

स्वाध्याय

स्वमन्थन

स्वावलम्बन

UTTAR PRADESH RAJARSHI TANDON OPEN UNIVERSITY
(Established vide U.P. Govt. Act No. 10, of 1999)



Indira Gandhi National Open University



UP Rajarshi Tandon Open University

BCA-12
RDBMS LAB

FIRST BLOCK : RDBMS Design

SECOND BLOCK : Practical on RDBMS

**THIRD BLOCK : RDBMS Lab Introduction to MS
Access**

Shantipuram (Sector-F), Phaphamau, Allahabad - 211013



Uttar Pradesh
Rajarshi Tandon Open University

BCA-12 RDBMS LAB

Block

1

RDBMS DESIGN

UNIT 1

RDBMS TERMINOLOGY 5

UNIT 2

OVERVIEW OF LOGICAL DATABASE DESIGN 36

UNIT 3

OVERVIEW OF NORMALISATION 57

/

COURSE INTRODUCTION

Relational database management system has permeated every segment of commercial world. They are required as the backbone of any information system. Presently almost all popular industrial database management system supports Relational Model although many extensions of this model have been in existence. Some of these extensions include advanced features such as object-oriented databases support for graphical objects, very large objects etc.

This course is an attempt to provide you with the basic information about relational database management system development. This course is a lab course but the attempt in this course is not to develop a static lab manual, such manuals already exists as the help manuals of respective RDBMS. The attempt here is to practically visualize the concepts of RDBMS and apply them to design and implement RDBMS in a given problem situation.

The course is divided into three blocks.

Block 1 covers the basic concepts of RDBMS, and various design issues relating to RDBMS with the help of several examples.

Block 2 is an attempt at providing various problem cases and their solutions. This block is not in the conventional format.

Block 3 provides details of an RDBMS package. Please note whenever, you want to shift from one RDBMS package to other you just need to go through the changed reference manual. You must carry the concepts given in block 1 and block 2, as they will be required for any relational database management system application development.

Finally, RDBMS technologies are growing at a very fast pace. You must keep up pace with it, especially in the client-server levels of databases. The logical design which has been discussed here will also help you in design of server site database application. The client site development require various application development tools which may be available with the DBMS such as report writers, form designers, ER diagramming tools etc. or it may be using a particular programming environment such as Visual Basic.

BLOCK INTRODUCTION

This being the first block of the course, an attempt has been made to define and consolidate concepts with the help of examples. The important concepts, which one must be able to describe, have been discussed in the block.

Please note that this block is the backbone for your practical implementations, therefore, must be given maximum attention.

One must be clear about these basic concepts in order to use a lot of functions and facilities, which does exist in an RDBMS. This block also visits the concepts relating to normalisation and various principles relating to it, they will be extremely useful from the viewpoint of table design.

The block is divided into three units.

Unit 1 defines the basic concepts relating to RDBMS. One of the main concepts defined in the block is of keys, Integrity and constraints. It also defines several terms relating to RDBMS and their logical implications.

Unit 2 focuses on the logical design of RDBMS. The key model covered in this unit is ER model, which is primary model for logical database design in the industry.

Unit 3 is on the principles of normalisation and tries to present various theories of normalisation for proper table/relation design.

In this block the process of DBMS design has been discussed only at conceptual schema level. However, the design of a DBMS is not complete at the design of relations. It should be further done at following two levels:

- **External Schema:** We must design various views for the types of users and their access rights.
- **Internal Schema:** We must predict a size of a relation and predict the type of hardware/software requirements. In addition, the fine-tuning of DBMS based on the expected types of queries by creating indexes as desired in an application.

However, these designs can be best explored only with experiences and company policies in real situations.

FURTHER READINGS

1. C.J. Date, *An Introduction to Database Systems*, Sixth Edition, Addison Wesley.
2. Silberschatz, Korth and Sudarshan, *Database System Concepts*, Third Edition, McGraw Hill Publications.
3. Bipin C. Desai, *An Introduction to Database Systems*, Asian Students Edition, Galgotia Publications.

UNIT 1 RDBMS TERMINOLOGY

Structure

- 1.0 Introduction
- 1.1 Objectives
- 1.2 Some Definitions
 - 1.2.1 Database
 - 1.2.2 Database-management System
 - 1.2.3 Instances and Schemas
 - 1.2.4 Traditional File Oriented Approach
 - 1.2.5 Benefits of Conventional or centralised DBMS
 - 1.2.6 Data Independence
 - 1.2.7 Data Dictionary
 - 1.2.8 Database Security
- 1.3 The Definitions related to Relational Model
 - 1.3.1 Domain Definition
 - 1.3.2 A Relation
- 1.4 Relational Data Integrity
 - 1.4.1 Candidate Keys
 - 1.4.2 Primary Key
 - 1.4.3 Foreign Keys
 - 1.4.4 Referential Integrity
 - 1.4.5 Candidate Keys and Nulls
- 1.5 Data Dictionary Checklist
- 1.6 Summary
- 1.7 Model Answers

1.0 INTRODUCTION

A database system is essentially a sophisticated, computerized record keeping system, a repository for a collection of computerized data files. A database system maintains information and makes that information available on demand, for this purpose a database system provides set of facilities to perform such operations. For example, adding new empty files in database, inserting new data in existing files, retrieve data, update data, delete data, query the database etc. Thus we can say that a database system stores data and provides information related to data. Although some people use "data" and "information" synonymously, but some prefer to distinguish between two. Data refers to the values actually stored in the database and information refers to the meaning of those values as understood by user.

The benefits of database system over any traditional file system are very obvious as data in database (at least in large system), is integrated as well as shared, thus, a database eliminates redundancy and unnecessary repetition of data, and also as a consequence, database lets multiple users access the same piece of data.

The most important advantage of the database is to maintain the integrity i.e., it ensures that changes made to the database by the authorized users do not result in a loss of data consistency and guard against accidental damage to the database.

In this unit, we present a brief introduction to the principles of database system. We also compare the pros and cons of database system and file oriented system. As we know that the emphasis of this course is very much on the relational approach, thus, this unit covers the theoretical foundation of that approach— namely, the relational model. Please note that most of our examples are using a general syntax, which is close to SQL, but is not SQL, and at most places self-explanatory.

1.1 OBJECTIVES

At the end of this unit, you should be able to:

- describe database and database management system;
- compare database and file oriented approach;
- describe various features of database like data consistency and security;
- describe relational model; and
- describe relational data integrity.

1.2 SOME DEFINITIONS

You will be revisiting these definitions as they are most commonly referred in any reference notes on RDBMS.

1.2.1 Database

The collection of data, usually referred to as database, contains information about one particular enterprise. It is a collection of related information stored so that it is available to many users for different purpose. It is a kind of electronic file, which has a large number of ways of cross-referencing checks so that the user can reorganise and retrieve data in the way he wants.

1.2.2 Database-management System

A database-management system (DBMS) consists of a collection of interrelated data and a set of programs to access those data. The primary goal of the DBMS is to provide an environment that is both convenient and efficient to use in retrieving and storing database information. Thus we can say that, a database management system is a combination of hardware and software that can be used to set up and monitor a database, and can manage the updating and retrieval of database that has been stored in it:

Most DBMS have the following facilities/capabilities:

- Creating of a file, addition of data, deletion of data, modification of data; creation, addition and deletion of entire files.
- Retrieving data collectively or selectively.
- The data stored can be sorted or indexed at the user's discretion and direction.
- Various reports can be produced from the system. These may be either standardised reports or that may be specifically generated according to specific user definition.
- Mathematical functions can be performed and the data stored in the database can be manipulated with functions to perform the desired calculations.

- To maintain data integrity and database use.
- Data integrity for multiple users.
- Providing form based interface for easy accessibility and data entry.

The DBMS interprets and processes user's requests to retrieve information from a database. From the following figure we can depict that DBMS serves as an interface in several forms.

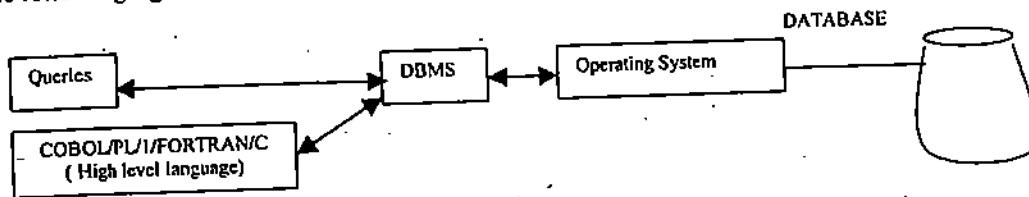


Figure 1: Database Management System

1.2.3 Instances and Schemas

Databases change over time as information is inserted and deleted. The collection of information stored in the database at a particular moment is called an instance of the database. The overall design of the database is called the database schema. If schemas change at all, they change very infrequently.

A database schema corresponds to the programming-language type definition. A variable of a given type has a particular values at a given instant. Thus, the values of a variable in programming languages corresponds to an instance of a database schema.

Database systems have general schemas, partitioned according to the levels of abstraction. At the lowest level is the physical schema, at the intermediate level is the logical schema, and at the highest level is a subschema. In general, database systems support one physical schema, one logical schema and several subschemas.

1.2.4 Traditional File Oriented Approach

The traditional file-oriented approach to information processing for each application has a separate master file and its own set of personal files. An organisation needs flow of information across these applications also, and this requires sharing of data, which is significantly lacking in the traditional approach. One major limitation of such a file-based approach is that the programs become dependent on the files and the files become dependent upon the programs.

Although such file-based approaches which came into being with the first commercial applications of computers did provide an increased efficiency in the data processing compared to earlier manual paper record-based systems as the demand for efficiency and speed increased, the computer-based simple file oriented approach to information processing started suffering from the following significant disadvantages.

Data redundancy and inconsistency: Different files may have different formats and the programs may be written in different programming languages as they are developed by different programmers. Moreover, the same information may be duplicated in several places (files). For example, the address and telephone number of a particular customer may appear in a file that consists of saving-account records and in a file that consists of checking-account records. This redundancy leads to higher storage and access cost. It may lead to data inconsistency; that is, the various copies of the same data may no longer agree.

Difficulty in accessing data: Suppose we need to access information about all the customers of a particular scheme. During the initial stages of development of the system this kind of query might not have been known, so no application program would be on hand to meet it. Say we have the application program that generates list of all the customers along with the scheme names. Thus we will have to run the latter program and sort the customers of particular scheme manually, or for each query we will have to write a new application program. Clearly we can see that accessing data is not easy in these cases.

Data isolation: Since data is scattered in various files, which might be in different formats thus, it is difficult to write a new application to retrieve appropriate data.

Integrity Problems: The data values stored in the database must satisfy certain types of consistency constraints. For example, the balance of a bank account may never fall below a prescribed amount (say, Rs. 500). Developers enforce these constraints in the system through hard-coding these conditions. When new constraints are added, it is difficult to change the program to enforce them.

Atomicity problems: Consider the situation in which during some transaction, say transferring money say Rs. 1000 from A's account to B's account, the power fails or the system goes down, leaving the state of database in an inconsistent state. Thus, the transactions should be atomic i.e., they must happen in their entirety or should not happen at all. It is difficult to ensure this property in a conventional file-processing system.

Security problems: Every person should not be allowed to access the database for security purposes. Since application programs are added to the system in an ad-hoc manner, it is difficult to ensure such security constraints.

1.2.5 Benefits of Conventional or Centralised DBMS

Let us discuss some specific advantages of the centralised control of data.

1. **Redundancy can be reduced:** As we know that in nondatabase systems (file oriented systems) each application has its own private files, because of which there exists considerable redundancy in stored data, resulting in wastage of storage space. Although database systems try to eliminate all possible redundancy, but we do not mean to suggest that all redundancy can be eliminated or necessarily should be eliminated. Sometimes there are sound business or technical reasons for maintaining several distinct copies of the same stored data. However, we do mean to suggest that any such redundancy should be carefully controlled i.e., the DBMS should be aware of it, if it exists, and should assume responsibility for "propagating updates".
2. **Inconsistency can be avoided to some extent:** Let us assume that in EMPLOYEE database, there exist two entries for the employee "ABC". This duplication is not only leads to redundancy but also leads to inconsistency as only one value of "ABC" might change during any modification of the data, hence two values in the database will not agree with each other. Clearly a database that is in an inconsistent state will supply incorrect or contradictory information to its users. It should be clearly noted that if the given fact is represented by a single entry, then such inconsistency can be avoided. Alternatively, if the redundancy is not removed but is controlled, then the DBMS could guarantee that the database is never inconsistent.
3. **The data can be shared:** With DBMS it is possible to share data in the database among various applications and also it is possible to satisfy the data requirements of new applications without having to create any additional stored data.
4. **Standards can be enforced:** With central control of database, the DBA (Database Administrator) can ensure that the data represents all applicable standards like corporate, installation, departmental, industry, national and international standards etc.

5. **Security restrictions can be applied:** A DBA can ensure that the database is accessed only through proper channels and by authorised users. For this purpose DBA defines some security rules, which are checked whenever there is an attempt to access the sensitive data.
6. **Integrity can be maintained:** The problem of integrity is the problem of ensuring that the data in the database is accurate and correct. Centralised control of DBMS can help in avoiding the problems of inconsistency as explained in the 2nd point, by permitting the DBA to define integrity rules to be checked whenever any data update, delete or insert operation is attempted.
7. **Conflicting requirements can be balanced:** Knowing the overall requirements of the enterprise – as opposed to the requirements of individual users – the DBA can so structure the system as 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 applications.

1.2.6 Data Independence

Most of the advantages of database systems, we have seen earlier, are fairly obvious. However, apart from these benefits i.e., reduced redundancy, maintaining integrity and consistency, sharing of data and balancing conflicting requirements, one further point, which might not be so obvious needs to be added to the list- namely, the provision of data independence. In fact we could say that rather than being an advantage, it is an objective for the database systems.

Before understanding data-independence, let us try to understand about data dependence. Data dependence is the way in which the data is organised in secondary storage, and the technique for accessing it, are both dictated by the requirements of the application under consideration, and moreover that knowledge of that data organisation and that access technique is built into the application logic and code.

Let's have a file EMPLOYEE, indexed on field “Emp_No”. In an older system, the application will know the fact that the index exists, and also knows the file sequence as defined by that index, and the internal structure of the application will be built around that knowledge. So, clearly such an application is data-dependent because it is impossible to change the storage structure (i.e., how the data is physically stored) or access technique (i.e., how it is accessed) without affecting the application, probably drastically.

In database systems, data dependence is undesirable for two reasons--

Different applications will need different views of the same data. Suppose there are two applications, A and B. Let A stores records in decimal and B stores records in binary. If application C using A and B is data dependent then C will not be able to integrate A and B. So, major work of programmers, done in applications A and B will be wasted.

But, it will still be possible to integrate the two files and to eliminate the redundancy, provided the DBMS is ready and able to perform all necessary conversions between the stored representations chosen and the form in which each application wishes to see it.

The DBA must have the freedom to change the storage structure or access technique in response to changing requirements, without having to modify existing applications. If this doesn't happen then for slight change in the way of accessing data would lead to change the program completely. This is wastage of scarce and valuable resource.

Thus provision of data independence is a major objective of database system. Following the definition of data dependence, thus, data independence can be defined as “the immunity of applications to change in storage structure and access techniques”- which clearly implies that applications concerned do not depend on any one particular storage structure or access technique.

We can also say that data independence is the ability to modify a schema definition in one level without affecting a schema in the next higher level.

Let us consider data type conversion on a particular field on each access. In principle, the difference between what the application sees and what is actually stored might be quite considerable, but that difference should not affect the user accessing the fields, hence this difference should be transparent to the user. To amplify this remark, we need to see the aspects of database storage structure that might be subject to variation.

1. **Representation of numeric data:** A numeric data field might be stored in internal arithmetic form (e.g., in packed decimal) or as a character string. Either way, the DBA must choose an appropriate base (e.g., binary or decimal), scale (fixed or floating point), mode (real or complex), and precision (number of digits). Any of these aspects might be changed to improve performance or to conform to a new standard or for many other reasons.
2. **Representation of character data:** A character string field might be stored using any of several distinct coded character sets or "forms-of-use" (e.g., ASCII, EBCDIC).
3. **Units of numeric data:** The units of numeric field might change from inches to centimeters, for example, during a process of metrication.
4. **Data encoding:** We can be able to represent data in coded value, e.g., we can store "Red" as 1, "Blue" as 2 and "Green" as 3, etc., but the user should be able to access them as Red, Blue and Green only.
5. **Structure of stored records:** Two existing records might be combined into one, and alternatively, a single stored record type might be split into two.
6. **Structure of stored files:** A given stored file can be physically implemented in storage in a wide variety of ways. For example, it might be entirely contained within a single storage volume (e.g., a single disk), or it might be spread across several volumes on several different types; it might or might not be sorted based on certain attribute; it might be indexed or it might have one or more embedded pointer chains; it might or might not be accessible via hash-addressing.

None of the above considerations should affect applications in any way. The database should be able to grow without affecting existing applications.

There are two levels of data independence:

1. **Physical data independence** is the ability to modify the physical schema without causing application programs to be rewritten. Modifications at the physical level are occasionally necessary to improve performance.
2. **Logical data independence** is the ability to modify the logical schema without causing application programs to be rewritten. Modifications at the logical level are necessary whenever the logical structure of the database is altered. It is more difficult to achieve.

The concept of data independence can be called as similar to abstract data types in modern programming languages. Both hide implementation's details from the users, to allow users to concentrate on the general structure, rather than two-level implementation details.

1.2.7 Data Dictionary

Database management system provide a facility known as data definition language (DDL), which can be used to define the conceptual scheme and also give some details about how to implement this scheme in the physical devices used to store the data. This definition includes:

- All entity sets and their associated attribute as well as the relationships among the entity sets.
- Any constraints that have to be maintained, including the constraints on the value that can be assigned to a given attribute and the constraints on the value assigned to different attributes in the same or different records.

These definitions, which can be described as metadata about the data in the database, are expressed in the DDL of the DBMS and maintained in a compiled form (usually as a set of tables). The compiled form of the definition is known as a **data dictionary**, **directory**, or **system catalog**. The data dictionary contains information on the data stored in the database and is consulted by the DBMS before any data manipulation operation.

Thus information pertaining to the structure and usage of data contained in the database, the metadata, is maintained in the data dictionary. The data dictionary is the database itself, documents the data. Each database user can consult the data dictionary to learn what each piece of data and the various synonyms of the data fields mean.

In an integrated system (i.e., in a system where the data dictionary is a part of the DBMS) the data dictionary stores information concerning the external, conceptual, and internal levels of the database. It contains the source of each data-field value, the frequency of its use, and an audit trail concerning updates, including the "who" and "when" for each update.

Currently data dictionary systems are available as add-on to the DBMS. Standards have yet to be evolved for integrating the data dictionary facility with the DBMS so that the two databases, one for metadata and the other for data, can be manipulated using a unified DDL/DML.

1.2.8 Database Security

Generally the terms "security" and "integrity" are considered to be same, but these two concepts are quite distinct. **Security** refers to the protection of data against unauthorised disclosure, alteration or destruction; **integrity** refers to the accuracy or validity of data. In other words,

- Security involves ensuring that users are allowed to do the things they are trying to do;
- Integrity involves ensuring that the things they are trying to do are correct.

Now the Question arises that why do we need Security?

The very obvious reason for the security is to prevent the misuse of the database by intentional (malicious) and accidental causes.

Accidental loss of data consistency may result from—

- Crashes during transaction processing;
- Anomalies caused by concurrent access to the database;
- Anomalies caused by the distribution of data over several computers;
- Logical errors that violate the assumption that transactions preserve the database consistency constraints.

The loss of data due to malicious access to data results from following:

- Unauthorised reading of data (theft of information);

- Unauthorised modification of data ;
- Unauthorised destruction of data.

The various aspects to the security problem are:

1. **Legal, social and ethical aspects** (for example, does the person making the request, say for withdrawing money from a bank, has legal rights for making the requested transaction?)
2. **Physical controls** (for example, is the computer or terminal room always locked or otherwise guarded.)
3. **Policy Questions** (for example, how does the enterprise owning the system decide who should be allowed access and to what?)
4. **Operational problems** (for example, is a password scheme is used, how are the passwords themselves kept secret? how often are they changed?)
5. **Hardware controls** (for example, does the processing unit provide any security features, such as the storage protection keys or a privileged operation mode?)
6. **Operating system security** (for example, does the underlying operating system erase the contents of storage and data files when they are finished with?)

The unit of data or "data object" that might need to be protected can range all the way from an entire database or collection of relations on the one hand, to the specific data item at a specific attribute position within a specific tuple within a specific relation on the other.

There are two approaches to security and control, discretionary and mandatory control.

In **discretionary control**, a given user will typically have different access rights on different objects. Different users will typically have different rights on the same object. Discretionary schemes are, thus, flexible. Discretionary control can be compared with the access control in the UNIX operating system. In UNIX file system, access permissions of user, group and others (ugo) are associated with each file. All three (ugo) has the combination of read, write and execute permission on the file.

In **mandatory control**, by contrast, each data object is tagged or labeled with a certain classification level, and each user is given a certain clearance level. A given data object can then be accessed only by users with the appropriate, clearance. Mandatory schemes are, thus, comparatively rigid.

Most DBMS support either discretionary control or mandatory control or both. In fact most of the DBMS support discretionary control, and some systems support mandatory control as well, thus, discretionary control is, thus, more likely to be encountered in practice.

