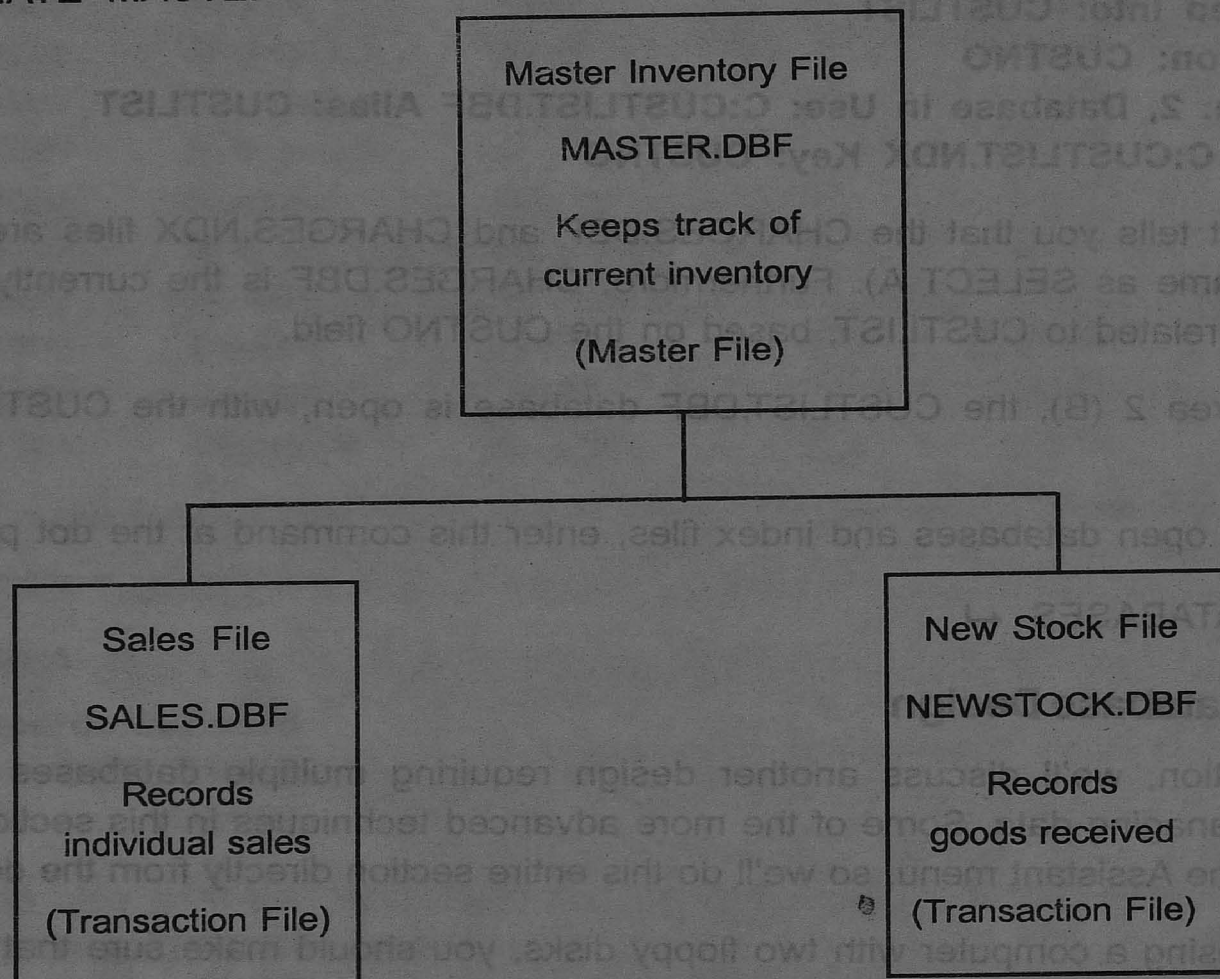


Figure 4.29: Inventory Databases

Give the MASTER file this structure:

Structure for database: MASTER.DBF

Field	Field Name	Type	Width	Dec
1	CODE	Character	5	
2	TITLE	Character	15	
3	QTY	Numeric	5	
4	PRICE	Numeric	5	2
5	REORDER	Numeric	5	

When dBASE asks if you want to input data records now, answer. Yes and type the following data:

CODE	TITLE	QTY	PRICE	REORDER
AAA	Rakes	30	3.50	25
BBB	Hoes	30	4.50	25
CCC	Shovels	30	5.00	25

Then enter these commands:

USE MASTER ↵

LIST ↵

Three records appear:

Record#	CODE	TITLE	QTY	PRICE	REORDER
1	AAA	Rakes	30	3.50	25
2	BBB	Hoes	30	4.50	25
3	CCC	Shovels	30	5.00	25

The first record tells you that product code AAA is rakes, that you have 30 in stock, that the purchase price is \$3.50, and that you reorder when the stock on hand gets below 25. You have 30 hoes in stock (product code BBB), each costing \$4.50, and you reorder when stock gets below 25. Product code CCC is shovels. You have 30 in stock, each costing \$5.00. Reorder when stock is below 25.

Now let's create a database to keep track of new stock received. Call it NEWSTOCK with this command

CREATE NEWSTOCK ↵

Structure it like this:

Structure for database: NEWSTOCK.DBF

Field	Field Name	Type	Width	Dec
1	CODE	Character	5	
2	QTY	Numeric	5	
3	PRICE	Numeric	12	2
4	DATE	Date	8	

Say Yes when dBASE asks "Input data now? (Y/N)". Assume that you've just received two orders from wholesalers, one order of ten rakes, each costing \$4.00, and another order for six shovels, each costing \$4.50. Furthermore, let's assume that you received them on March 1, 1986. To add these new items to NEWSTOCK, type the following data:

CODE	QTY	PRICE	DATE
AAA	10	4.00	03/01/86
BBB	6	4.50	03/01/86

When dBASE asks for data from Record #3, press Return and the dot prompt will appear: So if you now

```
USE NEWSTOCK ↵
LIST ↵
```

you'll see your new stock listed in database format:

Record#	CODE	QTY	PRICE	DATE
1	AAA	10	4.00	03/01/86
2	BBB	6	4.50	03/01/86

Now you need to come up with a method to update the master inventory so that it reflects the new goods received.

Updating Database with UPDATE

The dBASE UPDATE command allows you to update the contents of one database based upon information from another. You can specify that the update either add, subtract or replace entire fields. This is best explained with an example. Suppose that you want to add the new stock items to the MASTER file. Furthermore, if there is a change in the price for an item, you want the MASTER file to record the new price. In that case, replace the existing price in the MASTER file with the price in the NEWSTOCK file.

Let's review the contents of both files first. If you

```
USE MASTER ↵
LIST ↵
```

the original inventory appears:

Record#	CODE	TITLE	QTY	PRICE	REORDER
1	AAA	Rakes	30	3.50	25
2	BBB	Hoes	30	4.50	25
3	CCC	Shovels	30	5.00	25

That is, you have 30 rakes in stock, at a wholesale price of \$3.50. You have 30 hoes, wholesale priced at \$4.50. You have 320 shovels, wholesale priced at \$5.00. Now let's

```
USE NEWSTOCK ↵
LIST ↵
```


to see new items in stock:

Record#	CODE	QTY	PRICE	DATE
1	AAA	10	4.00	03/01/86
2	BBB	6	4.50	03/01/86

You've received ten product AAA (rakes) at \$4.00 each. You've also received six product BBB (hoes) at \$4.50 each. So add these items to your inventory, and note that the wholesale price of rakes has increased from \$3.50 to \$4.00. Here is the procedure.

First, identify a key field, one that relates the contents of the MASTER file with the NEWSTOCK file. Both files must have this field in common. In this example, CODE is the key field, because you want dBASE to add ten items of product code AAA to the MASTER file, and six of product code BBB.

Second, you must open both the MASTER and NEWSTOCK files simultaneously. Use the SELECT command to do so. For this example, open the MASTER file in a work area labeled A, and the NEWSTOCK file in a work area labeled B. Here are the correct commands:

```
SELECT A ↵
USE MASTER ↵
SELECT B ↵
USE NEWSTOCK ↵
```

Now, specify the MASTER file by selecting work area A. Then use the UPDATE command to perform the update. Here are the commands:

```
SELECT A ↵
UPDATE ON CODE FROM NEWSTOCK REPLACE QTY WITH; QTY + B- > QTY,
PRICE WITH B- > PRICE ↵
```

These commands mean, "Update the MASTER file from the data in NEWSTOCK using CODE as the comparison (key) field; replace the QTY field with its current value plus the value of the QTY field in NEWSTOCK ($QTY + B- > QTY$), and replace the PRICE with the PRICE from the NEWSTOCK file ($B- > PRICE$)." The $B- >$ symbol is used to specify data from the NEWSTOCK file opened in work area B. (The arrow symbol is formed by typing a hyphen followed by a greater-than sign.)

As soon as the update procedure is complete, list the contents of the MASTER file:

```
LIST ↵
```

You'll see these records:

1	AAA	Rakes	40	4.00	25
2	BBB	Hoes	36	4.50	25
3	CCC	Shovels	30	5.00	25

There are now 40 rakes (AAA) in stock, because you've received 10. The price of rakes is now \$4.00, as opposed to \$3.50, because you REPLACED PRICE. There are now 36 hoes (BBB) in stock because you received 6. The price of hoes is still \$4.50. Shovels (CCC) were not affected, because the NEWSTOCK file did not have any information about shovels.

Now, let's discuss updating the MASTER file from the SALES database. As stated in the dBASE manual, the FROM (transaction) file in an UPDATE command must be sorted and indexed by the key field. You can see that these are certainly not sorted by the key field, CODE. If you

```
USE SALES ↵
LIST ↵
```

you'll see these records:

1	AAA	Rakes	3	15.00	03/01/86
2	BBB	Hoes	2	12.50	03/01/86
3	CCC	Shovels	3	21.00	03/01/86
4	AAA	Rakes	2	10.00	03/01/86
5	CCC	Shovels	4	26.50	03/01/86
6	AAA	Rakes	2	11.00	03/02/86
7	CCC	Shovels	1	7.50	03/02/86
8	BBB	Hoes	2	12.50	03/01/86
9	AAA	Rakes	5	23.50	03/02/86

You can use INDEX in this case, so let's try it.

```
INDEX ON CODE TO SALES ↵
LIST ↵
```

Now the inventory is sorted by product code:

1	AAA	Rakes	3	15.00	03/01/86
2	AAA	Rakes	2	10.00	03/01/86
3	AAA	Rakes	2	11.00	03/02/86
4	AAA	Rakes	5	23.50	03/02/86
5	BBB	Hoes	2	12.50	03/01/86
6	BBB	Hoes	2	12.50	03/01/86
7	CCC	Shovels	3	21.00	03/01/86
8	CCC	Shovels	4	26.50	03/01/86
9	CCC	Shovels	1	7.50	03/02/86

Now open the MASTER and SALES files, along with the SALES index file to perform the update. First, let's cancel all previous SELECT assignments by typing this command:

CLEAR ALL ↵

Now select the MASTER file as A, and the SALES file and index as B:

SELECT A ↵

USE MASTER ↵

SELECT B ↵

USE SALES INDEX SALES ↵

To perform the update, select the MASTER file, and perform the update, subtracting the quantities in the SALES file from the quantities in the MASTER file.

SELECT A ↵

UPDATE ON CODE FROM SALES REPLACE QTY WITH; QTY - B- > QTY ↵

To see the effect, LIST the MASTER file.

LIST ↵

dBASE displays:

Record#	CODE	TITLE	QTY	PRICE	REORDER
1	AAA	Rakes	28	4.00	25
2	BBB	Hoes	32	4.50	25
3	CCC	Shovels	22	5.00	25

There are now 28 rakes in stock because you've sold 12. There are 32 hoes because you've sold 4, and 22 shovels because you've sold 8. dBASE subtracted the appropriate quantities from the MASTER file base upon the quantities and product codes in the SALES file.

Now, to see which items need to be reordered, type this command:

LIST FOR QTY < REORDER ↵

The result is

Record#	CODE	TITLE	QTY	PRICE	REORDER
3	CCC	Shovels	22	5.00	25

The amount of shovels in stock (22) has fallen below the reorder point (25).

Now, to keep future exercises in order, let's "unassign" the SELECT commands by typing this command:

CLOSE DATABASE ↵

You now have a managerial problem on your hands. The MASTER file is accurate, but SALES and NEWSTOCK still have data in them. If you were to add new records to these transaction databases and do another update, the MASTER file would then be incorrect. The UPDATE command would add or subtract these items a second time from the MASTER file. Therefore, you must come up with a managerial scheme for getting rid of data you've already updated. If you wished to update the MASTER file daily, a good approach might be to do the following:

1. Use the NEWSTOCK and SALES files during the course of the day to record goods received and sold.
2. At the end of the day, print a REPORT of all sales and goods received from the SALES and NEWSTOCK files for a permanent record.
3. UPDATE the MASTER file from the SALES and NEWSTOCK files.
4. DELETE and PACK (or ZAP) all records from the SALES and NEWSTOCK files, so future updates are not confused with previous updates.

The disadvantage to this approach is that you lose all the data from the SALES and NEWSTOCK files. Here's a better approach which leaves the SALES and NEWSTOCK file intact. Let's use NEWSTOCK as the example. Suppose that on March 2 you receive 10 rakes at \$4.00 each. Let's

USE NEWSTOCK ↵

APPEND ↵

and add the following information:

CODE : AAA :
QTY : 10 :
PRICE : 4.00 :
DATE : 03/02/86 :

When you LIST, you'll see the new record added to the list

Record#	CODE	QTY	PRICE	DATE
1	AAA	10	4.00	03/01/86
2	BBB	6	4.50	03/01/86
3	AAA	10	4.00	03/02/86

When you do another UPDATE, you won't want Records #1 and 2 to be used again. So move only the newest entry to a file named TEMP.

COPY TO TEMP FOR DTOC(DATE) = "03/02/86" ↵

If you now

USE TEMP ↵

LIST ↵

you'll see

Record#	CODE	QTY	PRICE	DATE
3	AAA	6	4.00	03/02/86

Now you can update the MASTER file from TEMP without worrying about updating Records #1 and 2 again. That is, you can

SELECT A ↵

USE MASTER ↵

SELECT B ↵

USE TEMP ↵

SELECT A ↵

UPDATE ON CODE FROM TEMP REPLACE QTY WITH;

QTY + B - > QTY, PRICE WITH B - > PRICE ↵

Note: Remember to sort or index the Temp file first. Since there was only one record in your TEMP file, you had no need to sort or index it first.

When you LIST the MASTER file now, you'll see

1	AAA	Rakes	38	4.00	25
2	BBB	Hoes	32	4.50	25
3	CCC	Shovels	22	5.00	25

Ten rakes have been added to the inventory. Your NEWSTOCK file still contains records of all goods received.

Reviewing Multiple Databases

In this chapter you have learned many techniques for managing multiple databases:

Data are often divided into separate databases to avoid unnecessary repetitions, as in the example of customer names and addresses in one database and individual sales transaction in another.

A *common field* is required to link two databases. The common field should have the same name, data type, and width in both database.

In one database, each record usually contains some unique identifying code. This code is called the *key field* and is used to link multiple databases. Examples of key fields include customer numbers, part numbers, and account numbers.

To set up a View between two files, select **View** from the **Create** menu and assign a name to the View. Identify the database and index file names, the common fields, and fields to view using menu options.

To open an existing View, highlight the Set Up option from the main menu bar and select View.

To print a report from two databases, open the View and create the report using the **Create** and **Report** options.

To modify an existing View, highlight the **Modify** option from the Assistant menu, and select View.

The SELECT and SET RELATION commands allow you to open up to ten databases simultaneously and set up relationships between them.

Inventory systems often use multiple databases in a **Master File Transaction File** relationship.

The dBASE III PLUS UPDATE command, available only from the dot prompt, allows you to update the values in a Master file with values in a Transaction File.

Note: Remember to sort or index the Temp file first. Since there was only one record in your TEMP file, you had no need to sort or index it first.

When you LIST the MASTER file now, you'll see

1	AAA	Rakes	35	4.00	25
2	BBB	Hoes	32	4.50	25
3	CCC	Shovels	22	5.00	25

Ten rakes have been added to the inventory. Your NEWSTOCK file still contains records of all goods received.

Reviewing Multiple Databases

In this chapter you have learned many techniques for managing multiple databases. Data are often divided into separate databases to avoid unnecessary repetitions, as in the example of customer names and addresses in one database and individual sales transactions in another.

A common field is required to link two databases. The common field should have the same name, data type, and width in both databases.

In one database, each record usually contains some unique identifying code. This code is called the key field and is used to link multiple databases. Examples of key fields include customer numbers, part numbers, and account numbers.

Unit - V

Section 5.1 File Maintenance and Performance

In this chapter, we'll discuss options from the Assistant menu which are generally used for applications involving many or large database files. This will be a "catch all" chapter for options and techniques used with the Assistant menu. After this chapter, you'll learn more about the dot prompt and the dBASE programming language.

The Menu Tool Kit

If you highlight the Tools option from the Assistant menu, you'll see these options:

- Set drive**
- Copy file**
- Directory**
- Rename**
- Erase**
- List structure**
- Import**
- Export**

These options help you manage files and make backups of important data. Let's take a closer look at each of these options.

Set Drive

When selected, the Set drive option displays a menu of disk drives (A:, B:, C:). The drive you select becomes the default drive where dBASE looks for all database, index report format, and other files. This option works the same as the SET DEFAULT command from the dot prompt.

Copy File

The Copy file option allows you to make a copy of any file and is particularly good for making backups. Once selected, this option will display a menu of existing files on the directory. You select a file to copy. Then you'll see a menu of disk drives (A:, B:, C:). After you select a disk drive, a prompt will ask that you enter a name for the copy. Type any valid file name and include a three letter extension if you wish.

Let's look at a couple of examples for making backups. Suppose that you want to make a backup copy of your MAIL.DBF database. You could select Copy file and MAIL.DBF as the file to copy. Then, enter the name MAIL.BAK as the file to copy to. When completed, the MAIL.BAK file will be identical to the MAIL.DBF file. If something happens to MAIL.DBF, you have a backup copy stored under the file name MAIL.BAK. You should repeat the backup process occasionally so that the copy is a reasonable facsimile of the original database.

If you have a hard disk, you can use this method to make backup copies onto floppy disks. Select the Copy file option and the file that you want to copy. Make sure there is a disk in Drive A, and then select A: as the drive to copy to. Create a file name for the copy. Because the copied file is going to a different disk drive, you can use the same name as the original file or add an extension that indicates the date of your backup (MAIL.FEB).

The Copy file option is useful, but it is not a substitute for the DOS COPY and BACKUP commands. If you plan on using a computer often, it will be worth your time to learn about the DOS COPY command. The DOS BACKUP command is particularly useful if you use a hard disk.

Directory

The Directory option lets you see the names of existing files on a disk. When selected, it displays these options:

*dbf	Database Files
*ndx	Index Files
*fmt	Format Files
*lbl	Label Files
* frm	Report Files
*txt	Text Files
*vue	View Files
*qry	Query Files
*scr	Screen Files
*	All Files

You can move the highlighting to the type of file you wish to view, and press Return to select the option. dBASE will display all the files with the file-name extension requested. (Note: You can also view All files using the last option.)

Rename

The Rename option allows you to change the name of an existing file. However, you should avoid changing the file-name extensions (.DBF, .NDX, and so forth) because they are used to identify the types of files. Like the Copy file option, this option displays a list of existing files and asks that you select one to rename. Then you simply type the new name for the file.

Erase

The Erase option allows you to remove a file from the disk drive permanently. You should use this option with caution because once a file is deleted, it cannot be recalled.

List Structure

This option displays the structure of the currently open database—a useful, quick reminder of field names in a database, particularly when working from the dot prompt. (The equivalent

dot-prompt command is also LIST STRUCTURE.) For example, selecting List structure while the MAIL.DBF database is in use would display this structure:

Structure for database: MAIL.DBF

Number of data records: 5

Date of last update : 03/05/86

Field	Field Name	Type	Width	Dec
1	LNAME	Character	15	
2	FNAME	Character	10	
3	ADDRESS	Character	25	
4	CITY	Character	15	
5	STATE	Character	5	
6	ZIP	Character	10	
7	PHONE	Character	13	
* * Total * *			94	

Import and Export

The Import and Export options allow you to exchange data from a PFS:FILE database. See Appendix A for a complete discussion of interfacing dBASE III PLUS with other software systems.

Data Catalogs

Data catalogs are valuable for hard-disk users who have to manage many files on a single directory. They also help you to keep files organized and make it easier to work with general applications. For example, you could set up a catalog for your mailing list that only provides access to the index files, reports, and screens that are relevant to the MAIL.DBF database.

Even though you can access a catalog from the Assistant menu, you must create it from the dot prompt. So, press Esc until the dot prompt appears.

To start a catalog, enter the SET CATALOG command along with a name for the catalog. For this example, we'll create a catalog named MAILLIST. If you are using a floppy-disk system, type these commands:

```
SET DEFAULT TO B ↵  
SET CATALOG TO MAILLIST ↵
```

If you are using a hard-disk system, enter this command:

```
SET CATALOG TO MAILLIST ↵
```

dBASE will see that no such catalog exists and will display the prompt:

Create new catalog? (Y/N)

Answer Yes. Then dBASE will ask that you

Enter title for file MAILLIST.CAT:

Type the title Mailing List Manager, and press Return. dBASE will then inform you that the

File catalog is empty.

Let's put some files into the catalog now. To put a database file into the catalog, type this command:

USE MAIL ↵

dBASE asks that you

Enter title for file MAIL.DBF:

Type this title:

Names and Addresses Database ↵

To put the MAIL index files into the catalog, enter this command:

SET INDEX TO NAMES,ZIP ↵

To put the ADDNAMES.FMT screen format into the catalog, enter this command:

SET FORMAT TO ADDNAMES ↵

dBASE will ask that you

Enter title for file ADDNAMES.FMT.

Type this title:

Screen to Enter and Edit Data -

To add a report to the catalog, use the MODIFY REPORT command. For example, to put BYNAME.FRM into the catalog, enter this command:

MODIFY REPORT BYNAME ↵

dBASE will, of course, give you an opportunity to modify the report. Because the purpose of this exercise is only to place the BYNAME.FRM report format in the catalog, you need not make any changes to the report. Just highlight the Exit option and select Save. dBASE will ask that you

Enter title for BYNAME.FRM:

Type this title:

Directory Listing ↵

In general, when you use the SET CATALOG TO command to open a catalog, you can use these commands to add files, and the titles you provide, to the catalog:

COPY

CREATE

INDEX

MODIFY

SET FORMAT

USE

To stop recording files to be included in the catalog, enter this command:

SET CATALOG OFF ↵

This will keep new files from being added to the catalog, but the catalog will still be active in the sense that menu selections will display only cataloged files.

To completely close a catalog so that all files once again become accessible, enter the SET CATALOG TO command without a file name:

SET CATALOG TO ↵

Now, let's get back to the Assistant menu and see what effect this catalog has. Type

ASSIST ↵

to bring back the Assistant menu. Then, highlight the Set Up option and select Catalog. Select a drive and MAILLIST.CAT from the submenu.

Nothing much appears to happen until you begin selecting items to work with. If you highlight Set Up and select Database file to open a database, only MAIL.DBF appears on the screen as an option, along with the title you previously entered:

Database file

MAIL.DBF

Names and Addresses Database

Select MAIL.DBF and answer Yes to the "In the file indexed?" prompt. The screen will display only the index files associated with this data catalog:

NAMES.ND

ZIP.NDX

If you select the Format for Screen option from the Set Up menu, only ADDNAMES.FMT will appear as an option. So, as you can see, a catalog allows you to define groups of related files. That means you can have all of your mailing list files in one catalog, your inventory in another, and accounts receivable in another.

An active catalog can also be useful when you are working from the dot prompt. The catalog query option (?), will display only files in the current catalog. For example, the command

SET CATALOG TO ? ↵

calls up a menu of all existing catalogs and allows you to pick one. Once you select a catalog, you can use the ? option in place of a file name in any command including MODIFY, REPORT, LABEL, SET INDEX, and USE. For example, entering the command

USE ? ↵

displays a menu of database files in the catalog. The command

SET INDEX TO ? ↵

displays a list of index files in the catalog. The commands

REPORT FORM ? ↵

and

SET FORMAT TO ? ↵

display catalog options for report and screen format files.

Data catalogs are just one method of organizing files. You can also develop menu-driven systems through the dBASE programming language and with the dBASE III PLUS applications generator.

Reusing Queries

As you continue working dBASE III PLUS, you may find it tedious to repeatedly use the Build a search condition option when you want to access specific records. For example, suppose that you have a database named SAMPLE.DBF which contains these records:

Record#	CUSTNO	PRODUCT	QTY	UNIT_PRICE	DATE	BILLED
1	1001	Floppy Disks	10	2.11	03/12/86	.T.
2	1002	8 Mhz Clock	2	16.39	03/15/86	.T.
3	1001	Color Card	1	101.00	03/31/86	.T.
4	1000	Turbo Board	1	550.00	04/01/86	.T.
5	1004	Video Cable	10	16.00	04/15/86	.T.
6	1002	RAM Disk	1	1100.00	04/11/86	.T.
7	1003	8 Mhz Clock	1	16.39	04/15/86	.T.
8	1000	RAM Disk	1	1100.00	04/18/86	.T.
9	1003	Tape Backup	1	1250.00	04/20/86	.T.
10	1003	40 Meg. Disk	1	450.00	04/20/86	.T.
11	1003	Floppy Disks	5	2.11	05/12/86	.F.
12	1004	8 Mhz Clock	1	16.39	05/15/86	.F.
13	1003	Color Card	1	101.00	05/31/86	.F.
14	1001	Video Cable	2	16.00	06/15/86	.F.
15	1001	8 Mhz Clock	2	16.39	05/15/86	.F.
16	1001	RAM Disk	1	1100.00	05/18/86	.F.
17	1004	Tape Backup	2	1250.00	06/20/86	.F.

Furthermore, you only want to work with unbilled accounts (records that contain BILLED = .F.). You can repeatedly specify BILLED = .F. in all your commands, or you can set up and save a Query. Use the CHARGES.DBF database to try an on-line Query.

Creating a Query Form

To create a Query, first be sure that CHARGES.DBF and its index files are open. Then, highlight the Create option and select Query. A prompt will ask that you enter a name for the query form. Follow the usual rules for creating file names (eight letter maximum, no spaces or punctuation). For this example, enter the file name NOTBILLD. The screen displays a blank query form.

To fill in the query form so that only records with the BILLED field equal to False are accessed, follow these steps:

- ✓ Select Field Name.
- ✓ Select BILLED from the submenu.
- ✓ Select Operator.
- ✓ Select Is False.

As you can see, filling in the query form is much like creating a search condition with Assistant menu options. The query form is now completed so that it look like Figure 5.1.

Saving a Query Form

Once you've filled in the query form, highlight the Exit option and select Save. You'll be returned to the Assistant menu.

Set Filter
Nest
Display
Exit
10:19:13 am

Field Name	BILLED
Operator	Is
Constant/Expression	False
Connect	
Line Number	1

Line	Field	Operator	Constant / Expression	Connect
1	BILLED	Is	False	
2				
3				
4				
5				
6				
7				

MODIFY QUERY || <C> || C: NOTBILLED.QRY || Opt : 1/5 || Ins ||
 Position selection bar - ↑↓. Select - ←→. Leave Menu - ←→.
 Enter an expression or constant for the filter condition.

Figure 5.1: Query Form to Access Unbilled Accounts

Activating a Query

You can activate a Query by following these steps from the Assistant menu screen:

- ✓ Highlight the Set Up option.
- ✓ Select Query.
- ✓ Select NOTBILLED.

Once activated like this, dBASE will operate as though only the records specified in the Query exist. For example, if you highlight

Retrieve, select List and Execute the command, only these records will be displayed:

Record#	CUSTNO	PRODUCT	QTY	UNIT_PRICE	DATE	BILLED
11	1003.	Floppy Disks	5	2.11	05/12/86	.F.
12	1004	8 Mhz Clock	1	16.39	05/15/86	.F.
13	1003	Color Card	1	101.00	05/31/86	.F.
14	1001	Video Cable	2	16.00	06/15/86	.F.
15	1001	8 Mhz Clock	2	16.39	05/15/86	.F.
16	1001	RAM Disk	1	1100.00	05/18/86	.F.
17	1004	Tape Backup	2	1250.00	06/20/86	.F.

If you select Report from the Retrieve menu, only these records will be displayed in your report. You can only BROWSE and EDIT records where BILLED = .F. while this Query is activated. In other words, you can work on these records without having to recreate search conditions for every command.

Closing a Query

Unfortunately, there is no simple way from the Assistant menu to deactivate a Query. To close a Query, leave the Assistant menu by pressing Esc, and enter this command at the dot prompt:

SET FILTER TO ↵

to close the query file. You can then enter the command

ASSIST ↵

to return to the Assistant menu and access your database without the filtering effects of the query form.

Modifying a Query Form

To change an existing query form, highlight the Modify option from the Assistant menu and select Query. Select the appropriate Query from the menu. You can modify the Query with the same options menu that you used to create the original query form.

Advanced Querying Techniques

Query forms can include the dollar (\$) operator for embedded character searches, as we discussed earlier. Query forms may also contain parentheses which help refine the logic of a search. For example, take a look at the Query in Figure 5.2.

Line	Field	Operator	Constant / Expression	Connect
1	CUSTNO	More than	1001	.AND.
2	BILLED	is	False	.OR.
3	DATE	More than or equal	85/81/86	

MODIFY QUERY | <C:> | C: TEST1.QRY | Opt : 1/5 | Ins |

Position selection bar - ↑↓. Select - ↵ Leave Menu - ←→ .

Select a field name for the filter condition.

Figure 5.2: Complex Query without Parentheses

Notice that this Query specifies

CUSTNO > 1001 .AND. BILLED = .F. .OR. DATE > = 05/01/86

or, in English, "Display records where the customer number is greater than 1001, and where the BILLED field is .F. or the date is greater than 05/01/86." If you LIST with this Query, these records will be displayed:

Record#	CUSTNO	PRODUCT	QTY	UNIT_PRICE	DATE	BILLED
11	1003	Floppy Disks	5	2.11	05/12/86	.F.
12	1004	8 Mhz Clock	1	16.39	05/15/86	.F.
13	1003	Color Card	1	101.00	05/31/86	.F.
14	1001	Video Cable	2	16.00	06/15/86	.F.
15	1001	8 Mhz Clock	2	16.39	05/15/86	.F.
16	1001	RAM Disk	1	1100.00	05/18/86	.F.
17	1004	Tape Backup	2	1250.00	06/20/86	.F.

Although you specified in the query form that the customer number must be greater than 1001, you see records with customer numbers equal to 1001. That's because the .OR. condition lists records with dates greater than or equal to 05/01/86, regardless of the customer number.

Now, take a look at this filter condition:

CUSTNO > 1001. .AND. (BILLED = .F. .OR. DATE > = 05/01/86)

Notice the use of parentheses which block off the .OR. portion of the Query. The parentheses cause the .OR. condition to be evaluated first. Therefore, in order to be displayed by this condition, a given record must either have .F. in the BILLED field, or a date greater than or equal to May 1, 1986. If a record meets either of those criteria, it then must further meet the criterion of customer number > 1001 to be included. Therefore customers with numbers less than or equal to 1001 are rejected by this Query.

You insert parentheses into query forms using the Nest option from the menu. When you select, Nest, the screen displays a menu:

Add:

Start: 0

End: 0

Remove:

Start: 0

End: 0

To place parentheses in a query form, select the Start option under Add:, and specify the line number where the parentheses should begin. (The line number appears in the left column of the query form.) Then, select the End option, and indicate the line where the parentheses should end. Figure 5.3 shows the sample Query with parentheses added to lines 2 and 3. Notice that the parentheses appear on the lines that separate portions of the query form.

After adding the parentheses, you can save and activate the query form in the usual manner. A list of records accessed by the modified query form appears on your screen:

Record#	CUSTNO	PRODUCT	QTY	UNIT_PRICE	DATE	BILLED
11	1003.	Floppy Disks	5	2.11	05/12/86	.F.
12	1004	8 Mhz Clock	1 1	6.39	05/15/86	.F.
13	1003	Color Card	1	101.00	05/31/86	.F.
17	1004	Tape Backup	2	1250.00	06/20/86	.F.

The Remove option under the Nest menu allows you to remove existing parentheses from a query form. The Display option from the Query menu will immediately display the first record in the database that meets the Query condition. Pressing PgDn will display other records that meet the search condition. This option allows you to check the validity of your Query before saving it via the Exit option.

Set Filter
Nest
Display
Exit
01:40:17 pm

Add
Start : 2
End : 3
Remove
Start : 8
End : 0

Line	Field	Operator	Constant / Expression	Connect
1	CUSTNO	More than	1001	AND
2	BILLED	Is	False	OR
3	DATE	More than or equal	85/81/86	
4				
5				
6				
7				

MODIFY QUERY || <C:>|| C: TEST1.QRY || Opt : 1/4 || Ins ||

Position selection bar - ↑↓. Select - ←→ Leave Menu - ←→

Enter the line number of the query form to remove left parenthesis.

Figure 5.3: Complex Query with Parentheses

Maximizing Search Performance

For any given search or query in a database, there will probable be several ways to accomplish the same goal. However, your search method significantly affects the time DBASE requires to find your records. One of the most important factors affecting the speed of a search is the use of the FOR (Build a search condition) and WHILE (Build a scope condition) options. Let's look at an example.

Suppose that you are using a database with the same structure as MAIL.DBF, but there are about a thousand records in the database. Of these, suppose that ten people have the last name Miller. How long does it take to pull out and view all the Millers in this database? Well, that depends on how you perform the search.

You can use the *Build a search condition* option to build the condition LNAME = 'Miller'. Then it will take about forty seconds to view all the records with a List or Report option. However, most of that time will be spent reading all of the records in the database that do not have the last name Miller, only to reject them from the display.

Seeking Indexed Records

An index file of last names for the database can speed this process up considerable. First, if the NAMES.NDX index is active for the MAIL.DBF database, and if it is the Master index, you can highlight the Position option from the Assistant menu and select Seek. Then enter

"Miller" ←

(Be sure to include the quotation marks.) The Seek option will immediately access the first Miller in the database. Because the database is already in alphabetical order, dBASE can quickly find the other Millers listed beneath the first one:

Mason	Marcia	123 Baker St.	San Clemente	CA
Miller	Muriel	P.O. Box 123	La Jolla	CA
Miller	Peter	1086 De Mayo	Del Mar	CA
Miller	Wendy	5441 Elm St.	Eugene	OR
Miller	Albert	High Point Lane	Scarsdale	NY
Miller	Fred	1 Hampton Ct.	Greenwich	CT
Miller	David	P.O. Box 2802	Boston	MA
Miller	Aaron	544 Oak St.	Encinitas	CA
Miller	Susita	186 Crest Dr.	Portland	ME
Miller	Cynthia	Tara	Atlanta	GA
Miller	Bill	Via Granada	Rancho	WA
Moses	Robert	321 Adams	San Diego	CA

Building a Scope Condition

Now, to keep dBASE from scanning through a thousand records to pull out all the Millers, you can tell dBASE to just keep displaying records as long as (WHILE) the last name is Miller. Follow these steps:

- ✓ Highlight Retrieve.
- ✓ Select List.
- ✓ Select Build a scope condition.
- ✓ Select LNAME.
- ✓ Select =.
- ✓ Type Miller.

You have built a command just as you would using the Build a search condition option. Notice, however, that the command you've built is

Command: LIST WHILE LNAME = "Miller"

rather than

Command: LIST FOR LNAME = "Miller"

When you select Execute the command, you'll notice another difference. It only takes dBASE around three seconds to display the Millers, rather than forty seconds. Of course, if this database contained 10,000 names, the time difference would be even more dramatic.

Although the Build a scope condition option is much faster than the Build a search condition option, it can only be used under the very specific circumstances described in this example. The WHILE command means, "Starting at the current record, keep accessing until the condition is no longer true." If the records are not sorted or indexed, and if you do not use the Seek option first, then the Scope condition will not work properly. Later in this book, we'll use the WHILE option in dBASE programs to gain maximum speed from a fully automated mailing list system.

Other Accessing Options

The Assistant menu offers other techniques for accessing records in a database, but these are generally used only with small databases where record numbers are easily memorized. We'll discuss these options briefly because they are of limited value in managing data.

The Position menu includes the Go To and Skip options. When you select Go To, dBASE asks you to enter a record number. You simply type a record number and press Return. dBASE will "point" to that record. If you then select Display from the Retrieve menu, or select Edit or Browse from the Update menu, the record whose number you entered will appear on the screen.

When you select the Skip option, dBASE will ask that you enter some number. dBASE will then move the record pointer by the number or records that you enter. For example, if you enter 10, the record pointer will skip down ten records. If you enter - 1, then the record pointer will skip back one record.

On the Retrieve menu, along with the Build a search condition and Build a scope condition, there is an option to Specify Scope. When you select this option, a submenu displays the options summarized in Table 5.1.

Option	Effect
Default Scope:	Has basically no effect. Change the default scope by selecting Specify Scope from the menu.
ALL	Specifies all records in the database.
NEXT	Asks for a number (n) and displays the next n records. For example, selecting Next and typing 20 displays the next twenty records (about a screenful).
RECORD	Allows you to define a particular record by its number.
REST	Displays records from the current pointer position to the end of the file.

Table 5.1: Specify Scope Menu Options

Let me reiterate that these menu options are of limited value in managing large databases. However, you may want to become familiar with them for your smaller searching and querying jobs.

Starting with the next chapter, we're going to start learning to program using the dBASE language and leave the Assistant menu once and for all. We'll begin with a discussion of a new way of storing information: memory variables in random access memory (RAM).

Reviewing File Maintenance and Performance

In this chapter you have learned how to maintain files and find records quickly:

To perform basic file maintenance, highlight Tools from the main menu bar, and select one of these options: **Set drive**, **Copy file**, **Directory**, **Rename**, **Erase**, **List structure**, **Import**, or **Export**.

To group files by a particular type of application, use the SET CATALOG command at the dot prompt, or use the **Set Up** and **Catalog** options from the Assistant menu.

To create Queries that you can save and reuse, complete the query forms using the **Query** option under the Create or **Modify** menus.