Audit Trail

The DBMS has certain routines that maintain audit trail or a journal. An audit trail or a journal is a record of an update operation made on the database. Audit trails may be used to trace the occurrence of an incorrect activity. The audit trail records-

Who (user or the application program and a transaction number),

When (time and date),

(from) **where** (location of the user and/or the terminal), and

What (identification of the data affected, as well as before-and-after image of that portion of the database that was affected by the update operation).

In addition, a DBMS contains routines that make a backup copy of the data that is modified. The current SQL standards support only the discretionary access control.

1.3 THE DEFINITIONS RELATED TO RELATIONAL MODEL

We define RDBMS as a system in which, at a minimum:

1. The data is perceived by the user as relations (and nothing but relations); and
2. The operators are at the user's disposal, for example, for data retrieval – the operators that generate new relations from old, include at least **SELECT**, **PROJECT**, and **JOIN**.

The Relational model is a way of looking at data– that is, it is a prescription for a way of representing data (namely, by means of tables), and a prescription for a way of manipulating such a representation (namely, by means of operators such as **JOIN**). More precisely, the relational model is concerned with three aspects of data: **data structure**, **data integrity**, and **data manipulation**.

The data structure aspect of RDBMS deals with defining the tables; it's various attributes etc. Tables are nothing but the logical structure in a relational system, not the physical structure, we can also say that tables represent an abstraction of the way the data is physically stored – an abstraction in which numerous storage-level details, such as stored record placement, stored record sequence, stored data encoding, stored record prefixes, stored access structures such as indexes, and so forth, are all hidden from user.

The data manipulation deals with the way data is retrieved from the database. We use the SQL query for the purpose of retrieving and manipulating data.

All likelihood, the database is always subject to numerous integrity rules; for example, employee salaries might have to be in the range of Rs. 10,000 to Rs. 40,000.

1.3.1 Domain Definition

For defining the concept of domain, we first need to discuss the term scalar. The individual data value, for example, individual employee name in the **EMP** Table, can be called as the **scalar**, thus we can say that Scalar value represent “the smallest semantic unit of data” in the sense that they are atomic as they are non-decomposable as far as Relational DBMS is concerned.

We define a domain to be a set of scalar values, all of the same type necessarily. For example, the domain of Employee Names is the set of all the employees in an organisation listed on the basis of their names.

Thus, domain is the pool of values, from which actual attributes are drawn. More precisely, each attribute must be “defined on” exactly one underlying domain, meaning that values of that attribute must be taken from a domain.

Most of the today's relational systems do not really support domains, even though they are actually one of the most fundamental ingredients of the entire relational model.

One question which naturally should come in our mind is that, what is the significance of the domains?

The importance of domains is in domain-constrained comparisons.

Let us consider three tables:

S (S#, SNAME, STATUS, CITY) which describes about Supplier
 P (P#, PNAME, COLOUR, WEIGHT, CITY) which describe about Parts
 SP (S#, P#, QTY) which describes about shipments.

Now let us consider two queries,

<pre>SELECT FROM P, SP WHERE P.P# = SP.P#;</pre>	<pre>SELECT FROM P, SP WHERE P.COLOUR = SP.QTY;</pre>
--	---

Of these two queries, the one on the left makes sense as it is the comparison of two attributes P.P# and SP.P# which are defined on the same domain. On the contrary the comparison on the right is comparison of two attributes P.COLOUR and SP.QTY, which are not defined on same domains, hence it is not of much sense. Hence operators based on comparison like join, union etc. will fail for the query on the right.

The above example is described using SQL to describe the concept of domain, although concept of domain is primarily conceptual. Sometimes the SQL query might be OK yet conceptually wrong for example, consider the SQL,

```
SELECT .....
FROM P, SP
WHERE P.WEIGHT = SP.QTY; Here since both WEIGHT and QTY are real numbers, and they could be equated, but logically comparison of these two entities might not be useful.
```

In broad sense we can say that a domain is nothing more or less than a data type, which could be built-in or user-defined but has logical implications.

1.3.2 A Relation

Till now we are considering relations analogical to tables itself. We might use them interchangeably but conceptually there is slight difference between both of above two. For that let us first define Relation based on domains.

A relation, R, say on a collection of domains D1, D2, ... Dn – not necessarily all distinct – consists of two parts, a heading and a body. In elaborate:

1. The **heading** consists of a fixed set attributes, or more precisely <attribute-name:domain-name> pairs, { <A1:D1>, <A2:D2>, ..., <An:Dn> }, such that each attribute Aj corresponds to exactly one of the underlying domains Dj (j = 1,2,..., n). The attribute names A1, A2, ..., An are all distinct.
2. The **body** consists of a set of **tuples**. Each tuple in turn consists of a set of <attribute-name:attribute-value> Pairs

{<A1:vi1>, <A2:vi2>, ..., <An:vin> } i = 1, 2, ..., m, where m is the number of tuples in the set.

The value 'm' i.e., number of tuples is called the cardinality and 'n' i.e. the number of attributes is called degree, of relation R.

Now try to differentiate between the concepts of relation and table.

So on analysing the above definition we can say that relation is an abstract kind of object and table is a reasonable concrete picture of such an abstract data type. But there are certain cases, which suggest that tabular representation or tables suggests something, which are not true with relations. For example, it clearly suggests that the rows of the table (i.e., tuples of the relation) are in a certain top-to-bottom order. but they are actually not.

Properties of Relations

- They do not contain duplicate tuples.
- Tuples in relations are unordered;
- Attributes in relations are unordered;
- All attribute values in a relation are atomic.

Kinds of Relations

Named relation: A named relation is a relation that has been defined to the DBMS. In SQL syntax we define relations by means of CREATE BASE RELATION or CREATE VIEW or CREATE SNAPSHOT etc.

Base Relation: A base relation is a named relation that is not a derived relation i.e., they are autonomous. We can manipulate the conceptual or "physical" relations stored in the database. By "physical" we mean that the relation corresponds to some stored data. This data may not be stored as a table and may actually be split horizontally or vertically and reside on one or more storage devices (at one or more sites). Such conceptual relations are referred as base relations.

Derived Relation: A derived relation is a relation that is defined by means of some relational expressions, in terms of other named relations. Ultimately they are derived from the base relations only. For example, view is a derived relation.

Views

For security and other concerns it is undesirable that all users should be able to see the entire relation, thus, it would be beneficial if we could create useful relations for different group of users, rather allowing all users to manipulate the base relation. Therefore, relation that is not a part of the physical database, i.e., a virtual relation, is made available to the users as view. To be useful, we assign the view a name and relate it to the query expression as

Create view <view name> as <query expression>

Example

Let EMPLOYEE be the base relation. We create the table for EMPLOYEE as follows:

Create table EMPLOYEE

(Emp_No	integer not null,
Name	char(25),
Skill	char(20),
Sal_Rate	decimal(10,2),
Dob	date,
Address	char(100))

For reasons of confidentiality, not all users are permitted to see the Sal_Rate of an employee. For such users the DBA can create a view, for example, EMP_VIEW defined as:

```

Create view EMP_VIEW as
  ( select Emp_No, Name, Skill, Dob, Address
    from EMPLOYEE)
    
```

Limitations of Views

Although views are a useful tool for queries, they represent significant problems if updates, insertions, or deletions are expressed with them. The difficulty is that a modification to the database expressed in terms of a view must be translated to a modification to the actual relations in the logical model of the database.

Since views represent only the selected attributes from the various relations which are abstracted as per our requirement. If we perform any operation (insert, delete or update) in the view, then we will perform the same operation in the base relations also maintaining both the integrity constraints. This task is very difficult to achieve, thus modifications are generally not permitted on view relations, except in limited cases. The general problem of database modification through views has been the subject of substantial research.

Views and Security

Views can be used to provide security mechanism. An example using SQL statements is:

```

CREATE VIEW SECVIEW AS
  SELECT S.S#, S.SNAME, S.CITY
  FROM S;
    
```

The view defines the data over authorisation is to be granted. The granting itself is done by means of the GRANT statement, e.g.,

```

GRANT SELECT ON SECVIEW TO Saurabh, Anita, Bharat;
    
```

Thus, allowing only limited portion available to the users. These grant permissions can also be revoked using REVOKE command.

Check Your Progress 1

Question 1. What are the advantages of using a database system?

.....

.....

.....

Question 2. What are the disadvantages of using a database system?

.....

.....

.....

Question 3. Consider a database PART_DETAIL:

MAJOR_PART#	MINOR_PART#	QUANTITY
P1	P2	2
P1	P3	4
P2	P3	1
P2	P4	3
P3	P5	9
P4	P5	8
P5	P6	3

- (a) Write a CREATE BASE RELATION and appropriate set of CREATE DOMAINS for the above relation.
- (b) Assuming this relation is included in some other database, say supplier-and-part, show the updates which system make to the catalog (data dictionary) to reflect your answer to part (a).
- (c) Write an appropriate set of DESTROY statements to cause the catalog updates of part (b) to be undone.
-
-

Question 4. A relation is defined to have a set of attributes and a set of tuples. Now, in mathematics the empty set is a perfectly respectable set; indeed, it is usually desirable that results, theorems, etc., that hold true for a set of n elements, should also hold true for a set of $n = 0$. Can a relation have an empty set of tuples? Or an empty set of attributes?

.....

.....

Question 5. Base relation STUDENT looks like:

STUDENT (STUDENT_ID, SEX, PART_TIME_JOB, STIPEND, COURSE, FEES)
PRIMARY KEY (STUDENT_ID)

Write security rules in the following statements using some pseudo-query approach.

- (i) User Anu RETRIEVE privileges over the entire relation.
 - (ii) User Smita INSERT and DELETE privileges over the entire relation.
 - (iii) Each user RETRIEVE privileges over the user's own tuple (only).
 - (iv) User Nath RETRIEVE privileges over the entire relation and UPDATE privileges over the STIPEND and FEES attribute (only).
 - (v) User Ram RETRIEVE privileges over the STUDENT_ID, STIPEND and FEES attributes only.
 - (vi) User John RETRIEVE privileges as for Ram and UPDATE privilege over the SALARY and FEES attributes (only).
-
-

1.4 RELATIONAL DATA INTEGRITY

What do we mean by database consistency?

At any given time, any given database contains some particular configuration of data values, and of course that configuration is supposed to "reflect reality".

We should know first that what is integrity? Integrity simply means to maintain the consistency of data, thus Integrity constraints provide a means of ensuring that changes made to the database by authorised users do not result in a loss of data consistency. Thus, integrity constraints guard against damage to the database.

The integrity constraints are of two types:

Key declarations: The stipulation that certain attributes form a candidate key for a given entity set. The set of legal insertions and updates is constrained to those that do not create two entities with the same value on a candidate key.

Form of a relationship: Many-to-many, one-to-many, one-to-one. A one-to-one or one-to-many relationship restricts the set of legal relationships among entities of a collection of entity sets.

1.4.1 Candidate Keys

Let R be a relation. Then a candidate key for R is a subset of the set of attributes of R, say K, such that:

Uniqueness property: No two distinct tuples of R have the same value for K
Irreducible property: No proper subset of K has the uniqueness property.

Every relation does have at least one candidate key, because relations do not contain duplicate tuples. A candidate key could be combination of many attributes also, if necessary. For example, if we say that the attribute combination {S#, P#} is a candidate key for base relation SP, we do not mean merely that no two tuples in the relation that happens to be the current value of SP have the same value for that combination; rather, we mean that all possible relations R that can be values of SP are such that no two distinct tuples of R have the same value for that combination.

Properties of a candidate key

1. They are defined on set of attributes. For example, {S#} is a candidate key for relation S etc.
2. A candidate that involves more than one attribute is said to be composite. A candidate key that involves exactly one attribute is said to be simple.
3. The requirement for irreducibility needs a little elaboration. Basically the point is that if we were to specify a "candidate key" that was not irreducible, the system would not be aware of true state of affairs, and thus would not be able to enforce the associated integrity constraint properly. For example, suppose we were to define the combination {S#, CITY} instead of S# alone, as a candidate key for the base relation S. Then the system will not enforce the constraint that supplier numbers are "globally" unique: instead, it will enforce only the weaker constraint that supplier numbers are "locally" unique within city. Thus, true candidate keys do not include any attributes that are irrelevant for unique identification purposes.
4. Finally, logical notion of candidate key should not be confused with the physical notion of a "unique index".

Candidate keys are important because they provide the basic tuple-level addressing mechanism in a relational system. For example if S# is a candidate key for the relation S then the expression, S WHERE S# = 'DEL0001' is guaranteed to yield at most one tuple. If SNAME is also the candidate key for the relation S, then the expression S WHERE SNAME = 'RAM' will also yield at most one tuple.

By contrast if we query on non-candidate key attribute then we end up yielding unpredictable number of tuples. For example, the expression S WHERE CITY = "DELHI" will yield unpredictable number of tuples.

1.4.2 Primary Key

The primary key is the kind of candidate key that is chosen by the database designer as the principal means of identifying entities within an entity set. The remainder, if any, is then called **alternate keys**.

1.4.3 Foreign Keys

Let R2 be a base relation, then a foreign key in R2, is a subset of the set of attributes of R2, say FK, such that:

1. There exists a base relation R1 (R1 and R2 not necessarily distinct) with a candidate key CK, and
2. For all time, each value of FK in the current value of R2 is identical to the value of CK in some tuple in the current value of R1.

Foreign Keys are also defined on set of attributes. For example, the foreign keys in relation SP in the suppliers-and-parts database are {S#}(here the base relation R1 is relation S) and {P#}(here the base relation R1 is relation P). Every value of a given foreign key is required to appear as a value of the matching candidate key. There is no requirement that a foreign key be a component of the primary key-or of some candidate key- of its containing relation.

Example

An employee may perform different roles in different projects, say RAM is doing coding in one project and designing in other. So, let's have relations EMPLOYEE and PROJECT. Let ROLE be the relationship describing the different roles required in the any project.

The relational scheme for the above three relations are:

EMPLOYEE (EMP#, Name, Designation)
PROJECT (PROJ#, Proj_Name, Details)
ROLE (ROLE#, Role_description)

In the above relations EMP# and PROJ# are unique and not null, respectively. As we can clearly see that we can identify the complete instance of the entity set employee through the attribute EMP#, thus EMP# is the primary key of the relation EMPLOYEE, similarly PROJ# is the primary key for the relation PROJECT.

Let ASSIGNMENT is a relationship between entities EMPLOYEE and PROJECT and ROLE, describing that which employee is working on which project and what is the role of the employee in the respective project. A possible representation of this relationship is by using the entities involved in the relationship.

ASSIGNMENT (Employee, Project, Role)

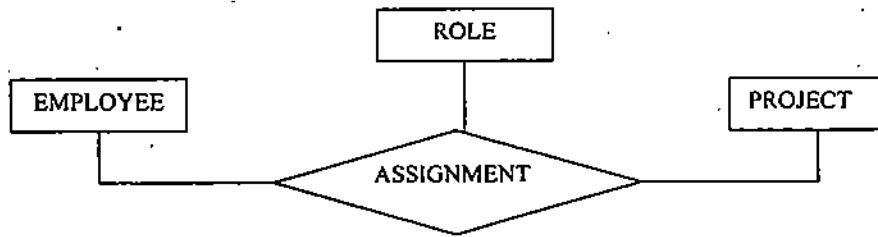


Figure 2: E-R diagram for employee role in development team

Let us consider sample relations:

EMPLOYEE

Emp#	Name	Designation
101	RAM	Analyst
102	SITA	Receptionist
103	ARVIND	Manager

PROJECT

PROJ#	Proj_name	Details
TCS	Traffic control System	For traffic shaping.
LG	Load Generator	To simulate load for input in TCS.
B++1	B++_TREE	ISS/R turbo sys

ROLE

ROLE#	Role_description
1000	Design
2000	Coding
3000	Marketing

ASSIGNMENT

EMP#	PROJ#	ROLE#
101	TCS	1000
101	LG	2000
102	B++1	3000

Using the unique identification properties of keys, we can replace the Employee, Project and role entities in ASSIGNMENT by their keys. The keys act as surrogates for their respective entities. Thus, we can define relational scheme for the relation ASSIGNMENT as follows:

ASSIGNMENT (EMP#, PROJ#, ROLE#)

It is clear that ASSIGNMENT is a relation that establishes a relationship among three "owner" relations. Such a relation may be thought of as an associative relation. The key of the associative relation is always the union of the key attributes of the owner relation. Thus, the key of the relation ASSIGNMENT is the combination of the attributes EMP#, PROJ# and ROLE#.

The attributes EMP#, PROJ# and ROLE# in the relation ASSIGNMENT are foreign keys.

Now after knowing the concept of foreign key, we can proceed to know about the actual integrity constraints namely Referential Integrity and Entity Integrity.

1.4.4 Referential Integrity

The database must not contain any unmatched foreign key values

The term "unmatched foreign key value" here means a foreign key value for which there does not exist a matching value of the relevant candidate key in the relevant target relation. In other words, the rule simply says: If B references A, then A must exist.

Database modifications can cause violations of referential integrity. We list here the test we must make for each type of database modification to preserve the following referential-integrity constraint:

RDDMS
Terminology

$\pi_{\alpha}(r_2) \subseteq \pi_k(r_1)$, here α is the foreign key (FK) and k is the key of relation R_1 (CK). π is the projection operator, r_1 and r_2 are the instance of relation R_1 and R_2 respectively.

Delete

In SQL query delete statement takes the form

```
DELETE target;
```

The target is a relational expression: all tuples in a target relation are deleted. Here is an example:

```
DELETE S WHERE STATUS < 20 ;
```

The target will often be just a restriction of some named relation.

A very obvious question arises in our mind that what should happen on an attempt to delete the target of a foreign key reference?

For example, an attempt to delete a supplier for which there exists at least one matching shipment? In general there are at least two possibilities:

RESTRICTED – The delete operation is “restricted” to the case where there are no such matching shipments (it is rejected otherwise)

CASCADE – The delete operation “cascades” to delete those matching shipments also.

So, if a tuple t_1 is deleted from r_1 , the system must compute the set of tuples in r_2 that reference t_1

$$\sigma_{t_1[k]}(r_2)$$

If this set is not empty, either the delete command is rejected as an error, or the tuples that reference t_1 must themselves be deleted. The latter solution may lead to cascading deletions, since tuples may reference tuples that reference, t_1 and so on.

Insert

The insert statement takes the form

```
INSERT source INTO target;
```

Where source and target are relational expressions that evaluate to type-compatible relations (in practice, the target will usually be just a named relation). The source is evaluated and all tuples of the result are inserted into the target. Here is an example:

```
INSERT (S WHERE CITY = "London") INTO TEMP;
```

TEMP here is assumed to be type-compatible with relation S.

If a tuple t_2 is inserted into r_2 , the system must ensure that there is a tuple t_1 in r_1 such that $t_1[k] = t_2[\alpha]$. That is,

$$t_2[\alpha] \in \pi_k(r_1)$$

Update

The update statement takes the form

UPDATE target assignment-comma-list ;

Where each assignment is of the form

Attribute := scalar-expression

The target is a relational expression, and each attribute is an attribute of the relation that results from evaluating that expression. All tuples in the target relation are updated in accordance with the specified assignments. Here is an example:

```
UPDATE P WHERE COLOR = 'Red'
      CITY := 'Paris' ;
```

In practice the target will often be just a restriction of some named relation.

But, what should happen on an attempt to update a candidate key that is the target of a foreign key reference? – For example, an attempt to update the supplier number for a supplier for which there exists at least one matching shipment? In general there are the same possibilities as for DELETE:

RESTRICTED – the update operation is “restricted” to the case where there are no matching shipments (it is rejected otherwise).

CASCADES – The update operation “cascades” to update the foreign key in those matching shipments. Also we must consider two cases for update: updates to the referencing relation (r_2); and updates to the referenced relation (r_1).

If a tuple t_2 is updated in relation r_2 , and the update modifies values for the foreign key α , then a test is made which should be similar to test in insert case. The system must ensure that $t_2[\alpha] \in \pi_K(r_1)$

If a tuple t_1 is updated in relation r_1 , and the update modifies values for the primary key $[K]$, then a test is made which should be similar to delete case. The system must compute $\sigma_{\alpha=([K])}(r_2)$ using the old value of t_1 (the value before the update is applied). If this set is not empty, the update is rejected as an error, or the update is cascaded in a manner similar to delete.

Now before understanding second type of Integrity constraint viz., Entity Integrity, we should be familiar to the concept of **NULL**.

Basically, nulls are intended as a basis for dealing with the problem of missing information. This kind of situation is frequently encountered in the real world. For example, historical records such as “Date of birth unknown”, or police records may include the entry “Present whereabouts unknown.” Hence it is necessary to have some way of dealing with such situation in our formal database systems, thus, Codd proposed an approach to this issue that makes use of special markers called nulls to represent such missing information.

A given attribute in the table might or might not be allowed to contain nulls – it will depend on the definition of the attribute in question, at least for base relations. If a given attribute is allowed to contain nulls, and a tuple is inserted into the relation and no value is provided to that attribute, the system will automatically place a null in that position. If a given attribute is not allowed to contain nulls, the system will reject any attempt to introduce a null into that position.

1.4.5 Candidate Keys and Nulls

The relational model has historically required that exactly one candidate key be chosen as the primary key for the relation in question, the remaining candidate keys, if any, are then alternate keys. So along with the primary key concept, the model has included the following rule known as **Entity Integrity Rule**, which states, no component of the primary key of a base relation is allowed to accept nulls. In other words, the definition of every attribute involved in the primary key of any base relation must explicitly or implicitly include the specifications of **NULLS NOT ALLOWED**.

Foreign Keys and Nulls
Let us consider the relation:

DEPT		
DEPT#	DNAME	BUDGET
D1	Marketing	10M
D2	Development	12M
D3	Research	5M

EMP			
EMP#	ENAME	DEPT#	SALARY
E1	Rahul	D1	40K
E2	Aparna	D1	42K
E3	Ankit	D2	30K
E4	Sangeeta		35K

Suppose that Sangeeta is not assigned any Department, thus, in the EMP tuple corresponding to Sangeeta, therefore, there is no genuine department number that can serve as the appropriate value for the DEPT# foreign key, thus, one cannot determine DNAME and BUDGET. This may be a real situation where the person has newly joined and is undergoing training and will be allocated to a department only on completion of the training. Thus, nulls in foreign key values may not be logical error.

So, based on Nulls we redefine Foreign Keys. Let R2 be a base relation, then a foreign key in R2 is a subset of the set of attributes of R2, say FK, such that:

- There exists a base relation R1 (R1 and R2 not necessarily distinct) with a candidate key CK, and
- For all time, each value of FK in the current value of R2 either is null or is identical to the value of CK in some tuple in the current value of R1.

Here referential integrity rule – “The database must not contain any unmatched foreign key values” – remains unchanged, but the meaning of the term “unmatched foreign value” is extended to refer to nonnull foreign key value for which there does not exist key value of the relevant candidate key in the relevant target relation

Check Your Progress 2

Question 1. Consider supplier-part-project database:

S		
S#	SNAME	CITY
S1	Smita	Delhi
S2	Jim	Pune
S3	Ballav	Pune

P			
P#	PNAME	COLOUR	CITY
P1	Nut	Red	Delhi
P2	Bolt	Blue	Pune
P3	Screw	White	Bombay

S

S#	SNAME	CITY
S4	Sita	Delhi
S5	Anand	Agra

P

P#	PNAME	COLOUR	CITY
P4	Screw	Blue	Delhi
P5	Cam	Brown	Pune
P6	Cog	Grey	Delhi

J

J#	JNAME	CITY
J1	Sorter	Pune
J2	Display	Bombay
J3	OCR	Agra
J4	Console	Agra
J5	RAID	Delhi
J6	EDS	Udaipur
J7	Tape	Delhi

SPJ

S#	P#	J#	QUANTITY
S1	P1	J1	200
S1	P1	J4	700
S2	P3	J2	400
S2	P2	J7	200
S2	P3	J3	500
S3	P3	J5	400
S3	P4	J3	500
S3	P5	J3	600
S3	P6	J4	800
S4	P6	J2	900
S4	P6	J1	100
S4	P5	J7	200
S5	P5	J5	300
S5	P4	J6	400

Using the sample data values in above table, tell the effect of each of the following operation:

1. UPDATE project J7, setting CITY to Nagpur.
2. UPDATE part P5, setting P# to P4.
3. UPDATE supplier S5, setting S# to S8, if the relevant update rule is restricted.
4. DELETE supplier S3, if the relevant rule is CASCADES.
5. DELETE part P2, if the relevant delete rule is RESTRICTED.
6. DELETE project J4, if the relevant delete rule is CASCADES.
7. UPDATE shipment S1-P1-J1, setting S# to S2. (shipment S1-P1-J1 means that in SPJ table the value for attributes S#, P# and J# is S1, P1 and J1 respectively)
8. UPDATE shipment S5-P5-J5, setting J# to J7.
9. UPDATE shipment S5-P5-J5, setting J# to J8
10. INSERT shipment S5-P6-J7
11. INSERT shipment S4-P7-J6
12. INSERT shipment S1-P2-*jjj* (where *jjj* stands for a default project number).

.....

.....

.....

.....

Question 2. The following two relations represent a database containing information about departments and employees:

DEPT (DEPT#, ... , MGR_EMP#, ...)
 EMP (EMP#, ..., DEPT#, ...)

Every department has a manager (MGR_EMP#); every employee has a department (DEPT#). Try to show the relationship diagrammatically and write a suitable database definition for this database.

.....

.....

.....

Question 3. The following two relations represent a database containing information about employees and programmers.

EMP (EMP#, ..., JOB, ...)
 PGMR (EMP#, ..., LANG, ...)

Every programmer is an employee, but the converse is not true. Write a suitable database definition for this database.

.....

.....

.....

1.5 DATA DICTIONARY CHECKLIST

This section is just for reference. You must visit this section again after you have gained experience with commercial database management systems.

Data dictionaries are used as the notation for representing structure of data items. So, we can say that data dictionaries are the data about the data. In data dictionaries we need to express;

Composition (sequence?) – how an item is made up of simpler ones (its attributes).

Repetition – items which are repeated in (c.g.) lists, arrays, etc.

Selection – values for items chosen from alternatives.

Optionality – items, which are not always present.

Symbols used in DD Notation names a meaningful name for every composite or basic data item

Symbols	Meanings in DD Notation
=	'is defined as', or 'is made up of'
+	'and'
{ }	Zero or more of whatever's inside i.e., repetition
n{ }m	between n and m (inclusive)
[]	one of the listed attributes is present
()	item inside is optional
" "	enclose literals (actual values)
**	enclose comments - define meaning of data, informally.

Let us take an example of the Tutorial List in a university put up on the Notice Board.

The data on this consists of:

- a Title
- a Version Number
- a Date
- a day and time and room and a list of students' names
- another day and time and room and list of students' names
- (yet) another day and time and room and list of students' names.

In this example, we have composition and repetition. But we need to know that how much repetition is required for the day, time and room of the tutorial.

Tutorial List Example

TutorialList = Title + VersionNumber + Date + {TutorialDetails}

TutorialDetails = DayOfWeek + TimeSlot + Room + StudentList

StudentList = {FamilyName + FirstName}

or.....

TutorialList = Title + VersionNumber + Date +
(DayOfWeek + TimeSlot + Room + {FamilyName + FirstName})

VersionNumber = Digit + "." + Digit

Digit = ["1"|"2"|"3"|"4"....]

TimeSlot = StartHour + "-" + EndHour

StartHour = ["9"|"10"|"11"|"12"....]

SDD Course Plan

CoursePlan = DateOfPlan + VersionNumber + Titles + {WeekDetails}

WeekDetails = *WeekNo. + StartDate + [TeachingWeek [NonTeachingWeek]

NonTeachingWeek = ["admin week"|"induction week"|"student centred learning"....]

TeachingWeek = 2{LectureDetails}2 + (TutorialDetails) + (PracticalWork)

LectureDetails = *Description of Lecture Content*

StartDate = Date

DateOfPlan = Date

Date = *date in format "dd-mm-yy"*

Data Dictionary - Conclusions

- Simple way of describing syntax of composite data.

- Meaning or Semantics captured informally, by: 'Meaningful' names for data.
- Comments explaining constraints and usage.
- Wide range of uses - any situation in which composite data is found:
 - particularly data 'on the move',
 - particularly data crossing the system boundary.
 - customers' source documents.
 - input data on data entry forms.
 - output data on reports/screens, etc.
 - Internal data needs more attention.....

Check List for Data Dictionary Review

Since these days more and more projects are implemented on Relational Database Management System (RDBMS) platform, thus we must have a checklist for the review of Data Dictionary on such a platform.

By adding few more questions, this checklist can be used to evaluate a Data Dictionary.

In the Data Dictionary, entity maps to a Database table and the attributes correspond to columns (field) of the table.

1. Each entity should have a primary key, which uniquely identifies a record in the table. This primary key can either be system-generated sequence or user defined unique value.
2. Each attribute should belong to a particular datatype (char, integer, date, number etc.).
3. Each attribute should have a precision defined (char (30), number (10,2) etc.).
4. Mandatory (required) or not mandatory characteristic of a attribute should be specified.
5. Default format (e.g., mm/dd/yyyy) of the attribute may be specified. This is useful if some CASE tool is used to generate the code.
6. The Data Dictionary (ER diagram) should display the relationships between different entities (One to Many, One to One, Many to One etc.)
7. Many to Many (M: M) relationship between entities should be broken down into One to Many (1:M) and Many to One (M: 1).
8. Foreign Keys should be defined to create relationships.
9. Foreign key in a detail table should be a Primary Key or a unique key of the Master table.
10. Constraints should be present on the columns and tables to enforce referential integrity.
11. Database Triggers, which get executed when changes (insert, update, delete) occur on the underlying tables, should be defined.
12. Approximate number of records in each table should be calculated.

13. Percentage increase in the number of records over a period of time (may be a year).
14. Database space allocated to each table.
15. Description of each entity and its attributes should be present.
16. Any procedures or functions operating on the tables should be defined.
17. Access privileges (select, insert, update, delete, all, etc.) for the entities should be specified.
18. Structure and attributes of the tables should satisfy the User Requirements.
19. Owner of the table should be specified.
20. Naming convention for the table and column names should follow the enterprise wide standards.

Additional points which should be checked to evaluate a Data Dictionary.

1. All Classes should be defined as per user requirements.
2. All the methods needed for each class should be defined in the Data Dictionary.
3. All the data members for each class should be defined.
4. Scope of the data members should be specified (Public or Private).
5. Pre and Post conditions should be defined for each method.
6. Parameter types passed to each method and the return type should be specified.

An Example Dictionary: Oracle Data Dictionary

There are many times when a DBA finds the need to generate a custom SQL report based on data dictionary and dynamic performance tables. There is a wealth of information contained in the DBA_, V\$, and X\$ series of tables and views, but to be of any use, the purpose of each table and its relationship to other tables in the Data Dictionary must first be understood.

The Oracle Data Dictionary is a collection of C constructs, Oracle tables and Oracle views. At the lowest level are the "hidden" C structures known as the X\$ tables. To even see their contents a DBA has to jump through a few hoops, and once you are there, they are not well documented, and have such logical attributes as "mglwmp" or "tabsrpr". It is suggested that administrators leave these to Oracle.

The next higher layer of the Oracle Data Dictionary is the \$ tables. These are more readable descriptions of the X\$ tables and have such names as COL\$ or TAB\$. While being a step above the X\$ structures, it is still suggested that the DBA only use them when it is really needed.

The third layer of the dictionary is the V\$ views and their cousins the V_\$ tables (actually twins, since for nearly all of the V\$ views there is a corresponding V_\$ view). These are the workhorses of the Data Dictionary, and what most DBA scripts should deal with.

The next layer of the dictionary is the DBA_ series of views. These views are made from the V\$ and \$ sets of views and tables and provide a more useful interface for viewing the insides of the Oracle Data Dictionary. USER_ and ALL_ views are based on the DBA_ views.

The USER_ and ALL_ views provide an important function since not all users are granted access to the DBA_ level views. The USER_ views provide the user with information on all of the objects they personally own while the ALL_ views provide information on all of the objects that they are allowed to access.

The following figure is a pseudo entity relationship diagram of the USER_views layer of the Oracle Data Dictionary. It is particularly handy as a printed reference.

One-to-One Relationship Must have a related entry in the table. They may have a related entry in the table.

The USER_VIEWS – Descriptions

The following is a description of Oracle provided USER_views. These views are derived from the DBA_level data dictionary views and contain information for the current user.

USER_ADMINCATALOG	Information about the current user's asynchronous administration requests.
USER_ADMINMASTERS	N-way replication information about the current user.
USER_ADMINOBJECTS	Replication information about the current user's objects.
USER_ADMINREGSNAPS	All snapshots in the database.
USER_ADMINRESOLUTION	Description of all conflict resolutions in the database.
USER_ADMINSCHEMAS	Replication information about the current user.
USER_ARGUMENTS	Arguments in object accessible to the user.
USER_AUDIT_OBJECT	Audit trail records for objects accessible to the user.
USER_AUDIT_SESSION	Records for the user's audit trail records regarding CONNECT and DISCONNECT.
USER_AUDIT_STATEMENT	Records for the user's audit trails involving statements (GRANT, REVOKE, AUDIT, NOAUDIT, and ALTER SYSTEM commands).
USER_AUDIT_TRAIL	Audit trail entries relevant to the user.
USER_CATALOG	Tables, Views, Synonyms and Sequences owned by the user.
USER_CLUSTERS	Descriptions of user's own clusters.
USER_CLUSTER_HASH_EXPRESSIONS	Hash functions for the user's hash clusters.
USER_CLU_COLUMNS	Mapping of table columns to cluster columns.
USER_COL_COMMENTS	Comments on columns of user's tables and views.
USER_COL_PRIVS	Grants on columns for which the user is the owner, grantor or grantee.
USER_COL_PRIVS_MADE	All grants on columns of objects owned by the user.
USER_COL_PRIVS_REC'D	Grants on columns for which the user is the grantee.
USER_CONSTRAINTS	Constraint definitions on user's own tables.
USER_CONS_COLUMNS	Information about accessible columns in constraint definitions.
USER_DB_LINKS	Database links owned by the user.
USER_DEPENDENCIES	Dependencies to and from user's objects.
USER_ERRORS	Current errors on stored objects owned by the user.
USER_EXTENTS	Extents comprising segments owned by the user.
USER_FREE_SPACE	Free extents in tablespaces accessible to the user.
USER_INDEXES	Description of the user's own indexes
USER_IND_COLUMNS	COLUMNS comprising user's INDEXes or on user's TABLES.

USER_JOBS	All jobs owned by this user.
USER_OBJECTS	Objects owned by the user.
USER_OBJECT_SIZE	Sizes, in bytes, of various pl/sql objects.
USER_OBJ_AUDIT_OPTS	Auditing options for user's own tables and views.
USER_REFRESH	All the refresh groups.
USER_REFRESH_CHILDREN	All the objects in refresh groups, where the user owns the refresh group.
USER_REPAUDIT_ATTRIBUTE	Information about attributes automatically maintained for replication.
USER_REPAUDIT_COLUMN	Information about columns in all shadow tables for user's replicated tables.
USER_REPCAT	Replication information about the current user.
USER_REPCATLOG	Information about asynchronous administration requests.
USER_REPCOLUMN_GROUP	Information about the current user's asynchronous administration requests.
USER_REPDDL	Arguments that do not fit in a single repcat log record.
USER_REPGENERATED	Objects generated for the current user to support replication.
USER_REPGROUPED_COLUMN	Columns in the all column groups of user's replicated tables.
USER_REPKEY_COLUMNS	Primary columns for a table using column-level replication.
USER_REPOBJECT	Replication information about the current user's objects.
USER_REPPARAMETER_COLUMN	All columns used for resolving conflicts in user's replicated tables.
USER_REPPRIORITY	Values and their corresponding priorities in all priority groups, which are accessible to the user.
USER_REPPRIORITY_GROUP	Information about all priority groups which are accessible to the user.
USER_REPPROP	Propagation information about the current user's objects.
USER_REPRESOLUTION	Description of all conflict resolutions for user's replicated tables.
USER_REPRESOLUTION_METHOD	All conflict resolution methods accessible to the user.
USER_REPRESOLUTION_STATISTICS	Statistics for conflict resolutions for user's replicated tables.
USER_REPRESOL_STATS_CONTROL	Information about statistics collection for conflict resolutions for user's replicated tables.
USER_REPSHEMA	N-way replication information about the current user.
USER_RESOURCE_LIMITS	Display resource limit of the user.
USER_ROLE_PRIVS	Roles granted to current user.
USER_SEGMENTS	Storage allocated for all database segments.
USER_SEQUENCES	Description of the user's own SEQUENCES.
USER_SNAPSHOTS	Snapshots at which the user can look.
USER_SNAPSHOT_LOGS	All snapshot logs owned by the user.
USER_SOURCE	Source of stored objects accessible to the user.
USER_SYNONYMS	The user's private synonyms.
USER_SYS_PRIVS	System privileges granted to current user.
USER_TABLES	Description of the user's own tables.
USER_TABLESPACES	Description of accessible tablespaces.

USER_TAB_COLUMNS	Columns of user's tables, views and clusters.
USER_TAB_COMMENTS	Comments on the tables and views owned by the user.
USER_TAB_PRIVS	Grants on objects for which the user is the owner, grantor or grantee.
USER_TAB_PRIVS_MADE	All grants on objects owned by the user.
USER_TAB_PRIVS_RECD	Grants on objects for which the user is the grantee.
USER_TRIGGERS	Triggers owned by the user.
USER_TRIGGER_COLS	Column usage in user's triggers.
USER_TS_QUOTAS	Tablespace quotas for the user.
USER_USERS	Information about the current user.
USER_VIEWS	Text of views owned by the user.

1.6 SUMMARY

A Database Management System (DBMS) consists of a collection of interrelated data and a collection of programs to access that data. Data refers to the values actually stored in the database and information refers to the meaning of those values as understood by the user. The data describes one particular enterprise. The primary goal of a DBMS is to provide convenient and efficient environment for people to use in retrieving and storing information.

Although file based systems were first commercial applications that provided increased efficiency in data processing compared to earlier manual paper record-based systems, but still these file based systems suffered from certain disadvantages, like, data redundancy and inconsistency, difficulty in accessing data, data isolation, integrity problems, atomicity problems and security problems.

Database systems are designed to store large bodies of information. The management of data involves both definition of structures for the storage of information and provision of mechanism for the manipulation of information.

DBMS provides a facility called DDL i.e., **data definition language**, which specifies the database scheme. DDL is expressed by set of definitions. These definitions are compiled into a set of tables that is stored in a special file called the **data dictionary**, thus, data dictionary contains metadata i.e., data about data.

The relational data model is based on the collection of tables. Tables are formed of attributes and tuples. The user of the database system may query these tables, insert new tuples, delete tuples, and update (modify) tuples.

Domains are the pools of values, from which actual attributes are drawn, and Relation on collection of domains consists of two parts, a heading and a body. The heading consists of a fixed set attributes, and the body consists of a set of tuples. Views are "virtual relations" defined by a query expression. Views are useful mechanism for simplifying database queries, but modification of the database through views may have potentially disadvantageous consequences. Therefore, database systems severely restrict updates through views.

Integrity constraints ensure that changes made to the database by authorised users do not result in a loss of data consistency. Domain constraints specify the set of possible values that may be associated with an attribute. Such constraints may also prohibit the use of null values for particular attributes.

Referential-integrity constraints ensure that a value that appears in one relation for given sets of attributes also appears for a certain set of attributes in another relation.

1.7 MODEL ANSWERS

Check Your Progress 1

- Question 1.
- Reduced Redundancy
 - Sharing
 - Enforcement of standards
 - Security
 - Integrity of Data
 - Fine tuning of database
 - Data independence

Question 2. Some of the disadvantages are as follows:

- Security might be compromised (without good controls)
- Integrity might be compromised (without good controls)
- Additional hardware might be required
- Performance overhead might be significant
- Successful operation is crucial (the enterprise might be highly vulnerable to failure)
- The system is likely to be complex (though such complexity should be concealed from the user).

Question 3. (a) Data Definition (It is not in SQL)

```
CREATE DOMAIN P# CHAR (6);
CREATE DOMAIN QTY NUMERIC (9);
CREATE BASE RELATION PARTR_DETAIL
(MAJOR_PART# DOMAIN ( P# ),
MINOR_PART# DOMAIN ( P# ),
QUANTITY DOMAIN ( QTY ))
PRIMARY KEY (MAJOR_PART#, MINOR_PART#);
```

Note: If the PART_DETAIL relation were a part of supplier-and-part database, we would probably need additional specifications in the relation definition.

```
FOREIGN KEY (RENAME MAJOR_PART# AS P#) REFERENCES P;
FOREIGN KEY (RENAME MINOR_PART# AS P#) REFERENCES P;
```

(b) We show the new catalog entries.

DOMAINS

DOMNAME	DATATYPE
P#	CHAR (6)
QTY	NUMERIC (9)

RELATIONS

RELNAME	DEGREE	CARDINALITY	RELTYPE
PART_DETAIL	3	0	Base

ATTRIBUTES

RELNAME	ATTRNAME	DOMNAME
PART_DETAIL	MAJOR_PART#	P#
PART_DETAIL	MINOR_PART#	P#
PART_DETAIL	QUANTITY	QTY

Note that the CARDINALITY entry in the RELATIONS relation for the RELATIONS relation itself will also need to be incremented by one. Also, we have shown cardinality of PART_DETAIL as 0, not 7 (despite the fact that in Question our relation PART_DETAIL contains 7 tuples), because, of course, the relation will be empty when it is first created.

- (c) DESTROY BASE RELATION PART_DETAIL;
DESTROY DOMAIN QTY;
DESTROY DOMAIN P#;

Question 4. A relation with an empty set of Tuples is perfectly reasonable, and indeed common (it is analogous to file with no records). In particular, of course, every base relation has an empty set of tuples when it is first created. It is usual, though a trifle imprecise, to refer to a relation with no tuples as an empty relation.

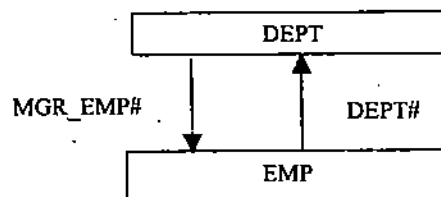
Question 5. What is perhaps less immediate obvious is that a relation with an empty set of attributes is perfectly reasonable too! In fact, such relations turn out to be of crucial importance - much as empty sets are crucially important in general set theory, or zero is important in ordinary arithmetic.