To activate a Query, select the **Query** option under the **Set Up** menu.

For maximum speed in searching a database, open an index file of the field that you want to search. Use the **Seek** option from the **Position** menu. Then use the **Build a scope condition** option under the **Retrieve** menu to build a WHILE condition.

5.2 Understanding Memory Variables

A Computer's main memory is called Random Access Memory, or RAM. ALL of this memory is available when the computer is turned on : whatever was stored in RAM is lost when you shut off the power. With dBASE III PLUS, you can store data in temporary RAM. Similar to the way that you store data in database fields or disk. Data in RAM are stored in memory variables. Memory Variables are temporary storage places for pieces of information that you are using to solve a problem. With dBASE III PLUS you can store 256 of these Variables. The name that you create for a memory variable can be up to ten characters long.

Let's start to understand memory variables by examining the arithmetic capabilities of RAM. We will work with the dBASE dot prompt, so press the Esc Key to leave the Assistant menu and bring up to dot prompt.

MANAGING DATA IN RAM,

Using RAM memory in your Computer is much like using any pocket calculator, you ask dBASE to calculate some numbers, and it displays the answers. Let's put the computer's main memory (RAM) to work now. Next to the dot prompt, type this command:

```
? 1 + 1 ↵
```

dBASE responds with

2

the sum of 1 Plus 1 Let's give it a tougher problem. Let's ask dBASE to

```
? 25/5 ↵
```

It responds with

5.00

The quotient of 25 divided by 5 Not bad - dBASE is performing as well as a \$5.00 calculator.

Now you can give it an even tougher problem. Suppose that you need to know what the cost of an item selling for \$ 181.93 is, if you must pay 6% sales Tax. That in, you need to know how much

181.93+6% of 181.93 is Type

```
? 181.93 + (.06 * 181.93) ↵
```

dBASE tells you that the total cost in

192.8458

The four decimal places give you more accuracy than necessary for most applications, but it beats paper and pencil you can also work with non-numeric data (called character strings or just strings) in RAM too. For instance, if you type

```
? 'Hi' + 'there' ↵
```

you get the result

```
Hithere
```

Notice that when you “added” two strings they were linked together rather than summed. At first you might think that dBASE was naturally clever enough to figure this out on its own but it was not dBASE’s idea. Rather, it was the fact that you enclosed “Hi” and “there” in apostrophes that told dBASE to link rather than sum. So does this mean that if you enclose the once in I+I in apostrophes, it will also link rather than sum? try it type

```
? 'I' + 'I' ↵
```

and, Yes, dBASE responds with

```
11
```

two once linked together, not summed. The apostrophes told dBASE to treat the ones as character strings, not as numbers. This leaves you another possibility : Hi + there

(no apostrophes). Type

```
? Hi + there ↵
```

dBASE informs you that it can’t perform this operation :

```
Variable not found
```

```
?
```

```
? Hi + there
```

```
Do you want some help? (Y/N) ↵
```

It looks like these apostrophes carry quite a bit of meaning in RAM. There is a very good reason for this, as you will see in a moment. For now, keep in mind that if you want to do math with numbers, do not use apostrophes (I+I). If you want to link strings together, you must use apostrophes (? ‘Hi’ + ‘there’). Let’s explore the reason for the error that occurred when you attempted to? Hi + there (no apostrophes)

Storing data to memory variables with store

How do you store data in memory variables? First, take a name for the memory variable, and ask dBASE to store some information in it. A Variable in the Computer sense is exactly

the same as a variable in the mathematical sense. That is if you know that the variable X equals 10 and that the variable Y equals 5, then you know that $X+Y = 15$. The same is true with computers. To store a value (such as 10) to a variable (such as X) you use the STORE command. Name your first variable x, and store 10 under that name :

```
STORE 10 TO X ↵
```

dBASE displays the brief message

```
10
```

Now create another variable name Y and store 5 under that name :

```
STORE 5 TO Y ↵
```

dBASE displays

```
5
```

When you ask dBASE for the sum of $X + Y$, like this :

```
? X+Y ↵
```

It should respond with 15, which it does :

```
15
```

while the variables are invisible to you right now, you can take a look at them by typing this command :

```
DISPLAY MEMORY
```

dBASE displays your variable names and what you have stored in them :

X	Pub N 10	(10.00000000)
Y	Pub N 5	(5.00000000)
2 variables defined, 18 bytes used		
254 variables available, 5982 bytes available		

It informs you that two memory variables exist : X and Y. Further more, you know that each is Numeric, and that the value of X is 10 and the value of Y is 5 since they are Numeric, you can do basic math them. For example, you can subtract them :

```
? X-Y ↵
```

you get the result

```
5
```

which is the difference of 10-5 To multiply them, type

```
? X*Y ↵
```

you get the product

50

which is 10 times 5. If you want to divide the numbers stored in the memory variables, type

```
? X/Y ↵
```

you get the quotient the answer to 10 divided by 5

you are not limited to simple equations. For example, if you want to get the answer to X plus Y times X, type

```
? X+Y*X ↵
```

The result in

60

dBASE automatically follows the standard order of precedence in math computation. That is, when an equation involves both multiplication and addition, the multiplication is performed first you can alter the order of computations by using parentheses :

```
? (X+Y)*X ↵
```

you get the result

150

In this case, addition was performed first. At this point, you have stored data in two memory variables X and Y so if you ask dBASE to

```
? A+B ↵
```

you get an error message

variable not found

```
? ↵
```

```
? A+B ↵
```

Do you want some help? (Y/N)

because you have asked dBASE to sum A and B, variables that you have not yet used for storing data.

If you again examine the memory variables by typing

DISPLAY MEMORY ↵

you see that you have numbers stored in X and Y :

X	Pub N	10	(10.00000000)
Y	Pub N	5	(5.00000000)

Earlier you got a syntax error when you asked dBASE to ? Hi + There because memory variables HI and THERE do not exist of course, you could create a couple of memory variables called HI and THERE. That is, you can

STORE 'Hello' TO HI ↵

and then

STORE yourself TO THERE ↵

Now you can type the command

? Hi + THERE ↵

and dBASE will respond with

Hello yourself

That is, the contents of memory variables HI and THERE linked together why are they linked? Let 's see what you have stored in your memory variables

DISPLAY MEMORY ↵

Now you have four memory variables :

X	Pub N	10	(10.00000000)
Y	Pub N	5	(5.00000000)
HI	Pub C	"Hello"	
THERE	Pub C	"yourself"	

Memory variables HI and THERE are of the character type. You told dBASE that they were character types by putting apostrophes around them when you stored them (STORE 'Hello' to HI) an important difference between fields and memory variables here when you define types of data in fields, you specify C or N when you CREATE the database. In memory variables, dBASE automatically assumes that data stored without apostrophes (STORE 10 TO X) are numbers, and data stored with apostrophes (STORE 'Hello' TO HI) are characters.

If you wish to link the words Hi and There rather than asking for the contents of the memory variables HI and THERE, use apostrophes :

? 'Hi' + 'there' ↵

This gives the on-screen result

Hi there

The apostrophes told dBASE that you literally wanted to link the words Hi and There To use the same principle with X and Y, ask dBASE to

? X+Y ↵

you get

15

as the answer, the sum of Numeric Variable x (10) Plus Numeric variable Y(5). On the other hand, if you use apostrophes,

? 'X' + 'Y' ↵

you get

X Y

This is literally an X and a Y linked together.

The important aspect of memory variables you should remember is that they are not permanent like database are RAM memory is temporary. Disk storage is permanent when you QUIT dBASE and turn of the computer, your data bases are still safe and sound on disk. However, memory variables are erased completely

Math Functions

Besides simple addition, subtraction multiplication, and division dBASE III PLUS can work with exponent and logarithms. The symbols ^ or * * can be used as an exponent symbol. For example, to see the result of three squared, ask dBASE to display 3 ^ 2

? 3^2 ↵

dBASE responds with 9.00

To see the value of three cubed, using the * * option type

? 3*3 ↵

dBASE respond with

27.00

To see the cube root of 27, raise 27 to the 1/3 power :

? 27^(1/3) ↵

dBASE responds with

3.00

The SQRT function will display the square root of any positive number. For example, to see the square root of 81, type this command :

? SQRT (81) ↵

dBASE responds with

9.00

you can use memory variables in place of numbers, of course. For example, you can store 81 to a memory variable called Z :

STORE 81 TO Z ↵

Then ask for the square root of Z :

? SQRT (Z) ↵

dBASE displays the square root of 81 :

9.00

The ROUND and INT (integer) functions are used to control the number of decimals displayed. For example, store the number 1.234567 to a memory variable called TEST :

STORE 1.234567 To TEST ↵

To see the number rounded to two decimal places, use the ROUND function :

? ROUND (TEST, 2) ↵

dBASE displays

1.230000

The comma two (,) in the ROUND function specifies two decimal places. To see the same number rounded to four decimal places, use the command

? ROUND (TEST,4) ↵

dBASE displays

1.234600

To see the integer (whole number) portion only, you can round to zero decimal places

? ROUND (TEST,0) ↵

which displays

1.000000

similarly the INTL Integer function will displays the number with out decimal places

? INT (TEST) ↵

which displays

1

There is a difference between rounding and using the INT function, however, for example, if you store 1.9999 to a variable called TEST

STORE 1.9999 TO TEST ↵

and print TEST rounded to zero decimal places

? ROUND (TEST,0) ↵

you get the number rounded up to 2

2.0000

However, the INT function truncates the decimal portion without rounding. So if you print the integer portion of TEST

? INT (TEST) ↵

you get

1

For those of you who use logarithms in your work, dBASE III PLUS provides the EXP and LOC functions.

For example, to view the natural exponent (e) of 1, type this command

? EXP(1) ←

dBASE responds with

2.72

To see the natural logarithm of 2.72 type the command

? LOG (2.72)

and dBASE displays

1.00

If you need more decimal just use them in the number, you are working with For example, the command

? EXP(1.00000000) ←

displays

2.7182818

You can store values to memory variables without the use of the STORE command if you like. If you use only a variable name, followed by the equal sign, and then some data to store, this will have the same effect as the STORE command. For example, the command

X = -123

Stores the number -123 to variable X. To prove this, type the command

? X

and dBASE displays

-123

The ABS function changes a negative number to a positive number For example, the command

? ABS(X)

displays

123

Section 5.3 Creating Command Files With Modify

A command file is a disk file that contains a series of commands. A command file is actually a computer program, and from now, we will use these words interchangeably.

The basic procedure for working with command files goes something like this. First you create the command file using the **MODIFY** command. Then you run the command file by asking dBASE to Do the command file. If you find mistakes in the command file, you have to edit it. Correcting these errors in a command file is called debugging. Let's deal with creating command files first.

For your first command file, you will create a program to print mailing labels. This process might seem a bit redundant, but a mailing labels program is an ideal example for explaining the basics of programming : Loops and decision making. Later, you will use the same skills.

Let's write a mailing labels command file now we will call this command file **LABELS**, and it on the same disk as the **MAIL.DBF** database. The command to create a new command file, or edit an existing one is **MODIFY COMMAND** plus the name of the command file. So from the dot prompt, enter this command :

```
MODIFY COMMAND LABELS ↵
```

dBASE displays the prompt

```
Edit : LABELS . PRG
```

and gives you a blank screen on which to write the command file. Let's go ahead and type it. Make sure that you type it exactly as it appears here :

```
? TRIM (FNAME), LNAME ↵
```

```
? ADDRESS ↵
```

```
? TRIM(CITY) +;, '+ STATE + ZIP
```

If you make errors while you are typing the program, you can move the cursor around to make changes. The cursor control keys are the same as those in the **EDIT** and **APPEND** modes and are displayed at the top of the screen when you have typed it exactly as shown, save it by pressing **^w** or **^End**.

You have just written your first command file and stored it on the data disk as **LABELS.PRG** (As you have probably dBASE added the **.PRG**, which stands for program). Let's run the program.

Running command files with do

Let's tell dBASE to use **MAIL.DBF** as the database :

```
USE MAIL ↵
```


Now to run the command file, tell dBASE to

Do LABELS ↵

and you will see one of the records from MAIL.DBF in label format :

Betsy Smith
222 Lemon Dr.
Newyork, NY 01234

dBASE did all three lines in the command file in the order they were placed. That is, dBASE printed the first name with the blanks trimmed off followed by the last name (? TRIM (FNAME), LNAME). Then it printed the address (? ADDRESS). Finally dBASE printed the city followed by a comma, the state, and the ZIP (? TRIM (CITY) +', '+STATE + ZIP). This is the first mailing label.

Let's review the steps :

- ✓ Create the command file using MODIFY COMMAND
- ✓ Name it LABELS.PRG.
- ✓ Save the command file by pressing ^w or ^End
- ✓ Ask dBASE to Do LABELS

dBASE executes each of the command in order. That is dBASE reads the command file left to right, top to bottom, just you read English. The results print in the order that you specify.

That is not bad for a first command file, but you can see one major weakness: if only prints one label. To get dBASE to print all the labels in the database, you must set up a Loop to tell dBASE to go back to the first command and repeat the steps in the command file until each record in the database has been printed in label format.

Setting up loops in programs with do while and enddo

dBASE has a pair of commands called Do WHILE and ENDDO which can be used in a program to repeat a series of commands indefinitely. All you have to do is enclose the commands to be repeated between a Do WHILE and an ENDDO command. Every Do WHILE begins a loop. Which will be ended by an ENDDO command you also need to tell dBASE what the loop. Let's give it a try in your LABELS program.

First, ask dBASE to

MODIFY COMMAND LABELS ↵

dBASE will redisplay LABELS.PRG and let you make some changes :

? TRIM(FNAME), LNAME
? ADDRESS
? TRIM(CITY) +', '+ STATE + ZIP

To make dBASE print labels for every person in the mailing list, you need to put these commands in side a loop, press ^N to make room for a new line, and add the Do WHILE command at the top of the program. You will also need to add a SKIP command to have dBASE Skip down to the next name in the database as it performs the commands in the loop. Then add ENDDO to end the loop. Finally, a RETURN command will tell dBASE to return to the dot prompt after the program has been executed. The command file should be like this :

```
DO WHILE. NOT. EOF ()  
? TRIM (FNAME), LNAME  
? ADDRESS  
? TRIM (CITY) +', '+ STATE + ZIP  
SKIP  
ENDDO  
RETURN
```

Now save the edited version of the command file with ^w or ^End when the dot prompt reappears, type

```
USE MAIL ↵  
DO LABELS ↵
```

to get

```
Betsy Smith  
222 Lemon Dr.  
NewYork, NY 01234  
Record no. 1  
Ruth Doe  
1142 J. St.  
Los Angeles, CA 91234  
Record no. 2  
Lucy Smithsonian  
461 Adams St.  
San Diego, CA 92122-1234  
Record no. 3  
John Q. Smith  
123 A St.  
San Diego, CA 92123  
Record no. 4  
Andy Apple by  
345 Oak St.  
Los Angeles, CA 92123  
Record no. 5
```


Note: Your labels may come out in a different order you can easily specify any order using the index files.

If something goes wrong when you try this, check to see if you typed the program exactly as it appears in the book. If your program seems to be running on and on endlessly. Press Esc to cancel the run and get back to the dot prompt. Then you will need to MODIFY COMMAND LABELS again and correct your program. So that it exactly matches the LABELS.PRG program

Now all the names on the database are on the screen in a mailing label format. Let's summarize this:-

In command file, you told dBASE to Do WHILE.NOT.EOF(). (i.e) "Do everything between here and the ENDDO command as long as you have not reached the EOF(), end of the database file".

The next three lines in the command file then print one label. Then the SKIP command causes dBASE to SKIP down to the next record in the data base. The ENDDO command marks the end of the loop, but the loop is repeated because the EOF() (end of the file) has not been reached yet.

Hence when another label is printed, dBASE has reached the end of the database. This causes the loop to end and the first command under the ENDDO command to be processed. The RETURN command tells dBASE to return is that they have record numbers on them you can get rid of the record numbers by asking dBASE to SET TALK OFF Let's run the program again.

Let's first

USE MAIL ↵

Then, tell dBASE to get rid of the record numbers with

SET TALK OFF ↵

If you have a printer hooked up, you can also

SET PRINT ON ↵

Now let's run the LABELS program. Tell dBASE to

DO LABELS ↵

and you will see that the record numbers have been removed :

Betsy Smith

222 Lemon Dr.

New York NY 01234

Ruth Doe

1142 J. St.

Los Angeles, CA 91234

Lucy Smithsonian

461 Adams St.

San Diego, CA 92122-1234

John Q. Smith

123 A St.

San Diego, CA 92123

Andy Apple by

345 Oak St.

Los Angeles, CA 92123

If you did SET PRINT ON, type the command to

SET PRINT OFF ↵

Otherwise, everything that you type on the screen will also go to your printer.

Printing Blank Lines

There's a problem here. Most mailing labels are one inch tall, and the names are spaced evenly on each one. If you modify your command file then you get correctly so let's

MODIFY COMMAND LABELS ↵

Now press the ↓ key four times to get the cursor on the SKIP command. The press ^N three times to make room for three new lines when you finish your command file is :

Do WHILE . NOT . EOF ()

? TRIM (FNAME), LNAME

? ADDRESS

? TRIM (CITY) + ', ' + STATE + ZIP

-

SKIP

ENDDO

RETURN

Now put in commands to print three blank lines on each mailing label : (i.e) begin each blank line with a ? command.

your program in

```
DO WHILE . NOT . EOF ( )
```

```
? TRIM (FNAME), LNAME
```

```
? ADDRESS
```

```
? TRIM (CITY) + ', ' + STATE + ZIP
```

```
?
```

```
?
```

```
?
```

```
SKIP
```

```
ENDDO
```

```
RETURN
```

Save it with ^w now let's

```
USEMAIL ↵
```

```
DO LABELS ↵
```

You will see the names and address properly formatted for one-inch mailing labels :

Betsy Smith

222 Lemon Dr.

NewYork NY 01234

Ruth Doe

1142 J. St.

Los Angeles, CA 91234

Lucy Smithsonian

461 Adams St.

San Diego, CA 92122-1234

John Q. Smith

123 A. St.

San Diego, CA 92123

Andy Apple by

345 Oak St.

Los Angeles, CA 92123

The labels have the extra three blank lines between them you may wonder why you must repeatedly type USE MAIL. The USE command returns the database to its first record. If you don't USE the database at this point, you will get nothing on the screen when you type Do LABELS

DO LABELS ↵

All you get in the dot prompt. No labels were printed because dBASE is already at the end of the file. Prove this by typing this question :

? EOF () ↵

dBASE responds with

.T.

Which in dBASE's way of saying, "True".

dBASE returns to the top of the database when you type

GO TOP ↵

Then type this command to see your labels :

DO LABELS ↵

It's to your benefit to put the GO TOP command right in to the command file. Add GO TOP to your program by editing the command file with MODIFY COMMAND LABELS your command is :

GO TOP

DO WHILE . NOT . EOF ()

? TRIM (FNAME), LNAME

? ADDRESS

? TRIM (CITY) +', '+ STATE + ZIP

?

?

?

SKIP

ENDDO

RETURN

Then the program will always start with dBASE at the top of the database. The command file would print a label for the first record skip to the next record, go back to the first record, print that label again, skip to the next record, back to the first record... on and on. The command file would print countless mailing labels for the first record in the database.

Making a Loop Easier to Read

Now you have a good mailing label program to use with the MAIL database file dBASE will understand and execute the commands in this file each time that you tell it to DO LABELS. However, you could spruce it up a bit so that it is easier for the human programmer to read.

(i.e) The program is LABELS.PRG :

```
* * * * * Mailing Labels Program Go Top
```

```
DO WHILE . NOT . EOF ( )
```

```
? TRIM (FNAME), LNAME
```

```
? ADDRESS
```

```
? TRIM (CITY) +', '+ STATE + ZIP
```

```
?
```

```
?
```

```
?
```

```
SKIP
```

```
ENDDO
```

```
RETURN
```

Programmers often put titles and comments in their programs as notes to themselves. The comments do not have any effect on the actual program : they are just reminders to the person who wrote the program. To put comments in dBASE programs like this, start the line with an asterisk (*). A lot of asterisks make the line stand out, but only one is necessary.

Also, there is a blank line between the GO TOP command and the start of the DO WHILE LOOP. This line emphasizes the loop. Also, all the commands inside of the DO WHILE LOOP are indented which emphasizes the loop. If you want to make your command file look like this one, just MODIFY COMMAND LABELS. Then press ^N to make room for the title put at least one asterisk in front of the title that you type. Then position the cursor under the GO TOP command, and press ^N Then position the cursor next to the ? TRIM (FNAME) line, press ^V (Insert on) and hit the space bar a few times to indent the line. Do the same for the other lines within the loop. Then save it with ^W (Incidentally if you need to delete a line from a command file, you can use ^Y).

Section 5.4 Designing and Developing Programs

When you create a command file (program), you are actually Writing Software. There are five parts to writing dBASE software. First, you design the program by determining its purpose. Second, unless your program will use an existing database, you must design a database structure. Third, to outline the program, you write pseudo code. Fourth you must write the program. In dBASE you do this by typing MODIFY COMMAND and then typing the program on the screen. Fifth, you run the program to test it. Because you will often make mistakes when you write program, you need to make corrections to debug the program. Let's examine each of these steps in detail.

Step 1 : Outline the General Idea

To write the general idea of a program on paper. For example, let's you want to develop a fancy mailing list system that's quick and easy to use. The general idea might be something like this :

This system will be designed to manage a mailing list, and will operate from a menu of choices when first run, the system will display these mail menu options :

1. Add New names and addresses.
2. Sort the mailing list
3. Print names and addresses
4. Edit data
5. Exit the system

The option to print names and addresses will allow you to specify types of individuals to print data for. The system will be completely "menu-driven." That is, once you Do the main command file, jobs like adding new names, sorting, printing mailing labels, and editing will be performed by simply selecting menu options or answering questions on the screen. The system will be completely automated, so that an individual with no knowledge of dBASE III PLUS could still manage the mailing list.

Now that you have the general idea defined, you need to design a database that will support it.

Step 2 : Design the Database Structure

Structure the database like this :

Record#	FieldName	Type	Width	Dec
1	LNAME	C	20	0
2	FNAME	C	15	0
3	ADDRESS	C	25	0
4	CITY	C	20	0
5	STATE	C	5	0
6	ZIP	C	10	0
7	PHONE	C	13	0

you have already created a database with this structure, MAIL, so this step is done already.

Step 3 : Develop Pseudo Code

It's a little easier to write a program if you first write a reasonable facsimile of it plain English. Doing so is called writing Pseudocode. When you write pseudocode, you should try to specify the logic and series of events that will occur in the program. So that when you have to translate the program to actual dBASE language, much of the task is already defined. Here's a pseudo coded example of the mail menu program for your mailing system :

COMMAND FILE NAME : MENU.PRG

PURPOSE : Present a menu of options for managing the mailing list.

Pseudo Code

- ✓ Set dBASE talk Off
- ✓ Use the mailing list database
- ✓ Set menu choice to 0
- ✓ Repeat the mail menu until option to exit is selected
- ✓ Clear the screen
- ✓ Display the mail menu like this :

Mail menu

1. ADD new names
2. SORT data
3. PRINT labels
4. EDIT data
5. EXIT the mailing system

- ✓ Ask which option is desired
- ✓ If option 1 Selected, APPEND new data
- ✓ If option 2 Selected, SORT by last and first name
- ✓ If option 3 Selected, PRINT mailing labels
- ✓ If option 4 Selected, EDIT data.
- ✓ Redisplay the mail menu (as long as option 5 was not selected)
- ✓ EXIT the mailing system.

Note that you have defined the logic of the MENU program you have also given it a title and mentioned its purpose. If it takes a long time to write the actual program, you can refer back to the pseudo code for reference the pseudo code describes the task in English, but it looks like a program too. This intermediate step makes the next step a little easier. From this point, you can write the actual program using proper dBASE III PLUS commands and syntax.

Step 4 : Writing the Program

Once you have a pseudo code out line of the program, you need to write the actual program. In dBASE, you use MODIFY COMMAND for this. Let's now write the actual program you cannot be as liberal with your sentences as you were in the pseudocode, because dBASE cannot understand English Type.

MODIFY COMMAND MENU ↵

which brings you a blank screen to work with. Now you can type the actual MENU program so that it looks like program 5.1.

```
* * * * * Mailing List System mail menu
```

```
SET TALK OFF
```

```
USE MAIL
```

```
STORE 0 TO CHOICE
```

```
* * * * * Present mail menu
```

```
DO WHILE CHOICE < 5
```

```
    CLEAR
```

```
    ? 'Mail menu'
```

```
    ?
```

```
    ? ' 1. Add new names'
```

```
    ? ' 2. Sort data'
```

```
    ? '3. Print labels'
```

```
    ? '4. Edit data'
```

```
    ? '5. Exit the mailing system'
```

```
    ?
```

```
INPUT Enter your choice (1-5) from above: TO CHOICE * * * * * Perform appropriate task based on CHOICE
```

```
DO CASE
```

```
    CASE CHOICE = 1
```

```
        APPEND
```

```
    CASE CHOICE = 2
```

```
        INDEX ON LNAME + FNAME TO NAMES
```

```
        USE MAIL INDEX NAMES
```

```
    CASE CHOICE = 3
```

```
        DO LABELS
```

```
    CASE CHOICE = 4
```

```
        EDIT
```

```
    END CASE
```

```
ENDDO
```

Program 5.1 MENU PRG for mailing list system.

Now let's discuss how the actual program resembles the pseudo code that you outlined. First, the command file sets the dBASE talk off, and uses MAIL as the database. Then it stores a zero to a memory variable called CHOICE. This is so that the DO WHILE CHOICE < 5 Condition will be true when dBASE first enters the loop. Then it clears the screen and displays the mailmenu using ? commands. Then it displays the question, "Enter your choice (1-5) from above" and waits for an answer when an answer has been entered, the program performs the desired option. If the choice was 1 dBASE goes in to APPEND mode. If the choice was 2, dBASE will sort by last name ; if the choice was 3, dBASE will Do our other command file, LABELS and so forth.

After the selected option has been performed, dBASE will eventually reach the ENDDO command at the bottom of the command file. The loop will repeat, redisplaying the mail menu and question. If the choice is 5, Exit the Mailing System, the DO WHILE CHOICE < 5 condition for the loop will cause the ENDDO not to repeat the loop, and the program will end.

Step 5 : Run and Test the Program

To test your program, you still need to run it, so type this command :

DO MENU ↵

After the screen is cleared, the mail menu is displayed :

Mail Menu

1. Add new names
2. Sort data
3. Print labels
4. Edit data
5. Exit the mailing system

Enter your choice (1-5) from above :

If you now type 1 dBASE will enter the APPEND mode :

```
LNAME   : _      :
FNAME   :         :
ADDRESS :         :
CITY    :         :
STATE   :         :
ZIP     :         :
PHONE   :         :
```

So, any one who types the command DO MENU can add new data to the database without typing the APPEND command. This is because the program has a line which states CASE CHOICE = 1, and beneath that is the APPEND command. Because CHOICE did = 1

dBASE goes into APPEND mode. You can then add as many names to the mailing list as you wish. When you exit the APPEND mode the mailmenu reappears :

Mail Menu

1. Add new names
2. Sort data
3. Print data
4. Edit data
5. Exit the mailing system

Enter your choice (1-5) from above

Why? Because none of the case clauses below the APPEND command in the program will be true dBASE won't do the commands in CASE CHOICE = 2 or CASE CHOICE = 3 and so forth, because CHOICE = 1 when dBASE reaches the ENDDO command, CHOICE is less than 5, so the program loops around up to DO WHILE CHOICE < 5 command, and the menu is redisplayed, and the INPUT question reappears. You could type another option now, and what ever option you select (1-5) will be stored in CHOICE. Then, the appropriate function will take place (APPEND, INDEX, and so forth).

Note in the command file that if you select options, Printlabels, dBASE is told to Do a different command file, your LABELS.PRG program. As you may recall, the last line in LABELS in the RETURN command. When one command file calls another as in this case were MENU calls LABELS the RETURN command tells dBASE to go back to where it left off in the first program therefore, MENU will DO LABELS, and LABELS will print the mailing labels and then RETURN to the next command in MENU.

We are assuming here that your program ran correctly the first time and you got one of dBASE's many error messages. The computer catches them right away. The most common errors that occur in command files are syntax errors. Which result from misspelling a command, a field name or a variable name.

For example,

Suppose that when you asked dBASE to DO MENU, you got the message

Variable not found

?

CASE CHOICE = 1

Terminate command file ? (Y/N)

At first you may not see the problem after you see that the word CHOICE is mis spelled dBASE attempted to find a memory variable called CHOICE, and could not.

If you answer Y to the question "Terminate command file ?" dBASE will return to dot prompt. Then you can MODIFY COMMAND MENU to correct the error.

Unit - VI

6.1 Revisiting Abstract Data types

We can use abstract data types to group related columns into objects. For example, columns that are part of address information can be grouped into an ADDRESS - TY data Type Via the create type command:

```
Create types    ADDRESS_TY as object
(Street        VARCHAR 2 (50),
City           VARCHAR 2 (25),
State          CHAR(2),
ZIP            NUMBER;
```

The Create type Command in the preceeding listing creates an ADDRESS_TY abstract data type. You can use ADDRESS_TY when creating additional database objects for examplly the folowing create type command creates the PERSON_TY data type using the ADDRESS_TY data type as the data type for its Address Column:

```
Create type    PERSON_TY as object
(Name         VARCHAR 2 (25),
Address       ADDRESS_TY);
/
```

Because the Address Column of the PERSON_TY data type uses the ADDRESS_TY data type, it holds not one value but four - the four Columns that constitute the ADDRESS_TY data type.

6.2 Security for Abstract Data types

The previous example assumed that the same user owned both the ADDRESS_TY data type and the PERSON_TY data type. What if the owner of ther PERSON_TY data type were different from the ADDRESS_TY data type's onwer? For example, what if the account named Dora own the ADDRESS_TY data type, and the user of the account named George tries to create the PERSON_TY data type? George executes the following Command:

```
Create type    PERSON_TY as object
(Name         VARCHAR 2 (25),
Address       ADDRESS_TY);
/
```

If George does not own the ADDRESS_TY abstract data type, then oracle will respond to this crerate type command with the following message:

Warning: Type created with compilation errors.

The Compilation errors are caused by problems creating the constructor method - a special method created by oralce whent the data type is created. Oracle cannot resolve the reference to the ADDRESS_TY data type, since George does not own a data type with that

name, George could issue the create type command again (using the or replace clause) to specifically reference Dora's ADDRESS_TY data type.

```
Create or replace type PERSON_TY as object
(Name      Varchar 2 (25),
Address Dora.ADDRESS_TY);
```

Warning : Type created with compilation errors.

To see the errors associated with the datatype creation, use the show errors command:

Show errors

Errors for type PERSON_TY:

Line / COL ERROR

```
-----
0/0   PL / SQL: Compilation unit analysis terminated
3/11  PLS-00201: identifier 'DORA.ADDRESS_TY' must be declared.
```

George will not be able to create the PERSON_TY datatype (which includes the ADDRESS_TY datatype) unless Dora first grants him EXECUTE privilege on her type. The following listing shows this grant:

grant Execute on Address_TY: to George

Now that the proper grants are in place, George can create a data type that is based on Dora's ADDRESS_TY datatype, as shown in the following listing.

```
create or replace type PERSON_TY as object
(Name      VARCHAR 2 (25),
Address Dora.ADDRESS_TY);
```

Using another user's data types is not trivial. For example, during insert operations, you must specify the owner of each type. George can create a table based on his PERSON_TY data type (which includes Dora's ADDRESS_TY data type) as shown in the following listing:

```
Create table      CUSTOMER
(Customer_ID      NUMBER,
Person            PERSON_TY);
```

If George owned the PERSON_TY and ADDRESS_TY data types an insert into CUSTOMER would use the format:

```
insert into CUSTOMER Values
(1, PERSON_TY ('Some Name',
ADDRESS_TY ('street value', 'city value', 'ST', 11111))).
```


Since George does not own the ADDRESS_TY data type. This command will fail. During the insert, the ADDRESS_TY constructor method is used and Dora owns it. Therefore the insert command must be modified to specify Dora as the owner of ADDRESS_TY. The following example shows the corrected insert statement, with the reference to Dora shown in bold:

```
insert into CUSTOMER Values
(1, PERSON_TY ('some Name;
Dora.ADDRESS_TY ('Street value', 'city value,' 'ST', 11111)))
```

Can George use a synonym for Dora's data type? No. Although George can create a synonym named ADDRESS_TY:

```
Create synonym ADDRESS_TY for Dora.ADDRESS_TY;
```

this synonym cannot be used:

```
create type PERSON_2_TY
(Name          VARCHAR 2 (25),
Address       ADDRESS_TY);
```

```
Create type PERSON_2_TY
```

```
*
```

```
ERROR at line /:
```

```
ORA -22863: synonym for data type DORA.ADDRESS_TY not allowed
```

As shown by the error message, you cannot use a synonym for another user's datatype. Therefore you will need to refer to the datatype's owner during each insert command.

NOTE

When you create a synonym, oracle does not check the validity of the object for which you are creating a synonym. When you create synonym x for y, oracle does not check to make sure that y is a valid object name or valid object type. The validity of that object is only checked when the object is accessed via the synonym.

6.3: Indexing Abstract Datatype Attributes

Now the CUSTOMER table was created, the CUSTOMER table contains a normal column (Customer-ID) and a person column that is defined by the PERSON_TY abstract data type:

```
Create table CUSTOMER
(Customer_ID  NUMBER,
Person       PERSON_TY);
```

The Name column is an attribute of the data type that defines the person column, so refer to the attributes as person Name:


```
Select Custc er_ID, C. person.Name
from CUSTO ER C;
```

You refer to attributes within the ADDRESS_TY data type by specifying the full path through the related columns. For example, the street column is referred to as person. Address. street, fully describing its location within the structure of the table. In the example, the city column is referenced twice: Once in the list of columns to select, and once within the Where clause. In each reference, a correlation variable is required

```
Select C. Person. Name,
C. Person. Address.City
from CUSTOMER C
where C Person Address. City like 'F%';
```

Because the city column is used with a range search in the where clause, the oracle optimizer may be able to use an index when resolving the query. If an Index is available on the city column, then oracle can quickly find all the rows that have city values starting with the letter F, as requested by the query.

To create an index on a column that is part of an abstract datatype, you need to specify the full path to the column as part of the create index command. To create an index on the city column (which is part of the Address column), you can execute the following command.

```
create index I - CUSTOMER$CITY
on CUSTOMER (person. Address. City);
```

This command will create an index named I - CUSTOMER\$ CITY on the Person Address.City column. Whenever the city column is accessed, the oracle optimizer will evaluate the SQL used to access the data and determine if the new Index can be useful to improve the performance of the access.

When creating tables based on abstract datatypes, you should consider how the columns within the abstract data types will be accessed. If like the city column in previous example. Certain columns will commonly be used as part of limiting conditions in queries then they should be indexed. In this regard, the representation of multiple columns in a single abstract data type may hinder your application performance, since it may obscure the need to index specific columns or the person columns. It is important to remember that the optimizer, when evaluating query access paths will consider the columns individually. In addition, remember that indexing the city column in one table that uses the ADDRESS_TY data type does not affect the city column in a second table that uses the ADDRESS_TY data type.

For example: if there is a second table named BRANCH that uses the ADDRESS_TY data type, then its city column will not be indexed unless you create an index for it. The fact that there is an index on the city column in the CUSTOMER table will have no impact on the city column in the BRANCH table.

Thus, one of the Advantages of abstract datatypes - the ability to improve adherence to standard for logical data representation - does not extend to the physical data representation. You need to address each physical implementation separately.

Implementing object views

The CUSTOMER table created in the previous section assumed that a PERSON_TY data type already existed when implementing object - relational database applications. Oracle provides object views as a means for defining objects used by existing relational tables.

The order of operations was as follows:

1. create the ADDRESS_TY data type.
2. create the PERSON_TY datatype, using the ADDRESS_TY data type.
3. create the CUSTOMER table, using the PERSON_TY data type.

There are two main benefits to using object views. First, they allow you to create abstract data types within tables that already exist. Since you can use the same abstract datatypes in multiple tables within your application, you can improve your application's adherence to standard representation of data and your ability to reuse existing objects. Since you can define methods for the abstract datatypes, the methods will apply to the data within any new tables you create as well as to your existing tables. Second, object views provide two different ways to enter data into the base table. The flexibility of data manipulation for object views - being able to treat the base table as either a relational table or an object table - is a significant benefit for application developers.

Manipulating Data Via Object Views

You can update the data in the CUSTOMER table via the CUSTOMER_ov object view, or you can update the CUSTOMER table directly. By treating CUSTOMER as just a table, you can insert data into it via a normal SQL insert Command, as given following insert into CUSTOMER values

```
(123, 'SIGMUD', '47 HAFFNER RD', 'Lewiston', 'NJ' 22222);
```

This insert Command inserts a single record into the CUSTOMER table. Even though you have created an object view on the table, you can still treat the CUSTOMER table as a regular relational table.

Since the object view has been created on the CUSTOMER table, you can insert into CUSTOMER via the constructor methods used by the view. When you insert data into an object that uses abstract datatypes. You specify the names of the constructor, methods within the insert command. The example in the following listing inserts a single record into the CUSTOMER_ov object view using the CUSTOMER_TY. PERSON_TY and ADDRESS_TY constructor methods.

```
insert into CUSTOMER_ov values
(2 3 4,
PERSON_TY ('EVELYN',
ADDRESS_TY ('555 HIGH ST,' 'LOWLANDS PARK', 'NE', 3333))));
```

Since you can use either method to insert values into the CUSTOMER_ov object view, you can standardize the manner in which your application performs data manipulation. If your inserts are all based on abstract data types, then you can use the same kind of code for inserts regardless of whether the abstract datatypes were created before or after the table,

Using INSTEAD OF Triggers

If you create an object view you can use INSTEAD OF triggers to tell oracle how to update the underlying tables that are part of the view. You can use INSTEAD OF triggers on either object views or standard relational views. For example, if a view involves a join of two tables. Your ability to update records in the view is limited. However, if you use an INSTEAD OF trigger. You can tell oracle how to update, delete or insert records in tables when a user attempts to change values via the view. The code in the INSTEAD OF trigger is executed in place of the insert, update or delete command you enter.