- (i) CREATE SECURITY RULE SEC51
GRANT RETRIEVE ON STUDENT TO Anu;
- (ii) CREATE SECURITY RULE SEC52
GRANT INSERT, DELETE ON STUDENT TO Smita;
- (iii) CREATE SECURITY RULE SEC53
GRANT RETRIEVE
ON STUDENT WHERE STUDENT.STUDENT_ID = USER ();
TO ALL;
- (iv) CREATE SECURITY RULE SEC54
GRANT RETRIEVE, UPDATE (STIPEND, FEES)
ON STUDENT TO Nath;
- (v) CREATE SECURITY RULE SEC55
GRANT RETRIEVE (STUDENT_ID, STIPEND, FEES)
ON STUDENT TO Ram;
- (vi) CREATE SECURITY RULE SEC56
GRANT RETRIEVE (STUDENT_ID, STIPEND, FEES)
UPDATE (STIPEND, FEES)
ON STUDENT TO John;

Check Your Progress 2

- Question 1.
1. Accepted
 2. Rejected (candidate key uniqueness violation)
 3. Rejected (violates RESTRICTED update rule)
 4. Accepted (supplier S3 and all shipments for supplier S3 in the relation SPJ are deleted).
 5. Rejected (violates RESTRICTED delete rule as P2 is present in relation SPJ).
 6. Accepted (project J4 and all shipments for project J4 from the relation SPJ are deleted).
 7. Accepted
 8. Rejected (candidate key uniqueness violation as tuple S5-P5-J7 already exists in relation SPJ).
 9. Rejected (referential integrity violation as there exists no tuple for J8 in relation J).
 10. Accepted
 11. Rejected (referential integrity violation as there exists no tuple for P7 in relation P).
 12. Rejected (referential integrity violation – the default project number jii does not exist in relation J).

Question 2. The diagram could be shown as below:



Since the database involves a referential cycle (there is a referential path from each of two relations to itself), either one of the two foreign keys must be permitted to accept nulls or the integrity checking must be deferred. The data definition (an outline) is:

```

CREATE BASE RELATION DEPT
  (DEPT#....,
   ....,
   MGR_EMP# ...)
  PRIMARY KEY (DEPT#);
  
```

```

CREATE BASE RELATION EMP
  (EMP#....,
   ....,
   DEPT# .....)
  PRIMARY KEY (EMP#)
  FOREIGN KEY (DEPT#) REFERENCES DEPT
  DELETE CASCADES
  UPDATE CASCADES;
  
```

Question 3.

Here also we show just the CREATE BASE RELATION statements

RDBMS
Terminology

```
CREATE BASE RELATION EMP
  (EMP#.....,
   ..... ,
   JOB ...)
  PRIMARY KEY (EMP#) ;
```

```
CREATE BASE RELATION PGMR
  (EMP#.....,
   ..... ,
   LANG ..... )
  PRIMARY KEY (EMP#)
  FOREIGN KEY (DEPT# ) REFERENCES EMP
  DELETE CASCADES
  UPDATE CASCADES;
```

This example illustrates the point that a foreign key can in fact also be a candidate key of its containing relation. Relation EMP lists all employees, and relation PGMR lists just those employees that are programmers; thus, every employee number appearing in PGMR is also a foreign key, referring to the primary key of EMP. Note that the two relations can be regarded as representing, respectively, an entity supertype (employees) and an entity subtype (programmers). In fact, the example is typical of the way entity supertypes and subtypes would be represented in a relational database.

Note that there is another integrity constraint that also needs to be maintained in this example – namely, the constraint that a given employee must be represented in relation PGMR and only if the value of EMP.JOB for that employee is “Programmer”. This constraint is not a referential constraint.

UNIT 2 OVERVIEW OF LOGICAL DATABASE DESIGN

Structure

- 2.0 Introduction
- 2.1 Objectives
- 2.1 The Steps of Database Design
 - 2.2.1 Conceptual Design
 - 2.2.2 Schema Refinement
 - 2.2.3 Physical database Design and Tuning
- 2.3 ER Model
- 2.4 ER Model Basics
 - 2.4.1 Entity
 - 2.4.2 Entity Type and Entity Set
- 2.5 Attributes
 - 2.5.1 Attribute
 - 2.5.2 Key Attributes in Entity Types
 - 2.5.3 Composite vs. Simple attributes
 - 2.5.4 Single vs. Multivalued Attributes
 - 2.5.5 Derived vs. Stored Attributes
 - 2.5.6 Null Values
 - 2.5.7 Value Sets of Attributes
- 2.6 Relationship
 - 2.6.1 Relationship
 - 2.6.2 Degree of Relationship Type
 - 2.6.3 Structural Constraints
- 2.7 Weak Entities
- 2.8 Components of an E-R diagram
- 2.9 ER Diagram Development Examples
- 2.10 Summary

2.0 INTRODUCTION

When an organisation decides to use database system then it is required to study the needs of the organisation. In the case of a small organisation with few users, having small volume of data, with no need for online query or update, a database system may not be necessary. In an organisation with a large number of users, and where decision making is distributed, the need for concurrent access to shared data is addressed by a database system.

In an organisation where large number of users and applications exist, the database system provides data independence, insulating these users and applications from changes. For the database to meet its objective, its design must be complete and consistent. All the significant inputs should be used in the design process, including the inputs of the users. The external schema allows multiple views of the data contained in the database. Designing a database system requires gathering details about the applications and transactions that are to be supported and the classes of users that will use the system.

The principal topic of this unit is entity-relationship model, which is basic step in the database design. The entity-relationship (E-R) data model is based on a perception of a real world that consists of a set of basic objects called entities, and of relationships among these objects. The E-R model is one of several semantic data models; the semantic aspect of data model lies in the attempt to represent the meaning of the data. The E-R model is extremely useful in mapping the meanings

and interactions of real-world enterprise onto a conceptual schema. Because of this utility, many database-design tools draw on concepts from the E-R model.

2.1 OBJECTIVES

At the end of this unit, you should be able to:

- recognise and define various entities in the enterprise;
- define relationship among various entities;
- gather information about these entities and relationships that should be stored in the database;
- recognise integrity constraints or business rules that hold;
- make the Entity-Relationship diagram.

2.2 THE STEPS OF DATABASE DESIGN

The three basic steps involved are:

2.2.1 Conceptual Design

It includes the several semantic data models. The Semantic aspect of the model lies in the attempt to represent the meaning of the data. For the conceptual design, the E-R models are used.

Entity Relation diagrams provides you with a graphical methodology for designing the structure of your database. There is a well-defined mapping from E-R diagrams into a relational schema making E-R diagrams a very practical tool for Database design.

2.2.2 Schema Refinement

It is called Normalisation. It checks relational schema for redundancies and related anomalies. Normalisation is a very important component of good database design. Redundancies in data not only makes databases larger, but also creates the potential for update and delete anomalies which can cause a database to become inconsistent. The normalisation will be discussed in the next unit.

2.2.3 Physical database Design and Tuning

It considers typical workloads and further refines the database design. Normally, it is handled by a Database Administrator.

2.3 ER MODEL

The ER model is a:

- High-level conceptual data model.
- Frequently used for conceptual design of database applications.

The E-R diagram (ERD) was originally proposed by Peter Chen for the design of relational database systems. The E-R model is extremely useful in mapping the meaning and interactions of real-world enterprises. Before making E-R diagram, we identify the following:

1. What are entities and relationships in the enterprise?
2. What information about these entities and relationships should we store in the database?
3. What are the integrity constraints or business rules that hold?

So a set of primary components is identified for the ERD; data objects, attributes, relationships and various type indicators. The primary purpose of the ERD is to represent data objects and their relationships.

For example, in general we can represent two entities and relationship between them as follows:

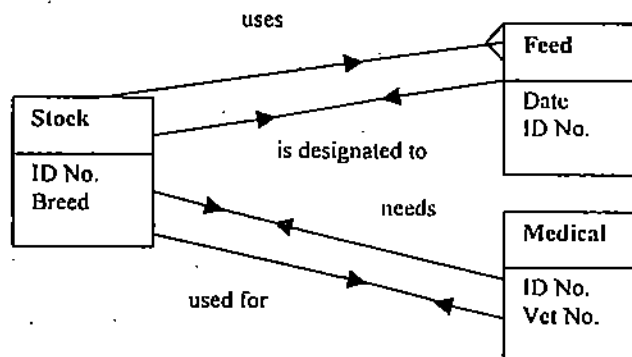


Figure 1: Representing relationship between various entities

Large database management system vendors implements ER model, example Oracle CASE, ERWIN etc.

2.4 ER MODEL BASICS

2.4.1 Entity.

An entity is a data object, a thing, an occurrence or event, a role, an organisational unit, a place or a structure in the real world that is distinguishable from all other objects.

For example, Employee or Department in a Company or an Enterprise is different entities.

The other examples could be:

Data Object - Anything that produces or consumes information.

Thing - A report or a display.

Occurrence - A Telephone Call.

Event - An Alarm.

Role - Salesperson.

Organisational Unit - Accounting Department.

Place - Warehouse.

Structure - File.

An entity may be concrete like person or Book, or it may be abstract like holiday or concept.

2.4.2 Entity Type and Entity Set

Entity Type: It defines a set of entities that have the same attributes

- Represented as a rectangle in ER diagram.
- Entity type describes the schema for a set of entities that share the same structure.

Entity Set: An entity set is a collection of similar entities, which share same properties or attributes. Each entity set has a key and each attribute has a domain. The individual entities that constitute a set are said to be extension of the entity set.

Example is set of all employees.

We can assume Employee of a university as one of the Entity set and Mr. ABC and Mr. XYZ as extensions of the above entity set.

2.5 ATTRIBUTES

2.5.1 Attribute

Attributes are descriptive properties possessed by each member of entity set. They can be used to—

- name an instance of an entity;
- describe the instance;
- makes reference to another instance in another table.

Attributes Formally

- Defined as function $A | A:E \rightarrow P(V)$, where $P(V)$ represents powerset of value set V .
- Powerset is set of all subsets
- Value of attribute A for entity e is $A(e)$
- $A(e)$ must be a singleton for single-valued attributes
- For a composite attribute A the value set V is the Cartesian product of sets: $P(V_1), P(V_2), \dots, P(V_n)$ where V_1, V_2, \dots, V_n are values sets for the simple components attributes that form A :

$$V = P(V_1) \times P(V_2) \times \dots \times P(V_n)$$

In some cases values for the identifier are unique, although this is not a requirement. For example, the various entities along with their attributes can be listed as follows:

Employee
Employee-Id
Name
Address
Tel No.
Date of Birth
Date of Joining
Department
Designation
Salary

Department
Dept-Id
Dept-name
Budget

ACCOUNT
Name
Account No.
Address
Phone No.
Balance
YTD Deposits
YTD Withdraws

CHEQUE
Amount
Payee
Check No.
Date

2.5.2 Key Attributes in Entity Types

- Key (or uniqueness) constraints can be applied to entity types;
- Key attribute's values are distinct for each individual entity;
- Each key attribute has its name underlined;
- Key must hold for every possible extension of the entity type.

2.5.3 Composite vs. Simple attributes

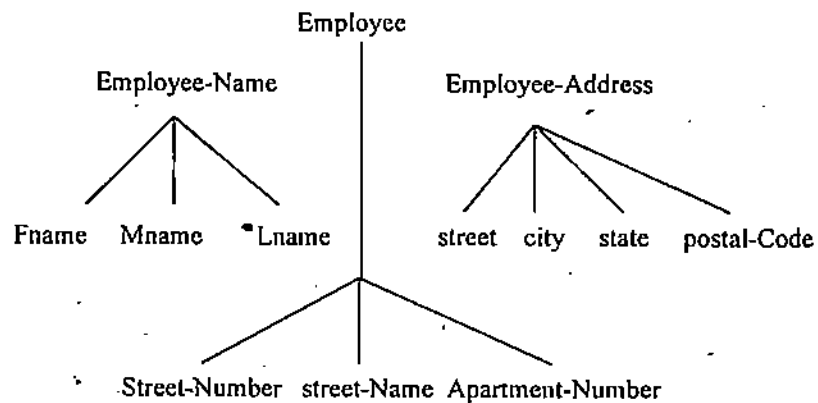
Simple attributes are the one's that can not be divided into sub-parts, for example, Employee-Id in the entity Employee, Account No. in the entity Account.

On the other hand Composite attributes can be divided into subparts. Example, Name (First Name, Middle Name, Last Name), Date (Day, Month, Year) etc.

Entity Set:

Composite Attributes:

Component Attributes:



2.5.4 Single vs. Multivalued Attributes

Single valued attributes have single value for the particular entity, while multivalued have set of values for particular entity.

Example, in the entity Employee, Employee-Id is the single valued attribute. Other examples for single valued attributes can be age, car, color etc.

Telephone Number is the multivalued attribute as an employee can have zero, one or more telephones.

We can represent Attributes as follows:

- Parenthesis (()) for composite attributes; and brackets ({ }) for multivalued attributes;
- Assume a person can have more than one residence and each residence can have multiple telephones, then the attribute AddressPhone can be represented as:

```
{AddressPhone({Phone(AreaCode,PhoneNum)},Address(Street(Street-Number, Street-Name,  
Apartment-Number),City,State,PostalCode) ) }
```

2.5.5 Derived vs. Stored Attributes

Value of derived type of attribute can be derived from the values of other related attributes or entities, for example, the age of an entity Employee can be derived from his Date of Birth.

On the other hand stored attributes are the attributes whose values are directly stored in the entity set, for example, Date of joining, Address etc.

2.5.6 Null Values

A Null value is used in the place when an entity does not have any value for an attribute, for example, in entity Employee, Telephone Number can have the null value as an employee may not have any telephone. Certain attributes like Employee-Id, Name can never take null value.

2.5.7 Value Sets of Attributes

Value Sets: Value sets specify the set of values that may be assigned to a particular attribute of an entity. Value set is the same as domains. They are not displayed on the ER diagram.

For example, age is between 16 and 70. S/he can join the company at the age of 16 years or onwards and at the age of 70 s/he gets retired.

Also Employee Name is the value set, as it is set of strings of alphanumeric characters.

2.6 RELATIONSHIP

2.6.1 Relationship

It is defined as association among two or more entities. Example, Rakesh works in Pharmacy department. Here Rakesh and Pharmacy department, are two different entities. The relationship among them is that one works in another.

In general sense we can say that an employee works in some department.

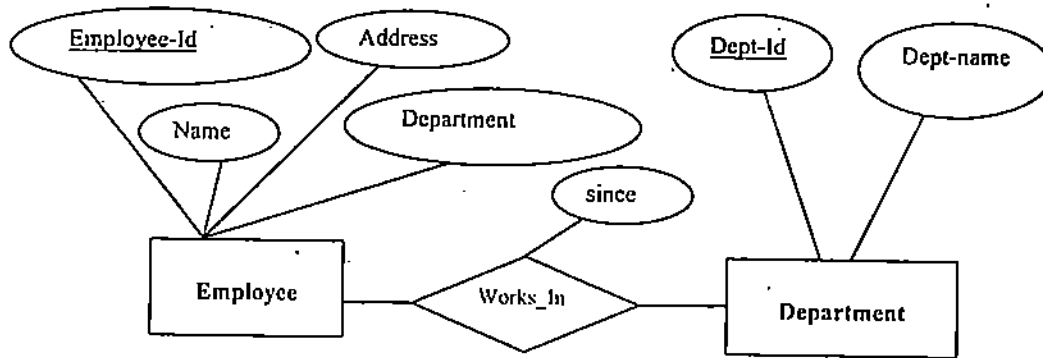


Figure 2: ER Diagram for representing employee works_in department

Relationship Set: Relationship set is defined as the collection of similar relationships. An n-ary relationship set R relates n entity sets E1 ... En; each relationship in R involves entities e1 (in E1), ..., en (in En). Relationship sets can also have descriptive attributes (example, the since attribute of Works_In).

In translating a relationship set to a relation, attributes of the relation must include:

- Keys for each participating entity set (as foreign keys). This set of attributes forms superkey for the relation.
- All descriptive attributes.

The following SQL statement will create the table works_in

```

CREATE TABLE Works_In (
Employee-Id INTEGER,
Dept-Id CHAR(6),
since DATE,
PRIMARY KEY (Employee-Id, Dept-Id),
FOREIGN KEY (Employee-Id) REFERENCES Employee,
FOREIGN KEY (Dept-Id) REFERENCES Department)
    
```

- Relationship Type defines an association between n entity types E1, ..., En, and also defines the structure of the relationship set
- A relationship set is a subset of the Cartesian product: E1 × E2 × ... × En.
- Each Ei is said to 'participate' in the relationship type.

2.6.2 Degree of Relationship Type

The degree of a relationship type is the number of participating entity types.

For example, in Figure 2, WORKS_IN, is a relationship of degree 2 i.e., it is binary relationship.

The relationship Supply among entities - supplier/part/project is of degree 3 therefore it is ternary relationship.

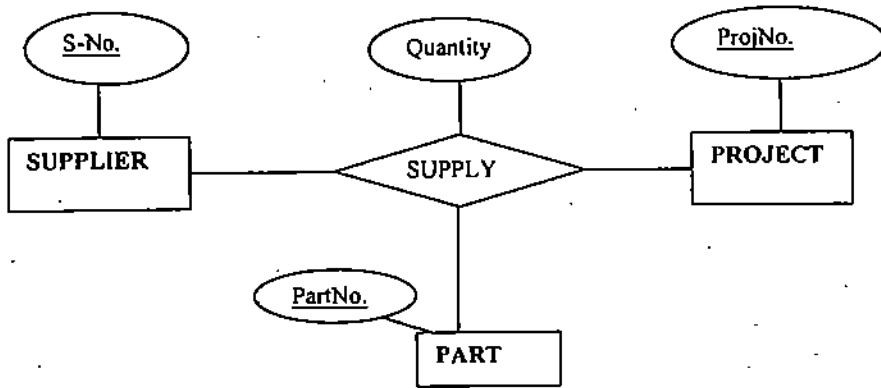


Figure 3: ER Diagram for the relationship SUPPLY among entities SUPPLIER, PROJECT and PART

Same entity type could participate in different relationship types, or in different 'roles' in same relationship type. Each entity type that participates in a relationship type plays a particular role in the relationship type. The role name signifies the role that a participating entity from the entity type plays in each relationship instance.

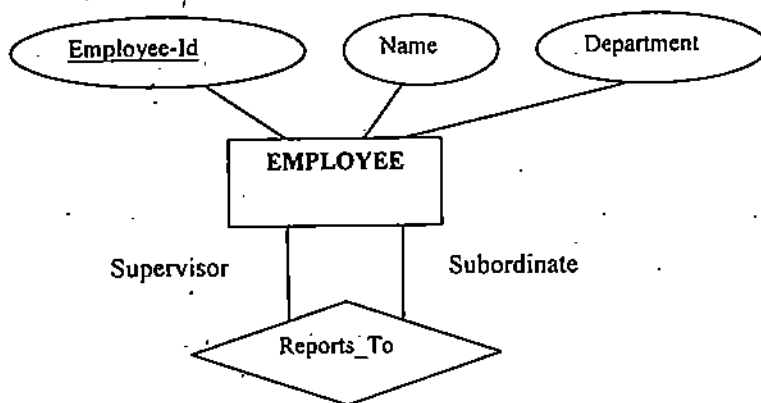


Figure 4: ER Diagram for the relationship Reports_To representing various roles of employee in an organisation

2.6.3 Structural Constraints

Two kinds of structural constraints can be placed on relationship types. These are:

- Cardinality Ratio;
- Participation Constraints.

Cardinality Ratio

It defines the number of entities to which another entity can be associated via a relationship set. It is sometimes referred to as Mapping Cardinalities.

For a binary Relationship set R between entity sets A and B, the mapping Cardinalities must be one of the following:

One-to-one: An entity in A is associated with at most one entity in B, and an entity in B is associated with at most one entity in A.

For example, in an ideal society, A relationship between husband and wife is One-to-one.



Figure 5: ER Diagram to explain one-to-one relationship

One-to-Many: An entity in A is associated with any number of entities in B. An entity in B, however, can be associated with at most one entity in A.

For example, each department has at most one manager, according to the key constraint on the relation *Manages*, but a manager can manage one or more departments. Thus this is an example of One-to-many relationship.

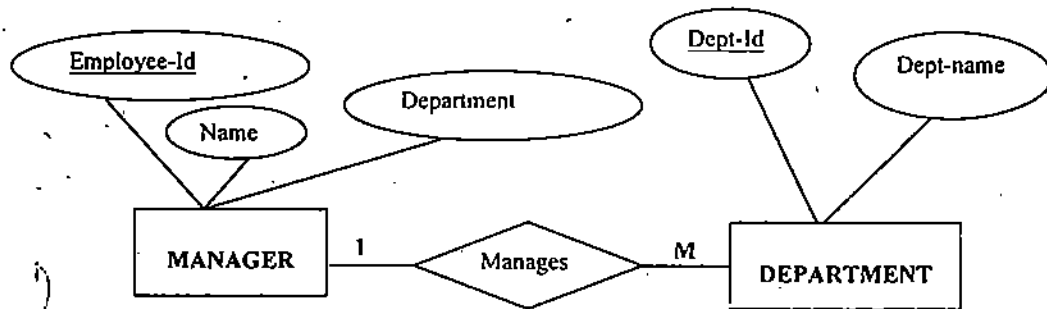


Figure 6: ER Diagram to explain one-to-many relationship

Many-to-one: An entity in A is associated with at most one entity in B. An entity in B, however, can be associated with any number of entities in A.