Let's consider two of Talbot's most-used tables: WORKER and LODGING. The WORKER table has three columns:

Name, Age and Lodging. The LODGING table has four columns: Lodging, Long Name, Manager and Address. You could select all of the columns from both tables by joining them on the Lodging column, as shown here:

```
Select    WORKER.Name,
          WORKER.Age,
          WORKER.Lodging,
          LODGING.Lodging,
          LODGING.Long Name,
          LODGING.Manager
          LODGING.Address
from      WORKER, LODGING
Where     WORKER. Lodging = LODGING.Lodging;
```

Now you can select which columns you want from each table and use this query as the basis for a view - For example, you can select the worker name, lodging. and lodging manager name from the two tables:

```
Create    View WORKER_LOADING_MANAGER as
Select    WORKER.Name,
          LODGING.Lodging,
          LODGING.Manager
from      WORKER, LODGING
where     WORKER.Lodging = LODGING.Lodging;
```


You can select values from this view - and if you use INSTEAD of triggers, you can also perform data manipulation via this view

```
Select*
from WORKER _LODING _MANAGER
where Name = 'BART SARJEANT';
```

Name	LODGING	MANAGER
BART SARJEANT	CRANMER	THOM CRANMER

Bart Sarjeant resides in the lodging named CRANMER, managed by Thom Cranmer. However he has decided to move to the WEITBROCHT lodging instead. You can indicate this in one of two ways. You can perform the update directly against the WORKER table:

```
Update WORKER
Set Lodging = 'WEITBROCHT'
Where Name = 'BART SARJEANT';
```

or you can use an INSTEAD OF trigger and update via the WORKER_LODGING_MANAGER View

In the following listing, an INSTEAD of trigger for the WORKER_LODGING_MANAGER view is shown:

```
Create trigger WORKER_LODGING_MANAGER_update
instead of UPDATE on WORKER_LODGING _MANAGER
for each row
```

```
begin
if: old Name <>: new.Name
then
update WORKER
Set Name = : new. Name
Where Name = : old.Name;
end if;
if: old.Lodging <>:new.Lodging
then
update WORKER
SET LODGING = :NEW.LODGING
Where Name = :old.Name;
```

```

end if;
if:old.Manager<>:new. Manager
then

    update LODGING
        Set Manager = :new.Manager
    Where Lodging = :old.Lodging;
end if;
end;
/

```

In the First part of this trigger, the trigger is named and its purpose is described via the instead of clause. The trigger has been named to make its function clear: it is used to support update commands executed against the WORKER_LODGING_MANAGER view. It is a row-level trigger; each changed row will be processed, as shown below.

create trigger WORKER_LODGING_MANAGER_UPDATE instead of UPDATE on WORKER_LODGING_MANAGER for each row.

The next section of the trigger tells oracle how to process the update. In the first check within the trigger body, the Name value is checked. If the old Name value is equal to the new Name value, then no changes are made. If the values are not the same, then the WORKER table is updated with the new Name value:

```

begin

    if:old.Name <>:new.Name
    then
        update WORKER
            Set Name = :new.Name
        where Name = :old.name;
    end if;

```

In the next section of the trigger, the Lodging value is evaluated. If the Lodging value has changed, then the WORKER table is updated to reflect the new lodging value for the WORKER:

```

if:old. Lodging<>:new.Lodging
then
    update WORKER
        Set Lodging = :new.Lodging
    where Name = :old.Name;
end if;

```


In the final section of the trigger, the Manager column is evaluated; if it has been changed, then the LODGING table is updated with the new value:

```

if: old. Manager<>:new.Manager
then
  update LODGING
    Set Manager = :new.Manager
  where Lodging = :old.Lodging;
end if;
end;
/

```

Thus, the view relies on two tables - WORKER and LODGING and an update of the view can update either or both of the tables. The INSTEAD OF trigger, introduced to support object views is a powerful tool for application development.

You can now update the WORKER_LODGING_MANAGER view directly and have the trigger update the proper underlying table for example, the following command will update the

```

WORKER table:
update WORKER_LODGING_MANAGER
Set Lodging = 'WEITBROCHT'
Where Name = 'BART SARJEANT';
/ row updated
verify the update by querying the worker table:
Select Name, Lodging
from WORKER
where Name = 'BART SARJEANT';

```

NAME	LODGING
BART SARJEANT	WEITBROCHT

You can also update the LODGING table via the WORKER_LODGING_MANAGER view, making John Peletier the new manager of Bart Sarjeant's lodging.

```

update WORKER_LODGING_MANAGER
Set Manager = 'JOHN PELETIER'
Where Name = 'BART SARJEANT';
/ row updated.

```

Verify the change by selecting John Peletier's row from the LODGING table:

```
Select*
from LODGING
Where Manager = 'JOHN PELETIER';
```

LODGING	LONGNAME
MANAGER	ADDRESS
ROSE HILL	ROSEHILL FOR MEN
JOHN PELETIER	RFD 3, N.EDMESTON.
WEITBROCHT	WEITBROCHT ROOMING
JOHN PELETIER	320 GENEVA, KEENE

INSTEAD of triggers are very powerful. As shown in this example, you can use them to perform operations against different database tables, using the flow-control logic available with in PL/SQL.

Methods

When you created an object view on the CUSTOMER table, you defined abstract data types that the table used. Those data types help to standardize the representation of data, but they serve another purpose as well. You can define methods that apply to data types, and by applying those data types to existing tables, you can apply those methods to the data in those tables

Consider the CUSTOMER_OV object view:

Create or replace View CUSTOMER_ov (Customer_ID, Person) as

```
select Customer_ID,
```

```
PERSON_TY (Name)
```

```
ADDRESS_TY (Street, City, State, Zip)
```

```
from CUSTOMER;
```

This view applies te PERSON_TY and ADDRESS_TY abstract data types to the data in the CUSTOMER table. If any methods are associated with those data types, they will apply to the data in the table. When you create an abstract data type, you use the create type command. When you create a method, you use the create type body command.

Syntax for creating methods

Before creating the body for a method, you must name the method within the data type declaration. For example, let's create a new data type, ANIMAL_TY. The data type definition for ANIMAL_TY is shown in the following listing:


```

Create or replace type ANIMAL_TY as object
(Breed          VARCHAR 2 (25),
Name           VARCHAR 2 (25),
Birth Date    DATE,
member function AGE (Birth Date IN DATE) return Number),
/

```

The ANIMAL_TY data type could be used as part of an object view on the BREEDING table. In the BREEDING table, the name of an off spring its sex, its Parents, and its birth date are recorded.

Within the create type command, this line:

```
Member function AGE (Birth Date IN DATE) return NUMBER.
```

Names the function that is a "member" of the ANIMAL_TY datatype. Since AGE will return a value, it is a function; if no values were returned. To define the AGE function, use the create type body Command, whose full syntax is shown in the Alphabetical Reference.

Let's create the AGE function as a member function within the ANIMAL_ty datatype. The AGE function will return the age, in days, of the animals:

```

create or replace type body ANIMAL_TY as
member function Age (Birth Date DATE) return NUMBER is begin.
RETURN ROUND (Sys Date - Birth Date);
end;
end;
/

```

If you had additional functions or procedures to code, you would specify them within the same create type body command, before the final end clause.

Now that the AGE function has been created, it is associated with the ANIMAL_TY datatype. If a table uses this data type, then it can use the AGE function. However, there is a problem with the way in which the function is specified. The AGE function, as previously defined in the create type body command, does not explicitly restrict updates. The AGE function does not update any data, but there is no guarantee that the type to call a member function within a SQL statement, you need to make sure that the function cannot update the database. Therefore, you need to modify the function specification in the create type command. The modification is shown in bold in the following listing:

create or replace type ANIMAL_TY as object

(Breed VARCHAR 2 (25),

Name VARCHAR 2 (25),

Birth Date DATE,

Member function AGE (Birth Date INDATE) return NUMBER, PRAGMA
RESTRICT_REFERENCES (AGE, WNDS));

You can use the AGE function within a query If ANIMAL_TY is used as the data type for a column named animal in the table named ANIMAL. Then the table structure could be as follows:

Create table ANIMAL

ID NUMBER,

Animal ANIMAL_TY);

insert into ANIMAL Values

(11, ANIMAL_TY ('COW', 'M1M1', To_DATE ('01-JAN-1997', 'DD-MON-YYYY'))).

You can now invoke the AGE function, which is part of the ANIMAL_TY data type. Like the attributes of an abstract data type, the member methods are referenced via the column names that use the data types (in this case, Animal.AGE):

Select A.Animal. AGE (A.Animal.Birth Date)

from ANIMAL A;

A.ANIMAL.AGE (A. ANIMAL.BIRTHDATE)

1034

6.4: How oracle works

The following steps describe an oracle configuration in which the user and the associated server process are located in separate machines and are connected through a network:

1. An instance is executed in a computer where oracle is configured (most often called host or data base server)
2. A computer executing an application (a local machine or a client work station) executes the application in a user process. The Client application tries to establish a connection with the server using the appropriate Nets driver.
3. The server executes the appropriate Net 8 driver and detects the request for connection from the application, thus creating a (dedicated) server process on behalf of the user process.
4. The user executes a SQL declaration and submits the transaction for example, the user changes a name in a table's row.

5. The Server process receives the declaration and checks the shared pool of the shared SQL area containing an identical SQL declaration. If a shared SQL area is found, the server process checks the user's access privileges to data requested, and this pre-existing area is used to process the declaration; otherwise, a new shared SQL area is allocated to the declaration, allowing it to be analysed and processed.
6. The Server process recovers all the data values needed from the real data file (table) or those that are stored in the system global area.
7. The Server process changes data in the system global area. The DBWn process permanently writes the changed blocks to disk, if this is an efficient operation. Due to the submitted transaction, the LGWR process immediately records the transaction to the online redolog file.
8. When the transaction is successful, the server process sends a message through the network to the application otherwise, an appropriate error message is transmitted.
9. In this procedure, other back ground processes are executed, observing the conditions that require intervention. In addition, the database server also manages other user transactions and prevents conflicts among the transactions that request the same data.

These steps describe the most basic level of the operations performed by Oracle.

DEFAULTS

CONSTRAINT

TABLE CONSTRAINT

PCTFREE

PCTUSED

Unit - VII

7.1: Creating And Maintaining Tables

In the chapters about the SQL Language you learned how to perform some table operations. Such as SELECT, INSERT, UPDATE and DELETE. In the previous chapter you saw that during the design of a table's structure you can add constraints to enforce the data integrity.

Before creating a table, you must analyze and define the data-base's structure and functionality, following the seven steps of system development. The tables should be created only after the logical structure is ready. It is important to note that tables containing new types of objects cannot be imported by versions of oracle previous to Oracle 8.

The create table Command

The complete syntax of the CREATE TABLE Command is included at the end of this Chapter. However, below is a summary of the syntax. The CREATE TABLE Command, creates a new table through the following operation:

- (i) Definition of the columns
- (ii) Definition of the table organization.
- (iii) Integrity constraints
- (iv) Tablespace of the table
- (vi) Storage characteristics
- (vii) An optional cluster
- (viii) Query data
- (ix) Rate of parallelism used in creation of the table, and default rate of parallelism for the queries of the table.
- (x) Definition of Partitions
- (xi) Arrangement by index or Leap (the storage area that allows random access).

Basic Syntax

CREATE TABLE Schema.table (Column/datatype)

DEFAULT expr

column_Constraint,.....Columnx)

PCTFREE integer

PCTUSED integer

INITRANS integer

MAXTRANS integer

TABLESPACE tablespace

STORAGE Storage _ Clause

ENABLE enable - clause

DISABLE disable - Clause As subquery PARALLEL

Paralle - Clause

CACHE

NOCACHE (Column data type)

RECOVERABLE

UNRECOVERABLE table_Constraint.

Arguments

Schema

An optional Parameter that indicates the Schema to which the table belongs

Table

The name of the table to be created.

Datatype

Specifies the column's data type. Uptc 254 columns are allowed

DEFAULTS

This is optional, and indicates a default value for the column.

CONSTRAINT

This is optional and indicates some integrity control that applies to the column, for example, it must contain a value.

TABLE CONSTRAINT

This is Optional and indicates some integrity control that applies to the table, for example, the primary key.

PCTFREE

This is optional and indicates the percentage of each data Block that will be reserved for failure updating in the table. The default value is 10.

PCTUSED

This is optional and indicater the minimum percentages of used space that oracle must maintain before a data block is eligible for a row insertion. The default value is 40.

INITTRANS

The default value is 1. It can vary from 1 to 255 and indicates the number of transactions assigned inside the data block of the table.

MAXTRANS

Indicates the maximum number of transactions that can update a data block.

TABLESPACE

This is optional. It indicates a tablespace different from that used as default.

CLUSTER.

This is optional, it indicates that the table is part of a cluster.

ENABLE

This is optional. It links the integrity restrictions.

DISABLE

This is optional, it disables the integrity restrictions.

Before we start creating a simple table let's analyze the following table, based on the knowledge, we have so far, and try to understand its structure.

CREATE TABLE:

```
emp C
empno    NUMBER (5) PRIMARY KEY,
ename    VARCHAR2 (15) NOT NULL,
Job      VARCHAR 2 (10),
Mgr       NUMBER (5),
hiredate  DATE DEFAULT (sysdate),
Sal       Number (7,2)
Comm      NUMBER (7,2),
dept no   NUMBER (3) NOT NULL.
```

CONSTRAINT dept_fkey REFERENCES dept)

PCTFREE 10

PCTUSED 40

TABLESPACE USERS

STORAGE (INITIAL 50 K

NEXT 50 K

MAX EXTENTS 10

PCT INCREASE 25);

This command creates a table called EMP in the users tablespace. It initially allocates 50 k bytes of space, and specifies 10% of free space and 40% of space that is used for the PCTUSED

Parameter. The Empno Column is of type number with a size of 5, and is the primary key of the table,. The Ename column can contain, data up to 15 characters and does not allow null values. The Hiredate column defaults to the systemdate if no value is specified. The dept no column does not allow null values and has a FOREIGN KEY constraint, which references the DEPT table.

Let's create a simple table.

```
SQL> create table newone  
Z (firstcol number ) (4),  
3 secondcol Varchar (30),  
4 thirdcol date);
```

Table created.

The structure of the table is;

```
SQL> describe newone;
```

Name	Null?	Type
------	-------	------

FIRSTCOL		NUMBER (4)
SECONDCOL		VARCHAR 2 (30)
THIRDCOL		DATE

```
SQL >
```

In the next example, the primary key (index) and some constraints are created.

```
SQL> create table newagain
```

```
(2) (first numbe (4) Constraint P/C new Primary Key,
```

```
Second date default sys date,
```

```
third varchar 20 not null);
```

Table created.

Now, Let's try to insert some seconds:

```
SQL> insert into newagain values (2222, 'bbbb');
```

```
insert into newagain values (2222, 'bbbb')
```

*

```
ERROR ON row /;
```

```
ORA-00936:missing expression.
```

Because the Second column has a default option, we've omitted the specification. This causes an error. When you don't specify the names of the fields, all the fields must be given. The following shows the proper way to specify this information.

```
SQL> insert into newagain values (2222, 'bbbb');
```

```
1 row created.
```

Here there is no error, since we assigned a blank value for the field. In addition, by specifying values only for the first and third columns, as shown below, no error resulted:

```
SQL> insert into newagain (first, third) values
```

```
(3333, 'CCCC');
```

```
/ row created.
```

Let's try to insert a null value in the third column. Note that the constraint blocked this action:

```
SQL> insert into newagain values (4444; 12/12/1999 null);
```

```
insert into newagain values (4444, '12/12/1999', null)
```

```
*
```

```
ERROR on row 1;
```

```
ORA-01400; it is not possible to insert
```

```
Null in ("SCOTT", "NEWAGAIN", "THIRD")
```

Below is the result of the insertion. Note that the third row has the data. value generated by sys date while the second row accepts the blank value that we specified.

```
SQL> Select*from newagain;
```

```
FIRST      SECOND  THIRD
```

```
1111 10/10/99  aaaa
```

```
2222          bbbb
```

```
3333 05/12/99  CCCC
```

The following example creates a table identical to EMP. This will help us understand constraints. Be sure to specify a name for the constraint when you create it;.

CREATE TABLE SCOTT.emp

(Empno NUMBER CONSTRAINT P/<_emp PRIMARY KEY,

ename.VARCHAR2(10), CONSTRAINT nn_ename Not Null

CONSTRAINT upper_ename

CHECK (ename=UPPER (ename),

Job VARCHAR 2 (9),

MGR NUMBER

CONSTRAINT fk_mgr

REFERENCES

Scott emp(Empno),

hiredate DATE

DEFAULT SYS DATE,

Sal NUMBER (10,2)

Constraint CK_SAL

check(SAL>500),

DEFAULT NULL,

Comm NUMBER (9,2)

CONSTRAINT nn_dept.no

dept no NUMBER (2)

NOT NULL

CONSTRAINT fk_dept no

PCTFREE 5 PCTUSED 75;

REFERENCES SCOTT.dept

(deptno)

To demonstrate the above example, we will create. Several tables, in the first example, we create a PRIMARY KEY Constraint to avoid duplicate Codes, and a CHECK Constraint, which inturn, checks to see if the contents of the field are all upper case.

SQL> create table test a constr.

2 (empno NUMBER CONSTRAINT P/<_emp PRIMARY KEY,

3 ename VARCHAR 2 (10) CONSTRAINT nn_ename NOT NULL

4 CONSTRAINT upper_enmae

5 CHECK (ename=upper (ename),

6 Job VARCHAR 2 (9))

Now we insert a row, where the Ename field is not upper case.

SQL> insert into testaconstr values (111,

'Ramal 1 ho', 'manager');

insert into testaconstr values (111,

'Rama 1 ho,' 'Manager')

*

ERROR on row 1:ORA-02290: chekcing constraint (SCOTT.UPPER_ENAME)

violatedSQL> insert into testaconstr values (111, 'RAMALHO', 'Manager');

1 row created.

SQL> Select.* from testaconstr;

EMPNO	ENAME	JOB
111	RAMALHO	Manager

Now insert another record, containing the same code for the Empno field:

SQL> insert into testaconstr values (111, 'SILVA', '(CLERK)');

*

ERROR on row 1;

ORA - 00001: Unique Constraint (SCOTT.PK_EMP2) violated

The Constraint Prevents the incorrect registration:

SQL>run

```
1 Create table democonstr 2
2 (empno NUMBER CONSTRAINT P/<_emp3 PRIMARY KEY,
3  ename VARCHAR 2 (10),
4  Job VARCHAR 2 (9),
5  Mgr NUMBER CONSTRAINT F/<_Mgr
6  REFERENCES democonstr 2 (empno),
7  hiredate DATE DEFAULT SYSDATE.
8*
```

Now let's insert a row in which an employee is the Manager of his department, and therefore his code appears in both the Empno and Mgr columns:

SQL> insert into democonstr2 values

(2222, 'SAROUT', 'Designer', 2222, '12/05/1999');

1 row created.

Let's try to insert another record. In this example, the manager's record has not been created yet:

1*insert into democonstr2 values

(3333, 'ARANTES', Writer, 1111, '12/05/1999')

insert into democonstr2 values

(3333, 'ARANTES', Writer, 1111, '12/05/1999')

*

ERROR on row 1

ORA-02291: integrity constraint (SCOTT.FK_MGR)

Violated-

Parent key not found

In this case, you first have to register the manager's record, or assign the value NULL to it, an option that is accepted by the column Constraint, as shown in the next example:

SQL> insert into democonstr2 values

(3333, 'ARANTES', Writer, null, '12/05/1999')

1 row created

SQL> Select * from democonstr2;

EMPNO	ENAME	JOB	MGR	HIREDATE
2222	SAROUT	Designer	2222	12/05/99
3333	ARANTES	Writer		12/05/99

And now our third example:

Create table democonstr 3

(empno NUMBER CONSTRAINT P/<emp4 Primary Key,

ename VARCHAR 2 (10)

Sal NUMBER (10,2) CONSTRAINT c/k_sal

CHECK (Sal>500).

deptno NUMBER (2) CONSTRAINT nn_deptno NOTNULL

CONSTRAINT FK_dep

REFERENCES Scott. dept (deptno))

Let's try to insert a record with a salary below \$500. A constraint immediately interrupts the attempt:

```
SQL> insert into democonstr 3 values
```

```
(333, 'SCORMIN', 400, 20);
```

```
insert into democonstr 3 values (333, 'SCORMIN', 400, 20)
```

```
ERROR on row 1;
```

```
ORA-02290: checking Constraint (SCOTT.CK_SAL) violated
```

Now insert the correct record, changing the amount from \$400 to \$800, and executing the RUN command

```
SQL> C/400/800
```

```
1*insert into democonstr3 values
```

```
(333, 'SCORMIN', 800, 20)
```

```
SQL>run
```

```
1*insert into democonstr3 values
```

```
(333, 'SCORMIN', 800, 20)
```

```
1 row created.
```

Now let's try to insert a record with an incorrect department Code:

```
SQL>insert into democonstr3 values
```

```
(444, 'TEODOZ10', 1200, 80)
```

```
insert into democonstr3 values (444, 'TEODOZ10', 1200, 80)
```

```
*
```

```
ERROR on row 1:
```

```
ORA-025291:integrity restriction (SCOTT.FK_DEP) violated-
```

```
Parent key not found.
```

Because of the Constraint, the operation was voided.

7.2: Creating a Table with the Contents of Another Table

Using the AS option, we can create a table with the SELECT command to recover the structure and data of another table. The next example creates a table called NEWEMP with all the columns of the EMP table and only the rows of department 20:

```
SQL> create table newemp
2 as select * from emp
3 where dept no = 20;
```

Table created.

```
SQL > Select * from newemp;
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	12/17/80	800		20
7566	JONES	MANAGER	7839	04/02/81	2975		20
7788	SCOTT	ANALYST	7566	04/19/87	3000		20
7876	ADAMS	CLERK	7788	05/23/87	1100		20
7902	FORD	ANALYST	7566	12/03/81	3000		20

Using the PARALLEL option

To create or read a table that uses all the parallel execution servers possible, use the PARALLEL OPTION. Such a resource will depend on the computer where the query is executed.

```
SQL > creates table newemp 2
2 Parallel
3 as select * from emp;
```

Table created.

Use parallelism in the creation of a table, since oracle uses parallel execution servers to create a table. After the table is created, searches in the table are also faster, because the same parallelism rate is used to access it.

Constraint Disabling

the following table is created with a primary Key. However, it will remain disabled. This is an interesting resource, used when the table is created from another table. The data is ready, and the creation of the table takes less time. Since the checking is omitted:-

```
SQL> CREATE TABLE dept 1
2 (deptno NUMBER (2) PRIMARY KEY DISABLE,
3 d name VARCHAR 2 (10),
4 1OC VARCHAR 2 (9);
```

However this option will also the creation of duplicate keys, as the next two commands show;

```
SQL > insert into dept 1 values (10 'aaa', 'aaaa')
```

1 row created.

```
SQL > insert into dept 1 Values (10, 'bbbb', 'bbbbbb');
```

1 row created

Modifying Tables

There are Several reasons for making changes to tables. These include adding or removing columns and constraints; changing characteristics of columns, such as data type, size, and default value; changing storage parameters or the space used by data blocks, and removing or disabling triggers and constraints associated to the table.

Consideration and Limitations

Not all the parameters of a table can be changed. A column can have its width increased or, if there are no rows in the table, it can be decreased. The INITIAL AND MINEXTENTS parameters cannot be changed, while changes in the remaining storage parameters will only be effective after the allocation of new extents.

The ALTER TABLE Command

The Command responsible for modifying tables is ALTER TABLE. Its Syntax is similar to the syntax of the CREATE TABLE Command. The only difference is that you specify the parameter you want to change. Following are some examples.

To change the space used by the data blocks, use the following:

```
ALTER TABLE Newemp
PCTFREE 30
PCTUSED 60;
```

Using the PARALLEL Option

A table created in the default serial mode can have its processing parallel through the PARALLEL option:

```
ALTER TABLE Newemp
PARALLEL;
```

Enabling a Constraint

To enable a Constraint that was disabled or created with the DISABLE option, use the ENABLE option. In the next example the primary key of the DEPT 1 table is reactivated:


```
SQL > ALTER TABLE DEPT 1
      2  ENABLE PRIMARY KEY;
ALTER TABLE DEPT 1
```

*

ERROR on row 1

ORA-02437: it is not possible to validate
(SCOTT.SYS_Coo1275

This error occurred because there are two rows with the same contents for the primary key, as shown below".

```
SQL > SELECT * FROM DEPT 1;
```

DEPTNO	DNAME	LOC
10	aaaa	aaaa
10	bbbb	bbbb

After we delete the redundant row, we change the table again. Now we are successful:

```
SQL> DELETE FROM DEPT 1 WHERE DNAME = 'aaaa';
```

1 row deleted.

```
SQL > ALTER TABLE DEPT 1
```

```
      2  ENABLE PRIMARY KEY;
```

Table changed.

Changing a column

To change the characteristics of a column, you use the MODIFY OPTION, and specify the characteristics for the column. The following example adds a NOT NULL Constraint to the Sal Column of the NEWEMP Table:

```
ALTER TABLE Newemp
```

```
MODIFY (Sal NUMBER CONSTRAINT nn_Sal NOT NULL);
```

After this modification, we can change the value to 0, but we cannot assign NULL to the column;

```
SQL > update newemp
```

```
      2 Set sal = 0 where empno = 7369;
```

1 row updated.

```
SQL > update newemp Set Sal = null where empno = 7369;
```

```
update newemp set Sal = nul where empno = 7369
```

*

ERROR on row /:

ORA - 01407: it is not possible to update

("SCOTT"."NEWEMP"."SAL") to NULL

When you change a column, be sure to respect the constraints of the table. Next, we will change the width of the state column to 4:

```
ALTER TABLE dept 1
```

```
MODIFY (State Varchar(4));
```

To add a default value to a column;

```
ALTER TABLE NEWEMP2
```

```
MODIFY (SAL DEFAULT 0), (COMM DEFAULT 0);
```

Adding Columns

By using the ADD option, we can add new column to the table. In the following example, we add a state column to the DEPT 1 table:

```
SQL > alter table dept 1
```

```
2 add (state Varchar (2));
```

Table Changed.

```
SQL > describe dept 1;
```

Name	Null?	Type
DEPTNO	NOT	NULL NUMBER (2)
DNAME		VARCHAR 2 (10)
LOC		VARCHAR 2 (9)
STATE		VARCHAR 2 (2)

Eliminating a column

To eliminate a column, use the DROP option. In the next example, we will eliminate the comm Column from the NEWEMP2 TABLE:

SQL> ALTER TABLE NEWEMP2

2

SQL > Select * from newemp2;

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	12/17/80	800		20
7499	ALLEN	SALESMAN	7698	02/20/81	1600	300	30
7521	WARD	SALESMAN	7698	02/22/81	1250	500	30

/ 4 rows selected

SQL > alter table newemp2

2 drop (COMM);

Table changed.

SQL > select * from newemp2;

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	DEPTNO
7369	SMITH	CLERK	7902	12/17/80	800	20
7499	ALLEN	SALESMAN	7968	02/20/81	1600	30

It is not possible to delete an object type column, a key column in a partitioned key, or a column that is the PARENT KEY of another table, Also you cannot delete a column from nested tables.

Deleting a Table

To remove a table from a database, use the DROP TABLE Command;

Syntax:

DROP TABLE Schema.name_of_table

Example:

SQL> drop table demol;

Table dropped.

If the deleted table has a primary or unique key referenced by the FOREIGN KEYS of other tables, and you want to delete a constraint of the other table, use the CASCADE CONSTRAINTS Option:

DROP TABLE table CASCADE CONSTRAINTS;

7.3 Index-organized Tables

Index-organized tables consists of a table with rows grouped in the same sequence as the primary key i.e., physically the rows of the table are in the same order as the index entries. This type of table allows fast access to queries that involve the query of a specific piece of data or a predetermined range of results. Updating this type of table reduces storage space, since it eliminates the duplication of key columns in the indexes and tables. This type of table is created with the CREATE TABLE Command and the ORGANIZATION INDEX Option, which indicates that the table will be index-organized. In this case, the primary key must be specified. You can also specify a ROW OVERFLOW Option which preserves the cluster of indexes through the storage of column values, that exceed the space assigned in a separate extent.

The overflow tablespace is defined as a Percentage of the block size. When the size of a row is greater than the value specified by the PCTTHRESHOLD parameter, the values of columns that are not a key are stored in the overflow tablespace. The following example shows all the parameters for the creation of an index organized table. Initially, we have created a tablespace called Test 3a, using the CREATE TABLESPACE Command:

```
SQL > Connect System / Manager
Connected.
```

```
SQL> CREATE TABLESPACE test 3a
```

```
2   DATAFILE 'tablespace - file 5.dbf' size 2m
3   MINIMUM EXTENT 64k
4   DEFAULT STORAGE INITIAL 128K NEXT 128k)
5   LOGGING;
```

Tablespace created.

Then, we create the index_organized table:

```
SQL> CREATE TABLE test 3
```

```
2   title char (20),
3   doc_1d NUMBER,
4   Value NUMBER.
5   Comments  VARCHAR 2 (512),
6   CONSTRAINT PK_docindex PRIMARY KEY
   (title, doc_1d)
7   ORGANIZATION INDEX TABLESPACE test 3a
8   PCTTHRESHOLD    20
9   OVERFLOW TABLESPACE test 3a;
```

Table created.

In the example, the ORGANIZATION INDEX option indicates that the key and non-key columns are in an index defined by the columns that specify the primary key [title doc.id]

CREATE TABLE command Syntax

Here is the complete Syntax of the CREATE TABLE Command:

CREATE TABLE [schema] table

[C{Column datatype [DEFAULT expr] [WITH ROWID]

[SCOPE is [schema.] scope_table_name]

[Column_Constraint]---

1 table_Constraint 1 REF (ref_column_name) WITH ROW ID

/SCOPE FOR(ref_column_name) IS

[Schema.] Scope_table_name}

[.{column data type [DEFAULT expr] [WITH ROW ID]

[SCOPE IS [schema.] Scope_table_name]

[Column_Constraint]---

/ table_constraint / REF (ref_Column_name) WITH ROWID

/ SCOPE FOR (ref_Column_name) is

[schema.] Scope_table_name}]]---

[{[ORGANIZATION{HEAP / INDEX}

/ PCTTHRESHOLD [INCLUDING Column_name]

[OVERFLOW [Physical_attributes_clause

/ TABLESPACE tablespace]---

/ PHYSICAL_ATTRIBUTES_CLAUSE

/ tablespace TABLESPACE

/ LOB (10b_item [.10b_item---]) STORE AS

[LOB_Segname]

[C TABLESPACE tablespace

/ STORAGE Storage_Clause

/ CHUNK integer

/ PCTVERSION integer

/ CACHE

/ NOCACHE LOGGING/ NOCACHE NOLOGGING

/INDEX [10b_index_name]