For example, two sisters have a common father, thus, forming an example of many-to-one relationship.

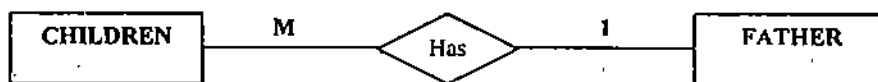


Figure 7: ER Diagram to explain many-to-one relationship

Many-to-Many: An entity in A is associated with any number of entities in B and an entity in B is associated with any number of entities in A.

For example, to understand Many-to-many relationship we can again consider the example of *Works_In* relationship in Figure 2. An employee can work in many departments and a department can have many employees.

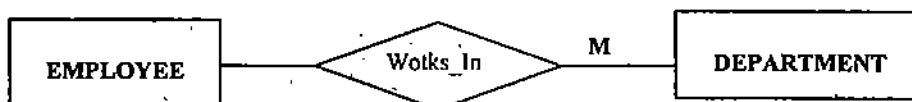


Figure 8: ER Diagram to explain many-to-many relationship

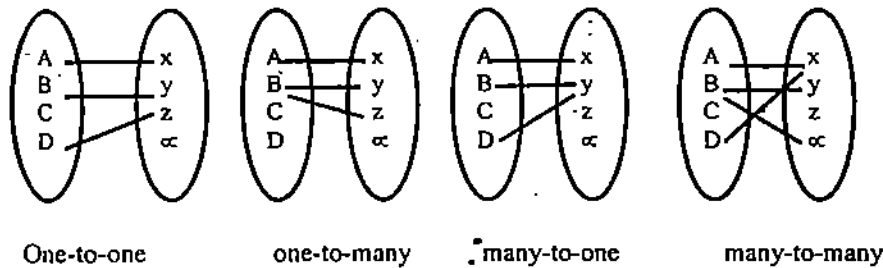


Figure 9: Relationship Cardinality

Translating ER Diagrams with Key Constraints

Before translating ER Diagram, we need to discuss that why do we need key constraints

Consider the situation: A father has two daughters, now there has to be some way for father to identify his daughters uniquely. Thus he gives them a unique name, thus in this case name becomes the key.

Thus we can very well observe that in Many side of the relationships there has to be some key for the unique identification.

We can see it in the following Diagram:

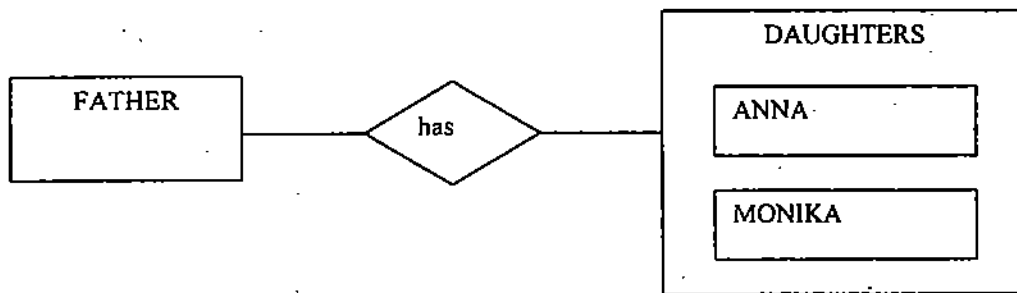


Figure 10: Importance of key in one-to-many relationship

Our next step is to understand how do we translate ER Diagrams with key constraints?

Map relationship to a table

Note that in the example of Works_In relationship, on assuming the constraint that an employee can work only in one department, Dept-Id becomes the key and we will have to make separate tables for Employees and Departments. (Please refer to figure 2) These tables look like follows:

The table Employee

Employee-Id	Name	Address	Department
1918	Amit	GH-9, ND-87	SND
2223	Mini	2, LIC Colony, ND-86	NMG
1672	Vineet	303, Bhera Enclave, PV, ND	VOIP

The table Department

Dept-Id	Dept-name
SND	Satellite Networking Division
NMG	Network Management Group
VOIP	Voice over Internet Protocol

Since each employee works in a unique department, we can combine Employee and Departments, thus, we have the table **Works_In**.

The table **Works_In**

Employee-Id	Dept-Id	Since
1918	SND	02-10-98
2223	NMG	07-05-95
1672	VOIP	22-07-96

The SQL for creating the tables **Employee**, **Department** and **Works_In** will be like as follows:

```
CREATE TABLE Employee(
Employee-Id INTEGER,
Name CHAR(25),
Address CHAR(50),
Department CHAR(6))
```

```
CREATE TABLE Department(
Dept_Id CHAR(6),
Dept-name CHAR(25) )
```

```
CREATE TABLE Works_In(
Employee-Id INTEGER,
Dept_Id CHAR(6),
since DATE,
PRIMARY KEY (Employee-Id, Dept_Id),
FOREIGN KEY (Employee-Id) REFERENCES Employee,
FOREIGN KEY (Dept_Id) REFERENCES Department)
```

Participation Constraints

The participation of an entity set E in a relationship set R is said to be total if every entity in E participates in at least one relationship in R. If only some entities in E participate in relationships in R, the participation of entity set E in relationship R is said to be partial.

Let us try to understand total relationship. Let us consider each department has a unique manager, we could instead combine Manages and Departments and have the table **Manages**.

The table **Manages**

Manager-Name	Employee-Id	Dept-Id	Since
Pratyush DasGupta	1002	SND	02-10-98
Mamta Bansal	1076	NMG	07-05-95
Mohan Hebbar	1115	VOIP	22-07-96

The SQL for creating the table will be like as follows:

```
CREATE TABLE Manages(
Manager-Name CHAR(25),
Employee-Id INTEGER,
Dept-Id CHAR(6),
since DATE,
```

```
PRIMARY KEY (Employee-Id, Dept-Id),
FOREIGN KEY (Employee-Id) REFERENCES Employees,
FOREIGN KEY (Dept-Id) REFERENCES Departments)
```

Let's try to find, does every department has a manager?

If answer is yes i.e., Every department has a manager, then this is a participation constraint: the participation of Departments in Manages is said to be total as every Dept-Id value in Departments table must appear in a row of the Manages table (with a non-null Employee-Id value).

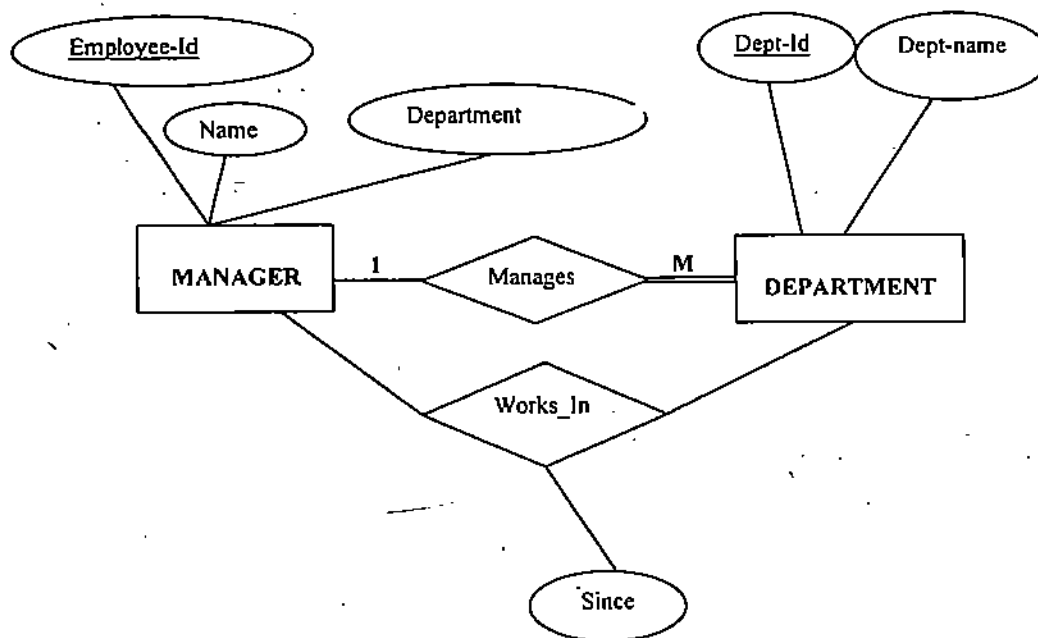


Figure 11: ER Diagram to explain total participation

Example of Partial Relationship

As we know that there are some schemes like LTC (Leave Travel Certificate) etc. These schemes can be called as Flexible Benefit Schemes. An employee can claim these FBSs if his or her spouse is not claiming it. Thus, out of husband and wife only one is eligible for FBP. Thus, we can define a relationship is eligible for between set of employees and FBP. Out of set all the employees in the company all employees may or may not be eligible for FBP, thus, this relationship is a partial relationship.

Participation Constraints in SQL

We can capture participation constraints involving one entity set in a binary relationship, but little else (without resorting to CHECK constraints).

```

CREATE TABLE Dept_Mgr ( Dept-Id CHAR(6), Dept-name CHAR(25), Employee-Id INTEGER
NOT NULL, since DATE, PRIMARY KEY (Dept-Id), FOREIGN KEY (Employee-Id)
REFERENCES Employees, ON DELETE NO ACTION)
    
```

Existence Dependencies

Another class of constraint is existence dependencies. It can be defined as - 'If the existence of entity X depends on the existence of entity Y, then X is said to be existence dependent on Y. Practically, if Y is deleted, so is X.'

Let us consider the relationship CCI between the entities CANDIDATE, COMPANY and INTERVIEW. A candidate is offered a job in the company if he appears in the interview and

qualifies it. Thus JOB_OFFER is an entity, which is dependent on the existence of the entity INTERVIEW. If there is no INTERVIEW then there is no JOB_OFFER. Thus, entity JOB_OFFER is existentially dependent on entity INTERVIEW.

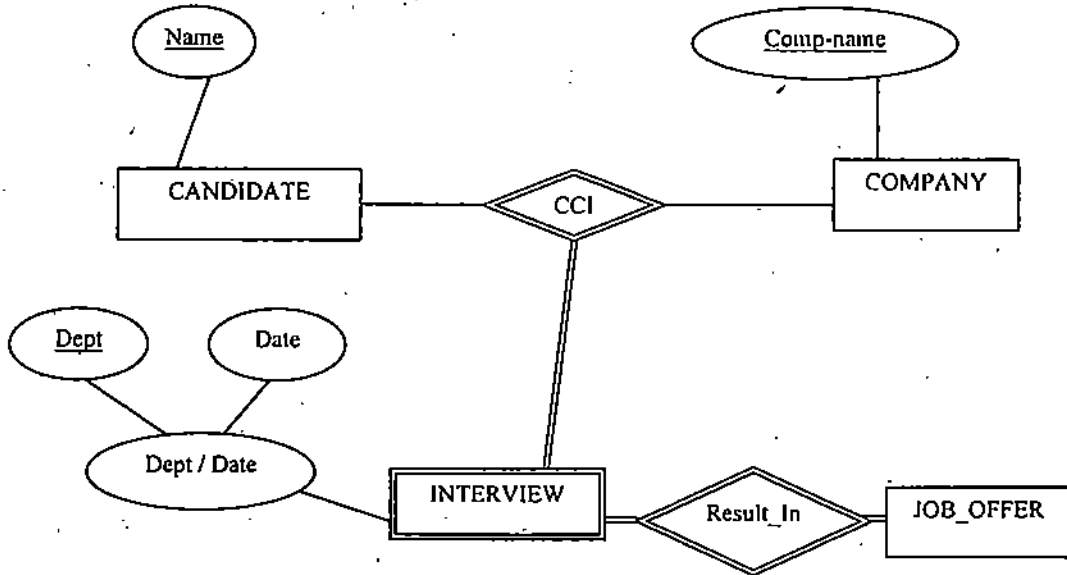


Figure 12: ER Diagram to explain existential dependency.

2.7 WEAK ENTITIES

An entity set may not have sufficient attributes to form a primary key. Such an entity set is termed as weak entity set. An entity that has a primary key is termed as a strong entity set. A weak entity can be identified uniquely only by considering the primary key of another (owner) entity.

- Owner entity set and weak entity set must participate in a one-to-many relationship set (i.e., one owner, many weak entities).
- Weak entity set must have total participation in this identifying relationship set ('existence dependency').

Entities may not be distinguished by their attributes but by their relationship to another entity. Let us establish a relationship, DEDUCTIONS, between the entity EMPLOYEE and DEPENDENTS as shown in Figure 13. In this case the instances of the entity from the set DEPENDENTS are distinguishable only by their relationship with an instance of an entity from the entity set EMPLOYEE. The relationship set DEDUCTIONS is an example of an identifying relationship and the entity set DEPENDENTS is an example of a weak entity.

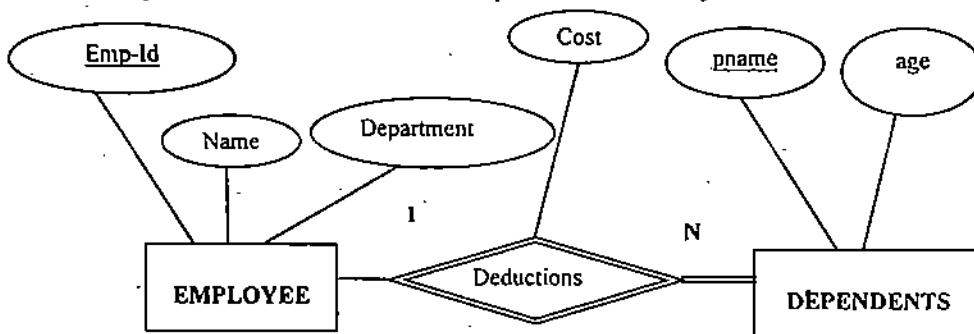


Figure 13: Converting an attribute association to a relationship and explaining weak entity set.

Identifying Relationship

- Identifying relationship is the relation type that relates a weak entity to its 'identifying owner'.
- Not every existence dependency results in a weak entity.
- Although a weak entity set does not have a primary key, we nevertheless need a means to distinguish among all those entities in the entity set that depend on one particular strong entity. The discriminator of a weak entity set is a set of attributes that allows this distinction to be made. The discriminator of a weak entity set is also called the partial key of the entity set.
- Primary Key is set of attributes that uniquely identify weak entities related to the same owner entity. It is formed by the Primary Key of the strong entity set on which the weak entity set is dependant, plus the weak entity set's discriminator.

For instance, let us assume The EMPLOYEE 1672(Vineet Anand) has two DEPENDENTS, Daksha Anand, his spouse and Swastik Anand, his son. These are distinct and can be distinguished from each other. The organisation can have another Anand in its employ (say Bharat Anand with Employee-Id as 1888), who has dependents Meena Anand, spouse and Sumit Anand, son.

Now note that the two instance (Vineet Anand, son) of the weak entity set DEPENDENTS associated with different instances of the strong entity set EMPLOYEE are not distinguishable from each other. They are nonetheless distinct because they are associated with different instances of the strong entity set EMPLOYEE. The primary key of a weak entity set is thus formed by using the strong entity set to which it is related, along with the discriminator of the weak entity. We rule out the case where a dependent such as Sumit Anand is the son of two different employes, namely his mother and father, since only one of them will claim him a deduction. However, if we allow this possibility, the relationship between EMPLOYEE and DEPENDENTS becomes many-to-many.

Translating Weak Entity Sets

Weak entity set and identifying relationship set is translated into a single table, and when the owner entity is deleted, all owned weak entities must also be deleted.

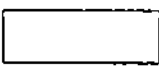


The SQL query for relationship with weak entity can be given as follows:

```
CREATE TABLE Deduction (pname CHAR(25), age INTEGER, cost REAL, Employee-Id
INTEGER NOT NULL, PRIMARY KEY (pname, Employee-Id), FOREIGN KEY (Employee-
Id) REFERENCES Employees, ON DELETE CASCADE)
```

2.8 COMPONENTS OF AN E-R DIAGRAM

Components of an E-R diagram

Following are the various components of an E-R diagram:

Symbol Name	Symbol	Meaning
Rectangles		It represents entity sets
Double Rectangles		It represents weak entity sets
Diamonds		It represents relationship sets

Symbol Name	Symbol	Meaning
Double Diamonds		It represents identifying relationship sets or types
Ellipses		It represents attributes
Underlined Ellipses		It represents key attributes
Double Ellipses		It represents multivalued attributes
Dashed ellipses		It represents derived attributes
		It represents composite attributes
Lines		It represents link attributes to entity sets and entity set to relationship sets
Double lines		It represents total participation of an entity in a relationship set i.e., total * participation of E2 in R
Numbers 0,1,N		It represents Cardinality ratio 1:N for E1:E2 in R
(min, max)		It represents structural constraints (min, max) on participation of E in R.

2.9 ER DIAGRAM DEVELOPMENT EXAMPLES

Example 1: A Company ABC develops applications for the clients. Same ideas apply when a company develops an application for their own personal use. Thus, the company can work in "user mode" and "developer mode". Try to find out all possible entities and relationships among the entities. Give a step-by-step procedure to make an E-R diagram for the process of the company when they develop an application for the client and use the diagram to derive an appropriate base table.

Solution: We are going to design a database to hold information about the clients and the applications of a system. We will call this CAPO, for Client Application Package Organiser.

First let us define the entities of the system. At present we have two objects—namely **Client** and **Application**. Each of these is a noun, and thus eligible to be entities in the database.

CLIENT

APPLICATION

Figure 14: Drawing Entities as Boxes

Taking these in turn, let's see if we can determine the correct entity types. Client is independent of anything else, and so is Application. So the entities clients and applications form an entity set.

Let us now consider what the relationship is between these two entities, if any. Obviously, the relationship among the entities depends on precise interpretation of spoken (or written) requirements, and we must pay meticulous attention to the contextual semantics - for example, just what do we mean by an application?

Is Accounts Receivable an application, or is the AR system installed at each client site regarded as a different application? Can the same application be installed more than once at a particular client site?

Before we answer these questions, have you noticed that another entity is trying to get into the act here? The client site seems to be another candidate entity. This is the kind of thing you need to be sensitive to at this stage of the process.

Let's deal with the relationship between Client and Site before coming back to Application. It is often easier to tackle what seems likely to prove simple before trying to resolve the apparently complex.

Each Client can have many sites, but each site belongs to one and only one client: This is a good example of a existential dependency and one-to-many relationship, and illustrates the precision and clarity of expression we employ when we read the following diagram.

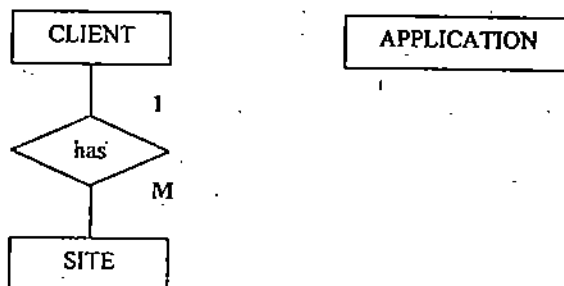


Figure 15: Adding a One-to-Many Relationship

Now the question arises that what entity type is Site?

We cannot have a site without a client, if there exist any site without a client then who would pay the company? Obviously a company gets the payment from the clients.

Now let's try and relate Application to the others: we would like to install the same application at many client sites. We also hope to install more than one application at some of these sites. That describes a many-to-many relationship between Site and Application:

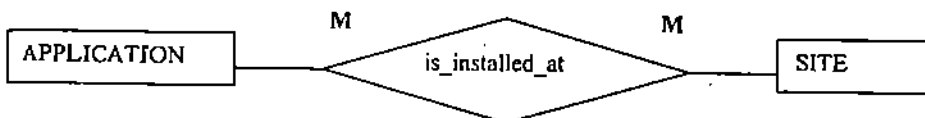


Figure 16: A Many-to-Many Relationship

To decompose the M:M relationship into 1:M relationship, we add a new entity called Installation in the above figure and by doing so we get the following figure:

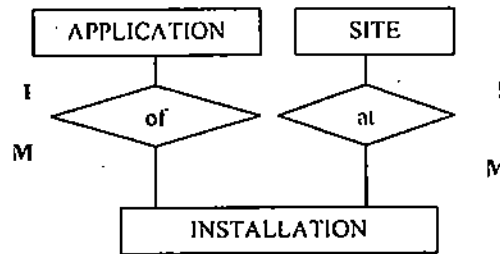


Figure 17: Resolving The Many-to-Many Relationship

Defining relationships in above figure:

- of - relationship describing installation of applications, and
- at - relationship describing installation at sites.

Note that entities can arise in one of two ways. Either we draw them because they were named in the specification, or as a result of resolving a M:M, as in this case. When we create a new entity in this way, we have to find a suitable name for it. This can often be based on the verb used to describe the M:M. Thus, from the statement we can install the application at many sites we get the name Installation.

Now we need to determine the correct type for the new entity. Always get the user to do this. Use the non-technical statement for each entity type to help decide which one describes it correctly. What we are concerned with here is the most effective way of uniquely identifying each record of this type. How do we want to identify each Installation?

Is an Installation independent of any other entity? i.e., can an entity Installation exist without being associated with the entities Client, Site and Application?

In this case, there cannot be an Installation until we can specify the Client, Site and Application. But since Site is existentially dependent on Client or we can say that Site is subordinate to Client, that means that Installation is a Combination entity type, identified by (Client) Site (it means Client or Site) and Application. We do not want to allow more than one record for the same combination of site and application.