```

[CTABLESPACE tablespace
/ STORAGE storage_clause
/ INITRAN integer
/ MAXTRANS' integer)---]]
/ NESTED TABLE nested_item STORE ASSTORAGE_table
/ LOGGING / NoLOGGING]]---
/ CLUSTER Cluster (Column [,Column]---)}}
[PARALLEL Parallel_Clause]
[PARTITION By RANGE (Column_list)
(PARTITION (Partition_name) VALUES LESS THAN
(value_list)
(physical_attributes_clause
/ TABLESPACE tablespace
/ {logging / nologging}}
[ENABLE Eenable_clause / DISABLE disable_clause]---
[AS subquery]
[CACHE / NOCACHE]
Physical_attributes_clause;:=
[PCTFREE integer
/ PCTUSED integer
INITRANS integer
MAXTRANS integer
STORAGE Storage_Clause]

```

Arguments

Schema

The schema that will contain the synonym. If the schema is omitted, oracle creates a synonym in the user's schema.

Table

The name of the table (or object table) to be created. A partitioned table cannot be a clustered table or an object table. or an object table.

OF object_type

Explicitly creates an object_type table. Each row will contain an instance of the object, and each instance will be assigned to only one object identifier (O,D) which is generated by the system when a row is inserted.

Column

Specifies the column name of table. A table can have upto 1,000 columns. You can only omit the column definitions when you use the A S clause attribute.

Specifies the name of the qualified column in are object item.

Datatype

This is the Column's data type. You can omit the data types only when the Command also designates a column as part of a foreign key in a referential integrity constraint. Oracle automatically attributes to the column the data type of the column corresponding to the referenced key in the referential integrity constraint.

The valid data types are REF Object_type, VARRAYs and nested tables.

DEFAULT

Specifies a value to be attributed to the Column when a subsequent INSERT Command omits a value to it. The data type of the expression must be equal to the data type of the column. The column also must be large enough to accomodate the expression.

Column ref_clause

Allows the detailed specification of a REF Column:

WITH ROWID

Stores the rowid and the REF value in the column or attribute. Storing a REF value with a rowid can improve the performance of the operations without referential integrity, but also uses more space. The default storage of REF values is without rowids.

SCOPE is Scope_table_name

Resricts the scope of the values of the REF Column to scope_table_name. The REF values for the Column must precede the REF values obtained in the object table specified in the clause. You can specify only one scope table for each REF column.

Scope_table_name

The name of the object table in which the object instances (the type as the REF column) are stored. The values of the REF Column Point to the objects in the scope table.

You must have the SELECT privilege or the SELECT ANY TABLE System privilege on the table.

Column_constraint

Defines as integrity cconstraint as part of the column definition.

table_constraint

Defines an integrity constraint as part of the table definition.

table_ref_clause.

Defines an integrity constraint as part of the column definition.

SCOPE FOR..... IS

Restricts the scope of values of the REF column in ref_column or ref_attribute to scope_table_name. The REF values for the column or attribute must come from the REF values obtained from the object table specified in the clause.

ref_column or ref_attribute

The name of the REF column of an object table, or a built-in attribute in an object column of a relational table. The values of the REF Column point to the objects of the scope table.

REF

The reference to a row in an object table. You can specify the name of a REF column of an object or relational table (ref_column), or a built-in REF attribute in an object column ref_attribute.

OIDINDEX

Specifies an index in the column of the hidden object identifier and / or the specification of the storage for the index. Both the storage_specification and the index must be specified.

Index

The name of the index in the hidden object identifier when not specified, Oracle generates a name.

Physical_attributes_clause

Specifies the values of the PCTFREE, PCTUSED, INITRANS, and MAXTRANS parameters and the storage characteristics of the table.

PCTFREE

Specifies the percentage of space in each data block of the table, object table OIDINDEX, or Partition reserved for future updates in the rows of the table. The PCTFREE value must be a value between 0 and 99. A block is filled with inserts of new rows when the value is 0. The default value is 10, and reserves 10% of each block for the updates in the existing rows. Allowing inserts of new rows to fill a maximum of 90% of each block.

PCTFREE has the same function in the PARTITION Clause, and in the Commands that create and change clusters, indexes, snapshots, and snapshot logs. The combination of PCTFREE and PCTUSED determines whether new rows will be inserted in the existing data blocks or in new data blocks.

PCTUSED

Specifies the minimum percentage of used space that oracle maintains of each data block in the table, object table OIDINEX or overflow data segment of the table organized by indexes. A block can have rows inserted when its physical space is below PCTUSED is specified as a positive integer between 1 and 99' its default value is 40.

PCTUSED has the same function in the PARTITION Clause, and in the commands that create and change clusters indexes, snapshots, and snapshot logs. PCTUSED is not a valid table characterisitic for index-organised tables (those with the ORGANIZATION INDEX option).

The sum of PCTFREE and PCTUSED must be less than 100 you can use PCTFREE and PCTUSED together to better use the space of a table

INIRANS

Specifies the initial number of transaction entries allocated to each block in the table, object, table OIDINEX, partition, index segment LOB, or data overflow segment. This number ranges from 1 to 255, and its default value is 1 Generally, the default value of INITRANS must not be changed. Each transaction in the block. The size of an entry depends on the operating system you are using.

This parameter guarantees that the minimum number of concurrent transaction updates the block and helps minimize the cost of allocating a transaction entry.

The INITRANS parameter has the same function in the PARTITION Clause, and in the Commands that create and change clusters indexes, snapshots, and shanpshot logs. The minimum value and default-value for a cluster or index is 2 instead of 1

MAXTRANS

Specifies the maximum number of concurrent transaction, which can be updated by a data block allocated to the table, object table OIDINDEX, partition index segment. LOB, or over flow data segments organized by indexes. This limit does not apply to queries. It can vary from 1 - 255 and its default value is a function of the data block size. The default value of MAXTRANS should not be changed.

If the number of concurrent transactions that update a block exceeds the INITRANS value, oracle, dynamically, allocates transaction entries in the block until the value of MAXTRANS is exceeded or the block runs out of free space.

MAXTRANS has the same function as PARTITION Clause, and in the commands that create and change clusters, indexes snapshots, and snapshot logs.

Storage_Clause

Specifies storage characteristics of the table, object table OIDINDEX, Partition, storage LOB index segment LOB, or overflow data segment in the table organised by indexes. This clause has performance ramifications in large tables. The storage should be allocated to minimize the allocation of the additional space.

TABLESPACE

Specifies the tablespace in which oracle creates the table, object table OIDINDEX, partition storage LOB index segment LoB, Or over flow data segment of the index Organised table. If this option is omitted, oracle creates an items in the default table space of the owner of the schema in which the table is contained. For non-partitioned tables, the value specified for TABLESPACE is the real physical attribute of the segments associated to the table. For partitioned tables the value specified for TABLESPACE is the default Physical attribute of the segments associated to all the partitions specified in the CREATE Command (and in the subsequent ALTER TABLE and ADD PARTITION Commands) unless you specify TABLESPACE in the PARTITION Clause.

LOGGING / NO LOGGING

Specifies the creation of the table (and any index needed due to the constraints) Partition, or LOB storage characteristics that will be recorded in the log repetition file. Also specifies that the subsequent operation of DIRECT LOADER (SQL*) and direct load INSERT to the table, Partition, or LOB. Storage are registered (LOGGING) or non-registered (NOLOGGING). If the LOGGING / NOLOGGING Clause is omitted the record attribute or table partition assumes the default value of the record attribute of the tablespace in which it resides.

If you omit the LOGGING / NO LOGGING Clause for LOBs, LOGGING will be used when CACHE is specified (Since you cannot have CACHE NOLOGGING) otherwise, the second attribute assumes the default value of the record attribute of the tablespace in which it resides.

For non-partitioned tables, the value specified for LOGGING is the real physical attribute of the segment - associated to the table. For partitioned tables, the value specified is the default physical attribute of the segments associated to all the partiones specified in the CREATE Command (and in subsequent ALTER TABLE and ADD PARTITION commands). Unless you specify LOGGING / NON LOGGING in the PARTITION Clause.

In LOGGING mode, data is modified with the minimum log (to mark new invlaid extensions and to write changes in the dictionary). When applied to media recovery, the invalid extension logs mark a range of blocks as logically corrupted because repetitive data is not totally recorded. Therefore, if you cannot lose thier table, make a backup after the operation.

The size of a repetition log generated by an operation in Nologging Mode is significantly smaller than the log generated with the LOGGING option.

If the database is executed in the ARCHNELOG Mode, the recovery of media from the backup made before the LOGGING operation restores the table. However, the recovery of media from the backup made after the NOLOGGING operation does not restore the table.

The table log attributes is independent of its item attributes. NO LOGGING is not a valid argument for the creation of index_organised tables.

ORGANIZATION INDEX

Specifies that the table is created as an index - Organized table. In an index - organized table, the rows of data are maintained in an index defined in the primary key of the table.

ORGANIZATION HEAP

Specifies that the rows of data in the table are stored in no special order. This is the default value.

Index_organized_table_clause

PCTTHRESHOLD integer

Specifies the percentage of space reserved in the index block for a row in the index-organized table. Any part of the row exceeding the specified limit is stored in the area. PCTTHRESHOLD should be a value between 0 and 50.

OVERFLOW

Specifies that the rows of the index-organised table that, exceed the specified limit are stored in the data segment defined in this clause. If OVERFLOW is not specified, the rows that exceed the limit of PCTTHRESHOLD are rejected.

INCLUDING Column_name

Specifies a Columns where a row of the index-organized table will be divided into the index and overflow parts. All the column after column_name should be stored in the overflow data segment. Column-name can be the name of the last column of the Primary key, or any other column that is not the PRIMARY KEY.

RECOVERABLE

This option no longer exist RECOVERABLE is not a valid argument for the creation of partitioned tables or tables with the LOB storage characteristics.

UNRECOVERABLE

This option no longer exists. It specifies that the creation of the table (and any index required due to constraints) will not be registered in the repetition log. This argument can only be specified with the As subquery clause. UNRECOVERABLE is not a valid argument for the creation of partitioned tables or index - organized tables.

LOB_Storage_Clause

LOB

Specifies the LOB Storage Characterisitcs

lob_item

The name of the LOB Column or LOB object attribute to which you explicitly define the tablespace and the storage characteristics different from those of the table.

STORE As Lob_Segname

Specifies the name of LOB data Segments. You cannot use Lob_Segment if you specify more than one lob_item.

/lob_Parameters

ENABLE STORAGE IN ROW

Specifies that the LOB value will be stored in line if its size is less than approximately 4,000 bytes, less the system control information.

DISABLE STORAGE IN ROW

Specifies that the LOB value will be stored offline regardless of its size. Note that the LOB locator is always stored in the row, regardless of where the LOB value is stored.

CHUNK integer

Specifies the number of bytes to be allocated for the manipulation of LOBs. If integer is not a multiple of the database's block size, Oracle will round up (in bytes) to the next multiple. For example, if the database's block size is 2,048 and the integer is 2050, Oracle will allocate 4,096 bytes, (two blocks) The maximum value is 32,768 (32k)

PCTVERSION integer

The maximum percentage of global space for the LOB, used created new versions of LOB. The default value is 10, which means that older versions of LOB data won't be overwritten until 10% of the global space of the LOB is used.

INDEXlob_index_name

The name of the LOB index segment. You cannot specify lob_index_name if you determine more than one lob_item in the associated List of lob _ item Note that you cannot change the LOB index using the ALTER INDEX Command, this can only be done with the ALTER TABLE Command.

NESTED TABLE.....STORE AS.....

Specifies storage - table as the name of the storage table, in which the rows of all the nested - item values reside you should include this clause when creating a table with nested columns. Nested - Item is the name of the column.

Storage - table

The name of the storage table created in the same schema and same tablespace as its parent table.

CLUSTER

Specifies that the table will be part of the cluster. The columns listed in this clause are the table columns corresponding to the cluster columns. Generally the cluster columns of a table are the columns or column that form its Primary key or part of it specifies a column of the table for each column in the cluster key. The columns are compared by position, not by name.

A clustered table uses the allocation of space of the cluster. However, do not use the PCTFREE, PCTUSED, INTRANS, OR MAXTRANS Parameter, the TABLESPACE option, or the STORAGE Clause with the CLUSTER Option.

Object tables cannot be part of a cluster. `table_partitioning - clause`

PARTITION BY RANGE

Specifies that the table will be partitioned at the value intervals of `column_list`

COLUMN_LIST

An ordered list of columns used to determine to which partition or row belongs. You cannot specify more than 16 columns in `column_list`. A `column_list` cannot contain a pseudo-column `rowid`, or any another column with the `rowid` or long data types.

PARTITION Partition_name

Specifies the physical partition clause if `Partition_name` is omitted, oracle generate a name in the form of `sys_on` for this partition.

VALUES LESS THAN

Specifies the Upper limit non-inclusive of the current partition.

Value_list

An ordered list of values corresponding to the `column_list` in the PARTITION BY RANGE Clause. You can replace the MAXVALUE argument with any literal in `value_list`. MAXVALUE specifies a maximum value that will always sort higher than any other value, including NULL.

The specification of a value for MAXVALUE other than the upper limit of the partition imposes an implicit integrity constraint on the table.

PARALLEL_CLAUSE

Specifies the rates of Parallelism in for the creation of the table, and the default rate of parallelism. for queries in the table already created. This is not a valid option for creating index organized tables.

enable_clause

Enables an integrity Constraint.

disable_clause

Disables an integrity constraint. The Constraint specified in the ENABLE and DISABLE Clauses of a CREATE TABLE Command should be defined in the Command. You can also enable or disable the constraints with the ENABLE and DISABLE arguments of the CONSTRAINT Clause if you define a constraint but do not enable or disable it explicitly. Oracle enables it by default.

You cannot use the ENABLE and DISABLE Clause in a CREATE TABLE Command to enable or disable the trigger.

As subquery

Inserts the rows returned by the subquery of the table after its creation. The number of columns in the table must be equal to the number of expressions in the subquery. The column definitions can only specify names, default values, and integrity constraints, not data types.

Oracle derives the data types and sizes from the subquery and also follows these rules for integrity Constraints.

Oracle automatically defines any NOT NULL Constraint in the columns of the new table, which existed in those corresponding columns of the selected table, if the subquery specifies the column instead of an expression containing it.

A CREATE TABLE Command cannot contain an AS Clause or the definition of a referential integrity constraint.

If a CREATE TABLE Command contains an AS Clause and a CONSTRAINT Clause, or an ENABLE Clause with the EXCEPTIONS Option, Oracle ignores the EXCEPTIONS option. If any row violates the constraint. Oracle will not create the table and returns an error message.

If all the expressions of the subquery are columns instead of expressions, you can omit the table definite columns. In this case, the names of the columns in the table will be the same as those in the subquery column.

For object tables, the subquery can contain an expression corresponding to the type of table or the number of high level attributes of the table type.

CACHE

Specifies that the most frequently accessed data blocks are put in the most recently used part of the RLU List of the buffer Cache, when a complete Scan of the table is done. This option is useful for small search tables.

As a Parameter of the LOB storage Clause, CACHE Specifies that Oracle preallocates and retains the values of LOB data in the memory for fast access.

This is not a valid argument for the creation of index organized tables.

NOCACHE

For data that is not frequently accessed, specifies that the accessed blocks for this table are put in the less used part of the RLU list of the buffer Cache, when a complete scan of the table is done. For LObs the value of LOB is put in the less used part of the RLU List and is not placed on the buffer cache. This is the default behaviour. However it is not a valid argument for the creation of index-organized tables.

As a parameter of the LOB storage clause NO CACHE specifies that the LOB values are not preallocated in the memory. This is the storage default LOB.

7.4: SQL*PLUS

This chapter shows you SQL*plus a tool that allows the handing of a data base through individual and interactive execution of the SQL Commands. In addition to the direct execution of SQL Command, SQL*plus allows the configuration of the PI / SQL Commands. PI/SQL is an oracle extension to the standard SQL while this may be executed both in the character and graphic modes, SQL*plus shows the result only in the character mode. SQL*plus is simple to operate and represents the fastest way to query and create quick reports By using the PL/SQL language extensions, you can create stored procedures.

To Configure SQL* plus, click on the start button then on programs / Oracle Home / Application Development SQL*plus

INTYPE File Assistant	☒ Application Development ▶
Oracle objects For OLE class Library	☒ Database Administration ▶
Oracle objects for OLE Help	☒ Migration Utilities ▶
☒ SQL Plus	☒ Network Administration ▶
☒ Web Publishing Assistant	☒ Documentation
Net Meeting	☒ Release Notes

If the user is not connected to the database the user name, Pass word, and host string are required, For the local data base, it is not necessary to specify the string.

Logon

User Name

Scott

Password

...

Host String

OK

Cancel

The database is then configured. If it is already configured, the SQL*Plus screen will be displayed immediately.

```
Oracle SQL*Plus
File Edit search options Help
SQL*Plus:Release 8.1-5-0-0-Production on Sun Feb 27 15:49:44 2000
C Copyright 1999 Oracle Corporation. All rights reserved
Connected to :
Oracle 8i Enterprises Edition Release 8-1-5-0-0 Production with the partitioning and java
options
PL/SQL Release 8-1-5-0-0 Production
SQL>1
```

The line with the SQL> prompt is where you enter the SQL Commands. The SQL* plus interface allows a SQL Command to occupy several lines.

It will execute the command only when the semicolon(:) is typed and the Enter key is pressed. To execute the last Command typed in the buffer the user types the Slash (/) and presses the Enter Key. Consider the two next examples in which the contents of the DEPT and EMP tables are displayed using only one command line for now, don't worry about the SQL Command that is shown, we will examine it later, The SELECT Command retrieves and displays the contents of a table. The asterisk (*) indicates that all the columns of the DEPT table should be displayed.

```
Oracle SQL *Plus
File Edit Search Options Help
SQL* plus Release 8.1.500 production on Sun Fb 27 15.49.44 2000
(C) Copy Right 1999 Oracle Corporation All rights reserved.
Connected to :
Oracle 8i Enterprise Edition Release 8.1.5.00 - produdction
With the partiontioning and java options
PL/SQL Release 8.1-5-0-0. production
SQL > Select - From dept;
DEPTNO      DNAME      LOC
-----
10          Account ING NEWYORK
20          RESEARCH   DALLAS
30          SALES      CHICAGO
40          OPERATIONS BOSTON
SQL>
```


Oracle SQL*Plus						
File Edit Options Help						
SQL > Select * From emp.						
EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM
7369	SMITH	CLERK	7902	17-DEC 80	800	
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300
7521	WARD	SALESMAN	7698	22-FEB 81	1250	500
7566	JONES	MANAGER	7839	02-APR-81	2975	
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400
7698	BLAKE	MANAGER	7839	01-MAY-81	2850	
7782	CLARK	MANAGER	7839	09-JUN-81	2450	
7788	SCOTT	ANALYST	7566	19-APR-87	3000	
7839	KING	PRESIDENT		17-NOV-81	5000	
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	
7876	ADAMS	CLERK	7788	23-MAY-87	1100	0
7900	JAMES	CLERK	7698	03-DEC-81	950	
7902	FORD	ANALYST	7566	03-DEC-81	3000	
7934	MILLER	CLERK	7782	23-JAN-82	1300	
14 rows selected						
SQL> 1						

In the next example, three lines are used to execute the SELECT Command, which shows the employees with salaries above \$ 2500 when you reach the end of a line, press the Enter key to go to the next line.

Oracle SQL * Plus	
File Edit Search Options HELP	
SQL> Select*from emp	
2 Where Sal > 2500	
3 :	

EMPND	ENAME	JOB	MGR	HIREDATE	SAL	COMM
7566	JOHNES	MANAGER	7839	02-APR-81	2975	20
7698	BLACKKE	MANAGER	783901	MAY 81	2850	30
7788	SCOTT	ANALYST	7566	19 APR-87	3000	20
7839	KING	PRESIDENT		17-NOV-81	5000	10
7902	FORD	ANALYST	7566	03-DEC-81	3000	20

MENUS

The following menus are useful in SQL * Plus

File Menu

The file menu contains the following options: open opens a file of SQL Commands that was previously written with the extension - Sql.

Save allows the writing to the buffer of a text file that receives the extension sql as default. The Replace sub option changes the contents of an existing file with the contents of the buffer. The Append sub option adds the contents of the buffer to the specified file.

Save As writes the contents of the buffer or the file that was configured with another name.

Spool stores the results of query in a file. As the default, the file created has the extension .lst. to disable spool for queries, you must activate the spool off option.

Run lists and executes the SQL Command or a PL / SQL block that is stored in the buffer.

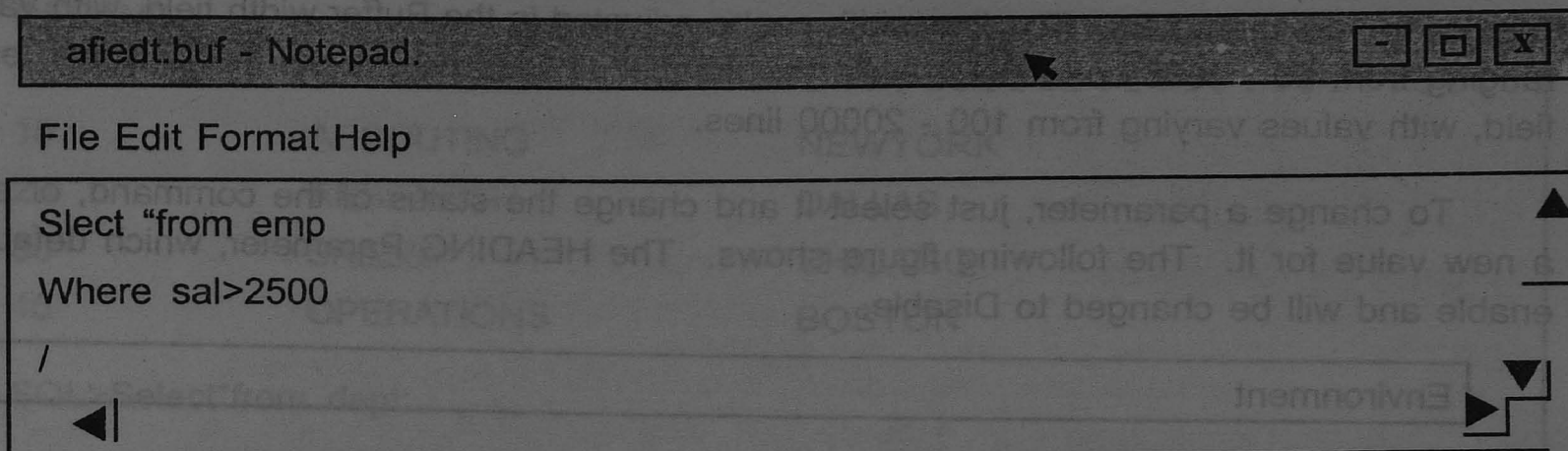
Cancel interrupts the operation that is being executed. Exit makes a commit of all the changes made to the data base.

Edit Menu

The Edit Menu includes the following options. Copy sends the selected text to the windows clipboard.

Paste, pastes the contents of the clipboard to the command line of SQL * plus, with a maximum size of 3,625 characters.

Clear, clears the contents of the screen and buffer. Editor opens an editor in which files of PL/SQL Commands and their settings can be edited. As the default, windows Notepad is selected. The Name of the file defaults to afiedt.buf. whenever the editor is called by the Invoke Editor Option, the contents of the buffer are automatically transferred to it.



FIND MENU

The find menu contains the Find and Find next options. Find opens a dialog box in which the user can type text to search for. After clicking on the Ok button in this dialogbox, the occurrence of the text is highlighted. Find Next finds the next occurrence. The search is always initiated from the current Screen; however, you can use the cursor keys if you want to start from the beginning of the text.

Options Menu

The options menu allows you to change SQL * Plus elements. The Environment option opens a dialog box that has two parts. The Define options list allows you to change the environment elements, such as variables and characteristics of SQL * plus the printing of column headers, and the formatting of numeric fields.

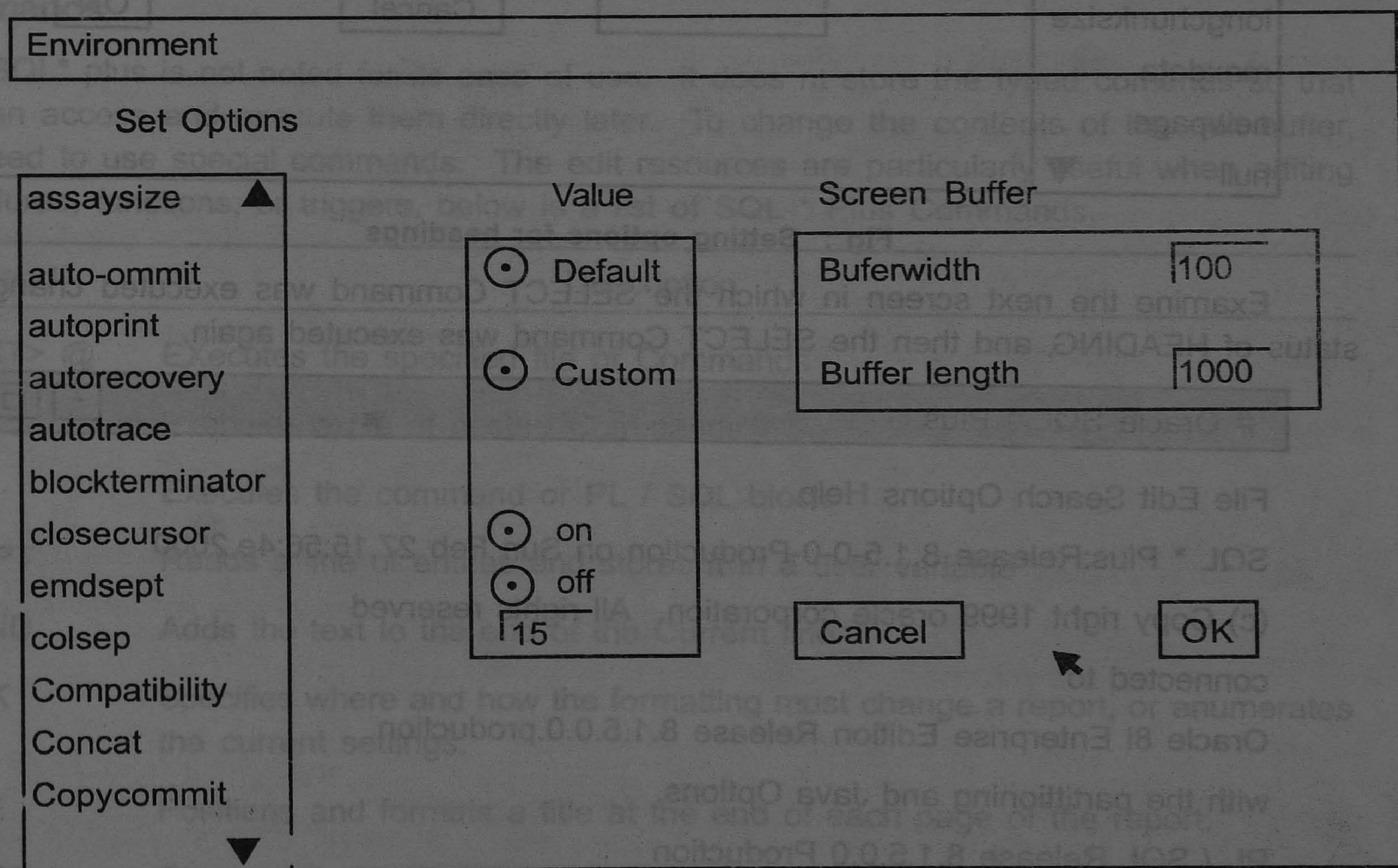


Fig : The Environment dialog

The screen Buffer area controls the number of characters and lines that are maintained and displayed by SQL * plus. As the default, the buffer is adjusted to display upto 1,0000

lines, with 100 characters. The line width canbe adjusted in the Buffer width field, with values ranging from 80 - 1000 characters, while the number of lines can be set in the Buffer length field, with values varying from 100 - 20000 lines.

To change a parameter, just select it and change the status of the command, or enter a new value for it. The following figure shows. The HEADING Parameter, which default to enable and will be changed to Disable.

Environment

Set Options

flagger ▲

flush

heading

headsep

Instance

linesize

logsource

long

longchunksize

maxdata

newpage

null ▼

Value

☐ Default

☐ Custom

☐ on

☒ off

Screen Buffer

Bufewidth

100

Buffer length

1000

Cancel

OK

Fig : Setting options for headings

Examine the next screen in which the SELECT Command was executed change the status of HEADING, and then the SELECT Command was executed again.

Oracle SQL * Plus

File Edit Search Options Help

SQL * Plus:Release 8.1.5-0-0-Production on Sun Feb 27.15:56:4e.2000

(c) Copy right 1999 oracie corporation. All rights reserved

connected to

Oracle 8i Enterprise Edition Release 8.1.5.0.0.production

with the partittioning and Java Options.

PL / SQL Release 8.1.5.0.0 Production

SQL > Select * from dept:

DEPTND	DNAME	LOC
10	ACCOUNTING	NEWYORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

SQL>Select*from dept:

10	ACCOUNTING	NEWYORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

SQL

Fig. The effect of disabling the column headings

Most of these Parameters can be changed through the SET Comman directly from the edit line.

Commands

SQL* plus is not noted for its ease of use. It does nt store the typed comands so that you can access and execute them directly later. To change the contents of the edit buffer, you need to use special commands. The edit resources are particularly useful when editing procedures, functions, or triggers, below is a list of SQL * Plus Commands.

Command	Description
<TTEXT> @	EXecutes the specified file of Commands
@@	Executes a file of nested Commands
/	Executes the command or PL / SQL block
ACCEPT	Reads a line of entries and stores it in a user variable
APPEND	Adds the text to the end of the Current line
BREAK	Specifies where and how the formatting must change a report, or enumerates the current settings.
BTITLE	Positions and formats a title at the end of each page of the report.
CHANGE	Changes the text of the current line
CLEAR	Reinitiates the current value of the adjustments made to the specified action

Command	Description
COLUMN	Specifies the attributes to a certain column and displays the current status of all columns.
COMPUTE	Computes and displays the summary lines and displays the compute settings.
CONNECT	Connects the user to the database
COPY	Copies data from a query to a local or remote table.
DEFINE	Specifies a user variable, and assigns to it a value of type char.
DEL	Deletes one or more lines of the buffer
DESCRIBE	Enumerates the definitions of the columns for the table, synonym or current / specified view
DISCONNECT	Executes a Commit of the pending changes, and disconnect the user from the data base without leaving SQL * plus.
EDIT	Invokes a word editor of the operating system, transferring to it the contents of the buffer.
EXECUTE	Executes a PL / SQL command.
EXIT	Terminates SQL * plus and returns to the operating system.
GET	Loads a text file to the SQL buffer
HELP	Enables the help system.
HOST	Executes a Command of the operating system without leaving SQL * plus.
INPUT	Adds one or more lines after the current line of the buffer
LIST	Lists one or more lines of the SQL buffer
PAUSE	Displays a blank line following another line containing text, and waits for the user to press Return, or displays two empty lines and waits for the user to answer.
PRINT	Displays the contents of a bind variable
PROMPT	Sends the specified message to the screen.
REMARK	Starts a Comment line

Command	Description
REPFOOTER	Positions and formats a footer at the end of each current report, or enumerates the current definition
REPHEADER	Positions and formats a header at the end of each current report or enumerates the current definition
RUN	Enumerates and executes the SQL command that is stored in the buffer
RUNFORM	Executes an application of the SQL * forms from SQL*plus.
SAVE	saves the contents of the SQL buffer to a text file.
SET	Attributes the contests to a system variable
SHOW	Displays the value of a system or environment variable
SPOOL	Stores the results of a query in a text file, and optionally sends them to a printer.
SQLPLUS	Starts SQL * plus rom the operating system.
START	Executes the contents of the file of the specified commands
STORE	Saves the environment attributes of the current session of SQL * plus in a text file.
TIMING	Records the time elapsed or displays the active timers.
TITLE	Positions and formats the title at the top of each page of the report.
UNDEFINE	Deletes one or more variables that were previously defined
VARIABLE	Declares a bind variable
WHENEVER	Stops the execution of SQL * plus if an operating
OSERROR	System Command generates an error
WHENEVER	Stops the execution of SQL * plus if a SQL or PI / SQL
SQLERROR	Command generates an error.

For help with Command suntax, type HELP name_of_Command:

HELP prompt

Editing the Command Line

Once a command is typed and executed, it is possible to perform several editing operations. The next table shows the specific syntax of the commands used in line editing:

Command	Abbreviation	Purpose
APPEND text	A text	Adds the text to the end of the line.
CHANGE / old/new	C/old/new	Changes the old text into the new next
CHANGE / text	C/text	Deletes the text of the line.
CLEAR BUFFER	CLBUFF	Deletes all the lines
DEL	(none)	Deletes the Current line
DELn	(none)	Deletes line n
DEL *	(none)	Deletes the current line
DELn*	(none)	Deletes from line n upto the current line
DEL LAST	(none)	Deletes the last line
DEL mn	(none)	Deletes all the lines from m to n
DEL * n	(none)	Deletes from the current line up to line n
INPUT	1	Adds lines
INPUT text	1 text	Adds the line with the specified text.
LIST	L	Enumerates all the lines.
LIST n	L nor n	Enumerates line n
LIST *	L*	Enumerates the current line.
LIST N*	Ln*	Enumerates from line upto the current line
LIST LAST	LLAST	Enumerates the last line.
LIST mn	Lmn	Enumerates from line m upto line n
LIST * n	L*n	Enumerates from the current line upto line n.

Correcting Errors

Typing errors are the main problem when executing a command. In the next example, the command has the field name "Deptno" typed as "Dep" when executed, SQL* plus shows an error. Examine the following:


```
SQL > Select * from emp
      2   where dep = 20
      3   and Sal > 2000;
where dep = 20
      *
```

ERROR on line 2:

ORA - 00904: invalid name in the column.

Changing a Line

To solve this problem, we can use the Command L2 to tell SQL * plus to display line 2. Then we use the C/dep dept no command to change the word "Dep" to "Dept no" and use RUN to execute the command line:

```
SQL > L2
```

```
2* where dep = 20
```

```
SQL> C/dep/dept no
```

```
2 * where dept no = 20
```

```
SQL > run
```

```
1   select * from emp
```

```
2   where dept no = 20
```

```
3*  and Sal > 2000
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7566	JONES	MANAGER	7839	04/02/81	2975		20
7788	SCOTT	ANALYST	7566	04/19/87	3000		20
7902	FORD	ANALYST	7566	12/03/81	3000		20

Adding a New line

To insert a new line after the Current line, you must use the INPUT Command. Consider the next example. After executing the first Command, we decide to add one more line to it:

```
SQL > run
```

```
1   Select * from emp
```

```
2   where deptno = 20
```

```
3*  and Sal > 2000
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7566	JONES	MANAGER	7839	04/12/81	2975		20
7788	SCOTT	ANALYST	7566	04/19/87	3000		20
7902	FORD	ANALYST	7566	12/03/81	3000		20

Just Press I to open a new line. You must type the rest of the Command and / or press the Enter key to open another line. Then, you see the inclusion in the line order by hiredate, which will organize the table by hire date:

SQL > i

4 order by hiredate;

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7566	JONES	MANAGER	7839	04/12/81	2975		20
7902	FORD	ANALYST	7566	12/03/81	3000		20
7788	SCOTT	ANALYST	7566	04/19/87	3000		20

Adding Text to the End of a Line

To add text to the end of a specific line of the buffer, use the APPEND command. First use the LIST Command, followed by the line you want to edit. Then enter the APPEND Command, followed by the text. If the text starts with a blank character, leave two spaces between the word "append" and the beginning of the word. Otherwise, it will be attached to the last character of the current line. Following is the procedure to include another restriction to the Second line of the buffer.

After executing the APPEND command, Press L to see the Command as a whole and Run to execute it. When you specify strings of text, include them inside single quotation marks.

SQL > 2

2* where dept no + 20

SQL > append and job = 'ANALYST'

2* where deptno = 20 and Job = 'ANALYST'

SQL > 1

1 Select * from emp.

2 where dept no = 20 and job = 'ANALYST'

3 and Sal > 2000

4* order by hiredate

SQL > run

```
1   Select * from emp.
2   where dept no = 20 and job = 'ANALYST'
3   and Sal > 2000
4*  order by hiredate
```

EMPNO	ENAME	JOB	MGR	HIREDATE
7902	FORD	ANALYST	7566	12/03/81
7788	SCOTT	ANALYST	7566	04/19/87

To insert a line before line 1, you must specify o (zero) folowed by the text of the new line.

Deleting Lines

To delete a line from the buffer, use the DEL Command. If you specify this command without any parameters, it will delete the current line. If it is specified with a number, it will delete the specified line.

SQL > del 3

SQL > 1

```
1   Select      *      from emp
2   where dept no =20 and job = 'ANALYST'
3*  order by hiredate
```

The DESCRIBE Command

This command is very useful when the developer wants to access the characteristics of a table's fields directly in the data dictionary. Let's say you forget the correct name of a column in a table. By using the DESCRIBE Command, you can have a quick view of the structure of the specified table.

Syntax

DESCRIBE name_of_table

Here are some examples:

SQL > describe dept

Name	Null?	Type
DEPTNO	NOT NULL	NUMBER(2)
DNAME		VARCHAR 2 (14)
LOC		VARCHAR 2 (13)

SQL> describe emp;

Name	Null?	Type
EMPNO	NOT NULL	NUMBER (4)
ENAME		VARCHAR 2 (10)
JOB		VARCHAR 2 (9)
MGR		NUMBER (4)
HIREDATE		DATE
SAL		NUMBER (7,2)
COMM		NUMBER (7,2)
DEPTNO		NUMBER (2)

The Column Command

While quite limited in its interaction and display capabilities SQL * plus offers some improvement when making changes to columns with the COLUMN Command. These are the Parameters of the Command:

ALI [AS] alias
 CLE[AR]
 FOLD-A[FTER]
 FOLD-BE[FORE]
 FOR[MAT] format
 HEA[DING] text
 JUS[TIFY] {L[EFT] / [CENTER] / C[ENTER] / R[IGHT]}
 LIKE {expr/alias}
 NEWL[INE]
 NEW_V[ALUE] Variable
 No PRI[NT] / PRI[NT]
 NUL[L] text
 OLD-V[ALUE] Variable
 ON / OFF
 WRA[PPED] WOR[D - WRAPPED] / TRU[NCATED]

We will examine some of these parameters and their uses in the following sections

Changing Headers

Column titles that are displayed by SQL * Plus reflect the name given to the column in the table. Often, it is not easy to understand the name, since there is no standard format. You can define a title for a column that is more easily understood by using the HEADING Parameter, as the following shows:

Syntax

COLOUMN name-of-column HEADING title

To change some of the column titles in the EMP table, insert the following Commands:

COLOUMN DEPTNO HEADING Department

COLUMN ENAME HEADING LastName

COLUMN SAL HEADING salary

SQL> COLUMN COMM HEADING Commission

SQL> COLOUMN HIREDATE HEADING Hired

SQL> Select * from emp whee dept no = 20

2;

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	12/17/80	800		20
7566	JONES	MANAGER	7839	04/02/81	2975		20
7788	SCOTT	ANALYST	7566	04/19/87	3000		20
7876	ADAMS	CLERK	7788	05/23/87	1100		20
7902	FORD	ANALYST	7566	12/03/81	3000		20

If the title has more than one word, enclose them in Single or double quotation marks.
If the title is too long for one line, use the Vertical bar to start a new line.

SQL> COLUMN empno HEADING "EMmployee / ID"

SQL > Select empno, ename, Sal from emp where dept no = 30;

Employee ID	Last Name	Salary
7499	ALLEN	1600
7521	WARD	1250
7654	MARTIN	1250
7698	BLAKE	2850
7844	TURNER	1500
7900	JAMES	950

To change the style of the line below the title, use the SETUNDERLINE Command followed by the desired Character:

SQL> set underline =

SQL > Select * from dept 2;

Department	DName	LOC
10	ACCOUNTING	NEWYORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

Formatting Numeric Columns

Using the FORMAT Parameter, you can change the appearance and precision of the display of a numeric column. As the default, it is displayed with a size of 10 characters. Values greater than this are rounded.

Syntax

COLUMN name - of - column FORMAT style

The format can contain numbers, commas, dollar signs, brackets, and leading zeros.

Element	Example	Description
9	9999	Displays a number in its position
0	0999	Displays a zero in its position
\$	\$9999	Inserts a dollar sign in front of the number
B	B9999	Displays zero values as blank
M I	9999MI	Displays "-" after a negative value.
S	S9999	Displays "+" for positive values and "-" for negative values.
PR	9999PR	Displays the negative value inside "<>"
D	99D99	Displays the decimal part separated from the integer part.
G	9G999	Displays the group separator in the specified position.
L	L999	Displays the current symbol in the position
,	9,999	Displays a comma in the position
.	99.99	Displays a period in the position
V	999V99	Multiplies the value by 10 ⁿ , where n is the value following V.
EEEE	9.999EEEE	Displays the number in scientific notation.
RN or rn	RN	Displays Roman numerals. The value can be between 1 and 3999.
DATE	DATE	Displays the value in the date format MM/DD/YY for Julian values.

To display the sal field with a dollar sign, you can use the following command:

```
SQL > COLOUMN SAL FORMAT $99,990
```

```
Select ename, Sal, from emp where dept no = 20
```

ENAME	SAL
SMITH	\$800
JONES	\$2,975
SCOTT	\$3,000
ADAMS	\$1,100
FORD	\$3,000

To change the width of a column, use the element A followed by the number of characters that the column must contain:

```
SQL > COLUMN ENAME FORMAT A4
```

```
SQL > Select ename, Sal, from emp where dept no = 20;
```

ENAME	SAL
SMITH	800
JONES	2,975
SCOTT	3,000
ADAMS	1,100
FORD	3,000

Copying Attributes from Columns

You can apply the formatting attributes from one column to another column. This saves you time when creating the formats. In this case, you must use the LIKE Clause of the COLUMN Command consider the following example:

```
SQL > column Comm like Sal
```

```
SQL > Select ename, Sal, Comm from emp where dept no = 30
```

ENAME	SAL	COMM
ALLEN	1,600	300
WARD	1,250	500
MARTIN	1,250	1,400
BLAKE	2,850	
TURNER	1,500	0
JAMES	950	

Displaying the Attributes of a column

To display the current attributes of a column, use the COLUMN Command followed by the name of the column:

COLUMN Column-name

Here is an example:

SQL > Column Sal;

COLUMN Sal on

FORMAT 99,9999

SQL>

To list the attributes of all the columns, enter the COLUMN Command without specifying a name for it:

COLUMN

To delete the definition of a Column's attributes, use the following syntax.

COLUMN Column - name CLEAR

To delete the definition of all the columns, use the COLUMNS clause of the CLEAR Command.

SQL > CLEAR COLUMNS

Columns cleared

Temporarily Disabling the Attributes of a column

If you want to temporarily disable the attributes of a column use the on / off option of the COLUMN Command. To disable the attributes, use:

COLUMN Column - name OFF

To enable the attributes, use:

COLUMN Column - name ON

The SAVE Command

SQL * plus allows you to store one or more commands in a file called a command file. This operation consists of writing the contents of the buffer in a text file that can be executed later. To SAVE the current command or PL / SQL block use the SAVE Command by a file name:

SAVE file _ name

SQL * plus adds the extension. Sql to the name provided. In the next example, we list the contents of the buffer. The command specifies the employees who do not work in department 10, by descending order of salary. Then we write a file called depd if 10.

SQL > 1

- 1 Select ename, sal, Comm from emp
- 2 Where dept no <> 10
- 3* order by sal desc

SQL > Save depd if 10

File depf if 10 created

The INPUT command allows you to enter Several command lines in the buffer without executing them. In the next example we create a query that shows the commissions of salesmen and format its columns. If this is a frequent task, we can save it in the machine for subsequent use. First, the contents of the buffer must be cleared.

SQL > CLEAR BUFFER

Then, the INPUT Command is typed without the semicolon at the end of the line and without the commands that will be written:

SQL > input

- 1 Column ename heading salesman
- 2 Column Sal heading Salary format \$99,9999
- 3 Column Comm like sal heading commission.
- 4 Select empno, ename, sal, comm
- 5 from emp "
- 6 where job = "SALESMAN"
- 7

After Pressing the Enter Key in the last lines we write the file :

```
SQL > Save Sales
Creating file sales
SQL >
```

The GET Command :

To load the contents of a command file to the buffer, use the GET Command followed by the name of the file:

```
GET file_name
```

```
SQL > get depd if 10
```

```
1   Select ename, Sal, Comm  from emp
2   where dept no <> 10
3*  order by Sal desc.
```