But what if we also sell multiple copies of packages, and need to keep track of each individual copy (license number) at each site? The answer is that we need another entity. We may even find that we need separate entities for Application and Package. This will depend on what attributes we want to keep in each - how far our requirements differ in respect of each of these.

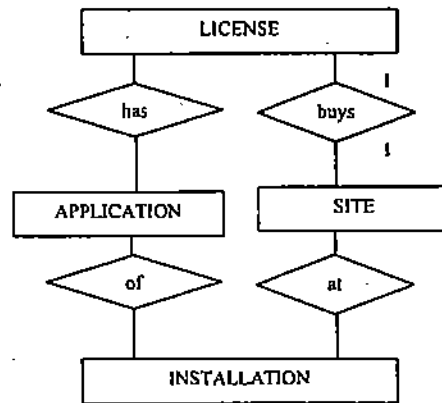


Figure 18: Adding More Entities

Defining Relationships in above figure:

has - relationship describing that each application has one or more licenses,

buys - relationship describing each site buys the license copies.

We might decide that License should be subordinate to Package: the best unique identifier for the entity could be Package ID and license serial number. We could also define License as identified by Client Site, Application and License Copy (serial number).

The only objection to the subordinate identification is that one of the key elements has an externally assigned value - we do not issue license numbers, and thus have no control over their length and data type (up to 15 mixed alpha-numeric characters?) nor even over their uniqueness and unchangeability. It is always safer to base primary keys on internally assigned values.

What if we make License subordinate to Package, but substitute our own Copy serial number for the software manufacturer's License number? It seems that the Client Site is not an essential part of the identifier. Come to think of it, we sometimes buy copies of packages for stock, and later sell them. And the client is quite free to move the copy of the package to another site. For both of these reasons, we should definitely not make Client/Site part of the primary key of License.

The above discussion is trying to emulate the kind of mental acrobatics you have to be prepared to undertake when designing the database, and particularly when typing the entities.

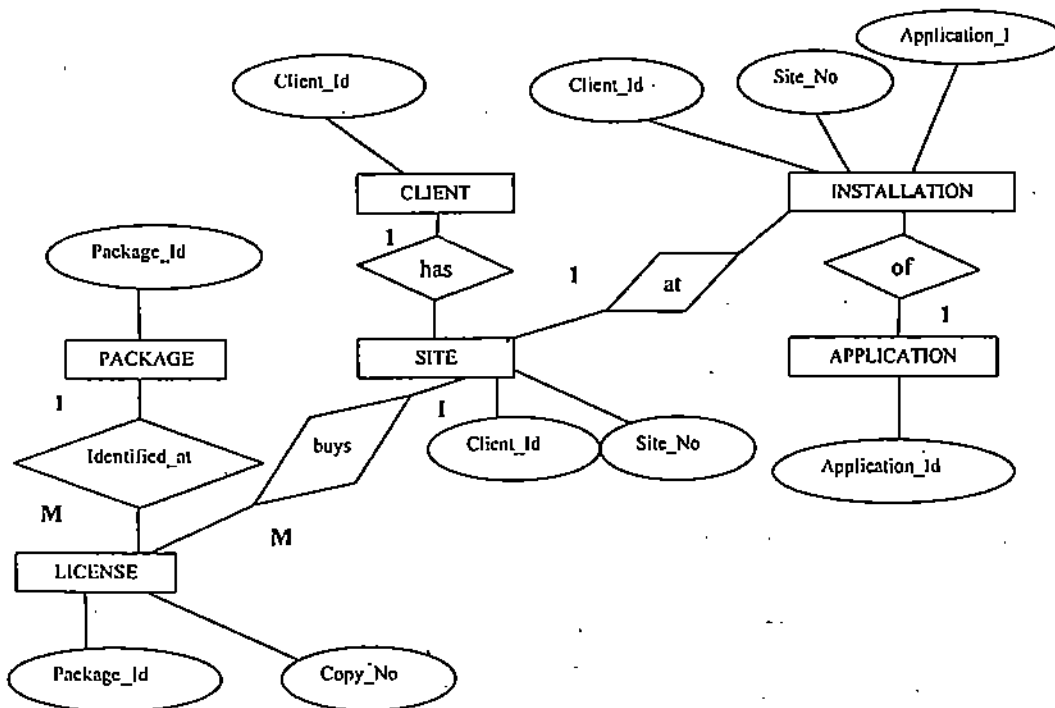


Figure 19: The Entity Types along with the attributes and keys

Let us list each entity identified so far, together with its entity type, primary keys, and any foreign keys.

Entity	Type	Primary Key	Foreign Keys
Client	Independent	Client ID	
Site	Subordinate	Client ID, Site No	
Application	Independent	Application ID	
Package	Independent	Package ID	
Installation	Combination	Client ID, Site No, Application ID	
License	Subordinate	Package ID, Copy No	Client ID, Site No

Now we are ready to switch on the computer, fire up Windows and any RDBMS say MS-Access or Oracle, and start defining the tables.

Example 2: An employee works for a department. If the employee is a manager then he manages the department. As an employee the person works for the project, and the various departments of a company control those projects. An employee can have many dependents. Draw an E-R diagram for the above company. Try to find out all possible entities and relationships among them.

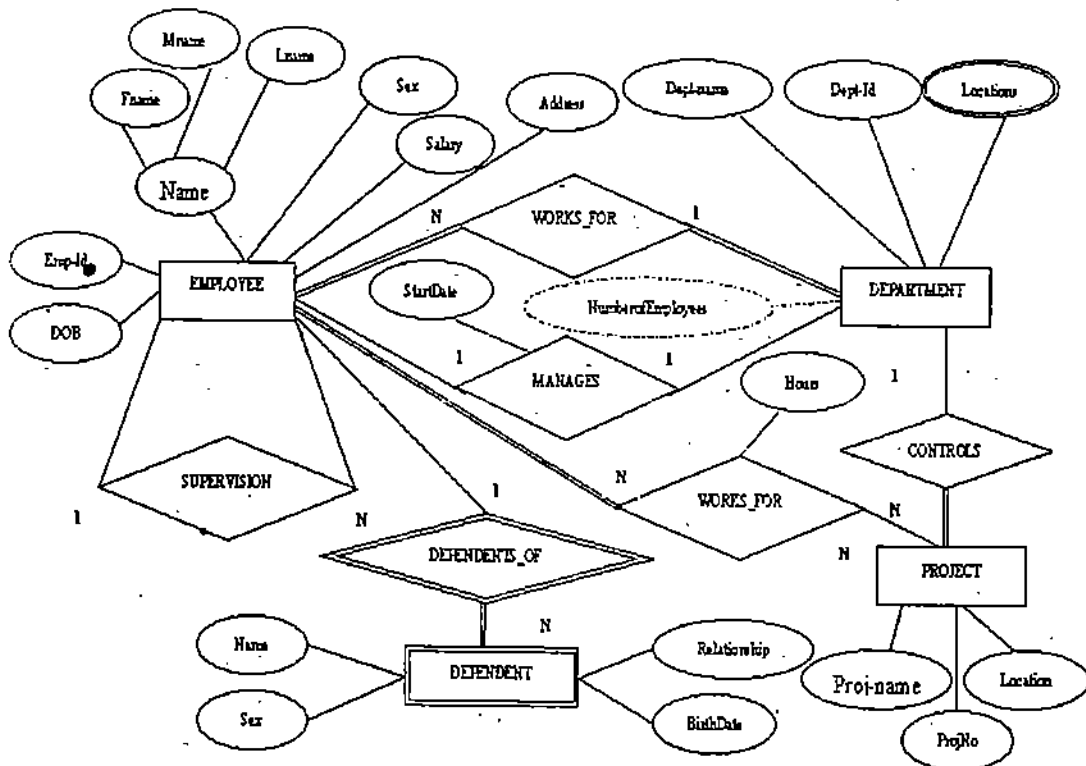


Figure 20: The E-R diagram for EMPLOYEE DEPARTMENT database

In the above E-R diagram, EMPLOYEE is an entity, who works for the department i.e., entity DEPARTMENT, thus works_for is many-to-one relationship, as many employees work for one department. Only one employee (i.e., Manager) manages the department, thus manages is the one-to-one relationship. The attribute Emp_Id is the primary key for the entity EMPLOYEE, thus Emp-Id is unique and not-null. The primary key for the entity DEPT are Dept-name and Dept_Id. Along with other attributes, NumberofEmployees is the derived attribute on the entity DEPT, which could be recognised the number of employees working for that particular department. Both the entities EMPLOYEE and DEPARTMENT participates totally in the relationship works_for as at least one employee works for the department, similarly an employee works for at least one department.

The entity EMPLOYEES work for the entity PROJECTS. Since many employees can work for one or more than one projects simultaneously, thus works_for is the N:N relationship. The entity DEPARTMENT controls the entity PROJECT, and since one department controls many projects, thus, controls in a 1:N relationship. The entity EMPLOYEE participates totally in the relationship works_for as at least one employee works for the project. A project has no meaning if no employee is working in a project.

The employees can have many dependents, but the entity DEPENDENTS can not exist without the existence of the entity EMPLOYEE, thus, DEPENDENT is a weak entity.

We can very well see the primary keys for all the entities. The underlined attributes in the eclipses represent the primary key.

Example 3: A supplier located in only one city, supplies various parts for the projects to different companies located in various cities. Let's call this database as supplier-and-parts database. Draw the E-R diagram for the supplier-and-parts database.

The E-R diagram for supplier-and-parts database is given as follows:

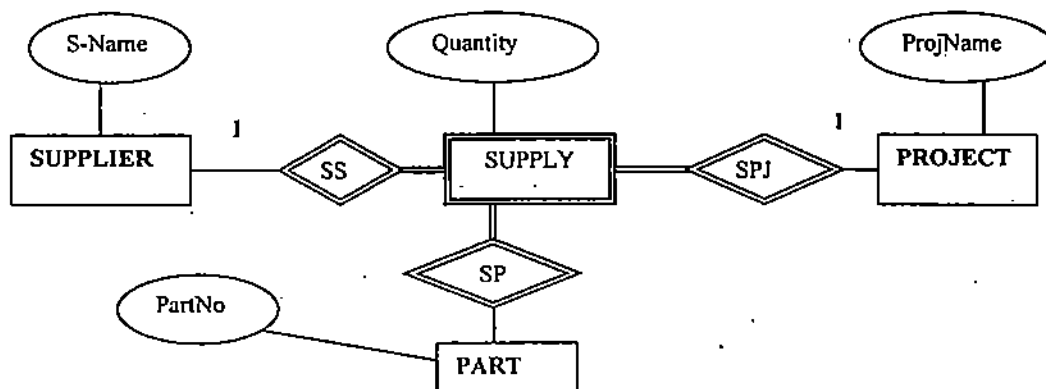


Figure 21: The E-R Diagram for supplier-and-parts database

2.10 SUMMARY

In this unit we discussed that **entity-relationship (E-R) data model** is based on a perception of a real world that consists of a set of basic objects called entities, and of relationships among these objects. E-R modeling is a **semantic modeling technique**, which is used primarily for the database-design process. It was developed to facilitate database design by allowing the specification of an enterprise schema. Such a schema represents the overall logical structure of the database. This overall structure can be expressed graphically by an E-R diagram.

An **entity** is a data object, a thing, an occurrence or event, a role, an organisational unit, a place or a structure in the real world that exists and is distinguishable from other objects. We accomplished the distinction by associating with each entity a set of attributes that describes the object. The **attributes** could be of various type— composite or simple, single or multivalued, derived or stored. A **null** value is used in the place when an entity does not have any value for an attribute.

A relationship is an association among several entities. The collection of all entities of the same type is an entity set, and the collection of all relationships of the same type is a relationship set. There are two kind of structural constraints that can be placed on relationship types. The cardinality ratio or mapping cardinalities express the number of entities to which another entity can be associated via a relationship set. They could be one-to-one, one-to-many, many-to-one and many-to-many.

Participation constraint on a relationship set is either total or partial depending on the fact whether all entities in the entity set participates in at least one relationship in the relationship set or only few entities in the entity set participates in the relationship in relationship set. Another form of constraint is existence dependency, which specifies that the existence of entity x depends on the existence of entity y.

An important task in database modeling is to specify how entities and relationships are distinguished. Conceptually, individual entities and relationships are distinct; from a database perspective, however, their difference must be expressed in terms of their attributes. To make such distinctions, we assign a primary key to each entity set. The primary key is a set of one or more attributes that, taken collectively, allows us to identify uniquely an entity in the entity set and a relationship in a relationship set. An entity set that does not have sufficient attributes to form a primary key is termed a weak entity set. An entity set that has a primary key is termed a strong entity set.

A database that conforms to an E-R diagram can be represented by a collection of tables. For each entity set and for each relationship set in the database, there is a unique table that is assigned the name of the corresponding entity set or relationship set. Each table has a number of columns, each of which has a unique name. Converting a database representation from an E-R diagram to a table format is the basis for driving a relational-database design from an E-R diagram.

UNIT 3 OVERVIEW OF NORMALISATION

Structure

- 3.0 Introduction
- 3.1 Objectives
- 3.2 Redundancy and Associated Problems
- 3.3 Role Of Normalization
- 3.4 Single-Valued Dependencies
- 3.5 Single Valued Normalizations
 - 3.5.1 The First Normal Form (1NF)
 - 3.5.2 The Second Normal Form (2NF)
 - 3.5.3 Third Normal Form (3NF)
 - 3.5.4 Boyce-Codd Normal Form (BCNF)
- 3.6 Desirable Properties Of Decompositions
 - 3.6.1 Attribute Preservation
 - 3.6.2 Lossless-Join Decomposition
 - 3.6.3 Dependency Preservation
 - 3.6.4 Lack of Redundancy
 - 3.6.5 Deriving BCNF
- 3.7 Multivalued Dependencies
- 3.8 Multivalued Normalization– Fourth Normal Form
- 3.9 The Fifth Normal Form
- 3.10 Rules of Data Normalization
 - 3.10.1 Eliminate Repeating Groups
 - 3.10.2 Eliminate Redundant Data
 - 3.10.3 Eliminate Columns Not Dependent On Key
 - 3.10.4 Isolate Independent Multiple Relationships
 - 3.10.5 Isolate Semantically Related Multiple Relationships
- 3.11 Summary
- 3.12 Model Answers

3.0 INTRODUCTION

We know that relations that form the database must satisfy some properties, for example, relations have no duplicate tuples, tuples have no ordering associated with them, and each element in the relation is atomic. Relations that satisfy these basic requirements may still have some undesirable properties, for example, data redundancy and update anomalies. We illustrate these properties and study how relations may be transformed or decomposed (or normalised) to eliminate them. Most such undesirable properties do not arise if the database modeling has been carried out very carefully using some technique like the Entity-Relationship Model that we have discussed but it is still important to understand the techniques in this chapter to check the model that has been obtained and ensure that no mistakes have been made in modeling.

The central concept in these discussions is the notion of functional dependency, which depends on the semantics of the data and which deals with what information in a relation is dependent on what other information in the relation.

This unit continues our discussion of design issues in relational databases. In general, the goal of a relational-database design is to generate a set of relational schemas that allows us to store information without unnecessary redundancy, yet also allows us to retrieve information easily. One approach is to design schemas that are in an appropriate normal form. Our prime objective in this

unit is to define the concept of functional dependence more precisely and then define normal forms using functional dependencies and using other types of data dependencies.

3.1 OBJECTIVES

At the end of this chapter, you should be able to:

- Define the concept of functional dependency;
- Discuss how to reason with the dependencies;
- Show how to use the dependencies information to decompose relations whenever necessary to obtain relations that have the desirable properties that we want without losing any of the information in the original relations.

3.2 REDUNDANCY AND ASSOCIATED PROBLEMS

Let us consider the following student relation.

Sno	Sname	Address	Cno	Cname	Instructor	office
1123	Rahul	D-27, Dewas	CS -01	Computer Organisation	Ashish Kumar	102
1123	Rahul	D-27, Dewas	CS -03	COBOL	Anurag Sharma	105
1123	Rahul	D-27, Dewas	CS-06	DBMS	Preeti Anand	103
1134	Aparna	B-III, Gurgaon	CS-06	DBMS	Preeti Anand	103

The above table satisfies the properties of a relation and is said to be in first normal form (or 1NF). Conceptually it is convenient to have all the information in one relation since it is then likely to be easier to query the database. But the above relation has the following undesirable features:

Repetition of information: A lot of information is being repeated. Student name, address, course name, instructor name and office number are being repeated often. Every time we wish to insert a student enrolment, say, in CS-06 we must insert the name of the course CS-06 as well as the name and office number of its instructor. Also every time we insert a new enrolment for, say Rahul, we must repeat his name and address. Repetition of information results in wastage of storage as well as other problems.

1. **Update Anomalies:** Redundant information not only wastes storage but also makes updates more difficult since, for example, changing the name of the instructor of CS-06 would require that all tuples containing CS-06 enrolment information be updated. If for some reason, all tuples are not updated, we might have a database that gives two names of instructor for subject CS-06. This difficulty is called the update anomaly.

Insertional Anomalies: Inability to represent certain information— Let the primary key of the above relation be (sno, cno). Any new tuple to be inserted in the relation must have a value for the primary key since existential integrity requires that a key may not be totally or partially NULL. However, if one wanted to insert the number and name of a new course in the database, it would not be possible until a student enrolls in the course and we are able to insert values of sno and cno. Similarly information about a new student cannot be inserted in the database until the student enrolls in a subject. These difficulties are called insertion anomalies.

3. **Deletion Anomalies-- Loss of Useful Information:** In some instances, useful information may be lost when a tuple is deleted. For example, if we delete the tuple corresponding to student 1123 doing CS-03, we will lose relevant information about course CS-03 (viz., course name, instructor, office number) if the student 1123 was the only student enrolled in that course. Similarly deletion of course CS-06 from the database may remove all information about the student named Aparna. This is called deletion anomalies.

The above problems arise primarily because the student relation has information about students as well as subjects. One solution to deal with the problems is to decompose the relation into two or more smaller relations.

Decomposition may provide further benefits, for example, in a distributed database different relations may be stored at different sites if necessary. Of course, decomposition does increase the cost of query processing since the decomposed relations will need to be joined, sometime frequently.

The above relation may be easily decomposed into three relations to remove most of the above undesirable properties:

S(sno,sname,address)

C(cno,cname,instructor,office)

SC(sno, cno)

Such decomposition is made using the principles of normalisation and is essential if we wish to overcome undesirable anomalies.

As noted earlier, normalisation often has an adverse effect on performance. Data which could have been retrieved from one relation before normalisation may require several relations to be joined after normalisation. Normalisation does however lead to more efficient updates since an update that may have required several tuples to be updated before normalisation could well need only one tuple to be updated after normalisation. It also results in overall size reduction of a database, and is MOST desirable for any database design.

Although in the above case we are able to look at the original relation and propose a suitable decomposition that eliminates the anomalies that we have discussed, in general this approach is not possible. A relation may have one hundred or more attributes and it is then almost impossible for a person to conceptualise all the information and suggest a suitable decomposition to overcome the problems. We therefore need an algorithmic approach to finding if there are problems in a proposed database design and how to eliminate them if they exist.

There are several stages of the normalisation process. These are called the first normal form (1NF), the second normal form (2NF), the third normal form (3NF), Boyce-Codd normal form (BCNF), the fourth normal form (4NF) and the fifth normal form (5NF).

So far as practical purposes, 3NF or the BCNF are quite adequate since they remove the anomalies discussed above for most common situations. It should be clearly understood that there is no obligation to normalise relations to the highest possible level. Performance should be taken into account and this may result in a decision not to normalise.

Intuitively, the second and third normal forms are designed to result in relations such that each relation contains information about only one thing (either an entity or a relationship). That is, non-key attributes in each relation must provide a fact about the entity or relationship that is being identified by the key. Again, a sound E-R model of the database would ensure that all relations either provide facts about an entity or about a relationship resulting in the relations that are obtained being in 3NF.

It should be noted that decomposition of relations has to be always based on principles that ensure that the original relation may be reconstructed from the decomposed relations if and when necessary. If we are able to reduce redundancy and not lose any information, it implies that all that redundant information can be derived given the other information in the database. Therefore information that has been removed must be related or dependent on other information that still exists in the database. That is why the concept of redundancy is important. Careless decomposition of a relation can result in loss of information. We will discuss this in detail later.

3.3 ROLE OF NORMALIZATION

In the E-R model, a conceptual schema using an entity-relationship diagram is built and then mapped to a set of relations. This technique ensures that each entity has information about only one thing and once the conceptual schema is mapped into a set of relations, each relation would have information about only one thing. The relations thus obtained would normally not suffer from any of the anomalies that have been discussed in the last section. It can be shown that if the entity-relationship is built using the guidelines that were presented in last unit, the resulting relations are in 3NF.

Of course, mistakes can often be made in database modeling specially when the database is large and complex or one may, for some reasons, carry out database schema design using techniques other than a modeling technique like the entity-relationship model.

For example, one could collect all the information that an enterprise possesses and build one giant relation (often called the universal relation) to hold it. This bottom-up approach is likely to lead to a relation that is likely to suffer from all the problems that we have discussed in the last section. For example, the relation is highly likely to have redundant information and update, deletion and insertion anomalies. Normalisation of such large relation will then be essential to avoid (or at least minimise) these problems.

To summarise, normalisation plays only some basic role in database schema design if a top-down modeling approach like the entity-relationship approach is used. Normalisation, however, plays a major role when the bottom-up approach is being used. Normalisation is then essential to build appropriate relations to hold the information of the enterprise.

Now to define the normal forms more formally, we first need to define the concept of functional dependence.

3.4 SINGLE-VALUED DEPENDENCIES

A database is a collection of related information and it is, therefore, inevitable that some items of information in the database would depend on some other items of information. The information is either single-valued or multivalued. Name of a person or his date of birth are single-valued facts; qualifications of a person or subjects that an instructor teaches are multivalued facts. We first deal only with single-valued facts and discuss the concept of functional dependency.

The concept of some information being dependent on other is quite basic; for example, at any time in a database, a student number would always determine the student name (a person may change his/her name but at any one time the person's name can be determined by the student number) and a subject with a given subject number would always have the same name. We formalise this concept.

Functional Dependency

We can say that the notion of functional dependency is a generalisation of the notion of key.

Superkey: Let R be a relation schema. A subset K of R is a superkey of R if, in any legal relation $r(R)$, for all pairs t_1 and t_2 of tuples in r such that $t_1 \neq t_2$, then $t_1[K] \neq t_2[K]$. That is, no two tuples in any legal relation $r(R)$ may have the same value on attribute set K .

The notion of functional dependency generalises the notion of superkey. We can define functional dependency as follows:

Definition: Let $\alpha \subseteq R$ and $\beta \subseteq R$. The functional dependency $\alpha \rightarrow \beta$ holds on R if, in any legal relation $r(R)$, for all pairs of tuples t_1 and t_2 in r such that $t_1[\alpha] \neq t_2[\alpha]$, it is also the case that $t_1[\beta] \neq t_2[\beta]$.

We also often express this idea by saying that " α determines β ", or that " β is a function of α ", or that " α functionally governs β ". Often, the notions of functionality and functional dependency are expressed briefly by the statement, "If α , then β ". It is important to note that the value β must be unique for a given value of α , i.e., any given value of α must imply just one and only one value of β , in order for the relationship to qualify for the name "function". (However, this does not necessarily prevent different values of α from implying the same value of β .)

Consider a relation R that has two attributes A and B . The attribute B of the relation is functionally dependent on the attribute A if and only if for each value of A no more than one value of B is associated. In other words, the value of attribute A uniquely determines the value of B and if there were several tuples that had the same value of A then all these tuples will have an identical value of attribute B . That is, if t_1 and t_2 are two tuples in the relation R and $t_1(A) = t_2(A)$ then we must have $t_1(B) = t_2(B)$.

A and B need not be single attributes. They could be any subsets of the attributes of a relation R (possibly single attributes). We may then write—

$R.A \rightarrow R.B$

if B is functionally dependent on A (or A functionally determines B). Note that functional dependency does not imply a one-to-one or one to many relationship between A and B although a one-to-one or many-to-one relationship may exist between A and B .

A simple example of the above functional dependency is when A is a primary key of an entity (e.g., student number) and B is some single-valued property or attribute of the entity (e.g., date of birth). $A \rightarrow B$ then must always hold. (why?)

Functional dependencies also arise in relationships. Let C be the primary key of an entity and D be the primary key of another entity. Let the two entities have a relationship. If the relationship is one-to-one, we must have $C \rightarrow D$ and $D \rightarrow C$. If the relationship is many-to-one, we would have $C \rightarrow D$ but not $D \rightarrow C$. For many-to-many relationships, no functional dependencies hold. For example, if C is student number and D is subject number, there is no functional dependency between them. If however, we were storing marks and grades in the database as well, we would have the functional dependency—

$(\text{student number}, \text{subject number}) \rightarrow \text{marks}$

and we might have the functional dependency

$\text{marks} \rightarrow \text{grades}$

The second functional dependency above assumes that the grades are dependent only on the marks. This may sometime not be true since the instructor may decide to take other considerations into account in assigning grades, for example, the class average mark

In the student database that we have discussed earlier, we have the following functional dependencies:

$sno \rightarrow sname$
 $sno \rightarrow address$
 $cno \rightarrow cname$
 $cno \rightarrow instructor$
 $instructor \rightarrow office$

These functional dependencies imply that there can be only one name for each sno, only one address for each student and only one subject name for each cno. It is of course possible that several students may have the same name and several students may live at the same address. If we consider $cno \rightarrow instructor$, the dependency implies that no subject can have more than one instructor (perhaps this is not a very realistic assumption). Functional dependencies, therefore, place constraints on what information the database may store. In the above example, one may be wondering if the following FDs hold

$sname \rightarrow sno$
 $cname \rightarrow cno$

Certainly there is nothing in the instance of the example database presented above that contradicts the above functional dependencies. However, whether above FDs hold or not would depend on whether the university or college whose database we are considering allows duplicate student names and subject names. If it was the enterprise policy to have unique subject names then $cname \rightarrow cno$ holds. If duplicate student names are possible, and one would think there always is the possibility of two students having exactly the same name, then $sname \rightarrow sno$ does not hold.

Functional dependencies arise from the nature of the real world that the database models. Often A and B are facts about an entity where A might be some identifier for the entity and B some characteristic. Functional dependencies cannot be automatically determined by studying one or more instances of a database. They can be determined only by a careful study of the real world and a clear understanding of what each attribute means.

We have noted above that the definition of functional dependency does not require that A and B be single attributes. In fact, A and B may be collections of attributes. For example,

$(sno, cno) \rightarrow (marks, date)$

When dealing with a collection of attributes, the concept of full functional dependence is an important one. Let A and B be distinct collections of attributes from a relation R and let $R.A \rightarrow R.B$. B is then fully functionally dependent on A if B is not functionally dependent on any subset of A. The above example of students and subjects would show full functional dependence if mark and date are not functionally dependent on either student number (sno) or subject number (cno) alone. This implies that we are assuming that a student may have more than one subjects and a subject would be taken by many different students. Furthermore, it has been assumed that there is at most one enrolment of each student in the same subject.

The above example illustrates full functional dependence. However the following dependence

$(sno, cno) \rightarrow instructor$

is not full functional dependence because $cno \rightarrow instructor$ holds.

As noted earlier, the concept of functional dependency is related to the concept of candidate key of a relation since a candidate key of a relation is an identifier which uniquely identifies a tuple and therefore determines the values of all other attributes in the relation. Therefore any subset X of the attributes of a relation R that satisfies the property that all remaining attributes of the relation are functionally dependent on it (that is, on X), then X is candidate key as long as no attribute can be removed from X and still satisfy the property of functional dependence. In the example above, the attributes (sno, cno) form a candidate key (and the only one) since they functionally determine all the remaining attributes.

Functional dependence is an important concept and a large body of formal theory has been developed about it. It is also associated with the concept of closure that helps us derive all functional dependencies that are implied by a given set of dependencies. A complete set of functional dependencies can be obtained, by studying the constraints in a relation relating to various attributes.

3.5 SINGLE VALUED NORMALISATIONS

Initially Codd (1972) presented three normal forms (1NF, 2NF, and 3NF) all based on functional dependencies among the attributes of a relation. Later Boyce and Codd proposed another normal form called the Boyce-Codd normal form (BCNF). The fourth and fifth normal forms are based on multivalued and join dependencies and were proposed later.

3.5.1 The First Normal Form (1NF)

A table satisfying the properties of a relation is said to be in first normal form. As discussed in the previous unit, a relation cannot have multivalued or composite attributes. This is what the 1NF needs to resolve.

A relation is in 1NF if and only if all underlying domains contain atomic values only.

Definition: A table (relation) is in 1NF if

1. There are no duplicated rows in the table.
2. Each cell is single-valued (i.e., there are no repeating groups or arrays).
3. Entries in a column (attribute, field) are of the same kind.

Notes: The order of the rows is immaterial; the order of the columns is immaterial.

The requirement that there be no duplicated rows in the table means that the table has a key (although the key might be made up of more than one column--even, possibly, of all the columns).

The first normal form deals only with the basic structure of the relation and does not resolve the problems of redundant information or the anomalies discussed earlier. All relations discussed in these notes are in 1NF.

For example, consider the following example relation--

student (sno, sname, dob)

Add some other attributes so it has anomalies and is not in 2NF.

The attribute `dob` is the date of birth and the primary key of the relation is `sno` with the functional dependencies $sno \rightarrow sname$ and $sno \rightarrow dob$. The relation is in 1NF as long as `dob` is considered an atomic value and not consisting of three components (day, month, year). The above relation of course suffers from all the anomalies that we have discussed earlier and needs to be normalised further.

3.5.2 The Second Normal Form (2NF)

The second normal form attempts to deal with the problems that are identified with the relation above that is in 1NF. The aim of second normal form is to ensure that all information in one relation is only about one thing.

Definition: A relation is in 2NF if it is in 1NF and every non-key attribute is fully dependent on each candidate key of the relation.

Note: Since a partial dependency occurs when a non-key attribute is dependent on only a part of the (composite) key, the definition of 2NF is sometimes phrased as, "A table is in 2NF if it is in 1NF and if it has no partial dependencies."

To understand the above definition of 2NF we need to define the concept of key attributes. Each attribute of a relation that participates in at least one candidate key of the relation is a key attribute of the relation. All other attributes are called non-key.

The concept of 2NF requires that all attributes that are not part of a candidate key be fully dependent on each candidate key. Let us consider the relation:

student (sno, sname, cno, cname)

and the constraints that a course number is given to only one course name; the functional dependencies are:

$sno \rightarrow sname$

$cno \rightarrow cname$

and assume that (sno, cno) is the only candidate key (and therefore the primary key), the relation is not in 2NF since `sname` and `cname` are not fully dependent on the key. The above relation suffers from the same anomalies and repetition of information as discussed above since `sname` and `cname` will be repeated. To resolve these difficulties we could remove those attributes from the relation that are not fully dependent on the candidate keys of the relations. Therefore we decompose the relation into the following projections of the original relation:

S1 (sno, sname)

S2 (cno, cname)

SC (sno, cno)

We may recover the original relation by taking the natural join of the three relations. If, however, we assume that `sname` and `cname` are unique and therefore we have the following candidate keys:

(sno, cno)

$(sno, cname)$

(sname, cno)

(sname, cname)

The original student relation in that case is in 2NF since the relation has no non-key attributes. The relation still has the same problems as before but it then does satisfy the requirements of 2NF. Higher level normalisation is needed to resolve such problems with relations that are in 2NF and further normalisation will result in decomposition of such relations.

3.5.3 Third Normal Form (3NF)

Although transforming a relation that is not in 2NF into a number of relations that are in 2NF removes many of the anomalies that appear in the relation that was not in 2NF, not all anomalies are removed and further normalisation is sometime needed to ensure further removal of anomalies. These anomalies arise because a 2NF relation may have attributes that are not directly dependent on the candidate key.

Definition: A relation R is in third normal form if it is in 2NF and every non-key attribute of R is non-transitively dependent on each candidate key of R.

To understand the third normal form, we need to define transitive dependence.

Let A, B and C be three attributes of a relation R such that $A \rightarrow B$ and $B \rightarrow C$. From these FDs, we may derive $A \rightarrow C$. As noted earlier, this dependence $A \rightarrow C$ is transitive, which exist because $A \rightarrow B$ and $B \rightarrow C$ exist.

The 3NF differs from the 2NF in that all non-key attributes in 3NF are required to be directly dependent on each candidate key of the relation. The 3NF therefore insists, in the words of Kent (1983) that all facts in the relation are about the key (or the thing that the key identifies), the whole key and nothing but the key. If some attributes are dependent on the keys transitively then that is an indication that those attributes provide information not about the key but about a non-key attribute. So the information is not directly about the key, although it obviously is related to the key.

Consider the following relation:

subject (cno, cname, instructor, office)

Assume that cname is not unique and therefore cno is the only candidate key. The following functional dependencies exist-

$cno \rightarrow cname$

$cno \rightarrow instructor$ (One course is being taught by only one instructor)

$instructor \rightarrow office$ (An instructor has only one office place)

We can derive $cno \rightarrow office$ from the above functional dependencies and, therefore, the above relation is in 2NF. The relation is however not in 3NF since office is not directly dependent on cno. This transitive dependence is an indication that the relation has information about more than one thing (viz., course and instructor) and should therefore be decomposed. The primary difficulty with the above relation is that an instructor might be responsible for several subjects and, therefore, his office address may need to be repeated many times. This leads to all the problems that we identified at the beginning of this unit. To overcome these difficulties we need to decompose the above relation in the following two relations-

s (cno, cname, instructor)

ins (instructor, office)

Relation **s** is now in 3NF and so is relation **ins**.

An alternate decomposition of the relation **subject** is possible—

s(cno, cname)

inst(instructor, office)

si(cno, instructor)

The decomposition into three relations is not necessary, since the original relation is based on the assumption of one instructor for each course.

The 3NF is usually quite adequate for most relational database designs. There are however some situations, for example the relation **student (sno, sname, cno, cname)** discussed in 2NF above, where 3NF may not eliminate all the redundancies and inconsistencies. The problem with the relation **student (sno, sname, cno, cname)** is because of the redundant information in the candidate keys. These are resolved by further normalisation using the BCNF.

3.5.4 Boyce-Codd Normal Form (BCNF)

The relation **student (sno, sname, cno, cname)** has all attributes participating in candidate keys since all the attributes are assumed to be unique. We, therefore, had the following candidate keys:

(sno, cno)

(sno, cname)

(sname, cno)

(sname, cname)

Since the relation has no non-key attributes, the relation is in 2NF and also in 3NF, in spite of the relation suffering the problems that we discussed at the beginning of this unit.

The difficulty in this relation is being caused by dependence within the candidate keys. The second and third normal forms assume that all attributes not part of the candidate keys depend on the candidate keys but does not deal with dependencies within the keys. BCNF deals with such dependencies.

Definition: A relation **R** is said to be in BCNF if $X \rightarrow A$ holds in **R**, and **A** is not in **X**, then **X** is a candidate key for **R**. In other words, a relation is in BCNF if it is in 3NF and if every determinant (left hand side of a functional dependency) is a candidate key.

It should be noted that most relations that are in 3NF are also in BCNF. Infrequently, a 3NF relation is not in BCNF and this happens only if,

- (a) The candidate keys in the relation are composite keys (that is, they are not single attributes),
- (b) There is more than one candidate key in the relation, and
- (c) The keys are not disjoint, that is, some attributes in the keys are common.

The BCNF differs from the 3NF only when there are more than one candidate keys and the keys are composite and overlapping. Consider for example, the relationship-

enrol (sno, sname, cno, cname, date-enrolled)

Let us assume that the relation has the following candidate keys

(sno, cno)

(sno, cname)

(sname, cno)

(sname, cname)

(we have assumed sname and cname are unique identifiers). The relation is in 3NF but not in BCNF because there are dependencies.

$sno \rightarrow sname$

$cno \rightarrow cname$

where attributes that are part of a candidate key are dependent on part of another candidate key. Such dependencies indicate that although the relation is about some entity or association that is identified by the candidate keys e.g., (sno, cno), there are attributes that are not about the whole thing that the keys identify. For example, the above relation is about an association (enrolment) between students and subjects and therefore the relation needs to include only one identifier to identify students and one identifier to identify subjects. Providing two identifiers about students (sno, sname) and two keys about subjects (cno, cname) means that some information about students and subjects that is not needed is being provided. This provision of information will result in repetition of information and the anomalies that we discussed at the beginning of this unit. If we wish to include further information about students and courses in the database, it should not be done by putting the information in the present relation but by creating new relations that represent information about entities student and subject.

These difficulties may be overcome by decomposing the above relation in the following three relations:

(sno, sname)

(cno, cname)

(sno, cno, date-of-enrolment)

We now have a relation that only has information about students, another only about subjects and the third contains the details about registration.

3.6 DESIRABLE PROPERTIES OF DECOMPOSITIONS

So far our approach has consisted of looking at individual relations and checking if they belong to 2NF, 3NF or BCNF. If a relation was not in the normal form that was being checked for and we wished the relation to be normalised to that normal form so that some of the anomalies can be eliminated, it was necessary to decompose the relation in two or more relations. The process of decomposition of a relation R into a set of relations R_1, R_2, \dots, R_n was based on identifying

different components and using that as a basis of decomposition. The decomposed relations R_1, R_2, \dots, R_n , are projections of R and are of course not disjoint otherwise the glue holding the information together would be lost. Decomposing relations in this way based on a recognise and split method is not a particularly sound approach since we do not even have a basis to determine that the original relation can be constructed if necessary from the decomposed relations. We now discuss desirable properties of good decomposition and identify difficulties that may arise if the decomposition is done without adequate care. The next section will discuss how such decomposition may be derived given the FDs.

Desirable properties of a decomposition are:

- Attribute preservation;
- Lossless-join decomposition;
- Dependency preservation;
- Lack of redundancy.

3.6.1 Attribute Preservation

This is a simple and an obvious requirement that involves preserving all the attributes that were there in the relation that is being decomposed.

3.6.2 Lossless-Join Decomposition

In this unit, so far we have normalised a number of relations by decomposing them. We decomposed a relation intuitively. We need a better basis for deciding decompositions since intuition may not always be correct. We illustrate how a careless decomposition may lead to problems including loss of information.

Consider the following relation.

enrol (sno, cno, date-enrolled, room-No., instructor)

Suppose we decompose the above relation into two relations enrol1 and enrol2 as follows:

enrol1 (sno, cno, date-enrolled)

enrol2 (date-enrolled, room-No., instructor)

There are problems with this decomposition but we wish to focus on one aspect at the moment. Let an instance of the relation enrol be-

Sno	cno	Date-enrolled	Room-No.	Instructor
1123	CS-01	20-06-1999	1	Akshay Kumar
1123	CS-02	26-09-1999	2	Anurag Sharma
1259	CS-01	26-09-1998	1	Preeti Anand
1134	CS-05	30-10-1999	5	Preeti Anand
2223	CS-06	05-02-1998	6	Shashi Bhushan

and let the decomposed relations enrol1 and enrol2 be-

Sno	cno	Date-enrolled
1123	CS-01	20-06-1999
1123	CS-02	26-09-1999
1259	CS-01	26-09-1998
1134	CS-05	30-10-1999
2223	CS-06	05-02-1998

Date-enrolled	Room-No.	Instructor
20-06-1999	1	Akshay Kumar
26-09-1999	2	Anurag Sharma
26-09-1998	1	Preeti Anand
30-10-1999	5	Preeti Anand
05-02-1998	6	Shashi Bhushan

All the information that was in the relation enrol appears to be still available in enrol1 and enrol2 but this is not so. Suppose, we wanted to retrieve the student numbers of all students taking a course from Preeti Anand, we would need to join enrol1 and enrol2. The join would have 11 tuples as follows:

Sno	cno	Date-enrolled	Room-No.	Instructor
1123	CS-01	20-06-1999	1	Akshay Kumar
1123	CS-02	20-06-1999	1	Anurag Sharma
1123	CS-01	20-06-1999	1	Preeti Anand
1123	CS-05	20-06-1999	5	Preeti Anand
1123	CS-06	20-06-1999	6	Shashi Bhushan

(add further tuples ...)

The join contains a number of spurious tuples that were not in the original relation Enrol. Because of these additional tuples, we have lost the information about which students take courses from Preeti Anand. (Yes, we have more tuples but less information because we are unable to say with certainty who is taking courses from Preeti Anand). Such decompositions are called **lossy decompositions**. A nonloss or lossless decomposition is that which guarantees that the join will result in exactly the same relation as was decomposed. One might think that there might be other ways of recovering the original relation from the decomposed relations but, sadly, no other operators can recover the original relation if the join does not (why?).

We need to analyse why some decompositions are lossy. The common attribute in above decompositions was Date-enrolled. The common attribute is the glue that gives us the ability to find the relationships between different relations by joining the relations together. If the common attribute is not unique, the relationship information is not preserved. If each tuple had a unique value of Date-enrolled, the problem of losing information would not have existed. The problem arises because several enrolments may take place on the same date.

A decomposition of a relation R into relations R1, R2, ..., Rn, is called a lossless-join decomposition (with respect to FDs F) if the relation R is always the natural join of the relations R1, R2, ..., Rn. It should be noted that natural join is the only way to recover the relation from the decomposed relations. There is no other set of operators that can recover the relation if the join cannot.

It is not difficult to test whether a given decomposition is lossless-join given a set of functional dependencies F . We consider the simple case of a relation R being decomposed into R_1 and R_2 . If the decomposition is lossless-join, then one of the following two conditions must hold

$$R_1 \cap R_2 \rightarrow R_1 - R_2$$

$$R_1 \cap R_2 \rightarrow R_2 - R_1$$

That is, the common attributes in R_1 and R_2 must include a candidate key of either R_1 or R_2 .

3.6.3 Dependency Preservation

It is clear that a decomposition must be lossless so that we do not lose any information from the relation that is decomposed. Dependency preservation is another important requirement since a dependency is a constraint on the database and if holds then we know that the two (sets) attributes are closely related and it would be useful if both attributes appeared in the same relation so that the dependency can be checked easily.

Let us consider a relation $R(A, B, C, D)$ that has the dependencies F that include the following:

$$A \rightarrow B$$

$$A \rightarrow C$$

etc.

If we decompose the above relation into $R_1(A, B)$ and $R_2(B, C, D)$ the dependency $A \rightarrow C$ cannot be checked (or preserved) by looking at only one relation. It is desirable that decompositions be such that each dependency in F may be checked by looking at only one relation and that no joins need be computed for checking dependencies. In some cases, it may not be possible to preserve each and every dependency in F but as long as the dependencies that are preserved are equivalent to F , it should be sufficient.