```
SQL > run
```

```
1   Select ename, sal, comm from emp
2   where dept no <> 10
3*  order by Sal desc
```

Salesman	Salary	Commission
SCOTT	\$3,0000	
FORD	\$3,0000	
JONES	\$2,975	
BLAKE	\$2,850	
ALLEN	\$1,600	\$300
TURNER	\$1,500	\$0
WARD	\$1,250	\$500
MARTIN	\$1,250	\$1,400
ADAMS	%1,100	
JAMES	\$950	
SMITH	\$800	

The RUN command will work only when the file contains SQL or PL / SQL commands. In this case, you must use the START Command.

The START Command

The START Command recovers the file and executes it immediately.

START file-name

If the file has the extension. sql, you do not have to specify it:

SQL > clear buffer

buffer contents removed

SQL > start sales

EMPNO	Salesman	Salary	Commission
7499	ALLEN	\$1,600	\$300
7521	WARD	\$1,250	\$500
7654	MARTIN	\$1,250	\$1,400
7844	TURNER	\$1,500	\$0

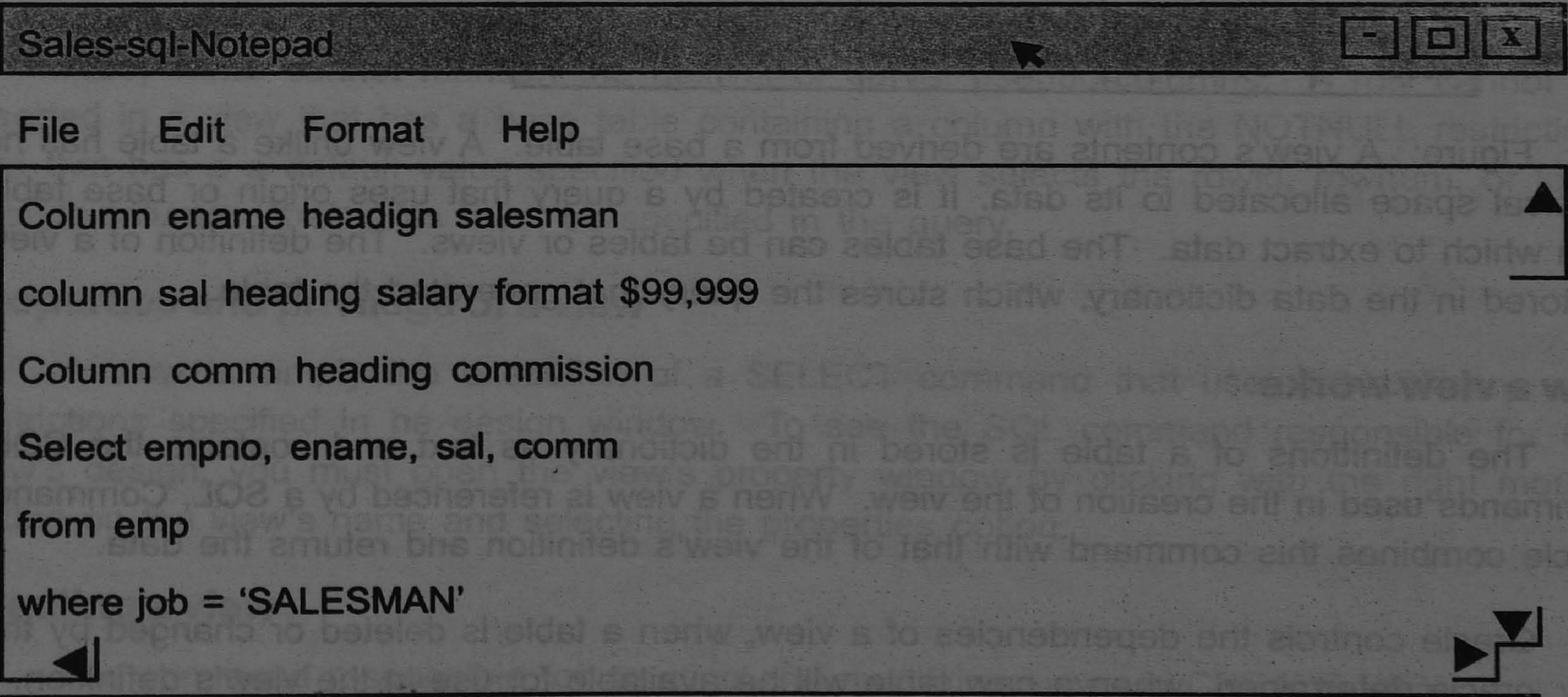
SQL >

The EDIT Command

Using the EDIT command we can ask the default editor of windows to format the contents of the command file:

SQL > EDIT Sales

This Command opens the editor.



Formatting file contents in Notepad.

SQL * plus is the main tool for oracle developers. It's essential that you master it if you want to access the database quickly and execute instant queries.

7.5. VIEWS

A view is a virtual or logical table that allows the viewing or manipulating of the contents of one or more tables in a window. A view looks like and works similarly to a normal table, except that its contents can be drawn from several different tables or even other views. The tables containing the origin columns are called base tables. A view reflects the current contents of the base table, and is different from a snapshot which displays only past data,

Base Table	EMP							
	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
	7329	SMITH	CLERK	7902	17-DEC-88	300.00	800.00	20
	7499	ALLEN	SALESMAN	7698	20-FEB-88	300.00	1600.00	30
	7521	WARD	SALESMAN	7698	22-FEB-88	5.00	1250.00	30
	7566	JONES	MANAGER	7839	02-APR-88		2975.00	20

View	Staff				
	EMPNO	ENAME	JOB	MGR	DEPTNO
	7329	SMITH	CLERK	7902	20
	7499	ALLEN	SALESMAN	7698	30
	7521	WARD	SALESMAN	7698	30
	7566	JONES	MANAGER	7839	20

Figure: A view's contents are derived from a base table. A view unlike a table has no physical space allocated to its data. It is created by a query that uses origin or base table from which to extract data. The base tables can be tables or views. The definition of a view is stored in the data dictionary, which stores the query that generated the table.

How a view works

The definitions of a table is stored in the dictionary as text and contains the SQL, Commands used in the creation of the view. When a view is referenced by a SQL, Command, Oracle combines this command with that of the view's definition and returns the data.

Oracle controls the dependencies of a view, when a table is deleted or changed by the view, oracle determined when a new table will be available for use in the view's definition.

Advantages of using a view

One major advantage of views is that they can restrict the viewing of a table's contents by limiting the columns that are displayed and the rows that are filtered. The same view can be used by different users and display distinct information depending on the way it was designed. A table for general use can be divided into specific views for certain users. For example a table with employee data can display just the name and phone number for general users, and can include the salary for those users who have the proper permission to see it, such as the human resources department.

Another advantage is that the execution of the SELECT Command is simplified, which otherwise would involve a great number of fields in many tables, and would require the specification of a schema in every column. In addition, the names of columns can be changed thus masking their origin and giving the user the opportunity to provide a name that is more descriptive or easier to remember.

A view can be referenced in the following SQL Commands:

- ▶ COMMENT
- ▶ DELETE
- ▶ INSERT
- ▶ LOCK TABLE
- ▶ UPDATE
- ▶ SELECT

Restrictions on using a view

Insert, update, and delete operations cannot be performed in a view formed by columns from more than one table when there is a grouping function, a Group By Clause, or restriction operators.

Also, views cannot mention the nextval or curval pseudo-columns. A row cannot be inserted in a view that has a base table containing a column with the NOTNULL restriction, and that has a default value specified when the view selects the rowid, rownum, or level pseudo-columns, their alias must be specified in the query.

Properties and privilege of a view

A view is simply the execution of a SELECT command that uses the columns and restrictions specified in the design window. To see the SQL command responsible for the view's design, you must open the view's property window by clicking with the right mouse button on the view's name and selecting the properties option.

Creating a view

The command responsible for creating and modifying views is CREATE VIEW.

Syntax

```
CREATE [OR REPLACE] View name As  
SELECT field (s) From table(s)  
FORCE / NOFORCE  
WHERE condition  
WITH CHECK OPTION CONSTRAINT Control  
WITH READ ONLY
```

OR REPLACE

Re-creates an existing view. It functions as an alternative to ALTER VIEW (which does not exist in Oracle), and is used to change an existing view without having to drop and grant privileges.

FORCE

Allows the creation of a view even when the base table does not exist, and the user doesn't have the privileges to create it. However, for the view to be used later, these situations must be resolved.

NOFORCE

Allow the creation of the view only when the user has the permissions and the specified tables exist. The default is NOFORCE.

WITH READ ONLY

Specifies that the INSERT, UPDATE or DELETE Commands cannot be executed in the view.

WITH CHECK OPTION

Allows the inclusion and update only of the rows that the view can select.

CONSTRAINT

The name attributed to the restriction of the CHECK OPTION CLAUSE

If the view has been created with GROUP BY, CONNECT BY, OR START WITH, OR by the DISTINCT Operator or group functions, the INSERT, UPDATE and DELETE commands cannot be executed in the view.

A View Based on a Table

The next examples were created with tables from user Demo. If you want to follow along, you must connect as User DEMO with the Password DEMO in SQL* plus.

Let's create two views based on data from a single table. A view can be created with all the columns of a table by using * to replace the names of the columns as shown below:

```
SQL > CREATE VIEW NEWDEP AS SELECT* FROM DEPARTMENT;
```

view created.

```
SQL SELECT * FROM NEWDEP;
```

Department_ID	NAME	LOCATION-ID
10	ACCOUNTING	122
20	RESEARCH	124
30	SALES	123
40	OPERATIONS	167
12	RESEARCH	122
13	SALES	122
14	OPERATIONS	122
23	SALES	124
24	OPERATIONS	124
34	OPERATIONS	123
43	SALES	167

The next example creates a view with specific columns from the EMPLOYEE table:

```
SQL > create view empbasic as
```

```
2 select employee_id, last_name, first_name,
```

```
Salary from employee;
```

Here's a partial list of the view's contents:

```
SQL > select * from empbasic where rownum < 5;
```

EMPLOYEE_ID	LAST_NAME	FIRST_NAME	SALARY
7369	SMITH	JOHN	800
7499	ALLEN	KEVIN	1600
7505	DOYLE	JEAN	2850
7506	DENNIS	LYNN	2750

A view Based on Two Tables (join view)

Next is an example of the creation of a view linking the DEPARTMENT and LOCATION tables. Let's examine the contents of each of them:

SQL >

SQL > select * from location;

LOCATION_ID	REGIONAL_GROUP
122	NEW YORK
124	DALLAS
123	CHICAGO
167	BOSTON

SQL > select * from department

Department_ID	NAME	LOCATION-ID
10	ACCOUNTING	122
20	RESEARCH	124
30	SALES	123
40	OPERATIONS	167
12	RESEARCH	122
13	SALES	122
14	OPERATIONS	122
23	SALES	124
24	OPERATIONS	124
34	OPERATIONS	123
43	SALES	167

Now, we are going to create a view using columns from both tables. Since there are two tables, you need to specify the name of each table before the name of the column as shown below:

SQL > create view gendep as

- 2 Select department-name, department.Location_id
- 3 location.regional-group
- 4 from department.location
- 5 where department.loctation_id=Location
- location_id:

Note that the WHERE clause was used, it displays the records from both tables that contain the same value in the Location_id field. Only 11 records corresponding to the 11 rows of the DEPARTMENT table are displayed, along with the regional_group field of the LOCATION table of the row with same Location_id. Without the WHERE Clause, a view with incorrect records would be created Each record of the DEPARTMENT table would be listed five times, once for each different code in the Location-id field.

SQL > Select * from gendep:

NAME	LOCATION_ID	REGIONAL_GROUP
ACCOUNTING	122	NEW YORK
RESEARCH	124	DALLAS
SALES	123	CHICAGO
OPERATIONS	167	BOSTON
RESEARCH	122	NEW YORK
SALES	122	NEW YORK
OPERATIONS	122	NEW YORK
SALES	124	DALLAS
OPERATIONS	124	DALLAS
OPERATIONS	123	CHICAGO
SALES	167	BOSTON

Instead of using the name of the table with the name of the column. We can create an alias for it, thus reducing the size of the command. To do that, use the desired alias, preferably with just one letter to minimize typing, and specify. This alias in the FROM Clause after the name of the department, as shown in the following example:

SQL > create view funcdep as

```

2      Select e.employee_id, e.Last_name,
e.first_name, e.Salary,
3      d. department_id d.name from employee e,
department.d
4      where e.department_id=d.department_id
```

Here's a partial list of the result:

SQL > Select * from funcdep order by name:

EMPLOYEE_ID	LAST_NAME	FIRST_NAME	SALARY	DEPARTMENT_ID	NAME
7782	CLARKY	CAROL	2450	10	ACCOUNTING
7839	KING	FRANCIS	5000	10	ACCOUNTING
7934	MILLER	BARBARA	1300	10	ACCOUNTING
7507	BAKER	LESLIE	2200	14	OPERATIONS
7609	LEWIS	RICHARD	1800	24	OPERATIONS
7676	SOMMERS	DENISE	1850	34	OPERATIONS

Using the SELECT Command in a view

A view functions as a normal table for most operations. The SELECT Command also works the same in a view. The next example shows a query that uses the WHERE clause.

SQL > Select * from empbasic where salary > = 3000

EMPLOYEE_ID	LAST_NAME	FIRST_NAME	SALARY
7569	ALBERTS	CHRIS	3000
7788	SCOTT	DONALD	3000
7799	FISHER	MATTHEW	3000
7839	KING	FRANCIS	5000
7902	FORD	JENNIFER	3000

SQL * Select * from gendep order by regional_group

NAME	LOCATION_ID	REGIONAL_GROUP
OPERATIONS	167	BOSTON
SALES	167	BOSTON
SALES	123	CHICAGO
OPERATIONS	123	CHICAGO
RESEARCH	124	DALLAS
OPERATIONS	124	DALLAS
SALES	124	DALLAS
ACCOUNTING	122	NEWYORK
RESEARCH	122	NEWYORK
SALES	122	NEWYORK
OPERATIONS	122	NEWYORK

Deleting a view

To delete the definition of a view from the data dictionary, use the DROP VIEW Command followed by the name of the view. This command does not affect the base tables.

Syntax

DROP VIEW name_of_view

Using views to change Tables

A view containing columns from a single table can be changed without any restrictions, and these changes can be displayed immediately in the base table.

A join view, a view, formed by column from several tables, can be changed if the SELECT Command that created it does not have:

- ▶ The DISTINCT operator
- ▶ The AVG, COUNT, GLB, MAX, MIN, STDDEV, SUM and VARIANCE functions.
- ▶ The UNION, UNION ALL, INTERSECT, and MINUS operations
- ▶ The Group By, HAVING, START, WITH of connect by clauses
- ▶ A rownum pseudo-column

If these restrictions are followed, you can use the UPDATE, INSERT or DELETE commands to change the data of the base tables that form the view.

To find out which columns may be changed, use one of the following views of the data dictionary:

Dictionary view

Description

USER_UP DATABLE_COLUMNS

Display all the columns of the user's table and views that can be changed.

DBA_UPDATABLE_COLUMNS

Display all the columns of the DBA's tables and views that can be changed.

ALL_UPDATABLE_VIEWS

Display all the columns of the tables that can be changed.

Next is an example showing the tables that can be changed in the Funcdep view:

SQL > SELECT Column - name, updatable

2 From user_updatable_columns

3 where table_name = 'FUNCDEP':

COLUMN_NAME	UPDATABLE
-----	-----
LAST_NAME	YES
FIRST_NAME	YES
SALARY	YES
DEPARTMENT_ID	NO
NAME	NO
EMPLOYEE_ID	YES

To understand the restrictions when changing composite views, you have to understand the concept of key-preserved table. A key-preserved table is a table in which the key column has all of its rows as part of the union result. In other words, it has all of its keys preserved when the view is formed.

The UPDATE Command

In the next example, let's change the contents of the Salary field in the Funcdep view. We'll give each employee in the accounting department a pay raise of 100%. The command to do this is:

```
SQL > update funcdep
2   set salary = salary * 2
3   where name = 'ACCOUNTING':
```

Compare these values to the old ones listed earlier in this chapter:

```
SQL > Select* from funcdep where name = 'ACCOUNTING':
```

EMPLOYEE_ID	LAST_NAME	FIRST_NAME	SALARY	NAME
7782	CLARKY	CAROL	4900	ACCOUNTING
7839	KING	FRANCIS	10000	ACCOUNTING
7934	MILLER	BARBARA	2600	ACCOUNTING

You are also limited by the integrity constraints of the base tables. In the next example, we try to change the contents of the employee_id field:

```
SQL > update funcdep
2   set employee_id = 9999
3   where last_name = 'CLARK':

update funcdep
```


ERROR on row 1:

ORA - 02292: integrity constraint (DEMO.SYS-C00801) violates-child record located

Following are two other situations:

SQL > update funcdep

2 set name = 'SALES_FORCE'

3 where name = 'SALES':

where name = 'SALES'

*

ERROR on row 3:

ORA-01779; cannot change a column that list a non key - preserved table.

SQL > update funcdep

2 set Last_name = 'CLARKY'

3 where Last _name = 'CLARKY'

1 row updated

when WITH CHECK OPTION is included in a view's definition. All the linked columns and all the repeated columns of the table are changed.

The INSERT Command

An INSERT command will work only when all the constraints of the base table are satisfied. We cannot insert a new row in the EMP table through funcdep. Since there is no department_id field in the EMPLOYEE table as the FOREIGN KEY.

Replacing a view

The replacement of a view simply changes its definition in the data dictionary.

A view can be replaced in one of two ways. One method is to delete the view with the DROP Command and then use the CREATE VIEW command to rebuild it. The second option does not delete the view, but uses the OR REPLACE option of the CREATE VIEW command. This second option replaces the definition of the view and preserves the existing security authorizations.

Viewing the Definitions of the view

The definitions of a view are stored in a view called USER_VIEWS in the data dictionary. The following shows the structure of such a view:

SQL > describe user _views:

Name	Null?	Type
-----	-----	-----
VIEW_NAME	NOT NULL	VARCHAR 2 (30)
TEXT_LENGTH		NUMBER
TEXT		LONG
TYPE_TEXT_LENGTH		NUMBER
TYPE_TEXT		VARCHAR 2 (4000)
OID_TEXT_LENGTH		NUMBER
OID_TEXT		VARCHAR 2 (4000)
VIEW_TYPE_OWNER		VARCHAR 2 (30)
VIEW_TYPE		VARCHAR 2 (30)

To see the name and text of the command that builds this view, use the view_name column and text:

SQL > Select view_name text from user_views:

VIEW_NAME

TEXT

EMPBASIC

Select employed_id Last_Name, first_name, Salary from employee

GENDEP

Select department, name, department, Location_id.

location.regional_group

from dep

NEWDEP

SELECT "DEPARTMENT_ID"."NAME", LOCATION_ID" FROM DEPARTMENT

SALES

SELECT SALESPERSON_ID. SALES_ORDER, CUSTOMER-ID

CUSTOMER.NAME

CUSTOMER,

P

Materialized views

A materialized views is a physical copy of the base table with the results moved to anothe schema object. Materialized views are also called snapshots. Because they are a king of photograph of the base table.

Designing Multiple tables and views simultaneously

You can create tables and views at the same time with the CREATE SCHEMA Command. In this commar d, you specify the defintions of all the tables and views, along with the access privileges of each of them. If, for some reason, one of the tables or views is not created, all the creation command are cancelled.

A schema is created by the CREATE USER command, which creates some of the scheme objects at the same time.

Syntax

```
CREATE_SCHEMA AUTHORIZATION name_of_Schema  
create_table_statement  
create_view_statement  
grant_statement
```

Arguments

name_of_Schema

This must be identical to the user name created by the CREATE USER command.

create_table_statement

A CREATE VIEW Command that is part of the schema. Several commands can be specified. You cannot use the final semicolon grant-statement.

create-view-statement

A CREATE VIEW Command that is part of the schema. Several commands can be specified. You cannot use the final semicolon.

grant-statement

An optional GRANT command used to attribute privileges. Cannot be returned with Semicolon.

This next example creates two table and a view.

```
CREATE          TABLE Dep_tap (  
  
Deptno         NUMBER (3) PRIMARY KEY,
```

```

DName          VARCHAR 2 (15),
LOC            VARCHAR 2 (25) )

CREATE TABLE Emp_tab(

Empno          NUMBER (5) PRIMRY KEY,
Ename         VARCHAR 2 (15) NOT NULL,
JOB           VARCHAR 2 (10),

Mgr           NUMBER (5),

Hiredate      DATE DEFAULT (Sysdate)

Sal           NUMBER (7,2),

Comm          NUMBER (7, 2),

Dept no       NUMBER (3) NOT NULL

```

```

CONSTRAINT Dept_fkey REFERENCES Dep_tap (Deptno)

```

```

GRANT SELECT ON Sales_Staff To human_resources:

```

```

CREATE SCHEMA AUTHORIZATION Scott

```

```

CREATE VIEW Sales_Staff As

```

```

SELECT Empno, Ename, Sal, Comm

```

```

From Emp_tap

```

```

WHERE Deptno = 30

```

```

WITH CHECK OPTION CONSTRAINT Sales_staff_Cnst

```

In this chapter you've learned that views are an important mechanism to help display tables and restrict database access through the limitation of columns. You've also seen how to mask the origin table or column by replacing the column schema with an alias. Mastering views is important for those who develop database applications.

Unit - VIII

8.1: Expressions and Operators

Oracle supports many of the common operators found in most languages. The operators act on the operands, i.e., on the elements that will be analysed by the operator. There are two classes of operator - unary operators, which act on only one operand, and binary operators, which act on two or more operands.

Precedence of Operators

There are several types of operators. When more than one type is present in an expression, they are evaluated according to the following precedence order;

1. Unary operators
2. Division & Multiplication
3. Addition & Subtraction
4. Comparison Operators
5. NOT Operators
6. AND Operator
7. OR Operator

Types of Operators

Below are the types of operators that are supported by Oracle:

Type	Name	Value/Description
Operate on numeric Operands	Multiplication	*
	Division	/
	Subtraction	-
	Addition	+
Character Manipulate strings	Concatenation	
Comparison		
Check for similarities & differences	Equal to	=
	Not equal to	!=
	Greater than	>
	Less than	<
	Greater than or equal to	>=
	Less than or equal to	<=
	Null	IS NULL

Logical

Manipulate Comparison NOT

expression AND

Set OR

Combine the results of UNION Returns the rows

two queries in one selected by any query

query INTERSECT Returns the rows selected by both queries

MINUS Returns the rows selected by the first but not the second query.

ORACLE SQL * Plus

File Edit Search Options Help

Sql> Select ename, hiredate from emp where sys date_hiredate <365.11;

E NAME HIREDATE

SCOTT 19-APR-87

ADAMS 23-MAY-87

Sql> Select ename, Sal, Sal * 1.3 from emp where Sal*1.3<=1,800;

ENAME SAL SAL*1.3

SMITH 1000 1300

WARD 1250 1625

MARTIN 1250 1625

ADAMS 1100 1430

JAMES 950 1235

MILLER 1300 1690

The next example shows how to concatenate, or link, character strings, using the || operator:

SQL > Select e name || 'works as' || job from emp;

ENAME	11 'Works as' 11 JOB
SMITH	works as Clerk
ALLEN	works as SALESMAN
WARD	works as SALESMAN
JONES	works as MANAGER
MARTIN	works as SALESMAN
BLAKE	works as MANAGER

The next example shows the use of a comparison operator. These operators are generally used to compare the contents of a column with a certain value.

SQL > Selct * from emp where comm <>0;

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7499	ALLEN	SALESMAN	7698	2/20/81	1600	300	30
7421	WARD	SALESMAN	7698	02/02/81	1250	500	30
7654	MARTIN	SALESMAN	7698	09/08/81	1250	1400	30

SQL > Select * from emp where Sal > 3000;

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT		11/17/81	5000		10

SQL > Select* from emp where Sal> = 3000

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7788	SCOTT	ANALYST	7566	04/19/87	3000		20
7839	KING	PRESIDENT		11/17/81	5000		10
7902	FORD	ANALYST	7566	12/03/81	3000		20

SQL> SELECT * FROM EMP WHERE SAL < 3000;

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	12/17/80	800		20
7499	ALLEN	SALESMAN	7698	02/20/81	1600	300	30
7521	WARD	SALESMAN	7698	02/22/81	1250	500	30
7566	JONES	MANAGER	7839	04/12/81	2975		20
7654	MARTIN	SALESMAN	7698	09/28/81	1250	1400	30
7698	BLAKE	MANAGER	7839	05/01/81	2850		30
7782	CLARK	MANAGER	7839	06/09/81	2450		10

7844	TURNER	SALESMAN	7698	09/08/81	1500 0	30
7876	ADAMS	CLERK	7788	05/23/87	1100	20
7900	JAMES	CLERK	7698	12/03/81	950	30
7934	MILLER	CLERK	7782	01/23/82	1300	10

11 lines selected

The next example shows the use of logical operators combined with other operators:

SQL > select*from emp where deptno = 30 and Sal > 2000;

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7698	LAKE	MANAGER	7839	05/01/81	2850		30

SQL > select*from emp where deptno <> 30 and Sal > 2000;

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7566	Jones	Manager	7839	04/02/81	2975		20
7782	Clark	Manager	7839	06/09/81	2450		10
7788	Scott	Analyst	7566	04/19/87	3000		20
7839	King	President		11/17/81	5000		10
7902	Ford	Analyst	7566	12/3/81	3000		20

SQL > Select * from emp where Sal < 1000 or Sal > 3000;

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	Smith	Clerk	7902	12/17/80	800		20
7839	King	President		11/17/81	5000		10
7900	James	Clerk	7698	12/3/81	950		30

If you combine more than one logical operator, pay attention to the precedence order. The AND Operator is analyzed first and then the OR Operator. Now examine the same example with different precedence orders. We want to list all the employees with salaries above \$2,500 and who work in department 10 or 30.

In this first example, Dept no = 30 and Sal > 2500 are analyzed as one condition and Dept no = 10 as another condition.

SQL > select * from emp where dept no = 10 or dept no = 30 and Sal > 2500

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7698	Blake	Manager	7839	05/01/81	2850		30
7782	Clark	Manager	7839	06/09/81	2450		10
7839	King	President		11/17/81	5000		10
7934	Miller	Clerk	7782	01/23/82	1300		10

To change the logic so that only employees of departments 10 or 30 who earn salaries above \$2,500 are listed, you have to place the first condition in paranthesis. This forces it to be analyzed first and its result is then compared to the second expression

Sal > 2500

SQL > Select * from emp where (dept no = 10 or dept no =30) and

Sal > 2500.

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7698	Blake	Manager	7839	05/01/81	2850		30
7839	King	President		11/17/81	5000		10

FUNCTIONS

A SQL function is a routine that performs a specific operation and returns the result. It is similar to a procedure, except that a procedure does not return any vlaue. A function can receive Parameters that will be used in its processing.

The basic syntax of a function is:

Func name (arg1, arg2, ARGn)

Arguments

Funcname

The name of the function, such as ABs or LOWER arg1 and arg2 are the optional parameters of the function. SQL Functions can be of two types. Single row functions, or scalar, and group functions, or aggregated. A single row function returns the result for each row of a table or view. A group function returns only one result for a group of rows.

For example, the LOWER function converts its argument to lower-case. In the following example, this function acts on all the rows of a table:

SQL > Select dname, lower(dname) from dept;

DNAME	LOWER (DNAME)
-----	-----
ACCOUNTING	accounting
RESEARCH	research
SALES	sales
OPERATIONS	operations

Those functions can be used in SELECT Commands that have the WHERE, START WITH or CONNECT By Clauses. However, they cannot be used with GROUP BY SQL > Select ename, lower(enmae) from emp where deptno = 10.

ENAME	LOWER (ENAME)
-----	-----
CLARK	Clark
KING	King
MILLER	milller

The group fucntions process a group of rows in the table, and return the result. Most of them act on all the rows of the default table. They accept the following parameters:

DISTINCT

Makes the function act only on the rows that have different values.

ALL

Makes the function consider all the values (rows), including duplicates. This is the default. With the exception of the COUNT funciton, all other functions ignore null values.

The group functions are

AVG

Returns the arithmetic average of n

Syntax

AVG ([DISTINCT / AU]n)

COUNT

Returns the number of rows fo the query.

Syntax

COUNT ({*/[DISTINCT/ALL] expr})

Expr represents the field or non-null expression.

MAX

Returns the maximum value of expr

Syntax

MAX ([DISTINCT / ALL] expr)

MIN

Returns the minimum value of expr.

Syntax

MIN ([DISTINCT / ALL] expr)

Following are several examples of these functions:

SQL > Select min (Sal), Max(Sal), avg(sal) from emp;

MIN(SAL)	MAX(SAL)	AVG(SAL)
-----	-----	-----
950	5000	2087.5

SQL> Select Count (*) from emp;

COUNT (*)

14

SQL> Select Count (COMM) from emp;

COUNT (COMM)

4

The above example returns only the rows with values other than NULL. Note the following example returns only the number of rows with unique department codes:

SQL > Select COUNT (dept no) from emp;

COUNT (DEPT NO.)

14

SQL> Select Count (distinct dept no) from emp;

COUNT (DISTINCT DEPT NO)

3

The INSERT Command

This command inserts a new row in the table and fills the columns with specific values. It is not necessary to assign contents to all fields.

Basic Syntax

INSERT INTO Scheme table (columns)

VALUES (Values)

This example inserts a new row in the DEPT table:

SQL> Select * from dept;

DEPT NO	DNAME	LOC
10	ACCOUNTING	NEWYORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

SQL> insert into dept values (50,'MARKETING', SAO PAULO');

1 line created

SQL > SELECT * FROM DEPT;

DEPT NO	DNAME	LOC
10	ACCOUNTING	NEWYORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON
50	MARKETING	SAO PAULO

The following example adds a new row with data for all of the fields. Note that you don't have to specify a name for the fields (columns), because the command assumes that the first value is assigned to the first column, the second value to the second column and so on.

SQL > INSERT INTO emp VALUES (7777, 'RAMALHO',
'MANAGER' 7902, SYSDATE, 1200, NULL, 20).

1 line created

SQL > SELECT * FROM EMP WHERE DEPTNO = 20;

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	12/17/80	800		20
7566	JONES	MANAGER	7839	04/02/81	2975		20
7788	SCOTT	ANALYST	7566	04/19/87	3000		20
7876	ADAMS	CLERK	7788	05/23/87	1100		20
7902	FORD	ANALYST	7566	12/03/81	3000		20
7777	RAMALHO	MANAGER	7902	05/10/89	1200		20

6 lines created.

This example inserts only some fields in the row. In this case, you must specify a SELECT clause with the names of columns to be used:

SQL > insert into (Select empno, ename, hiredate, From emp) values (8888, 'PARKER', Sysdate);

1 line created

SQL > Select * from emp;

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	12/17/80	800		20
7499	ALLEN	SALESMAN	7698	02/20/81	1600	300	30
7521	WARD	SALESMAN	7698	02/22/81	1250	500	30
7566	JONES	MANAGER	7839	04/02/81	2975		20
7654	MARTIN	SALESMAN	7698	09/28/81	1250	1400	30
7698	BLAKE	MANAGER	7839	05/01/81	2850		30
7782	CLARK	MANAGER	7839	06/09/81	2450		10
7788	SCOTT	ANALYST	7566	04/19/87	3000		20
7839	KING	PRESIDENT		11/17/81	5000		10
7844	TURNER	SALESMAN	7698	09/08/81	1500	0	30
7876	ADAMS	CLERK	7788	05/03/87	1100		20
7900	JAMES	CLERK	7698	12/03/81	950		30
7902	FORD	ANALYST	7566	12/03/81	3000		20
7934	MILLER	CLERK	7782	01/23/82	1300		10
7777	RAMALHO	MANAGER	7902	05/10/99	1200		20
8888	PARKER			05/10/99			

16 lines Selected

Data Integrity

When a row is inserted, oracle checks the table definitions to see whether any restrictions are being applied to the table. In the case of the EMP table, the EMPno field in a unique primary key, which does not allow duplicates. If we try to re-insert the record for parker, an error will be generated:

```
SQL > insert into (Select empno, ename, hiredate From emp)
```

```
Values (8888, 'PARKER', Sysdate);
```

```
insert into (Select empno, ename, hiredate from emp)
```

```
values (8888, 'PARKER', Sysdate)
```

```
*
```

```
ERROR on line /:
```

```
ORA - 00001:exclusive restrition (SCOTT. PK_EMP) violated
```

After inserting the row for employee 8888, the command generated an error, since the key, in this case, the employee number, does not allow duplicates.

The UPDATE Command

The UPDATE Command allows the updating of one or more fields of a row or group of rows in a table or a view. The contents of each field must be adjusted with the SET Clause. When more than one column is updated; the column = value pairs must be separated by commas.

Basic Syntax:

```
UPDATE Schema.table
```

```
SET Column = Value
```

```
WHERE condition
```

The next example shows that only some fields of employee 8888 are updated:

```
SQL> update emp set job = 'CLERK', MGR = 7839, SAL = 2000,
```

```
DEPT No = 10 WHERE EMPNO = 8888;
```

```
1 line undated.
```

```
SQL > Select * from emp where empno = 8888;
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
8888	PARKER	CLERK	7839	05/10/99	2000		10

The DELETE Command

The DELETE command removes row that match a specified condition from a table or view.

Basic Syntax

DELETE FROM Schema.table WHERE Condition

The next example inserts a row in the DEPT table, and then deletes it:

```
SQL> insert into dept VALUES (60, 'MEDIA', 'SAO JOSE');
```

1 line created.

```
SQL> Select * from dept;
```

DEPT NO	DNAME	LOC
10	ACCOUNTING	NEWYORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON
50	MARKETING	SAO PAULO
60	MEDIA	SAO JOSE

6 lines Selected.

```
SQL > delete from dept where deptno = 60;
```

1 line deleted.

```
SQL > Select * from dept;
```

DEPT NO	DNAME	LOC
10	ACCOUNTING	NEWYORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON
50	MARKETING	SAO PAULO

Transactions

In addition to the DML and DDL Commands we have seen so far, a third category of Commands called TPL (transaction process language) is included in SQL. The Commands in this category write to the database the changes made by the DML and DDL Commands, and recover the database's states when something goes wrong. Imagine a situation in which several tables must be updated to reflect a complete transaction. For example, when issuing a sales order, the system updates the inventory, creates a record in the billing table, and updates the payable accounts. If a problem occurs during the update process, one of the tables may not be updated. Oracle has two commands that allow writing data after a successful operation or after recovering the previous situation to the beginning of processing: COMMIT and ROLLBACK

COMMIT Writes a transaction to the database. Its basic syntax is : COMMIT [WORK]

ROLLBACK restores a transaction. It recovers the database to the position where it was after the last COMMIT Command was executed. Its syntax is :

ROLLBACK

The next example shows the contents of the DEPT table. Then a new row is inserted in the table. Imagine that in addition to this particular row, some rows were also inserted in other tables, or data was updated in other locations. In the middle of this process, there is a power outage and the tables are only partially updated.

```
SQL > insert into dept values (70, 'PLANT', 'DALLAS');
```

```
1 line created
```

```
SQL > Select * from dept;
```

DEPT NO	DNAME	LOC
-----	-----	-----
10	ACCOUNTING	NEWYORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON
50	MARKETING	SAO PAULO
70	PLANT	DALLAS

```
6 lines Selected.
```

It may seem that the row was correctly written to the database. However, it has not yet been written. The row will be written only when the session is ended, or when a COMMIT Command is executed.

SQL > roll back;

ROLLback Complete

SQL > Select * from dept;

DEPT NO	DNAME	LOC
-----	-----	-----
10	ACCOUNTING	NEWYORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

SQL>

Note that the marketing department and plant rows were not definitely written and therefore, these rows were eliminated when the recovery was performed. Here is the same example where COMMIT is executed: SQL> insert into dept values (70, 'PLANT', 'DALLAS').

1 line created

SQL > COMMIT;

Validation Complete

SQL> roll back;

Rollback Complete

SQL> Select * from dept;

DEPT NO	DNAME	LOC
-----	-----	-----
10	ACCOUNTING	NEWYORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON
70	PLANT	DALLAS

With these basic commends you are able to perform most recovery and table maintenance activities in an oracle database. The following, is the lists the main PL/SQL Commands, which improve the functionality of the SQL language.

PL / SQL Commands.

ALTER CLUSTER
 ALTER DATA BASE
 ALTER FUNCTION
 ALTER INDEX
 ALTER PACKAGE
 ALTER PROCEDURE
 ALTER PROFILE
 ALTER RESOURCE COST
 ALTER ROLE
 ALTER ROLLBACK SEGMENT
 ALTER SEQUENCE
 ALTER SESSION
 ALTER SNAPSHOT
 ALTER SNAPSHOT LOG
 ALTER SYSTEM
 ALTER TABLE
 ALTER TABLESPACE
 ALTER TRIGGER
 ALTER USER
 ALTER VIEW
 ANALYZE
 AUDIT
 COMMENT
 CREATE CLUSTER
 CREATE CONTROLFILE
 DROPCLUSTER
 DROP DATABASE LINK
 DROP FUNCTION
 EXPLAIN PLAN
 NO AUDIT
 RENAME
 REVOKE
 SAVEPOINT
 TRUNCATE.

8.2: TRIGGERS

Using triggers is one of the most practical ways to implement routines, thus granting integrity of data or operations. Triggers are routines or procedures that are used when an INSERT, UPDATE or DELETE Command is executed in a table or a view. The name "trigger" is appropriate; as these are triggered whenever the above mentioned commands are executed. In oracle 8i they can be written in PL / SQL, Java or even in C

A trigger is automatically executed without any action required by the user. A stored procedure, on the other hand, needs to be explicitly invoked. This is the main difference between triggers and stored procedures.

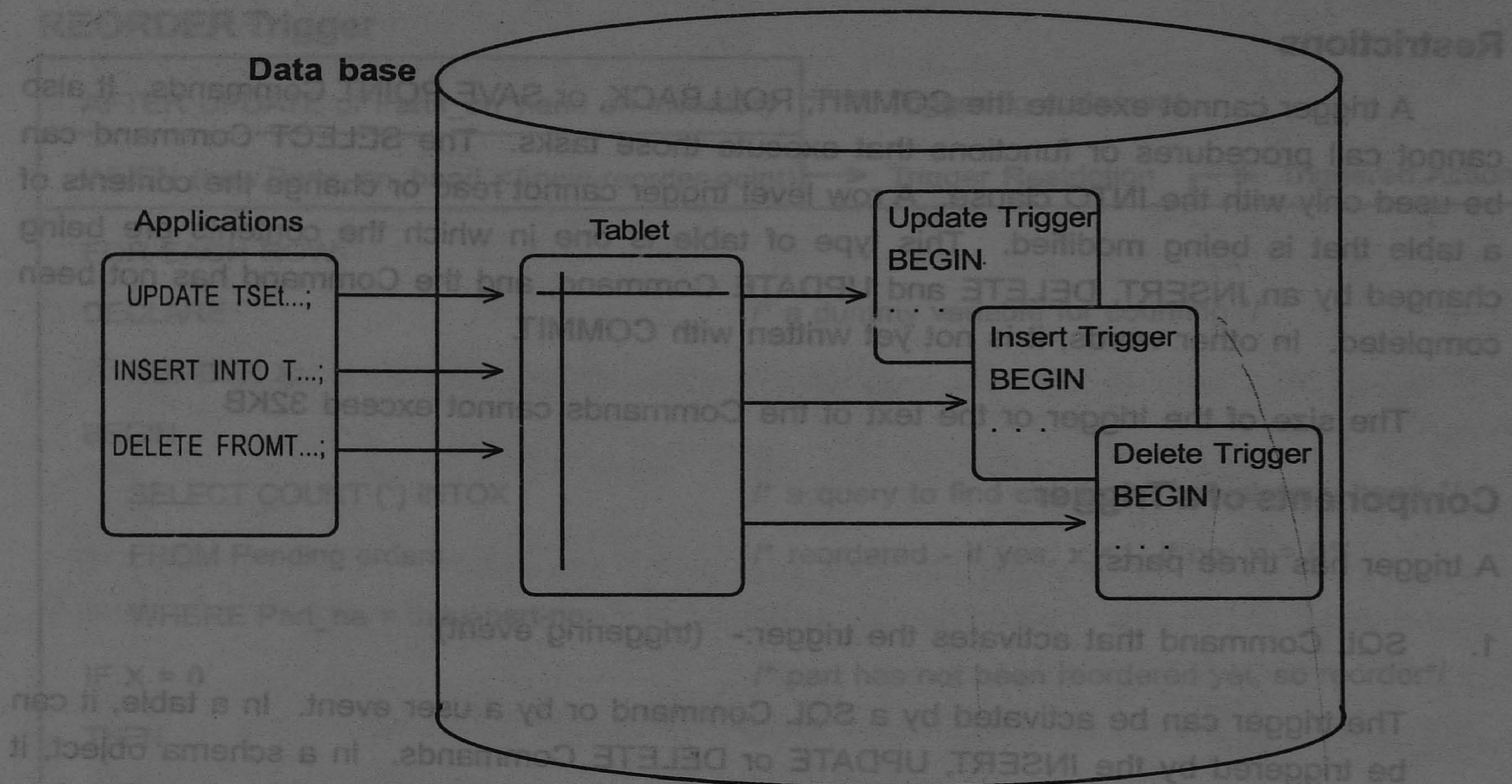


Fig.8.1

The anatomy of a Trigger.

The major application of a trigger is to create consistencies and access restrictions to the database, as well as implement security routines. Instead of giving control of the application to the table itself, triggers execute these controls, providing much more security to the handling of the data base.

Triggers are also used to:

- ▶ Create the contents of a column derived from other columns.
- ▶ Create Validation mechanisms involving searches in multiple tables
- ▶ Create logs to register the use of a table.
- ▶ Update other tables as a result of inclusions or changes in the current table.

A trigger can execute the commands contained in its blocks, or enable the stored procedures to execute certain tasks. A trigger is associated to a table and in some cases, to a view. When a view is used, the base table triggers are normally enabled.

A trigger can perform the role of a Constraint, forcing an integrity rule. If the task can be performed both by triggers and constraints, your first option should be constraint. In this case, the triggers should be used only when a referential integrity rule cannot be executed using the NOT NULL, UNIQUE KEY, PRIMARY KEY, FOREIGN KEY, CHECK, DELETE CASCADE, DELETE SET, and NULL Constraints.

Restrictions

A trigger cannot execute the COMMIT, ROLLBACK, or SAVE POINT Commands. It also cannot call procedures or functions that execute those tasks. The SELECT Command can be used only with the INTO clause. A row level trigger cannot read or change the contents of a table that is being modified. This type of table is one in which the contents are being changed by an INSERT, DELETE and UPDATE Command, and the Command has not been completed. In other words, it is not yet written with COMMIT.

The size of the trigger or the text of the Commands cannot exceed 32KB

Components of a Trigger

A trigger has three parts:

1. SQL Command that activates the trigger:- (triggering event)

The trigger can be activated by a SQL Command or by a user event. In a table, it can be triggered by the INSERT, UPDATE or DELETE Commands. In a schema object, it can be triggered through the CREATE, ALTER or DROP Commands, when the data base is loaded or shut down, or through an error message.

The INSERT, DELETE and UPDATE Commands enable a trigger. The same trigger can be invoked in more than one situation i.e., only when an INSERT Command is executed, or when an INSERT or DELETE command is executed.

2. Trigger restrictions

Represented by the WHEN clause, it specifies what condition must be true for the trigger to be triggered .

3. Trigger action

The PL / SQL block or Java or C routine executed by the trigger.

TYPES of Triggers

Oracle 8i has four types of triggers, depending on the application:

- ▶ Table triggers applied to DML Commands
- ▶ INSTEAD of triggers applied to views
- ▶ SYSTEM EVENT triggers applied to the database in the startup, shut down, and error handling operations.
- ▶ USER EVENT triggers applied to a user or object schema

REORDER Trigger

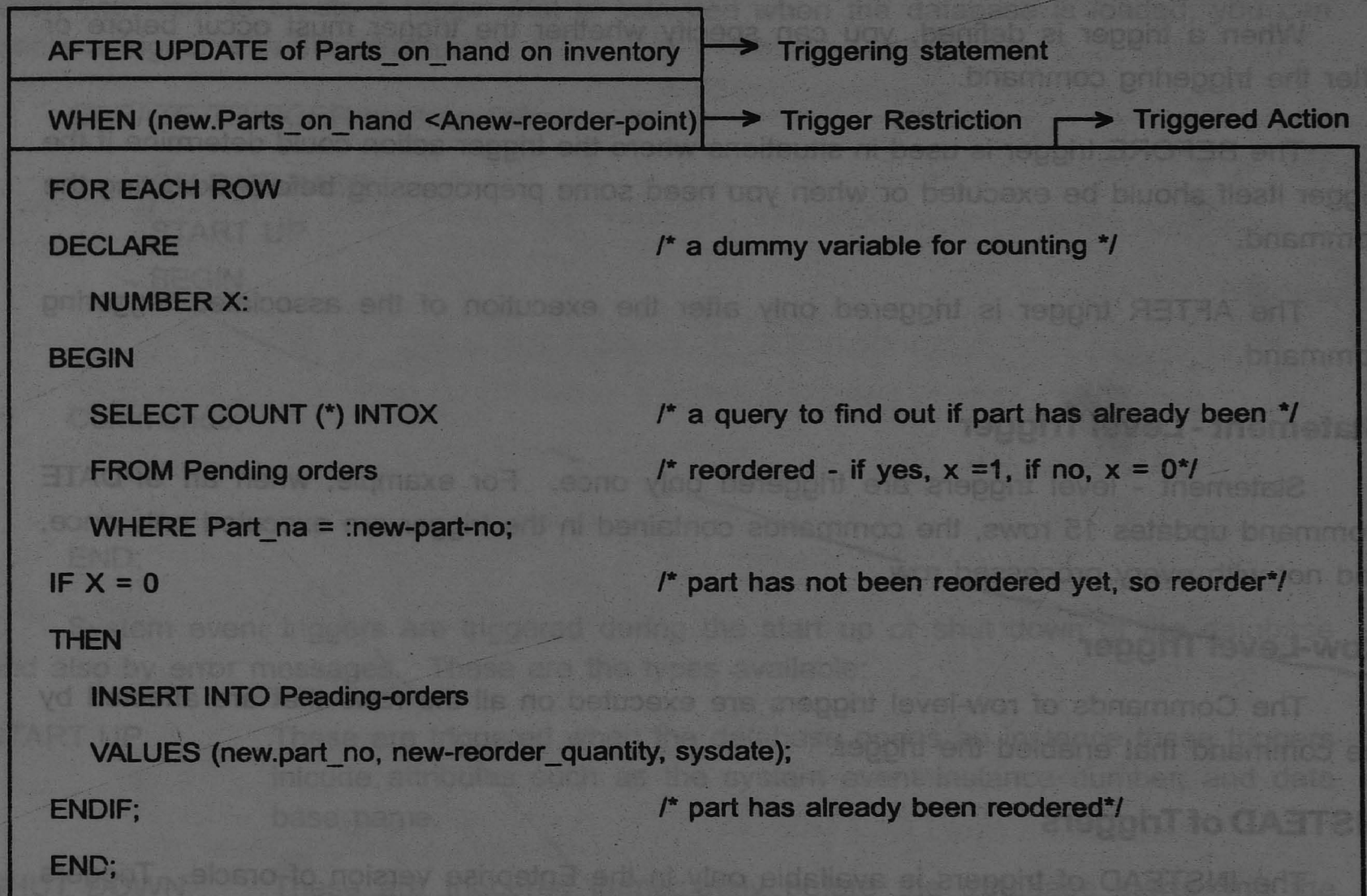


Figure : Example of a trigger.

Table Triggers

There are two distinct types of triggers that can be used in a table. The first type, statement level, is executed only when a triggering event occurs. The other type, row-level is used in every row of the table processed by the SQL Command.

A trigger can be activated before or after the execution of an activation Command (INSERT, DELETE and UPDATE). A table can contain upto 12 triggers associated with the activation

commands and the triggering event. There are six row-level triggers, and six statement - level triggers

- ▶ BEFORE INSERT
- ▶ AFTER INSERT
- ▶ BEFORE UPDATE
- ▶ AFTER UPDATE
- ▶ BEFORE DELETE
- ▶ AFTER DELETE

When a trigger is defined, you can specify whether the trigger must occur before or after the triggering command.

The BEFORE trigger is used in situations where the trigger action could determine if the trigger itself should be executed or when you need some preprocessing before executing the command.

The AFTER trigger is triggered only after the execution of the associated triggering command.

Statement - Level Trigger

Statement - level triggers are triggered only once. For example, when an UPDATE Command updates 15 rows, the commands contained in the trigger are executed only once, and not with every processed row.

Row-Level Trigger

The Commands of row-level triggers are executed on all the rows that are affected by the command that enabled the trigger.

INSTEAD of Triggers

The INSTEAD of triggers is available only in the Enterprise version of oracle. Triggers are a good way to change views that cannot be directly changed with the SQL DML commands. (INSERT, UPDATE and DELETE). These triggers are called INSTEAD OF triggers, because, unlike other triggers oracle calls the trigger itself rather than executing an instruction through it.

System Event and User Event Triggers

Triggers can be associated with system or schema object events of a user. The types available are:

- ▶ System Events
- ▶ Database Startup and shutdown
- ▶ Server error message events
- ▶ User events
- ▶ User Log on and log off
- ▶ DDL Commands (CREATE, ALTER and DROP)
- ▶ DML Commands (INSERT, DELETE and UPDATE)

The system event triggers can be defined at the database or schema level. For example, when you want to create a trigger that is activated when the database is loaded, you can create a trigger with this structure.

```
CREATE TRIGGER start up DB
```

```
ON DATABASE
```

```
START UP
```

```
BEGIN
```

```
...
```

```
Commands;
```

```
...
```

```
END;
```

System event triggers are triggered during the start up or shut down of the database and also by error messages. These are the types available:

START UP	These are triggered when the database opens an instance these triggers include attributes such as the system event instance number, and data base name.
SHUT DOWN	These are triggered immediately before the database starts and the shutdown process. These triggers include attributes such as system event, instance number, and database name.
SERVER ERROR	These are triggered whenever an error occurs. These triggers include attributes such as the system event and the error number.

The user event trigger occur when a user logs on, logs off, or executes DDL or DML commands. The attributes of the LOGON and LOGOFF triggers are the system event and the user name:

LOGON This is triggered after the user connection.

LOGOFF This is triggered when the user starts his or her connection.

DDL Command triggers, which are associated to a database or schema, have attributes such as the system event, the type of schema object and the name;

BEFORE CREATE and AFTER CREATE These are triggered when a schema object is Created

BEFORE ALTER and AFTER ALTER These are triggered when a schema object is modified

BEFORE DROP and AFTER DROP These are triggered when a schema object is eliminated from the schema or database

Creating a Trigger

A trigger is created with the **CREATE TRIGGER** Command. It has the following syntax:

Syntax

```
CREATE [OR REPLACE] TRIGGER [Schema.] trigger
```

```
{BEFORE / AFTER/ INSTEAD OF}
```

```
{DELETE
```

```
/INSERT
```

```
/ UPDATE [of column [, column].....]}
```

```
[OR {DELETE
```

```
/INSERT
```

```
/ UPDATE [of Column [, Column].....]}]....
```

```
ON [Schema.] {table / view}
```

```
[[REFERENCING { OLD [AS] old
```

```
/ NEW [AS] new } ...]
```

```
FOR EACH {ROW / STATEMENT} [WHEN [CONDITION]]]
```

```
pl / Sql _ block
```

Arguments

OR REPLACE

Re-creates an existing trigger. You use this option to change the definition of an existing trigger without having to delete it first.

Schema

This is the schema that will contain the trigger. If schema is omitted, oracle will create the trigger in the user's schema.

trigger

The name of the trigger to be created.

BEFORE

Makes oracle activate the trigger before executing the triggering command. For row triggers, this is the individual triggering before each affected row is changed. You cannot specify a BEFORE trigger in a view or object view.

AFTER

Makes oracle activate the trigger after the execution of a triggering command. You cannot specify an AFTER trigger in a view or object view.

INSTEAD OF

Makes oracle activate the trigger instead of executing the triggering command. As the default, the INSTEAD of triggers are enabled in each row. INSTEADof is a valid option only for views; you cannot specify an INSTEAD of trigger in a table.

DELETE

Makes oracle activate the trigger whenever a DELETE Command eliminates a row in a table

INSERT

Makes Oracle activate the trigger whenever an INSERT command adds a row to the table.

UPDATE

Makes oracle activate the trigger whenever an UPDATE Command changes the value in one of the columns specified in the of clause. If omitted, oracle will activate the trigger whenever an UPDATE Command changes a value in any column of the table.

You cannot specify an of clause with an INSTEAD OF trigger. Oracle activates the INSTEAD of Triggers whenever an UPDATE changes the value of any column in the view. You cannot specify nested tables of LOB columns in of clauses.

ON

On specifies the schema and the name of the table (table) or the view (view) of one of the following parameters when the trigger is created: table, object table, view, or object view.

If schema is omitted, oracle assumes the table is in its own schema. You are allowed to create triggers in indexed tables. You can also create a trigger in a table of the Sys schema.

table

The name of a table or an object table

View

The name of a view or an object view

REFERENCING

Specifies the correlation names. You can use the correlation names in the PL / SQL block, and in the WHEN clause in a row trigger. Specifically referencing the new and old values of the current row. The default correlation names are OLD and NEW. If your row trigger is associated to a table called OLD or NEW, use this clause to specify different names for the correlation (don't confuse the table name with the correlation name). If a trigger is defined as an object or view table, OLD and NEW refer to the object instances.

FOR EACH ROW

Designates a trigger such as a row trigger. Oracle triggers a row trigger for each row that is affected by a triggering command, which meets the optional constraints conditions defined in the WHEN clause.

Except for the INSTEAD of triggers, if this clause is omitted, the trigger is a triggering command: Oracle triggers a triggering Command only once, when the triggering command is submitted, as long as the optional conditions of the trigger constraint are met The INSTEAD of triggers are implicitly enabled for each row

WHEN (condition)

Specifies the trigger restriction, which is a SQL condition that must be met before oracle can activate the trigger. You can specify a trigger restriction only for a row trigger. Oracle evaluates this condition for every row affected by the triggering command.

You cannot specify trigger restrictions for the INSTEAD of trigger.

You can reference object columns and their attributes, VARRAY, nested tables or LOB columns. It is not possible to Call PL / SQL functions or methods in trigger restrictions.

Pl / Sql _ block

The PL / SQL block executed by oracle to activate the trigger.

Note: The PL / SQL block of a trigger cannot contain SQL transaction control commands (COMMIT, ROLLBACK, SAVEPOINT and SET CONSTRAINT)

Modifying a Trigger

A trigger cannot be directly modified. To change it you must re-create it with the CREATE Command. If a trigger had its privileges granted to other users, they remain valid as long as the trigger exists.

Enabling / Disabling a Trigger

When a trigger is created, it is automatically enabled and is triggered whenever the triggering command and the execution command is true. To disable the execution of the trigger, use the ALTER TRIGGER command with the DISABLE clause. To enable it again, use the ENABLE clause.

Syntax

```
ALTER TRIGGER name DISABLE / ENABLE;
```

Example

```
ALTER TRIGGER change _ Sal DISABLE;
```

Deleting Trigger

To delete a trigger, use the DROP TRIGGER Command:

Syntax

```
DROP TRIGGER name_ of_trigger;
```

This removes the trigger structure from the database and with draws the privileges that were granted to other users.

References to columns Inside a Trigger

You can access the value of a column inside a row - level trigger. Depending on the operation that is being performed. You may need to precede the column name with new or old. This is necessary because you may be able to obtain both the old and new values.

- ▶ For an INSERT Command, the values of the fields that will be written must be preceded by : new
- ▶ For a DELETE Command, the values of the row fields that are being processed must be preceded by :old.
- ▶ For an UPDATE Command, the original value that is being written is accessed with : old. The new values that will be written must be preceded by : new.

Take an example, which determines whether the value inserted is greater than 10000.

```
IF : new Sal > 10000 (Commands)
```

The following determines whether the new value attributed to the field is greater than the old value:

IF : new Sal < : Old. Sal (Commands)

The old and new values are available in BEFORE and AFTER triggers.

Validating the Data Entry

If you are using SQL * plus to create triggers, you will certainly notice its limitations as a program editor. Use a processing editor instead. Such as Note Pad, and cut and paste to insert data in a command row of SQL * plus . In the following example we will determine a maximum value for the salary field. No employee can earn more than \$8,000. In addition to activating the trigger on the insert operation, thus preventing a new employee from earning more than the limit, the update routines must be checked to make sure the salary of the existing employees is not changed to values that are not allowed. When the UPDATE Command is selected, the table columns are displayed, and the ones that changed can be marked and triggered when there are updates.

In this example, the Sal column must be marked. Since the trigger is a row-level trigger, for Each Row must be marked and the OLD AS and NEW AS fields must be filled out.

After entering this information , Click on the Body tab, and enter the following command block between BEGIN and END

```
if: new sal > 80000 then
    raise - application _ error (-20000, 'Error checking
    Sal/Incorrect Salary');
end if;
```

The IF command test Sal. If the column's content is greater than \$8,000, an internal procedure called raise - application - error is enabled. This procedure returns an error message for the application or sub program that invoked it. Specify the code -20000, a comma, and the appropriate error message.

A new row will not be written until the restriction imposed by the trigger is met.

Compilation Errors

Many errors can occur during the creation of a trigger. Two common situations are typing errors or the use of improper commands. The following trigger, created directly in SQL * Plus, will return an error message.

```
SQL > CREATE TRIGGER TESTASAL:
2  BEFORE INSERT OR UPDATE OF SAL
3  ON SCOTT. EMP
4  FOR EACH ROW
5  BEGIN
```



```

6    IF: NEW.SAL>8000
7    RAISE _APPLICATION_ERROR (-20000, 'INCORRECT VALUE')
8    ENDIF;
9    END;
10   /

```

After executing the command, a message is displayed.

Note: Trigger created with compilation errors,. To view the error that occurred in the last command executed use the SHOW ERRORS View. Look at the following:

```

SQL > SHOW ERRORS;
Errors for TRIGGER TESTASAL;

```

Row/Col	ERROR
3/4	PLS-00103: Encountered the symbol / "RAISE_APPLICATION_ERROR" When expecting one of the following : .(*)&-+ / mod rem then an exponent (* *) and or. The symbol "*" was substituted for "RAISE_APPLICATION_ERROR" to continue.
4/4	PLS-00103 : Encountered the symbol "END" when expecting one of the following : (*)& - + / mod rem then an exponent (* *) and or. Taking another look at the code. You will note the failure to include the THEN clause in the IF Command. To correct this error, eliminate the trigger and create it again:

```
SQL > DROP TRIGGER TESTASAL
```

Trigger dropped

```
SQL> CREATE TRIGGER TESTASAL
```

```

2    BEFORE INSERT OR UPDATE OF SAL
3    ON SCOTT.EMP
4    FOR EACH ROW
5    BEGIN
6    IF:NEW.SAL>8000 THEN
7    RAISE_APPLICATION_ERROR (-20000, 'INCORRECT VALUE');
8    END IF
9    END;
10   /

```

Trigger Created.

Now its time to test our trigger to see if it is working properly. Let's try to give Smith a pay raise to \$30,000:

```
SQL > update emp
      2      Set Sal = 30000
      3      Where ename = 'SMITH';
```

Update emp

*

ERROR on row /:

ORA - 20000 : INCORRECT VALUE

ORA - 06512 : in "SCOTT.TESTASAL", rows

ORA - 04088: error during the execution of trigger

'SCOTT.TESTASAL'

Note the text of error 20000 gives the reason for the error. If you, were using a graphical interface to include or update the record, you would see a windows message box.

Now let's try a smaller pay raise.

```
SQL > UPDATE EMP
      2      SET SAL = 2000
      3      WHERE ENAME = 'SMITH';
```

1 row update

The record was successfully updated, because the trigger condition was met.

Replicating with Triggers

Triggers are an efficient mechanism to replicate data among tables. We can easily create triggers that insert, delete, or modify another table based on the operations performed in the Current table.

To create the replication trigger. We will use the AFTER INSERT mode. This mode enables the trigger after including the new record in the current table. Using data obtained through new Name of field, we insert the new record in the NEWEMP 2 table, which was created already.

We should create a table with the same structure as the EMP table, but without the Comm Column.


```

CREATE TRIGGER REP_INS
AFTER INSERT ON SCOTT.EMP
FOR EACH ROW
BEGIN

```

```

    INSERT INTO SCOTT. NEWEMP2
VALUES
(: NEW. EMPNO, :NEW.ENAME, :NEW.JOB,:NEW.MGR,
:NEW.HIREDATE,:NEW.SAL,:NEW.DEPTNO);

```

```

END;

```

Now, we insert a record in the EMP table and check the results in the NEWEMP2 table:

```

SQL > INSERT INTO EMP

```

```

2 VALUES

```

```

*

```

```

3

```

```

(8000, 'PACHARES', 'CLERK', 7782, '05/10/1999', 2000, 0, 30)

```

1 row created.

```

SQL > SELECT *FROM NEWEMP2 WHERE EMPNO = 8000;

```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	DEPTNO
8000	PALHARES	CLERK	7782	05/10/99	2000	30

Now let's create another row, but omit the value for the Hiredate Column:

```

SQL > INSERT INTO EMP

```

```

2 (EMPNO, ENMAE, JOB, MGR, SAL, DEPTNO)

```

```

3 VALUES (800/, 'TREVOR', 'ANALYST', 7782, 3000, 30);

```

1 row created

```

SQL > SELECT * FROM NEWEMP2;

```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	DEPTNO
7369	SMITH	CLERE	7902	12/17/80	800	20
7499	ALLEN	SALESMAN	7698	02/20/81	1600	30
7521	WARD	SALESMAN	7698	02/22/81	1250	30
7566	JONES	MANAGER	7839	04/02/81	2975	20
7654	MARTIN	SALESMAN	7698	09/28/81	1250	30
7698	BLAKE	MANAGER	7839	05/01/81	2850	30
7782	CLARK	MANAGER	7839	06/09/81	2450	10
7788	SCOTT	ANALYST	7566	04/19/87	3000	20
7839	KING	PRESIDENT		11/17/81	5000	10
7844	TURNER	SALESMAN	7698	09/08/81	1500	30
7876	ADAMS	CLERK	7788	05/23/81	1100	20
7900	JAMES	CLERK	7698	12/03/81	950	30
7902	FORD	ANALYST	7566	12/03/81	3000	20
7934	MILLER	CLERK	7782	01/23/82	1300	10
8000	PALCHARES	CLERK	7782	05/10/99	2000	30
8001	TREVOR	ANALYST	7782		3000	30

16 rows selected.

look this example:

```
SQL> CREATE TRIGGER REP_DEL
      2  BEFORE DELETE ON EMP
      3  FOR EACH ROW
      4  BEGIN
      5  DELETE FROM NEWEMP2
      6  WHERE EMPNO = :OLD.EMPNO;
      7  END;
      8  /
```

Trigger Created.

Now we delete the record of employee palhares from the EMP table:

```
SQL > DELETE FROM EMP WHERE EMPNO = 8000;
```

1 row deleted

Now let's check if the employee record remains in the NEWEMP2 table:

```
SQL > SELECT * FROM NEWEMP2;
```


EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	DEPTNO
7369	SMITH	CLERE	7902	12/17/80	800	20
7499	ALLEN	SALESMAN	7698	02/20/81	1600	30
7521	WARD	SALESMAN	7698	02/22/81	1250	30
7566	JONES	MANAGER	7839	04/02/81	2975	20
7654	MARTIN	SALESMAN	7698	09/28/81	1250	30
7698	BLAKE	MANAGER	7839	05/01/81	2850	30
7782	CLARK	MANAGER	7839	06/09/81	2450	10
7788	SCOTT	ANALYST	7566	04/19/87	3000	20
7839	KING	PRESIDENT		11/17/81	5000	10
7844	TURNER	SALESMAN	7698	09/08/81	1500	30
7876	ADAMS	CLERK	7788	05/23/81	1100	20
7900	JAMES	CLERK	7698	12/03/81	950	30
7902	FORD	ANALYST	7566	12/03/81	3000	20
7934	MILLER	CLERK	7782	01/23/82	1300	10
8001	TREVOR	ANALYST	7782		3000	30

15 rows selected.

Everyrthing worked fine.

Obtaining Information about a Trigger

Oracle has a view in the data dictionary, accessible only by those with the DBA privilege, that contains data from all the triggers created for the database in use. This table is called DBA - TRIGGERS. Following is a description of its contents (using system / manager to access this table):

SQL > desc dba - triggers;

NAME	NULL?	TYPE
-----	-----	-----
OWNER	NOTNULL	VARCHAR2(30)
TRIGGER-NAME	NOT NULL	VARCHAR 2 (30)
TRIGGER-TYPE		VARCHAR 2 (16)
TRIGGER - EVENT		VARCHAR 2 (26)
TABLE-OWNER	NOTNULL	VARCHAR 2 (30)
TABLE-NAME	NOTNULL	VARCHAR 2 (30)

REFERENCE - NAMES	VARCHAR 2 (8)
WHEN _CLAUSE	VARCHAR2(2000)
STATUS	VARCHAR2 (8)
DESCRIPTION	VARHCAR (2000)
TRIGGER _ BODY	LONG

To view the user's triggers, you use the view USER_TRIGGERS:

SQL>describe user _trigger;

NAME	NULL?	TYPE
-----	-----	-----
TRIGGER-NAME		VARCHAR 2 (30)
TRIGGER-TYPE		VARCHAR 2(16)
TRIGGRING-EVENT		VARCHAR 2(75)
TABLE-OWNER		VARCHAR 2(30)
BASE-OBJECT-TYPE		VARCHAR 2(16)
TABLE-NAME		VARCHAR 2(30)
COLUMN-NAME		VARCHAR 2(4000)
REFERENCING-NAMES		VARCHAR 2(128)
WHEN-CLAUSE		VARCHAR 2(4000)
STATUS		VARCHAR 2(8)
DESCRIPTION		VARCHAR 2(4000)
ACTION-TYPE		VARCHAR 2(11)
TRIGGER_BODY		LONG

SQL > Select trigger-name, trigger-body from user - triggers

where trigger -name = 'REP_INS';

TRIGGER-NAME

TRIGGER-BODY

REP-INS

BEGIN

UPDATE NEWEMP2

SET

EMPNO =:NEW.EMPNO,

ENAME =:NEW.ENAME,

JOB.

In this chapter you learned about triggers and the events that trigger them.

Unit - IX

9.1. Table Replication

Distributed database systems often locally replicate remote tables that are frequently queried by local users. By having copies of data accessed in different nodes, the distributed database does not need to always send information through the network, thus helping to maximize the performance of the database application. Data can be replicated using snapshots or replicated master tables.

Database Security

Multuser database systems such as Oracle include security resources that control how a database is accessed and used. These security mechanisms prevent unauthorized access to the database and schema objects, control the use of the disk and system resources (such as the CPU time); and audit the user's actions. Each database user has an associated schema with the same name. A schema is a logical collection of database objects (tables, views, sequences, synonyms, indexes, clusters, procedures, functions, packages and database links). As the default, each user creates and has the access to all the corresponding schema objects.

There are two types of database security: system security and data security.

System security encompasses the mechanisms that control the access to and use of the database at a certain level; system security includes:

- ▶ Valid combinations of user name and password.
- ▶ The amount of available disk space for the schema objects of a user.
- ▶ A user's resource limits.

These mechanisms check whether a user is authorized to connect to a database, if the database audit is activated, and which system operations a user can perform. Database Security encompasses the mechanisms that control the access and use of the database at the schema object level. Data security includes:

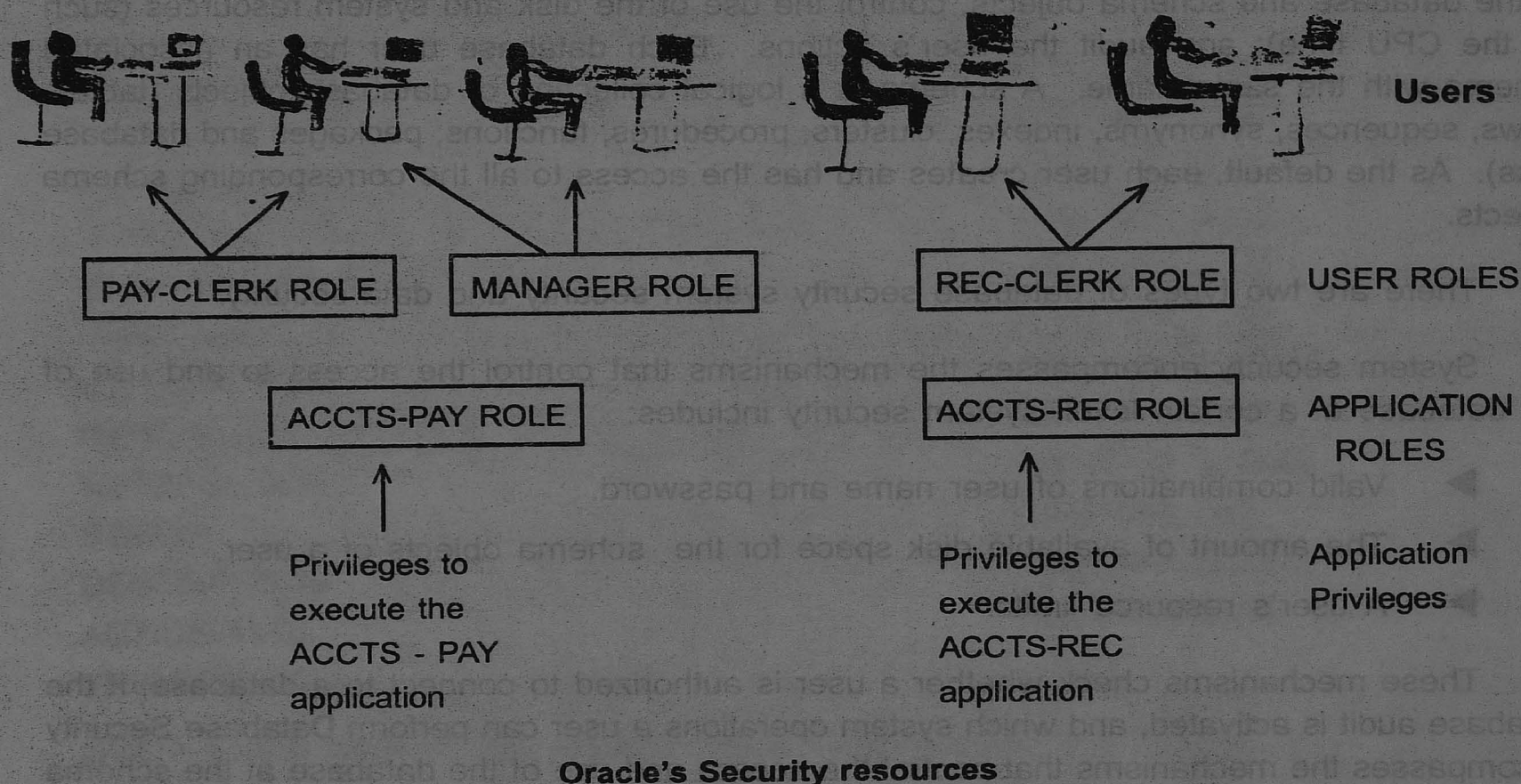
- ▶ which user has access to a specific schema object and what specific types of actions each one can perform on the schema object (for example, user Scott can issue the SELECT and INSERT declarations, but not the DELETE declarations on the EMP table).
- ▶ Auditing the actions, if they exist, for each schema object.

Security Mechanisms

Oracle provides arbitrary access control, which is a way to restrict access to privileged information. For a user to access a schema object he or she must have the appropriate privileges. Users with appropriate privileges can grant them to others, using whatever criteria they wish; for this reason, this type of security is called "arbitrary".

Oracle manages database security using several different resources. They include.

- ▶ Database users and schemes
- ▶ Privileges
- ▶ Roles
- ▶ Definitions and storage quotas
- ▶ Resource limits
- ▶ Audits



Database users and schemas

Each database in oracle has a list of user name. In order to access a database, an application of this type must be initialized and the user must be connected under a valid a user name. Each name has an associated password to avoid unauthorized use.

Security Domain

Each user has a security domain-a set of properties that determines things such as:

- ▶ Actions (privileges and roles) available for the user
- ▶ Tablespace quotas (available disk space) of the user
- ▶ System resource limits (for example, the processing time of the CPU) of the user.

Each property that contributes to the security domain of a user is discussed in the following sections.

Privileges

A privilege is a right to execute a certain type of SQL declaration. Some examples of privileges include:

- ▶ The right to connect to the database (create a session)
- ▶ The right to create a table in a schema
- ▶ The right to select rows in another person's table.
- ▶ The right to execute another person's stored procedure.

Oracle's database privileges can be divided into two distinct categories. System privileges and Schema object privileges.

System privileges

System privileges allow users to execute certain actions at the system level or in a certain type of schema object. For example, the privileges to create a tablespace or to delete rows in any database table are System privileges. Many of them are available only to the administrator and software developers.

Schema object privilege

Schema object privilege allows users to perform certain actions in a schema object that is itself specific. A privilege that can include, for example, the rows of a specific table is an object privilege. These privileges are granted to the final users so they can use the database application to perform specific tasks.

Granting privileges

Privileges are granted to users so they can change data in a database. A user can receive a privilege in two different ways:

- ▶ They can be explicitly granted to the users. For example, the privilege to insert records in the EMP table can be explicitly granted to user scott.
- ▶ They can be granted to roles (a named group of privileges), and then the roles can be granted to one or more user. For example, the privilege of inserting records in the EMP table can be granted to a role called clerk, which can then be granted to users Scott and Brian.

Since roles help make the management of privileges easier and more efficient, privileges are normally granted to roles than specific users. The following section provides details about roles and their use.

Roles

Oracle provides for the easy and controlled management of privileges by means of roles. Roles are named groups of related privileges that are granted to users and to other roles.

The following properties of roles allow for easier management of privileges.

Property	Description
Granting of limited privileges	Instead of explicitly granting the same set of privileges to many users, the database administrator can grant those privilege using a role. Then the database administrator can grant the role to each user.
Dynamic management of privileges	When the privileges of a group need to be changed, only the privileges of the role must be changed. The Security domains of all the users that have the group role are automatically reflected and the changes are made to the role.
Selective availability of privileges	The roles granted to a user can be selectively enabled (available to use) or disabled (not available). This allows the specific control of a user's privileges in certain situations.
Application awareness	A database application can be created to automatically enable or disable selected roles when a user tries to use the application.

Database administrators often create the roles for an application of this type. The DBA grants to this role all the privilege that are needed for the execution, and then grants that role to other roles or users. An application can have many different roles, or with each one having a different set of privileges.

The DBA can create of role with a password to prevent it from being used without the appropriate authorization. Generally, an application is created to activate the proper role when it is initialized. As a result, a software user does not need to know the password for an application role.

Audits

Oracle allows the selective audit (logged monitoring) of the user's actions in order to help investigate a suspect use of the database. The audit can be performed at three different levels: declaration, privilege, and schema object.

Advanced Formatting

Chapter 1 showed an example of a stock table from the newspaper. Here, you will actually manipulate that table to draw non-obvious information from it. For the sake of brevity, the stocks examined will be limited to a small number in three industries electronics, space and medical. These are the column commands in effect.

■ Column Net format 99.90

Column Industry format all

Column Company format a 18

Column close Today heading 'Close/Today' format 999.90

Column close yesterday heading 'close / year' format 999.90

Column volume format 999, 999, 999.

The first select simply retrieves the stocks in the three industries calculates the difference in closing prices between today and yesterday, and sorts the stocks in order by Industry and company, as shows Figure 14-1.

This is a good beginning, but to make it more meaningful some additional features must be added. A new column is calculated to show the percentage of change between one days trading and the next day's trading:

■ $(\text{Close Today} / \text{Close yesterday}) * 100 - 100$ As percent

The calculation is given the alias percent, and column formatting is put in place for it. Additionally, both company and Industry columns are cut back considerably to make room for additional columns, which will be added shortly of course, on a wide report, such as one with 132 columns, this may not be necessary. It is done here for space considerations.

■ Column percent heading percent / change format 9999.90

column company format a8 truncate

Column Industry heading 'Ind' format a5 truncate.

Select Industry . company

close yesterday. Close Today

$(\text{Close Today} - \text{close yesterday}) / \text{As Net}$

Volume from STOCK

Where Industry in ('ELECTRONICS', 'SPACE MEDICAL') order by Industry Company.

Industry Company	Close Yest	Close Today	Net	Volume
ELECTRONICS IDK	95.00	95.25	.25	9,443,523
ELECTRONICS MEMORY GRAPHICS	15.50	14.25	-1.25	4,557,992
ELECTRONICS MICRO TOKEN	77.00	76.50	-.50	25,205,667
MEDICAL AT SPACE	46.75	48.00	1.25	11,398,323
MEDICAL AUGUST ENTERPRISES	15.00	15.00	.00	12,221,711
MEDICAL HAYWAD ANTISEPTIC	104.00	106.00	1.75	3,358,561
MEDICAL KENTGEN BIOPHYSICS	18.25	19.50	1.25	6,636,863
SPACE BRANDON ELLIPSIS	32.75	33.50	.75	25,789,769
SPACE GENERAL ENTROPY	64.25	66.00	1.75	7,598,562
SPACE GENEVA ROCKETRY	22.75	27.25	4.25	22,533,944
SPACE NORTHERN BOREAL	26.75	28.00	1.25	1,348,323
SPACE OCKHAM SYSTEMS	21.50	22.00	.50	7,052,990
SPACE WONDER LABS	5.00	5.00	.00	2,553,712

FIGURE 14.1 Closing Stock Prices and Volumes

Break on

Next, a break on command is set up to put a black line between the end of an Industry group and the beginning of the next Industry, the sum of the daily volume by Industry is then computed. Note the co-ordination between breakon and compute sum in Figure 14.2

The break on an compute sum are now expanded as shown in Figure 14.3

The order of the columns in the break on is critical, as will be explained shortly. The compute sum has been instructed to calculate the volume on breaks of both the Industry and Report columns.

break on Industry Skip 1

Compute sum of volume on Industry

Select Industry

Company,

Close yesterday. Close Today.

(close Today/Close yesterday) As Net.

(Close Today/close yesterday) * 100 As percent,

volume

from STOCK

Where Industry in ('ELECTRONICS', SPACE, MEDICAL') Order by Industry, company.

Ind	Company	Close Yest	Close Today	Net	Percent Change	Volume
ELECT	IDK	95.00	95.00	.25	.26	9,443,523
	MEMORY G	15.50	14.25	-1.25	-8.06	4,557,992
	MICRO TO	77.00	76.50	-.50	-.65	25205,667
Sum						39,207,182
Medic	AT SPACE	46.75	48.00	1.25	2.67	11,398,323
	AUGUSTE	15.00	15.00	.00	.00	12,221,711
	HAYWARD	104.25	106.00	1.75	1.68	3,358,561
	KENTGEN	18.25	19.50	1.25	6.85	6,636,863
Sum						33,615,458
SPACE	BRANDON	32.75	33.50	.75	2.29	25,789,769
	GENERAL	64.25	66.00	1.75	2.72	7,598,562
	GENEVA,R	22.75	27.25	4.50	19.78	22,533,944
	NORTHERN	26.75	28.00	1.25	4.67	1,348,323
	OCKHAMS	21.50	22.00	.50	2.33	7,052,990
	WONDERL	5.00	5.00	.00	.00	2,553,712
Sum						66,877,300

Figure 14.2 Stock report with a break on Industry and Compute sum of volume

The volume is shown for each industry, but now a total volume for all industries has been added. The compute command has been expanded to allow you to compute the sum on Report. This must be co-ordinated with the break on:

```

break on Report on Industry Skip 1
Compute sum of volume on Industry Report
break on Report on INdustry Skip 1
Compute sum of volume on Industry Report

Select Industry
Company
Close yesterday
Close Today

```

(Close Today - Close yesterday) As Net,

(Close Today/Close yesterday)* 100 As percent

Volume

from STOCK

Where Industry in ('ELECTRONICS', SPACE', MEDICAL) order by Industry, company current portfolio.

Current Portfolio

Industry Listings

March 1st 2000

Ind	Company	Close Yest	Close Today	Net	Percent Change	Volume
ELECT	IDK	95.00	95.00	.25	.26	9,443,523
	MEMORY G	15.50	14.25	-1.25	-8.06	4,557,992
	MICRO TO	77.00	76.50	-.50	-.65	25205,667
Sum						39,207,182
Medic	AT SPACE	46.75	48.00	1.25	2.67	11,398,323
	AUGUSTE	15.00	15.00	.00	.00	12,221,711
	HAYWARD	104.25	106.00	1.75	1.68	3,358,561
	KENTGEN	18.25	19.50	1.25	6.85	6,636,863
Sum						33,615,458
SPACE	BRANDON	32.75	33.50	.75	2.29	25,789,769
	GENERAL	64.25	66.00	1.75	2.72	7,598,562
	GENEVA,R	22.75	27.25	4.50	19.78	22,533,944
	NORTHERN	26.75	28.00	1.25	4.67	1,348,323
	OCKHAMS	21.50	22.00	.50	2.33	7,052,990
	WONDERL	5.00	5.00	.00	.00	2,553,712
Sum						66,877,300

Portfoli.Sql

Figure 14.3 Stock report with break on and cumpute sum on Report for grand totals.

part 11: SQL and SQL * plus

This will produce a sum of volume for the entire report. This is the break on and compute that appear in figure 14.3. The placement of an Report in the break on command is unimportant: on Report will always be the final break.

Next, the report is given a top title and a bottom title, using the extensive formatting capabilities of title and btitle see the sidebar "title and btitle formatting commands" for an explanation of this.

title left 'current portfolio:-

right 'March 1st 2000' Skip 1-

Center 'Industry listing. skip2

btitle left 'Portfoli.sql':

Order of columns in break on.

You can get into trouble if the order of the columns in break on is incorrect. Suppose that you wish to report on revenues by company, division, department, and project (where a division has departments and departments have projects). If you input this:

break on project on Department on Division on company then the totals for each of these entities would be calculated every time the project changed, and they wouldn't be accumulated totals, only those for the project. This would be worthless. Instead, the break on must be in order from the largest grouping to the smallest, like this:

title and btitle formatting commands

The results of these title and btitle commands can be seen later in this chapter in Figure14.5

title left 'current portfolio

right x INDUSTRY Skip 1

Center 'Industry Skip 4

Left, right and center define where the string that follows is to be placed on the page. A dahs at the end of a line means another line of title commands follows, skip tells how many blank lines to print after printing this line. Text in single quotation marks is printed as is. Words not inside of single quotation marks are usually variables. If they've been defined by accept, NEW-VALUE, or define, their values will print in the title. If they have not been defined the name of the vairable will print instead.

btitle left 'port foli.sql on ' X Today -

right 'page' Format 999 Sql.pno:

Sql.pno is a variable that contains the current page number. Formatting commands can be placed anywhere in a title, and they will control formatting for any variables from there forward in the title or btitle unless another formatting command is encountered.

Adding views

In order for the report to be useful, computations of each stock in relation to its industry segment and to the whole are important. Two views are therefore created, shown next. The first summarizes stock volume, grouped by Industry. The Second summarizes total stock volume.

break on Row

SQL Plus also allows computes and breaks to be made on Row. Like on Report, the break on and Compute must be co-ordinated.

```
■ Create or replace view INDUSTRY as select Industry, Sum (volume) As volume  
from STOCK
```

```
Where Industry in ('ELECTRONICS' 'SPACE' 'MEDICAL')  
group by Industry.
```

```
■ Create or replace view MARKET as  
Select Sum (Volume) As volume  
from STOCK
```

```
Where Industry in ('ELECTRONICS', 'SPACE', 'MEDICAL')
```

This practice of creating views in a SQL Plus report is intended only for temporary views that have no particular use outside of the report. More widely used views that are shared by others and used as if they were regular tables in the database would be neither dropped nor created in SQL Plus start file that was used just for a report. They would simply be a part of the from Clause in the select statement columns used with title and btitle

Some additional column definitions are now added, along with a new column, Port of Ind (explained later in the chapter) see circle A in Figure 14.4.

Two columns that will appear in title and btitle are put in with the NEW-VALUE Command. Look at circle B in Figure 14.4 and its effect in circle in Figure 14.5. The column today was used to produce today's data in the btitle. How did this work? First of all, Today is an alias for a formatted sys Date in the select statement.

■ To_CHAR (Sys Date, 'FMMonth ddth.yyyy') As Today.

NEW_VALUE placed the contents of the Today Column (November 1st 1999 in this example) into a variable named X Today, which is then used in btitle.

btitle left 'portfoli.Sql on X today-

right 'page' format 999 Sql.Pno.

The variable could have had any name X Today was chosen to make it easy to spot in the listing, but it could even have the same name as the column Today, or something else, such as Date vor or XYZ. Portfoli.Sql is simply the name of the start file used to produce this report. It's a useful practice to print some where on the report the name of the start file used to create the report, so that if it needs to be rerun, it can be found quickly.

The last part of format 999 Sql.pno.Sql.pno. is a variable that always contains the current page number. You can use it anywhere in either ttitle or btitle. By placing it in the btitle after the word "right" it shows upon the bottom-right corner of each page.

The format 999 that precedes it designates the format for the page number. Any time a formatting command like this appears in ttitle or btitle, it defines the format of any number or characters in the variables that follow it all the way to the end of the ttitle or btitle command, unless another format command is encountered.

Column user no print

Column part of Ind heading 'Part Ind' Format 999.90

Column Today NEW-VALUE X TODAY noprint format al trunc

Column Industry NEW_VALUExINDUSTRY

ttitle left 'current portfolio' -

right x Industry Skip 1 -

Center 'Industry Listing' Skip4

btitle left 'port foli.Sql on 'X Today -

right 'page'format 999 Sql.Pno

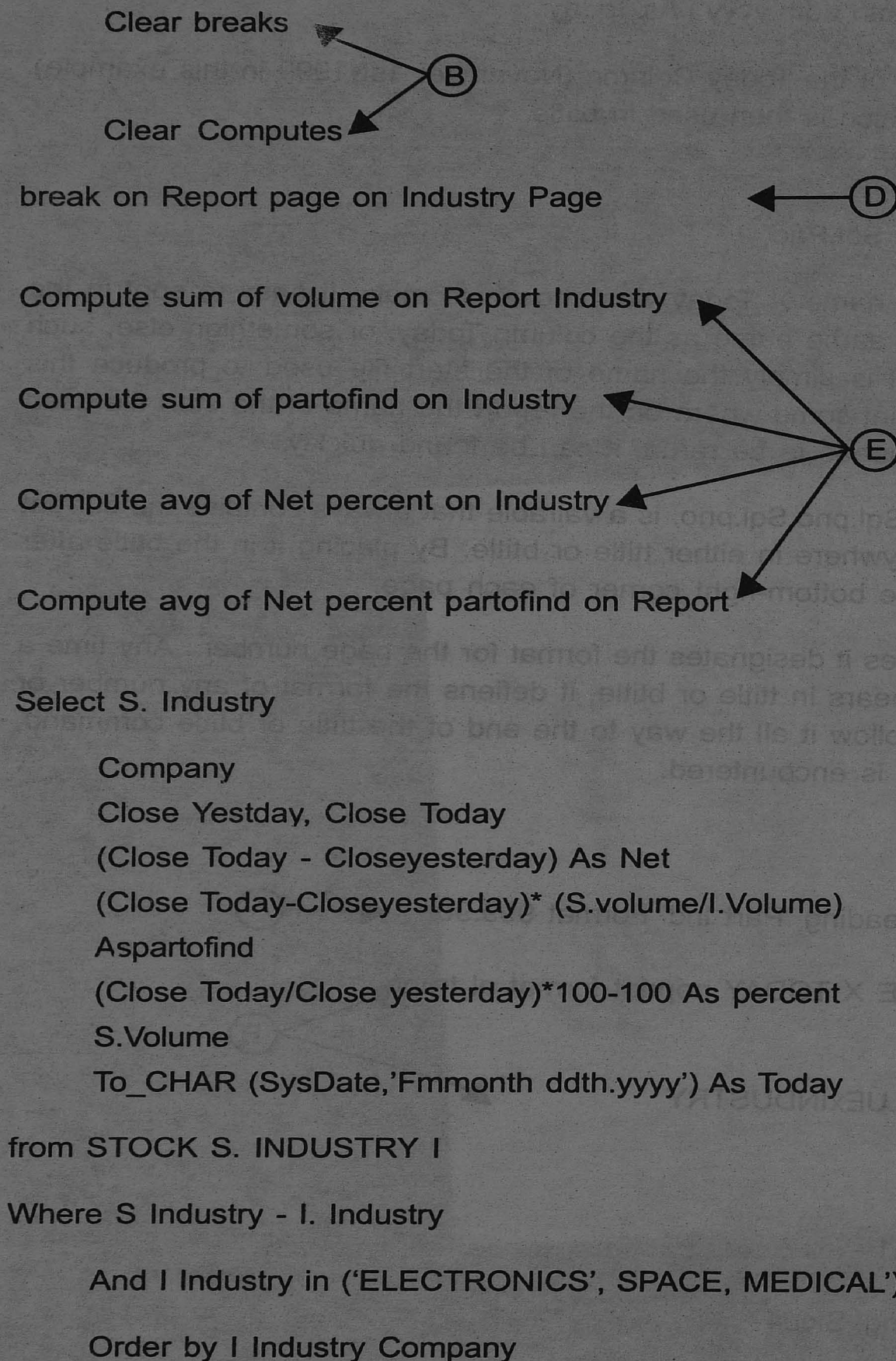


Figure 14.4 SQL Plus Commands for report by industry.

A warning about variables

There are two other items of importance in the column command:

■ Column Today NEW-VALUE x Today noprint format at trunc

Current portfolio					② → ELECTRONICS		
Industry listings							
Ind	Company	Close Yest	Close Today	Net	Part of Ind	Percent Chang	Volume
ELECT	IDK	95.00	95.25	.25	.06	.26	9443.523
	MEMORY G	15.50	14.25	-1.25	-.15	-8.06	4557992
	MICRO TO	77.00	76.50	-.50	-.32	-.65	25205667
avg				-.50		-2.82	
Sum					-.41		39.207,182
Portfoli.Sql on March 1st 2000					← ①		
page-1							

Current Portfolio					③ → MEDICAL		
Industry Listings							
Ind	Company	Close Yest	Close Today	Net	Part of Ind	Percent Chang	Volume
MEDIC	At space	46.75	48.00	1.25	.42	2.67	11.398323
	AUGUSTE	15.00	15.00	.00	.00	.00	12221.711
	HAYWARD	104.25	106.00	1.75	.17	1.68	3.358561
	KENTGEN	18.25	19.50	1.25	.25	6.85	6.636.863
avg sum				1.06	.85	2.80	33615458
Port foli Sql on March 1st 2000 ← ①					Page 2		

FIGURE 14.5 Report by industry, one industry per page

No print tells SQL Plus not to display this column when it prints the results of the SQL statement. Without it, the date would appear on every row of the report. Format a1 trunc is a bit more esteric. Dates that have been reformatted by TO-CHAR get a default width of about 100 characters even though the noprint option is in effect, SQL Plus gets confused and counts the width of the Today column when deciding whether linesize has been exceeded. The effect is to completely foul up the formatting of the rows as they are displayed or printed. By changing the format to a1 trunc, this effect is minimized.

Current Portfolio							(4) → SPACE
Industry Listings							
Ind	Company	Close Yest	Close Today	Net	Part of Ind	Percent Chang	Volume
SPACE	BRANDON	32.75	33.50	.75	.29	2.29	25789769
	GENERAL	64.25	66.00	1.75	.20	2.72	7598562
	GENEVA R	22.75	27.25	4.50	1.52	19.78	22.533944
	NORTHERN	26.75	28.00	1.25	.03	4.67	1.348323
	OCKHAMS	21.50	22.00	.50	.05	2.33	7.052990
	WONDERL	5.00	5.00	.00	.00	.00	2553.712
avg sum			(5) → 1.46		2.08	5.30	66877300
portfoli.Sql on March 1st 2000							Page 3

Current Portfolio							SPACE
Industry Listings							
Ind	Company	Close Yest	Close Today	Net	Part of Ind	Percent Chang	Volume
			(6) → .88		.19	2.66	139699940
Portfoli.Sql on March 1st 2000							page 4

FIGURE 14.5 Report by industry one industry per page (continued)

The other column with NEW-VALUE is Industry.

This column already has some column definitions in effect:

Column Industry heading 'Ind' format a5 trunc and now a new one is added as shown in the following listing (remember that different instructions for a column may be given on several lines).

Column Industry NEW-VALUE X Industry

Like Today, this column command inserts the contents of the column from the select statement into a variable called x Industry. The value in this variable is coordinated with the break on. It gets the value of Industry when the first row is read, and keeps it until a new value is encountered and the break on forces a new page as shown at circles 2, 3 and 4 in figures 14.5

Here's what SQL Plus does'.

1. Delays printing anything on a page until the break on detects a change in the value of industry or enough rows are retrieved to full the page.
2. Prints the title with the value x industry had before the value changed (or the page got full)
3. Moves the value of x Industry into an old-value variable and saves it.
4. Prints the rows on the page
5. Loads the new value into x Industry
6. Prints the title
7. begins collecting rows for the next page and goes back to the first step.

What this means is that if x industry had been placed in the title instead of the title it would contain the Industry for the following page, instead of the one being printed. MEDICAL (Circle 3) would be at the bottom of page 1 SPACE (Circle 4) would be at the bottom of page 2, and so on. To use the title properly for column values retrieved in the query you would use this.

■ Column Industry OLD-VALUE x Industry.

More on break on and compute

Circle in Figure 14.4 immediately precedes the break on and compute commands, once a compute command is in place, it continues to be active until you either clear it or leave SQLplus. This means you can have compute commands from previous reports that will execute on a current report. Producing all sorts of unwanted effects.

The break on command also persists, although any new break on command will completely replace it. It is good practice to clear breaks and clear computes just before setting up new ones.

Here are the options available for break on:

- break on column
- break on row
- break on page
- break on report

Column can be a column name an alias or an expression such as SUBSTR (Industry, 1,4). Each of these options can be followed by one of these actions.

- Skip n
- Skip page

or by nothing. break on column produces the action any time the value in the selected column changes break on row forces a break with every row. break on page forces a break every time a page is filled (in the current example, x Industry will always contain the value in the Industry column for the first row of the page with this option). break on report takes the specified action every time a report ends.

The Skip actions are to skip one or more lines (that is print them blank) or to go to the top of a new page. See circle D in Figure 14.4.

The Command compute on the other hand , can be reused for each type of computation and for one or more column at once. Circle E shows a variety of ways in which it can be used. Note how columns appear on either side of the on in the compute commands. No commas are used anywhere in either breakon or compute commands.

Here are possible computations.

Compute avg	Compute num
Compute count	Compute sum
Compute max	Compute Std
Compute min	Compute Var.

These have the same meanings for a column in a SQLPLUS report that AVGU, count (), MAX(), MIN(), STDDEU(), SUM() and VARIANCE () have in SQL

All of them except num ignore Nulls, and none of them is able to use the DISTINCT Keyword. Compute num is similar to compute count, except that compute count produces the count of non-Null rows, and compute num produces the count all rows.

Displaying current breaks and computes Entering just the word break or compute in SQLPLUS will cause it to display the break and computes it has in effect at the moment.

Running several computes at once

Figure14-4 Contains the following compute statements:

- Compute sum of volume on Report Industry
- Compute sum of partofind on Industry
- Compute avg of Net percent on Industry
- Compute avg of Net percent partofind on Report

Look at circles 5 and 6 in figure 14-5 and you'll see the effects of all of these. Observe that the avg calculation for each column appears on one line, and the sum (where there is one) appears on the following line. Further, the word sum and avg appear under the column that follows the word on in the compute statement.

They're missing at circle 6 for the grand totals because these are made on Report, and Report is not a column. So sum and avg can't appear under a column.

The volume sum at circle 5 is just for the space industry stocks. The volume sum at circle 6 is for all industries (page 4 of this report contains only the grand totals and averages). Note that the first compute here is for the sum of one column, volume, on both Report and Industry (Incidentally, the order of the column in a compute is irrelevant, unlike in break on).

The next compute for partofind on Industry, requires some explanation (partofind refers to its part of the industry point shift or change in stock price over all the stocks in the industry).

partofind comes from a portion of the select statement.

■ (Close Today - Close yesterday_ * S.volume / I.volume

As partofind

This calculation weights the net change in a stock versus the other stocks in its Industry based on the volume traded. Compare the actual Net change between BRANDON and NORTHERN in the SPACE industry. BRANDON changed only .75 but traded over 25 million Shares NORTHERN changed 1.25 but traded only about 1 million shares. The contribution of BRANDON to the upward shift in the space industry is thus considerably greater than that of NORTHERN; this is reflected in the relative values for the two under the partofind column.

Now compare the sum of the partofind column, 2.08 with the avg of the Net Column, 1.46 (from the very next compute). which is more representative of the change in stock price in this industry? It is the sum of partofind because its values are weighted by stock volume. This example is given to show the difference between a simple average in a column and a weighted average, and to show how summary information is calculated. In one case, It is done by averaging, in another it's done by summing.

This is not the place to launch a detailed discussion of weighted averages, percentages, or statistical calculations, but it is appropriate to point out that significant differences will result from various methods of calculation. This will often affect decisions that are being made so care and caution are in order.

The final compute calls for the average of Net, percent and partofind on Report. These values are shown at circle 6 as .88, .19, and 2.66 respectively. Look at the last line of figure 14.6. This is the same report as that in figure 14.5, with a few exceptions. It is on one page instead of four; its upper-right title is the date, not the industry segment: and its bottom line has an additional and different result for averages and totals:

Market Averages and Totals 1.09, 2.66, 139699940

These are the correct results, and they differ from those that are or can be calculated from the columns displayed using any compute statement, because the compute statements lack the weighting of the total industry volume so has was this result produced? The answer is in figure 14.7. At Circle F, the btitle was turned off and at circle G (just following the select that produced the main body of the report) is an additional select that includes the Label Market Averages and Totals, and a proper calculation of them.

This select statement is preceded by set heading off and title off. These are necessary because a new select will normally force a new toptitle to print. The set heading off command is used to turn off the column titles that normally would appear above this line. The btitle at circle F had to be turned off before the first select executed, because the completion of the first select would cause the btitle to print before the second select could even execute.

As shown in the sample report outputs in this Chapter (figures 14.2, 14.3, 14.5 and 14.6) when a compute command is used, the computed value is labeled with the function that was executed. For example, in figure 14-6, both an AVG and a sum function are computed; the output shows a label of "avg" for the AVG calculation, and "sum" for the sum calculation. You can override the default labels and specify your own Labels for computed functions in the compute command. See the Label clause of COMPUTE in the Alphabetical Reference for details. To avoid having the compute commands apply to the volume column of the second query the clear computes command is executed after the first query.

Current Portfolio

March 1st 2000

Industry Listings

Ind	Company	Close Yest	Close Today	Net	Part of Ind	Percent Chang	Volume
ELECT	IDK	95.00	95.00	.25	.06	.26	9443523
	MEMORY G	15.50	14.25	-1.25	-.15	-8.06	4557992
	MICRO TO	77.00	76.00	-.50	-.32	-.65	25205667
avg				-.50		-2.82	
Sum					-.41		39207182
MEDIC	AT SPACE	46.75	48.00	1.25	.42	2.67	11398323
	AUGUST E	15.00	15.00	.00	.00	.00	12221711
	HAYWARD	104.25	106.00	1.75	.17	1.68	3358.561
	KENTGEN	18.25	19.50	1.25	.25	6.85	6636863
avg				1.06		2.80	
Sum					.85		33615458

SPACE	BRANDON	32.75	33.50	.75	.29	2.29	25789769
	GENERAL	65.25	66.00	1.75	.20	2.72	7598562
	GENEVA R	22.75	27.25	4.50	1.52	19.78	22533944
	NORTHERN	26.75	28.00	1.25	.03	4.67	1348323
	OCKHAM s	21.50	22.00	.50	.05	2.33	7.052990
	WONDER L	5.00	5.00	.00	.00	.00	2553.712
AVG				1.46		5.30	
Sum					2.08		66877300
				.88	.19	2.66	
Market Averages and Totals				1.09			139699940
							2.66139699940

Figure 14.6 Revised report with correct market averages and totals

Set termout off and set termout on

Another useful pair of commands is set termout off and set termout on. The former is often used in a start file just before the spool command, and the latter is often

tttitle left 'current portfolio' right X today skip 1-

Center 'Industry Listings' Skip 2:

bttitle off



Clear breaks

Clear computes

break on Report on Industry skip 1

Compute sum of volume on Report Industry

Compute sum of partofind on Industry

Compute avg of Net percent on Industry

Compute avg of Net percent partofind on Report

Select S. Industry

Company

Close yesterday, Close Today,

(Close Today - Close yesterday) As net,

(Close Today - Close yesterday)* S. volume / I.Volume)

As partofind

(Close Today/ Close yesterday) * 100-100 As percent,

S. volume, user,

To_CHAR (Sysdate, 'FmMonth ddth.yyyy) As today

from Stock S. Industry I

Where S. Industry= I. Industry

And I Industry in ('ELECTRONICS', 'SPACE', 'MEDICAL') order by I Industry Company.

Set heading off

Ttitle off

Clear computes

Select 'Market Averages and Totals ← (G)

Sum(((Close Today - Close yesterday) * S. Volume) / M. Volume) As Net

AVg ((Close Today / Close yesterday))*

*100-100 As percent

Sum(S. volume) As volume

from STOCK S. MARKET M

Where Industry in ('ELECTRONICS, SPACE, 'MEDICAL)

FIGURE 14.7 How the correct market averages and totals were produced.

Used just after it. The effect is to suppress the display of the report to the screen for reports that are going to be printed, this saves time (they 'll run faster) and avoids the annoying rolling of data across the screen, the spooling to a file continues to work properly.

Variables in SQL Plus

If you're using SQL Plus interactively you can check on the current ttitle and btitle at any time by typing their names alone on a line. SQL immediately showsyou their contents

■ ttitle

ttitle ON and is the following 113 characters:

left 'Current portfolio right x Today

Skip 1 center 'Industry Listings 'Skip 2

Note the presence of x TODAY, which is a variable rather than its current contents. SQL Plus is capable of storing many variables - those used in ttitle and btitle are only some of them. The current variables and their values can be discovered by typing define:

■ define

DEFINE _SQLPlus_RELEASE = * 801050000* (CHAR)

DEFINE - EDITOR = *VI*(CHAR)

DEFINE - O-VERSION = *oraclesi Enterprise

Edition Release 8.1.5.0.0

with the partitioning and Java options

PL/SQL Release 8.1.5.0.0	- Production * (CHAR)
DEFINE - 0 - RELEASE	= *801050000* (CHAR)
DEFINE x INDUSTRY	= *SPACE* (CHAR)
DEFINE x TODAY	= * March 1st, 2000* (CHAR)

The EDITOR variable is the editor you use when you type the word edit. You saw how to set this up in your login. The other variables identify which version and release of Oracle you are using.

The last variables are defined in the stock market queries, and their contents are just those displayed in the reports. SQL Plus stores variables for title and btitle by defining them as equal to some value, and then allows you to use them whenever a query is being executed. In fact, variables can be used many places in a report other than in titles.

Suppose your stock data base were updated automatically by a pricing service and you wished to check regularly the closing prices and volumes on a stock-by-stock basis. This could be easily accomplished with variables in the where clause. Normally, you would type this query:

```
SQL> Select Company. Close yesterday; Close Today,
      Volume
      From Stock where Company = 'IDK';
```

Company	Close Yest	Close Today	Volume
IDK	95.00	95.25	9.443.523

Alternatively, you could type this into a start file named, for example, closing.sql:

```
SQL> Column close yesterday heading 'Close/Today,' format 999.90
SQL> Column close yesterday heading 'close / yest,' format 999.90
SQL> Column volume format 999.999.999
SQL> accept x Company prompt 'Enter company name:
SQL> Select Company, close yesterday, Close Today, Volume from STOCK
SQL> where company = '& x Company';
```

In this file x Company is a variable you have invented. Such as x Industry or x Today. accept tells SQL Plus to accept input from the keyboard and prompt tells it to display a message. Note that the variable name must be preceded by the ampersand (&) when it appears in the SQL.

select statement, but not when it appears in the accept or in a title. Then, when you type this:

■ Start closing .Sql

the screen displays this:

■ Enter company name:

you then type MEMORY GRAPHICS.SQLPLUS

will display the following:

■ Select company, close yesterday, close Today, volume

from Stock

Where company = '& x Company':

old 3: Where company = '& x company'

new 3: where company = 'MEMORY GRAPHICS'

Company	Close Yest	Close Today	Volume

MEMORY GRAPHICS	15.50	14.25	4557.992

First, it shows you the query you have setup. Next it shows you the where clause - first with the variable in it, and then with the value you typed. Finally, it shows you the results of the query.

If you then typed start closing. Sql again, but typed IDK for company name, the old and new would show this:

■ old 3: where company = 'MEMORY GRAPHICS'

new 3: where company = 'IDK'

and the result would be for IDK. It is unnecessary for the select and the old and new to be displayed each time. Both of these can be controlled the first by the set echo command and the second by set verify to see what these are set at currently, type the show command followed by the keyword verify or echo:

■ Show verify

Verify ON

Show echo

echo ON

The revised version of the start file looks like this:

Column close Today heading 'close / Today' Format 999.90 column close yester heading 'close /yest,' format 999.90 column volume format 999.999.999

```
set echo off
```

```
set verify off
```

```
set sqlcase upper
```

```
accept x company prompt 'Enter company name:'
```

```
Select company closeyesterday, close Today, volume from Stock
```

```
where company = '& x company'.
```

Set Sqlcase upper tells SQLPlus to convert anything keyed into the variable (using the accept) to uppercase before executing the query. This can be helpful if you've stored your data in uppercase (which is a good practice) but don't want to force people to type in uppercase any time they run a query. Now when the start file is executed, this is what happens:

```
Start Closing . Sql
```

```
Enter Company name: memory graphics
```

Company	Close Yest	Close Today	Volume
-----	-----	-----	-----
MEMORY	15.50	14.25	4.557.992

The use of variables in start files can be very helpful, particularly for reports in which the basic format of the report stays the same but certain parameters change, such as date, company division, stock, name project, client and so on. When the report starts, it asks the person using it for these details and then it turns. As just shown, typing only the word define will result in a list of all the variables currently defined typing define with just one name will show only that variable's contents:

```
define x company
```

```
DEFINE x company = "memory graphics" (CHAR)
```

After a start file has completed, any variables are help by SQL plus until you exit or intentionally undefine them:

```
undefine x company
```

An attempt now to see the variables value produces the following

```
define x Company
```

```
Symbol x company is UNDEFINED
```

you also can define a variable within the start file without using the accept command. Simply assign a value directly as shown here:

```
define x company = 'IDK'
```

Any place that & x company appears in the start file will have IDK substituted.

Other places to use variables.

Any variable that you define using either accept or define can be used directly in a btitle or ttitle command without using NEW-VALUE column Command, NEW-VALUE simply takes the contents of a column and issues its own define command for the variable name following NEW-VALUE. The single difference in using the variable in titles, as opposed to the SQL statement, is that in title the variable is not preceded by an ampersand.

Variables also can be used in a setup start file, as explained later in this chapter in "using mak. Sql".

9.2 Numeric Formatting

The default method SQL Plus used for formatting numbers is to right-justify them in a column, without commas, using decimal points only if the number is not an integer. The table NUMBERTEST contains columns named value 1 and 2, which have identical numbers in each column. These will be used to show how numeric formatting works. The first query shows the default formatting.

```
select value1, value 2 from NUMBERTEST:
```

VALUE 1	VALUE 2
0	0
0001	0001
1234	1234
1234.5	1234.5
1234.56	1234.56
1234.567	1234.567
98761235	98761235

Row five is Null. Notice how the decimal point moves from row to row. Just as column can be formatted individually in queries, the default format can be changed:

```
set numformat 9.999.999
```


Select value1, Value2 from NUMBERTEST:

0	0
0	0
1,234	1,234
1235	1235
1235	1235
1235	1235
#####	#####

Now, row eight is filled with pound signs. The problem with row eight is that the format defined is too narrow. You can fix this by adding another digit on the left as shown here.

■ Set numformat 99.999.9999

Select value1, Value2 from NUMBERTEST:

Value 1	Value2
0	0
0	0
1234	1234
1235	1235
1235	1235
1235	1235
98,761,235	98.761.235

Using Mask. Sql

Column used in one report may also need to be used regularly in several reports. Instead of retyping the formatting commands for these columns in every start file, it can be useful to keep all the basic column commands in a single file. This file might be called mask. sql. because it contains formatting (also called masking) information for columns for example, the file might look like the following.

■ REM File mask. Sql

Set numerdth 12

Set numformat 999,999.999.90

Column Net format 99.90

Column Industry format all

Column company format a18

Column close Today heading 'close / Today' format 999.90

Column Close yesterday heading 'Close / yet', format 999.90

column volume format 999,999,999.

define x Department = 'Systems planning Dept 3404'.

This is then effectively embedded in a start file (usually near the top) simply by including this line:

■ Start mask. Sql:

Numeric formatting options

The options work with both set numformat and the column format command

9999990	The count of nines and zeros determines the maximum digits that can be displayed
999,999,999,99	Commas and decimals will be placed in the pattern shown
999990	Displays a zero if the value is zero
099999	Displays numbers with leading zeros.
\$99999	A dollar sign is placed in front of every number.
B99999	The displays will be black if the value is zero. This is the default
99999M1	If the number is negative a minus sign follows the number. The default is for the negative sign to be on left.
99999PR	Negative numbers are displayed within < and >.
9.999EEEE	The display will be in scientific notation.
999V99	Multiplies number by 10 ⁿ , where n is the number of digits to the right of V. 999v99 turns 1234 into 123400.

Show all and spooling

You've seen several commands that use the set command, and whose current status can be determined using the show command. Such as feedback, echo, verify heading, and so on. There are actually about 50 such commands that can be set. These can all be displayed at once using this:

Show all

Unfortunately, these fly by so quickly on the screen that it's impossible to read most of them. You can solve this problem by spooling. Simply spool to a file, as in the previous example, execute show all, and then spool off. You can now look at the current status of all the set commands using your editor on the file. The rows returned by the show all command will be ordered alphabetically.

As an alternative, you can use the store command to save your current SQL Plus environment settings to a file (like mask.Sql). You can then execute this file at a later date to reset your environment settings.

Folding onto New Lines

Information retrieved from a database is often just fine with only one line of column titles, and columns of data stacked below them. Occasionally, however, a different kind of layout is preferable. Sometimes, this can be accomplished using literal columns of nothing but blank spaces, to properly position real data on more than one line and have it line up properly. These literal columns are given an alias in the SQL and then a blank heading in the column command. The technique parallels what was done for the total line in Figure 14.6 for example, look at this:

```

Column Industry format a14 truncate
column B format a21 heading
column company format a20
Column Volume format 999,999,999. Justify left
Select Industry, 'ASB',
      Company
      Close Yesterday, Close Today
      (Close Today-Close yesterday) As Net,
      Volume
from Stock
Where Industry in ('ADVERTISING', 'GARMENT')
order by Industry Company;
```

INDUSTRY	COMPANY	
CLOSE YESTERDAY	CLOSE TODAY	NET VOLUME
ADVERTISING	AD SPECIALTY	
31.75	31.75	0 18,333 876
ADVERTISING	MBK COMMUNICATIONS	
43.25	41	-2.25 10,022.980
ADVERTISING	NANCY LEE FEATURES	
13.5	14.25	.75 14,222.692
GARMENT	ROBERT JAMES APPAREL	
23.25	24	.75 19,032,481

A literal column of one space, given an alias of B, is defined by the column command as 21 characters wide with a blank heading. This is used specifically to move the company names over so that the columns line up as desired, with stock volume directly below the company name and column B for close Today and Net lining up with the blank.

fold - after and fold - before

Next, look at how the column command fold - after affects every column in the select:

Clear columns

Column Net format 99.90

Column A format a15 fold-after 1

Column Company format a20 fold-after1

Column close Today heading 'Close / Today' format 999.90-

fold - after

Column close yesterday heading 'Close / Yest.' format 999.90-

fold-after 1

Column Net fold - after 1

Column volume format 999,999.999 fold - after1

set heading off

Select Industry 11', 'AS A,

Company,

Close yesterday

Close Today, (Close Today - Close yesterday) As net,

Volume

From Stock

where Industry in ('ADVERTISING', 'GARMENT')

order by Industry, company:

The previous query produces the following output. Note that even though close Today had no number following its fold-after clause, SQLPLUS used a default of 1.

ADVERTISING

AD SPECIALTY

31.75

31.75

.00

18.333.876

ADVERTISING,

MBK COMMUNICATIONS

43.25

41.00

-2.25

10.022,980

ADVERTISING,

NANCY LEE FEATURES

13.50

14.25

.75

14.222.692

GARMENT,

ROBERT JAMES APPAREL

23.25

24.00

.75

19.032.481

Additional Reporting Controls

Many of the commands illustrated here, as well as as in other chapters, have options considerably beyond those used in these examples.

Unit - X

The concept of users is important for data base administrators to understand. Any one who wants to access oracle must be registered as a user of the database and must have privileges assigned to him or her outlining the tasks he or she will be able to execute. Roles are groups of privileges that can be assigned to users. These concepts are thoroughly explained in this chapter.

Users : one of the most important tasks of an administrator is controlling access to the database. Oracle relies on a mechanism that allows you to register a person called "user". Each registered user has an access password, which must be provided in various situations. Each user is then assigned individual privileges or roles.

10.1: The CREATE USER COMMAND

The CREATE user Command is responsible for the creation of new users. It creates a user or an account through which you can connect to the database, and establish the means by which oracle will allow the user access you can assign the following optional properties to the user.

- ▶ Standard tablespace
- ▶ Temporary tablespace
- ▶ Quotas to allocate space in the tablespaces
- ▶ Profile containing resource limits

Syntax

```
CREATE USER user IDENTIFIED {By password | EXTERNALLY | GLOBALLY AS 'CN = user'}
```

```
[DEFAULT TABLESPACE tablespace
```

```
| TEMPORARY TABLESPACE tablespace
```

```
| QUOTA { integer [K / M]
```

```
| UNLIMITED } ON tablespace
```

```
[QUOTA { integer [K / M]
```

```
| UNLIMITED } ON tablespace] . . .
```

```
| PROFILE profile
```

```
| PASSWORD EXPIRE
```

```
| ACCOUNT { LOCK / UNLOCK } . . .
```


Arguments

User

The name of the user to be created

IDENTIFIED

Indicates how oracle validates the user :

By Password

Requires the user to specify this password for the connection.

EXTERNALLY

Indicates that a user must be validated by an external service (such as an operating system service or even third parties).

GLOBALLY AS external name

Indicates that a user must be authenticated by oracle security service External name is the name X - 509 in oracle security service, which identifies this user

DEFAULT TABLESPACE

Identifies the standard table space for the objects created by the user. If this clause is omitted, the objects are sent to the system tablespace as default.

TEMPORARY TABLESPACE

Identifies the tablespace for the user's temporary segments. If this clause is omitted, they are sent to the system tablespace as default.

QUOTA

Allows the user to allocate space in the tablespace optionally establishing a quota of bytes integer. This quota is the maximum space that can be allocated in the tablespace by the user you can also use K or M to specify the quota in K bytes or M bytes. Note that a CREATE USER command can have several QUOTA clauses for different tablespace.

UNLIMITED

Allows the user to allocate unlimited space in the tablespace.

PROFILE

Reassigns the named profile to the user, and limits the database resources that can be utilized by the user. If this clause is omitted, oracle assigns the default profile to the user.

PASSWORD EXPIRE

Makes the user password expire. It must be changed before a new connection to the database can be completed.

ACCOUNT LOCK

Blocks the user account, disabling its access.

ACCOUNT UNLOCK

Unblocks the user account, enabling its access.

Here is an example (do not execute this command) :

```
CREATE USER james
```

```
IDENTIFIED BY welcome
```

```
DEFAULT TABLESPACE sports
```

```
QUOTA 10 M ON Sports
```

```
QUOTA 5 M ON Temp - ts
```

```
QUOTA 5 M ON System
```

```
PROFILE boxer
```

User James has the password welcome and uses the sports tablespace with 10 m bytes of space (Specified by QUOTA). There are 5 m bytes each in the temp - ts and system tablespaces. The limits for the resources of the user database are defined in the profile file called Boxer.

Creating a new user

To assign privileges, you must connect to the administrator using the system name and the manager keyword. To connect to the database as a new user, you must use the CONNECT command with the following format :

```
CONNECT user / password
```

```
CONNECT system / manager
```

Now let's use the for CREATE USER command both the name and password can be up to 30 characters long, without spaces or commas. If you enter a user name that has already been registered, the system will issue an error message and will not allow you to register the new user.

When the user is created, you have to assign him or her privileges. First, let's create a user called cindy with the password ramal ho :

SQL > CREATE USER Cindy IDENTIFIED By ramal ho, user created.

Before assigning the privilege, we will try to connect as Cindy:

SQL > connect cindy / ramal ho ;

ORA - 01045 : the user cindy needs the CREATE SESSION privilege ; logon denied.

There is an error in the above example because the user doesn't have any privileges assigned yet. Let's go back and connect to the system using the GRANT command.

10.2 : The GRANT command

The GRANT command has two distinct forms : one to distribute system privileges and the other to distribute object privileges (these are discussed later in this chapter) only the DBA can use the GRANT command to distribute system privileges.

Syntax

GRANT System privilege / role To user / role / public

WITH ADMIN OPTION

Arguments

Privilege

The name of the privilege to be assigned

user / role

The name of the user or role that is receiving the privilege.

WITH ADMIN OPTION

Allows a user or role that receives the privilege to assign it to other users, change it or even delete it. Let's connect as system and assign the RESOURCE privilege, which allows access to the database and the creation of tables, sequences, procedures, triggers, indexes and clusters :

SQL > connect system / manager ;

Connected.

SQL > grant RESOURCE to cindy ;

Grant operation successful.

From now on the user can perform any operation pertaining to the role received, such as, in the following example, the creation of a table :

```
SQL > connect cindy / ramal ho ;
```

Connected.

```
SQL > create table X
```

```
2 (a number) ;
```

Table created.

The next example shows, through SQL * Plus, the importance of understanding the concept of user, role, and privilege. After configuring SQL * Plus, we use the CONNECT command to make a new connection as user cindy. Note that the name was entered in correctly, and even with the correct password oracle denied access :

```
SQL > connect cindy / ramal ho ;
```

ERROR :

ORA - 01017 : invalid name of user / password ; logon denied. Now, we will connect correctly and access Scott's table :

```
SQL > connect
```

Enter the user name : Cindy

Enter the password : * * * * *

connected

```
SQL > Select * from emp ;
```

```
Select * from emp
```

*

ERROR on row 1 :

ORA - 00942 : the table or view does not exist.

This occurs because the EMP table, which was created by scott, did not assign any access privileges to there users. Therefore, each user is restricted to his or her own activities inside the database.

Deleting a User

You can remove a database user with the DROP USER command. This command removes both the user and all the objects contained in this user's schema you will need to specify the CASCADE clause of the command. Oracle also removes all the referential integrity associated to the objects of the removed user.

Syntax

```
DROP USER name <CASCADE>
```


In the next example user Arnold is removed :

SQL > connect system / manager

connected.

SQL > drop user arnold ;

users eliminated.

SQL > select * from all - users ;

USER NAME	USER - ID	CREATED
SYS	0	03/01/99
SYSTEM	5	03/01/99
OUTLN	11	03/01/99
DBSNMP	20	03/01/99
AURORA \$ ORB \$ UNAUTHENTICATED	25	03/01/99
MTSSYS	28	03/01/99
SCOTT	20	03/01/99
DEMO	26	03/01/99
ORDSYS	27	03/01/99
ORDPLUGINS	30	03/01/99
MDSYS	31	03/01/99
CTXSYS	35	03/01/99
CINDY	37	05/07/99

13 rows selected .

Privileges

A privilege is an authorization given to the user to access and manipulate a database object in a certain way. For example, one user can be assigned the privilege to select tables but not change them, while another user can read, update, and even change the structure of tables and other objects. As all database manipulating is done through SQL commands, a privilege grants to a user the right to use these commands.

There are two types of privileges : system privileges and object privileges.

System privileges

A system privilege is the right or permission to execute certain database actions in a specific type of database object. There are more than 70 types of privileges associated to the actions of a database. The name of the privilege is similar to the name of the action it executes.

For example, a privilege called ALTER TABLE grants a user the right to modify a table. The following is a list of the system privileges :

Action	Privilege
Analyze	ANALYZE ANY
Audit	AUDIT ANY AUDIT SYSTEM
Cluster	CREATE CLUSTER CREATE ANY CLUSTER ALTER ANY CLUSTER DROP ANY CLUSTER
Database	ALTER DATABASE
Database link	CREATE DATABASE LINK
Index	CREATE ANY INDEX ALTER ANY INDEX DROP ANY INDEX
Privilege	GRANT ANY PRIVILEGE
Procedure	CREATE PROCEDURE ALTER ANY PROCEDURE DROP ANY PROCEDURE EXECUTE ANY PROCEDURE
Profile	CREATE PROFILE ALTER PROFILE DROP PROFILE ALTER RESOURCE COST
Public database link	CREATE PUBLIC DATABASE LINK DROP PUBLIC DATABASE LINK
Public Synonym	CREATE PUBLIC SYNONYM DROP PUBLIC SYNONYM
Role	CREATE ROLE ALTER ANY ROLE DROP ANY ROLE GRANT ANY ROLE
Rollback segment	CREATE ROLLBACK SEGMENT

Action	Privilege
Sequence	ALTER ROLLBACK SEGMENT
	DROP ROLLBACK SEGMENT
	CREATE SEQUENCE
	CREATE ANY SEQUENCE
	ALTER ANY SEQUENCE
Session	DROP ANY SEQUENCE
	SELECT ANY SEQUENCE
	CREATE SESSION
	ALTER SESSION
	RESTRICT SESSION
Snapshot	CREATE SNAPSHOT
	CREATE ANY SNAPSHOT
	ALTER ANY SNAPSHOT
	DROP ANY SNAPSHOT
Materialized Views	ALTER MATERIALIZED VIEW
	CREATE MATERIALIZED VIEW
	DROP MATERIALIZED VIEW
Synonym	CREATE SYNONYM
	CREATE ANY SYNONYM
	DROP ANY SYNONYM
System	ALTER SYSTEM
Table	CREATE TABLE
	CREATE ANY TABLE
	ALTER ANY TABLE
	BACK UP ANY TABLE
	DROP ANY TABLE
	LOCK ANY TABLE
	COMMENT ANY TABLE
	SELECT ANY TABLE
	INSERT ANY TABLE
	UPDATE ANY TABLE

Action	Privilege
Tablespace	DELETE ANY TABLE
	CREATE TABLESPACE
	ALTER TABLESPACE
	MANAGE TABLESPACE
	DROP TABLESPACE
Transaction	UNLIMITED TABLESPACE
	FORCE TRANSACTION
	FORCE ANY TRANSACTION
Trigger	CREATE TRIGGER
	CREATE ANY TRIGGER
	ALTER ANY TRIGGER
	DROP ANY TRIGGER
User	CREATE USER
	BECOME USER
	ALTER USER
	DROP USER
View	CREATE VIEW
	CREATE ANY VIEW
	DROP ANY VIEW

Object privileges

An object privilege is the right to perform certain actions in a specific object, such as the right to include a row in a table. These privileges do not apply to all the objects of a database. Triggers, procedures, indexes, database links, and cluster do not have any object privileges. The following is a list of the object privileges.

Action	Privilege
ALTER	Tables and sequences
DELETE	Tables and views
EXECUTE	Procedures
INDEX	Tables
INSERT	Tables and views
REFERENCE	Tables
SELECT	Tables, view and sequences
UPDATE	Tables and views

When a user creates an object, such as a table, only the user can view it. The user / owner must grant a privilege or a role to access the table. A privilege can be granted with the GRANT command.

Assigning object privileges to a User / Role

A variation of the GRANT command allows you to assign object privileges to a user or role. The privileges assigned can be : SELECT, ALTER, DELETE, EXECUTE, INSERT, INDEX, REFERENCE, and UPDATE. The following objects can be assigned privileges : tables, views, sequences, snapshots, and synonyms. Any user who has authority can grant privileges on tables.

Syntax

GRANT ALL

SELECT

INSERT

DELETE

INDEX

ALTER

UPDATE [(column - name)] on table - name / view - name

To user / PUBLIC

WITH ADMIN OPTION.

Options

SELECT

To select data in a table or view

INSERT

To insert rows in a table or view

DELETE

To eliminate rows in a table or view

UPDATE

To update a table and, optionally, update only the specified columns

INDEX

To create or eliminate the indexes of a table

ALTER

To modify a table

ALL

To perform all the above privileges

Table - name

Names of existing tables (including the qualifier) in the database

View - name

Names of existing view (including the qualifier) in the database

Column - name

This is optional and determines the columns of tables or views specified in the ON clause. The names of the columns must not be qualified

USER

Name of the user to whom the privilege is assigned

PUBLIC

Means that all the current and new users have the privileges specified for the table or view

WITH ADMIN OPTION

Allows the user or role that receives the privilege to grant it to other users, modify it, or even delete it. The next example shows user scott granting the SELECT privilege so user cindy can access the EMP table :

```
SQL > connect scott / tiger ;
```

connected

```
SQL > grant select on emp to cindy ;
```

GRANT operation successful.

Now we connect to cindy and access the table :

```
SQL > select ename from emp ;
```

```
select ename from emp
```

*

ERROR on row 1 :

ORA - 00942 : table or view does not exist.

An error occurs what is the problem, since the privilege was already assigned? When a user makes a SELECT in another user's table, he or she must provide the schema of the table or object in other words, he or she must identify the table, owner, and even the database link, if needed. If the following example, you need to provide the SCOTT.EMP schema to access the EMP table :

```
SQL > select ename from scott emp ;
```

```
ENAME
```

```
-----
```

```
SMITH
```

```
ALLEN
```

```
WARD
```

```
JONES
```

If you try to access another one of scott's tables using this schema, but without assigning an object privilege for the user, you will have the same problem again connected as scott, we will assign to cindy the UPDATE privilege only for the DNAME column of the DEPT table :

```
SQL > grant update (DNAME) on dept to cindy ;
```

Then, connected as cindy, we will update two columns of the DEPT table ;

```
SQL > update scott dept
```

```
2      set dname = 'SALES FORCE'
```

```
3      where dept no = 30 ;
```

```
1 row updated.
```

```
SQL > update scott dept
```

```
2      set 10 c = 'MIAMI'
```

```
3      where dept no = 30 ;
```

```
Update scott dept
```

```
*
```

```
ERROR ON row 1 :
```

```
ORA - 01031 : insufficient privileges.
```

Note that it was not possible to update the column with out the proper privileges. In the previous example, user cindy was not assigned the upDATE privilege for all of the columns. Now we will try to insert a new row in the SCOTT . DEPT table :

```
SQL > insert into scott . dept values (50, 'MEDIA,' 'MIAMI')
```

```
insert in to scott . dept (50; MEDIA', 'MIAMI')
```

```
*  
ERROR or row 1 ;
```

ORA - 01031 : in sufficient privileges

Note that cindy is prevented from updating the table because she has not assigned the proper privileges.

Viewing users and privileges

Connected as scott, we will try to create a new user. SQL > create user arnold identified by terminator ;

```
create user arnold identified by terminator  
*
```

```
ERROR on row 1 ;  
ORA - 01031 ; insufficient privileges
```

Remember that high - level privileges are necessary to create a user. Connected as system / manager, we will create this user and view a list of all other users you can use one of the tables of the data dictionary to check the database users. The dictionary view that maintains this list is ALL_USERS.

```
SQL > connect system / manager ;
```

connected

```
SQL > create user arnold identified by terminator ;
```

user created.

```
SQL > select * from ALL_USERS ;
```

USERNAME	USER_ID	CREATED
SYS	0	03/01/99
SYSTEM	5	03/01/99
OUTLN	11	03/01/99
DBSNMP	20	03/01/99
MTSSYS	28	03/01/99
AURORA \$ ORB \$ UNAUTHENTICATED	25	03/01/99
SCOTT	26	03/01/99

DEMO	27	03/01/99
ORDSYS	30	03/01/99
ORDPLUGINS	31	03/01/99
MDSYS	32	03/01/99
ARNOLD	38	05/17/99
CTXSYS	35	03/01/99
CINDY	37	05/17/99

Revoking a system privilege

In addition to granting a privilege, you can also revoke it, by using the REVOKE command. To be revoked, a system privilege must have been granted with the ADMIN OPTION option.

```
SQL > revoke create session from arnold ;
```

Revoke successful.

To check the functionality of the command, connect as Arnold ;

```
SQL > connect arnold / terminator
```

ERROR ;

ORA - 01045 ; The user Arnold needs the privilege CREATE SESSION ; logon denied

Notice : you are not connected to oracle any more.

```
SQL >.
```

Oracle checked Arnold's privileges and noted he did not have the CREATE SESSION privilege any more thus preventing the connection to the database.

Revoking an object privilege

Now we show you how to revoke an object privilege that was granted to a user. In an example we have seen before user scott granted to cindy the SELECT privilege for the DEPT table. Now we will revoke this privilege using a SQL command to do that, you must connect as a user who has the authority to grant privileges. A user cannot grant or revoke privileges to himself or herself. In the following example, we have to connect as user scott. Then, we use the revoke command, specifying the privilege the object, and the user that will be revoked ;

```
SQL > connect scott / tiger
```

connected

```
SQL > revoke select on dept from cindy ;
```

Revoke successful.

To verify that the revoke operation worked, we connect as cindy and try to access the SCOTT.DEPT table :

```
SQL > connect cindy / ramal ho
```

connected

```
SQL > select * from scott.dept ;
```

```
select * from scott.dept
```

```
*
```

ERROR on row 1 ;

ORA - 01031 : insufficient privileges

Because the privilege has been revoked, cindy cannot access the table any more.

While cindy cannot view the DEPT table anymore, she still can change the contents of the Dname column :

```
SQL > update scott.dept
```

```
2      set dname = 'SALES'
```

```
3      where dept no = 30 ;
```

10.3 : Roles

A role is a group of privileges assigned to a name. Instead of granting eight privileges to a user, you can create a role that is assigned those eight privileges, and then assign that role to the user. The use of roles simplifies the administration of users.

Oracle 8i has several predefined roles, such as CONNECT, RESOURCE, DBA, EXP_FULL_DATABASE, and IMP_FULL_DATABASE. In addition to these roles, users can create other ones that meet the needs of their applications. Next, we will review Oracle's users, passwords, and roles :

User	Password	Role
SCOTT	TIGER	CONNECT and RESOURCE
SYSTEM	MANAGER	DBA
SYS	SYSI	CONNECT, RESOURCE and DBA EXP_FULL_DATABASE and IMP_FULL_DATABASE
DEMO	DEMO	CONNECT and RESOURCE.

If you are using Windows 95, the password of user sys is the serial number without the hyphens. This number can be obtained by clicking on the system icon on the control panel.

It appears after the identification of the user in a format similar to this : "2 877 - 411 - 1357921 - 04810". User sys is the most advanced of all. Therefore, use it only when you really need it, since it can execute some actions that may compromise the entire database and its contents.

Below are oracle 8i's predefined roles :

Role	Purpose
CONNECT	Allows access to the database
RESOURCE	Allows access to the database, and the creation of tables, sequences, procedures triggers indexes, and clusters.
DBA	All the system privileges. It allows granting the privilege to other users.
DELETE_CATALOG_ROLE	
EXECUTE_CATALOG_ROLE	Roles used to export views and packages of the data dictionary.
SELECT_CATALOG_ROLE	
EXP_FULL_DATABASE	
IMP_FULL_DATABASE	Roles used to export and import the database.
AQ_USER_ROLE	
AQ_ADMINISTRATOR_ROLE	Roles used for oracle's advanced querying.
SNMPAGENT	Role used by the Enterprise Manager or by the intelligent Agent.
RECOVERY_CATALOG_OWNER	Creates a user that has a recovery catalogs.
HS_ADMIN_ROLE	This role is used by a DBA who uses the heterogeneous services of oracle to access specific tables of the data dictionary.

To create a role, the user must have the CREATE ROLE Privilege. When created, a role can be assigned the option ADMIN OPTION, which allows the user to grant it to other roles or users, revoke the granted role, change the access to it, and remove it.

Creating a Role

The creation of a new role follows the same rules that apply to the creation of other objects we have seen so far when you create a new role, you can assign it some privileges or other roles that were previously created. The SQL command used to create a role is CREATE ROLE :

Syntax

CREATE ROLE role [NOT IDENTIFIED

| IDENTIFIED {By password | EXTERNALLY | GLOBALLY}]

Arguments

role

Name of the role to be created, Oracle recommends the role contain at least one single-byte character, independent of the database's set of characters, which may or may not contain multiple-byte characters.

NOT IDENTIFIED

Indicates that this role is authorized by the database and that no password is required to enable it.

IDENTIFIED

Indicates that the user must be authorized by the specified method before the role can be enabled with the SET ROLE command :

By password

The user must specify the password for Oracle to enable the role. It can contain only single-byte characters from the database's set of characters, regardless of whether this set of characters contains multiple-byte characters.

EXTERNALLY

Indicates that the user must be authorized by an external service (such as an operating system or a third-party service) before the role can be enabled. Depending on the operating system, the user must specify password before the role can be enabled.

GLOBALLY

Indicates that the user has authorization from Oracle security service to use the role before it can be enabled with the SET ROLE command or at the login.

If both the NOT IDENTIFIED and the IDENTIFIED options are omitted, the standard role will be NOT IDENTIFIED.

In the following example, we connect as system / manager to create a role called Basic 1 :

```
SQL > CREATE ROLE basic 1 ;
```

role created

Granting privileges and Roles to a Role

After creating a role, you can grant it privileges using the GRANT command. The privileges can be specific to that role or from other roles. If a role is granted a role, the newly created role assumes the privileges of those roles it may receive.

Syntax

GRANT role / name of privilege To user / role / PUBLIC WITH ADMIN OPTION

The next example grants the RESOURCE role to Basic 1 :

SQL > grant resource to basic 1 ;

Grant operation successful.

Granting a Role to a user

Now we assign the Basic 1 role to user Arnold :

SQL > grant basic 1 to arnold ;

Grant operation successful.

Next we will connect as the new user :

SQL > connect arnold / terminator

ERROR :

ORA - 01045 : the user Arnold needs the privilege CREATE SESSION ; logon denied

Notice : you are not connected to oracle any more.



TIP : While the RESOURCE role allows many operations in the schema objects you need the CREATE SESSION privilege (part of the CONNECT role) to connect to the database. To solve this problem, we will connect as system / manager and assign CONNECT to Basic 1.

SQL > Connect system / manager

connected.

SQL > grant connect to basic 1 ;

Grant operation successful

Next let's try to connect :

SQL > connect arnold / terminator ;

connected.

Viewing the Roles of a User

Oracle has some special views that control the privileges and roles one of these views is called USER_ROLE_PRIVS. It displays the roles assigned to the current user :

```
SQL > select * from user_role_privs ;
```

USERNAME	GRANTED_ROLE	ADM	DEF	OS
-----	-----	---	---	---
ARNOLD	BASIC 1	NO	YES	NO

```
SQL >
```

```
1 row updated,
```

Deleting a Role

The command responsible for the deletion of a role is DROP ROLE. The user must have the system privilege DROP ANY ROLE to remove a role or the role must have been created with the ADMIN OPTION option.

Syntax

```
DROP ROLE role
```

The next example removes the Basic 1 role oracle automatically updates all the grants, along with updating roles and users that depend on the role that is being removed. After removing the Basic 1 role that was granted by user Arnold, we try to connect to this user ; oracle does not allow the access, since the user does not have the access privilege any more :

```
SQL > connect system / manager
```

```
connected.
```

```
SQL > drop role basic 1 ;
```

```
Role eliminated
```

When we try to connect as user Arnold, an error will occur, because he has no associated privilege or role :

```
SQL > connect arnold / terminator
```

```
ERROR :
```

```
ORA - 01045 : the user Arnold needs the privilege CREATE SESSION ; logon denied
```

```
Notice : you are not connected to oracle anymore.
```


10.4 : Indexes

Oracle uses the concept of indexes. These are similar to indexes found in a book, where you can quickly find the pages containing a specific topic by locating the index, finding the topic, and turning to the designed page.

In the case of Oracle, the index is an optional structure associated to tables and clusters that allows a faster execution of the SQL commands.

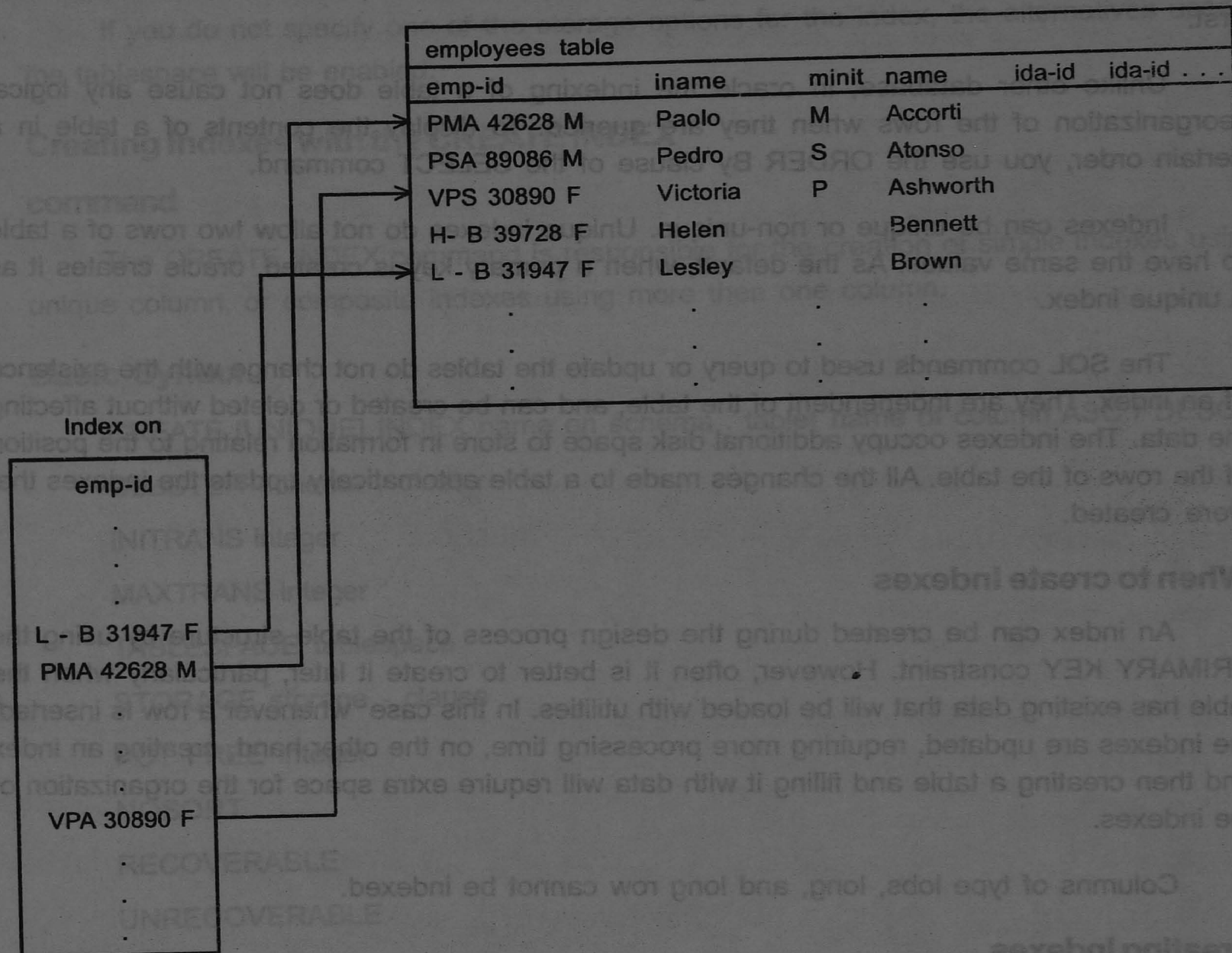


Figure 10.1 An index speeds row access in a table.

Although they do speed up access to a table's rows, indexes are not appropriate for all tables. Some situations must be considered before creating an index. The presence of data in a column generally indicates that an index must be created for it. However, columns that have many rows with null values are not good candidates for indexing. The same is true for those columns containing many identical values. Also, tables with a small number of rows should not be indexed. Columns that are used as primary keys, on the other hand, should be indexed. When creating a table through Navigator, the column that is indicated as having a primary key is automatically indexed. Those columns used as FOREIGN KEY also should be indexed.

A table can contain an unlimited number of indexes, but the more indexes there are, the more effort will be needed for management and updating. This can result in a performance hit, particularly in the update or insert operations, where there is an addition to creating and writing a new row, the database also has to update all the indexes. An index can be created using the contents of one or more columns of the database. When more than one column is used the index is called composite. When defining the index, the most important column should come first.

Unlike other databases, in Oracle the indexing of a table does not cause any logical reorganization of the rows when they are queried. To display the contents of a table in a certain order, you use the ORDER BY clause of the SELECT command.

Indexes can be unique or non-unique. Unique indexes do not allow two rows of a table to have the same values. As the default, when a primary key is created, Oracle creates it as a unique index.

The SQL commands used to query or update the tables do not change with the existence of an index. They are independent of the table, and can be created or deleted without affecting the data. The indexes occupy additional disk space to store information relating to the position of the rows of the table. All the changes made to a table automatically update the indexes that were created.

When to create Indexes

An index can be created during the design process of the table structure by using the PRIMARY KEY constraint. However, often it is better to create it later, particularly when the table has existing data that will be loaded with utilities. In this case, whenever a row is inserted, the indexes are updated, requiring more processing time. On the other hand, creating an index and then creating a table and filling it with data will require extra space for the organization of the indexes.

Columns of type LOB, LONG, and LONG RAW cannot be indexed.

Creating Indexes

There are several ways to create indexes. The following discussion gives details on each.

Creating Indexes with Constraints

When creating a constraint using UNIQUE or PRIMARY KEY, Oracle automatically creates an index if the constraint is enabled. No other command is necessary to create the index.

Through the PRIMARY KEY or UNIQUE constraint it is possible to create an index if the ENABLE and USING INDEX options are used, as in the following example :


```
CREATE TABLE emp ( emp NUMBER(5) PRIMARY KEY, . . . )
```

```
empno PRIMARY KEY USING INDEX
```

```
TABLESPACE users
```

```
PCTFREE 0 ;
```

If you do not specify one of the storage options for the index, the alternatives used for the tablespace will be enabled.

Creating Indexes with the CREATE INDEX

command

The CREATE INDEX command is responsible for the creation of simple indexes using a unique column, or composite indexes using more than one column.

Basic Syntax

```
CREATE [UNIQUE] INDEX name on schema . table( name of column ASC / DESC, ...)
```

```
CLUSTER schema . cluster
```

```
INITTRANS integer
```

```
MAXTRANS integer
```

```
TABLESPACE tablespace
```

```
STORAGE storage - clause
```

```
PCT FREE integer
```

```
NOSORT
```

```
RECOVERABLE
```

```
UNRECOVERABLE
```

Arguments

CLUSTER

Indicates that the name of the key is indexed to a cluster.

TABLESPACE

Indicates the name of the tablespace for the creation of the index, using a different tablespace for the index improves the performance since there will be fewer disk access conflicts.

PCTFREE

When an index is created, oracle reserves disk space, for the data blocks that are filled with their keys. PCTFREE indicates the space that should be left free and unused by the indexes.

INITTRANS and MAXTRANS

Indicate the initial value and maximum space allocated for the transactions of the data blocks is the indexed segment.

STORAGE

Specifies the parameters stored by the index. Now we will use some commands to create and handle an index. First, we will check the contents and the order of the rows of the DEPARTMENT tables of user Demo. To do that, we must be connected as this user :

```
SQL > CONNECT DEMO / DEMO
```

connected.

After the connection, we will take a look at this user's tables :

```
SQL > SELECT TABLE _ NAME FROM USER _ TABLES ;
```

```
TABLE _ NAME
```

```
-----
```

```
CUSTOMER
```

```
DEPARTMENT
```

```
EMPLOYEE
```

```
ITEM
```

```
JOB
```

```
LOCATION
```

```
PRICE
```

```
PRODUCT
```

```
SALARY _ GRADE
```

```
SALES_ORDER
```

Now we will take a close look at the DEPARTMENT Table :

```
SQL > select * from department
```

2/

DEPARTMENT_ID	NAME	LOCATION_ID
10	ACCOUNTING	122
20	RESEARCH	124
30	SALES	123
40	OPERATIONS	167
12	RESEARCH	122
13	SALES	122
14	OPERATIONS	122
23	SALES	124
24	OPERATIONS	124
34	OPERATIONS	123
43	SALES	167

Let's create an index called Depname based on the Name column of the DEPARTMENT table.

```
SQL > . create index depname on department (name) ;
```

index created.

After creating the index, a new query to the table will not show a physical order. However, the new index will be updated with each inclusion or change made to the database.

You cannot create an index that uses only one column if it was already used by another index. However, you can create a composite index that includes a column that was already used by another index

Consider the next example

```
SQL > create index indname on department (name) ;
```

```
create index idname on department (name)
```

ERROR on row 1;

ORA - 01408 : this list of columns is already indexed. Notice that you can use the name column in the creation of a composite index, even though you have already used that column is a previous index.

```
SQL > create index name loc on department (name, location - id) ;
```

Index created.

Note that the existence of these indexes does not change the execution of other SQL commands related to the table. Here is another example, where specific storage spaces are allocated for the index. First, we will connect as user scott :

```
SQL > connect scott / tiger
```

```
connected.
```

```
SQL >
```

```
SQL >
```

```
SQL > CREATE INDEX ind_ename on newemp 2 (ename)
```

```
2 TABLESPACE users
```

```
3 STORAGE (INITIAL 20K
```

```
4 NEXT 20 K
```

```
5 PCTINCREASE 75)
```

```
6 PCTFREE 0 ;
```

Creating a Function - Based index

Function - based indexes improve the queries that use a value returned by a function or expression, since they store this value by expression / function of the index itself. One advantage of using this type of index is the ability to include sorts based on keys containing words with accents or disregarding the size of the letter in the field. For example, when a database's contents include words in a language with accents, the use of national language support (NLS) helps to create a properly sorted index through the NLSSORT function.

The next example shows the creation of an index with all the names converted to uppercase, using the UPPER function :

```
CREATE INDEX X1 ON newemp 2 (UPPER (ename)) ;
```

Even when there are names in lowercase or combining lowercase and uppercase, the following SELECT command returns data from all the entries that include "smith" ;

```
SELECT * FROM newemp 2 WHERE UPPER (ename) = 'SMITH' ;
```

Now we will create an index based on the German language. It does not apply to the example tables we are using :

```
CREATE INDEX n/s - Index on t - table (NLSSORT (name, 'NLS_SORT = 'German'  
)) ;
```

Creating an index using key compression

Oracle allows the creation of an index using key compression, a technique that helps to eliminate the duplicated values of a key column. Key compression breaks the key into two parts : a prefix and a suffix, once created, the prefix is shared by all the entries of an index block. For example, if a name field contains 1,000 records of people, whose names start with "Maria" the index creates the prefix Maria, which is shared by all 1000 records. Besides saving space and improving performance, the use of compressed indexes is indicated when you have indexes formed by several columns.

To create a compressed index, just add the COMPRESS clause, as shown in the following example :

```
CREATE INDEX emp_test (ename)
```

```
COMPRESS
```

Key compression can be disabled with the ALTER INDEX command and the NOCOMPRESS clause :

```
ALTER INDEX emp_ename REBUILD NOCOMPRESS ;
```

Changing an Index

Changing an index is restricted to modifying the transaction and storage parameters. It is not possible to change the columns in the index. In this situation, the index must be deleted and re - created with a new definition you can change the parameters of any index, including those created by oracle for the PRIMARY KEY and UNIQUE KEY constraints, by using the ALTER INDEX command. The following command changes the index :

```
SQL > alter index pk_emp 3
```

```
2    INITRANS 5
```

```
3    MAXTRANS 10
```

```
4    STORAGE (PCT INCREASE 50) ;
```

When you change the transactions inserted in the INITRANS and MAXTRANS parameters of an index, the new INITRANS parameter applies just to the data blocks allocated while the new MAXTRANS parameter applies to all the blocks.

(current or subsequent) of an index

The storage of the INITIAL and MINEXTENTS parameters cannot be changed. All the new parameters for the storage of other parameters affect only the extents allocated to the index.

Re - Creating an Index

An Index can be re - created by using the ALTER INDEX command. This operation is the equivalent of deleting and reconstructing the index, with the advantage of reorganizing the fragmentation of a block and providing better performance. To re - create an existing index, use the following format :

```
ALTER INDEX index - name REBUILD ;
```

The REBUILD option should be specified right after the name of the index when the ALTER INDEX command includes other options :

During the creation of an index, oracle applies an access block to the DML commannds. However, you can re - create the index online, i.e without applying a block to the table. This operation is necessary in systems that are constantly running. For this type of reconstruction, you must add the ONLINE option to the ALTER INDEX or CREATE INDEX command :

SQL > CREATE INDEX indname on dept 1 (dname, loc) ONLINE ;

Index created.

SQL > alter index indname rebuild online ;

Index changed.

Obtaining Information about Indexes

With the USER_INDEXES view of the data dictionary, we can obtain information about the index of each user,

+ oracle SQL * PLUS

File Edit Search Options Help

SQL >

SQL > describe user_indexes ;

Name	Null?	Type
INDEX_NAME	NOT NULL	VARCHAR 2 (30)
INDEX_TYPE VARCHAR 2 (27)		
TABLE_OWNER	NOT NULL	VARCHAR 2 (30)
TABLE_NAME	NOT NULL	VARCHAR 2 (30)
TABLE_TYPE		VARCHAR 2 (11)
UNIQUENESS		VARCHAR 2 (9)
COMPPESSION		VARCHAR 2 (8)
PERFIX_LENGTH		NUMBER
TABLESPACE_NAME		VARCHAR 2 (30)
INI_TRANS		NUMBER
MAX_TRANS		NUMBER
INITIAL_EXTENT		NUMBER
NEXT_EXTENT		NUMBER
MIN_EXTENT		NUMBER
MAX_INCREASE		NUMBER
PCT_THERE SHOLD		NUMBER
PCT_COLUMN		NUMBER
INCLUDE_COLUMN		NUMBER
FREE LISTS		NUMBER
FREE LIST_GROUPS		NUMBER

PCT_FREE	VARCHAR 2 (3)
LOGGING	NUMBER
BLENEL	NUMBER
LEAF_BLOCKS	NUMBER
DISTINCT_KEYS	NUMBER
AVG_LEAF_BLOCKS_PER_KEY	NUMBER
AVG_DATA_BLOCKS_PER_KEY	NUMBER
CLUSTERING_FACTOR	VOROHAR 2 (8)
STATUS	NUMBER
NUM ROWS	
SOMOIC SIZE	NUMBER
#	

Figure 10.2 obtairing information about a user's indexes.

Following is a query to this table, where the indexes we have just changed appear (pk_emp3) :

SQL select index_name, table_name, uniqueness, ini_trans, max_trans from USER_indexes ;

INDEX_NAME	TABLE_NAME	UNIQUENESS	INI_TRANS	MAX_TRANS
INDNAME	DEPTI	NONUNIQUE	2	255
IND_ENAME	NEWEMP2	NONUNIQUE	2	255
PK_DEPT	DEPT	UNIQUE	2	255
PK_EMP	EMP	UNIQUE	2	555
PK_EMP2	TESTRACONSTR	UNIQUE	2	255
PK_EMP3	DEMOCONSTR 2	UNIQUE	5	10
PK_EMP4	DEMOCONSTR 3	UNIQUE	2	255
PK_NEWA	NEWAGAIN	UNIQUE	2	255
SYS_C001275	DEPTI	UNIQUE	2	255

Eliminating an Index

An index should be kept active only when it is being used effectively, keeping an index that is not used frequently reduces the performance of the system and occupies unnecessary disk space. Simply creating an index for a column does not mean that there will be production gains, since the performance is affected by several factors. If you test and see that there are no performance gains with an index, simply remove it. From time to time, an index can become fragmented. In this case, we recommend that you delete and re - create it. The SQL command responsible for the elimination of an index is DROP INDEX.

Syntax

DROP INDEX name

Example :

DROP INDEX depemp

To eliminate an index, it must be long to your schema. It is not possible to exclude only the index associated to a UNIQUE KEY or PRIMARY KEY constraint. In this case, you must eliminate or disable the constraint itself :

SQL > drop index pk_emp2 ;

drop index pk_emp2

*

ERROR on row 1 :

ORA - 02429 : it is not possible to eliminate the index used to create the Unique / Primary key

SQL > drop index indname ;

Index deleted.

Anbagam Printers, Madurai - 7

Copies - 2000/ 2007 - 08