Let F be the dependencies on a relation R which is decomposed in relations R_1, R_2, \dots, R_n . We can partition the dependencies given by F such that F_1, F_2, \dots, F_n are dependencies that only involve attributes from decomposed relations respectively. If the union of dependencies imply all the dependencies in F , then we say that the decomposition has preserved dependencies, otherwise not.

If the decomposition does not preserve the dependencies F , then the decomposed relations may contain relations that do not satisfy F or the updates to the decomposed relations may require a join to check that the constraints implied by the dependencies still hold.

Consider the following relation

sub(sno, instructor, office)

We may wish to decompose the above relation to remove the transitive dependency of office on sno. A possible decomposition is

S1(sno, instructor)

S2(sno, office)

The relations are now in 3NF but the dependency **instructor** \rightarrow **office** cannot be verified by looking at one relation; a join of S1 and S2 is needed. In the above decomposition, it is quite

possible to have more than one office number for one instructor although the functional dependency $\text{instructor} \rightarrow \text{office}$ does not allow it. Thus, a decomposition may be lossless, yet not dependency preserving. The correct decomposition for the above relation will be $S1(\text{sno, instructor})$ and $S2(\text{instructor, office})$

Generalisation
Normalisation

3.6.4 Lack of Redundancy

We have discussed the problems of repetition of information in a database. Such repetition should be avoided as much as possible.

Lossless-join, dependency preservation and lack of redundancy not always possible with BCNF.
Lossless-join, dependency preservation and lack of redundancy is always possible with 3NF.

3.6.5 Deriving BCNF

Should we also include deriving 3NF?

Given a set of dependencies F , we may decompose a given relation into a set of relations that are in BCNF using the following algorithm

So far we have considered the "recognise and split" method of normalisation. We now discuss Bernstein's algorithm. The algorithm consists of

- 1) Find out the facts about the real world.
- 2) Reduce the list of functional relationships.
- 3) Find the keys.
- 4) Combine related facts.

Once we have obtained relations by using the above approach we need to check that they are indeed in BCNF. If there is any relation R that has a dependency $A \rightarrow B$ and A is not a key, the relation violates the conditions of BCNF and may be decomposed in AB and $R - A$. The relation AB is now in BCNF and we can now check if $R - A$ is also in BCNF. If not, we can apply the above procedure again until all the relations are in fact in BCNF.

3.7 MULTIVALUED DEPENDENCIES

With modelling using the E-R Modelling technique, we noted difficulties that can arise when an entity has multivalued attributes. It was because in the relational model, if all of the information about such entity is to be represented in one relation, it will be necessary to repeat all the information other than the multivalued attribute value to represent all the information that we wish to represent. This results in many tuples about the same instance of the entity in the relation and the relation having a composite key (the entity_id and the multivalued attribute). Of course the other option suggested was to represent this multivalued information in a separate relation. The situation of course becomes much worse if an entity has more than one multivalued attributes and these values are represented in one relation by a number of tuples for each entity instance such that every value of one the multivalued attributes appears with every value of the second multivalued attribute to maintain consistency. The multivalued dependency relates to this problem when more than one multivalued attributes exist.

Consider the following relation that represents an entity employee that has one multivalued attribute proj:

emp (e#, dept, salary, proj)

We have so far considered normalization based on functional dependencies; dependencies that apply only to single-valued facts. For example, $e\# \rightarrow dept$ implies only one dept value for each value of $e\#$. Not all information in a database is single-valued, for example, proj in an employee relation may be the list of all projects that the employee is currently working on. Although $e\#$ determines the list of all projects that an employee is working on, $e\# \rightarrow proj$ is not a functional dependency.

So far we have dealt with multivalued facts about an entity by having a separate relation for that multivalued attribute and then inserting a tuple for each value of that fact. This resulted in composite keys since the multivalued fact must form part of the key. In none of our examples so far have we dealt with an entity having more than one multivalued attribute in one relation. We do so now.

The fourth normal form deals with multivalued dependencies. Before discussing the 4NF, we discuss the following example to illustrate the concept of multivalued dependency.

programmer (emp_name, qualifications, languages)

The above relation includes two multivalued attributes of entity programmer; qualifications and languages. There are no functional dependencies.

The attributes qualifications and languages are assumed independent of each other. If we were to consider qualifications and languages separate entities, we would have two relationships (one between employees and qualifications and the other between employees and programming languages). Both the above one programmer could have several qualifications and may know several programming languages. Also one qualification may be obtained by several programmers and one programming language may be known to many programmers.

The above relation is therefore in 3NF (even in BCNF) but it still has some disadvantages. Suppose a programmer has several qualifications (B.Sc, Dip. Comp. Sc., etc.) and is proficient in several programming languages; how should this information be represented? There are several possibilities.

Emp_name	qualifications	languages
Rahul	B. Sc.	FORTRAN
Rahul	B. Sc.	COBOL
Rahul	B. Sc.	PASCAL
Rahul	Dip. CS	FORTRAN
Rahul	Dip. CS	COBOL
Rahul	Dip. CS	PASCAL

Emp_name	qualifications	languages
Rahul	B. Sc.	NULL
Rahul	Dip. CS	NULL
Rahul	NULL	FORTRAN
Rahul	NULL	COBOL
Rahul	NULL	PASCAL

Emp_name	qualifications	languages
Rahul	B. Sc.	FORTTRAN
Rahul	Dip. CS	COBOL
Rahul	NULL	PASCAL

Other variations are possible (we remind the reader that there is no relationship between qualifications and programming languages). All these variations have some disadvantages. If the information is repeated we face the same problems of repeated information and anomalies as we did when second or third normal form conditions are violated. If there is no repetition, there are still some difficulties with search, insertions and deletions. For example, the role of NULL values in the above relations is confusing. Also the candidate key in the above relations is (emp name, qualifications, language) and existential integrity requires that no NULLs be specified. These problems may be overcome by decomposing a relation like the one above as follows:

Emp_name	qualifications
Rahul	B. Sc.
Rahul	Dip. CS

Emp_name	languages
Rahul	FORTTRAN
Rahul	COBOL
Rahul	PASCAL

The basis of the above decomposition is the concept of multivalued dependency (MVD). Functional dependency relates one value of A to one value of B while multivalued dependency $A \twoheadrightarrow B$ defines a relationship in which a set of values of attribute B are determined by a single value of A.

The concept of multivalued dependencies was developed to provide a basis for decomposition of relations like the one above. Therefore, if a relation like enrolment(sno, subject#) has a relationship between sno and subject# in which sno uniquely determines the values of subject#, the dependence of subject# on sno is called a trivial MVD since the relation enrolment cannot be decomposed any further. More formally, a MVD $X \twoheadrightarrow Y$ is called trivial MVD if either Y is a subset of X or X and Y together form the relation R. The MVD is trivial since it results in no constraints being placed on the relation. Therefore, a relation having non-trivial MVDs must have at least three attributes; two of them multivalued. Non-trivial MVDs result in the relation having some constraints on it since all possible combinations of the multivalued attributes are then required to be in the relation.

Let us now define the concept of multivalued dependency.

Definition: The multivalued dependency $X \twoheadrightarrow Y$ is said to hold for a relation $R(X, Y, Z)$ if for a given set of value (set of values if X is more than one attribute) for attributes X, there is a set of (zero or more) associated values for the set of attributes Y and the Y values depend only on X values and have no dependence on the set of attributes Z.

In the example above, if there was some dependence between the attributes qualifications and language, for example, perhaps, the language was related to the qualifications (perhaps the qualification was a training certificate in a particular language), and then the relation would not have MVD and could not be decomposed into two relations as above. In the above situation whenever $X \twoheadrightarrow Y$ holds, so does $X \twoheadrightarrow Z$ since the role of the attributes Y and Z, is symmetrical.

Consider two different situations.

- a) Z is a single valued attribute. In this situation, we deal with R(X, Y, Z) as before by entering several tuples about each entity).
- b) Z is multivalued.

Now, more formally, $X \twoheadrightarrow Y$ is said to hold for R(X, Y, Z) if t1 and t2 are two tuples in R that have the same values for attributes X and therefore with $t1[x] = t2[x]$ then R also contains tuples t3 and t4 (not necessarily distinct) such that

$$t1[x] = t2[x] = t3[x] = t4[x]$$

$$t3[Y] = t1[Y] \text{ and } t3[Z] = t2[Z]$$

$$t4[Y] = t2[Y] \text{ and } t4[Z] = t1[Z]$$

In other words if t1 and t2 are given by

$$t1 = [X, Y1, Z1], \text{ and}$$

$$t2 = [X, Y2, Z2]$$

then there must be tuples t3 and t4 such that

$$t3 = [X, Y1, Z2], \text{ and}$$

$$t4 = [X, Y2, Z1]$$

We are therefore insisting that every value of Y appears with every value of Z to keep the relation instances consistent. In other words, the above conditions insist that Y and Z are determined by X alone and there is no relationship between Y and Z since Y and Z appear in every possible pair and hence these pairings present no information and are of no significance. Only if some of these pairings were not present, there would be some significance in the pairings.

(Note: If Z is single-valued and functionally dependent on X then $Z1 = Z2$. If Z is multivalued dependent on X then $Z1 \neq Z2$).

- [Please note the decomposition as given in the example, MVD is a formalised theory for the decomposition.]

3.8 MULTIVALUED NORMALISATION- FOURTH NORMAL FORM

We have considered an example of Programmer (Emp name, qualification, languages) and discussed the problems that may arise if the relation is not normalised further. We also saw how the relation could be decomposed into P1 (Emp name, qualifications) and P2 (Emp name, languages) to overcome these problems. The decomposed relations are in fourth normal form (4NF), which we shall now define.

We are now ready to define 4NF. A relation R is in 4NF if, whenever a multivalued dependency $X \twoheadrightarrow Y$ holds then either

- (a) the dependency is trivial, or
- (b) X is a candidate key for R.

As noted earlier, the dependency $X \twoheadrightarrow \Phi$ Or $X \twoheadrightarrow Y$ in a relation $R(X, Y)$ is trivial since they must hold for all $R(X, Y)$. Thus, the decomposition proposed above is in 4NF.

In fourth normal form, we have a relation that has information about only one entity. If a relation has more than one multivalued attribute, we should decompose it to remove difficulties with multivalued facts.

Intuitively R is in 4NF if all dependencies are a result of keys. When multivalued dependencies exist, a relation should not contain two or more independent multivalued attributes. The decomposition of a relation to achieve 4NF would normally result in not only reduction of redundancies but also avoidance of anomalies.

3.9 THE FIFTH NORMAL FORM

The basic principle behind 5NF is the situation when a table exists having all key attributes.

Consider the following relation SPJ (Supplier#, Part#, Project#).

The constraints are:

- A supplier can supply any part to any project.
- A project may need many parts supplied by any supplier.
- A part may be supplied by any supplier.

Let us consider the following instance of SPJ:

Relation: SPJ

Supplier#	Part#	Project#
S1	P1	J1
S1	P2	J1
S1	P1	J3
S1	P2	J3
S2	P2	J3
S2	P2	J1
S2	P3	J1
S3	P1	J1
S3	P1	J2

In the above relation that Supplier S3 can supply Part2 cannot be represented till he supplies this part to some project, as the field project# is part of the only key to the relation which is (Supplier#, Part#, project#). On decomposing the relation into two tables SP and SJ we may encounter the following problem.

Decomposed First Relation: SP

Supplier#	Part#
S1	P1
S1	P2
S2	P2
S2	P3
S3	P1

Decomposed Second Relation: SJ

Supplier#	proJect#
S1	J1
S1	J3
S2	J3
S2	J1
S3	J1
S3	J2

On joining SP and SJ we get Relation: Lossy_SPJ

Supplier#	Part#	proJect#
S1	P1	J1
S1	P2	J1
S1	P1	J3
S1	P2	J3
S2	P2	J3
S2	P3	J3
S2	P2	J1
S2	P3	J1
S3	P1	J1
S3	P1	J2

We need the third decomposed Relation: PJ

Part#	proJect#
P1	J1
P2	J1
P1	J3
P2	J3
P3	J1
P1	J2

On combining Lossy_SPJ with the PJ relation we will get the original relation.

There may be other problems relating to the 2 decomposibility of SPJ relation and it can be decomposed into three possible relations. This three decomposibility is best explained by Join dependency and Fifth Normal Form. However, in this course we will not present a formal definition of the two as they are the least properly understood. A designer have to rely more on his common sense to get this normal form.

3.10 RULES OF DATA NORMALIZATION

1. **Eliminate Repeating Groups:** Make a separate table for each set of related attributes, and give each table a primary key.
2. **Eliminate Redundant Data:** If an attribute depends on only part of a multi-valued key, remove it to a separate table.
3. **Eliminate Columns Not Dependent On Key:** If attributes do not contribute to a description of the key; remove them to a separate table.
4. **Isolate Independent Multiple Relationships:** No table may contain two or more 1:n or n:m relationships that are not directly related.
5. **Isolate Semantically Related Multiple Relationships:** There may be practical constrains on information that justify separating logically related many-to-many relationships.

Let's explain these steps of normalisation through an example:

Let us make a list of all the employees in the company, in the original employee list, each employee name is followed by any databases that the member has experience with. Some might know many, and others might not know any.

Employee List				
Emp-ID	Emp-Name	Dept-Name	Company-Loc	Database Known
1	Gurpreet Malhotra	A	N-Delhi	Oracle
2	Faisal Khan	A	N-Delhi	Access
3	Manisha Kukreja	C	Agra	DBASE, Clipper, FoxPro
4	Sameer Singh	B	Bombay	DB2
5	Dave Jones	B	Bombay	DB2, Oracle

For the time being we are not considering Dept-Name and Company-Loc till step 3.

3.10.1 Eliminate Repeating Groups

If we give the query, "Find out list of employees ,who knows DB2".

For this query we need to perform an awkward scan of the list looking for references to DB2. This is inefficient and an extremely untidy way to retrieve information. Let us create another field termed as Database-ID as D-ID to codify name of database in numeric form.

The Emp-ID in the database table matches the primary key in the employee table, providing a foreign key for relating the two tables with a join operation. Now we can answer the question by looking in the database table for "DB2" and getting the list of Employees.

Employee Table (Not shown Dept-Name and Company-Loc)	
Emp-ID	Emp-Name
1	Gurpreet Malhotra
2	Faisal Khan
3	Manisha Kukreja
4	Sameer Singh
5	Dave Jones

Database Table		
D-ID	Emp-ID	Database
1	2	Access
2	4	DB2
2	5	DB2
3	3	FoxPro
4	3	DBASE
5	3	Clipper
6	1	Oracle
6	5	Oracle

3.10.2 Eliminate Redundant Data

In the Database Table, the primary key is made up of the Emp-ID and the D-ID (Database ID). This makes sense for the "Where Learned" and "Skill Level" attributes, since they will be different for every member/database combination. But the database name depends only on the D-ID. The same database name will appear redundantly, every time its associated ID appears in the Database Table.

Let us now understand how does anomalies occur.

1. We need reclassify a database, thus we give different D-ID to the database. The change has to be made for every Employee that lists that database. If you miss some, you'll have several Employees with the same database under different IDs. This is an update anomaly.
2. Suppose the last Employee listing a particular database leaves the group. His records will be removed from the system, and the database will not be stored anywhere! This is a delete anomaly.

To avoid these problems, we need **second normal form**. To achieve this, we isolate the attributes depending on both parts of the key from those depending only on the D-ID. This results in two tables: "Database" which gives the name for each D-ID, and "EmpDatabase" which lists the databases for each member.

Employee Table (Not shown Dept-Name, Company-Loc)	
Emp-ID	Emp-Name
1	Gurpreet Malhotra
2	Faisal Khan
3	Manisha Kukreja
4	Sameer Singh
5	Dave Jones

EmpDatabase Table	
Emp-ID	D-ID
2	1
4	2
5	2
3	3
3	4
3	5
1	6
5	6

DataBase Table	
D-ID	Database
1	Access
2	DB2
3	FoxPro
4	DBASE
5	Clipper
6	Oracle

Now we can reclassify a database in a single operation: look up the D-ID in the "Database" table and change its name. The result will instantly be available throughout the application.

3.10.3 Eliminate Columns Not Dependent On Key

Now consider the fields Dept-Name and Company-Loc in the employee table.

The Employee Table satisfies

first normal form - As it contains no repeating groups.

second normal form - As it doesn't have a multivalued key.

But the key is Emp-ID; and the Dept-Name and location describe only about Department, not a Employee. To achieve third normal form, they must be moved into a separate table. Since they describe a department, Dept-Code becomes the key of the new "Company" table.

The motivation for this is the same for second normal form: we want to avoid update and delete anomalies. For example, suppose no Employees from the department "A" were currently stored in the database. With the previous design, there would be no record of its existence, even though 2 past members were from "A".

Employee-Table-Final	
Emp-ID	Emp-Name
1	Gurpreet Malhotra
2	Faisal Khan
3	Manisha Kukreja
4	Sameer Singh
5	Dave Jones

Department-Table		
Dept-ID	Dept-Name	Company-Loc
1	A	N-Delhi
2	B	Bombay
3	C	Agra
4	D	Banglore
5	E	Hyderabaad

Rest of the Table remains same.

3.10.4 Isolate Independent Multiple Relationships

Isolate Semantically Related Multiple Relationships

Example

Let us now simply consider an example, simply giving Normalisation Table.

Unnormalised Data Items for Puppies
Puppy number
Puppy name
Kennel Code
Kennel Name
Kennel location
Trick ID 1...n
Trick Name 1...n
Trick Where Learned 1...n
Skill Level 1...n

FIRST NORMAL FORM		
Puppy Table		
Puppy Number	Primary Key	Every puppy gets a unique number.
Puppy Name		
Kennel Name		
Kennel Location		
Trick Table		
Puppy Number	Primary Key [composite]	
Trick ID	Primary Key [composite]	
Trick Name		We'll add a row for every trick learned by every puppy.
Trick Where Learned		
Skill Level		

TRICK TABLE				
Puppy Number	Trick ID	Trick Name	Where Learned	Skill Level
52	27	Roll Over	16	9
53	16	Nose Stand	9	9
54	27	Roll Over	9	5

SECOND NORMAL FORM		
Puppy Table		
Puppy Number	Primary Key	Every puppy gets a unique number.
Puppy Name		
Kennel Name		
Kennel Location		
Trick Table		
Trick ID		
Trick Name		We'll add a row for every trick learned by every puppy.
Puppy Tricks		
Puppy Number	Primary Key [composite]	
Trick ID	Primary Key [composite]	
Trick Where Learned		
Skill Level		

THIRD NORMAL FORM		
Puppy Table		
Puppy Number	Primary Key	Every puppy gets a unique number.
Puppy Name		
Kennel Code		
Kennels Table		
Kennel Code	Primary Key	
Kennel Name		
Kennel Location		
Trick Table		
Trick ID		
Trick Name		We'll add a row for every trick learned by every puppy.
Puppy Tricks		
Puppy Number	Primary Key [composite]	
Trick ID	Primary Key [composite]	
Trick Where Learned		
Skill Level		

Puppy Tricks and Costumes		
<i>Puppy Number</i>		
Trick ID		
Trick Where Learned		
Skill Level		
Costume		
FOURTH NORMAL FORM		
Puppy Table		
Puppy Number	Primary Key	Every puppy gets a unique number.
Puppy Name		
Kennel Code		
Kennels Table		
Kennel Code	Primary Key	
Kennel Name		
Kennel Location		
Trick Table		
Trick ID		
Trick Name		We'll add a row for every trick learned by every puppy.

Puppy Tricks		
Puppy Number	Primary Key [composite]	
Trick ID	Primary Key [composite]	
Trick Where Learned		
Skill Level		
Costumes Table		
Costume Number	Primary Key	
Costume Name		
Puppy Costumes		
Puppy Number	Primary Key [composite]	
Costume Number	Primary Key [composite]	

Kennel-Breeder-Breeds (Not in 4NF)		
Kennel Number		
Breeder		
Breed		

Kennel-Breeder-Breeds (Not in 4NF)		
Kennel Number	Breeder	Breed
5	Acme	Spaniel
5	Acme	Dachshund
5	Acme	Banana-Biter
5	Puppy Factory	Spaniel
5	Puppy Factory	Dachshund
5	Puppy Factory	Banana-Biter
5	Whatapuppy	Spaniel
5	Whatapuppy	Dachshund
5	Whatapuppy	Banana-Biter

Kennel-Breed (4NF)	
Kennel Number	Breed
5	Spaniel
5	Dachshund
5	Banana-Biter

Kennel-Breeder (4NF)	
Kennel Number	Breeder
5	Acme
5	Puppy Factory
5	Whatapuppy

FIFTH NORMAL FORM		
Puppy Table		
Puppy Number	Primary Key	Every puppy gets a unique number.
Puppy Name		
Kennel Code		
Kennels Table		
Kennel Code	Primary Key	
Kennel Name		
Kennel Location		
Trick Table		
Trick ID	Primary Key	
Trick Name		We'll add a row for every trick learned by every puppy.
Puppy Tricks		
Puppy Number	Primary Key [composite]	
Trick ID		
Trick Where Learned		
Skill Level		
Costumes Table		
Costume Number	Primary Key	
Costume Name		
Puppy Costumes		
Puppy Number	Primary Key	
Costume Number	Primary Key	
Kennel-Breed		
Kennel Number	Primary Key	
Breed		
Kennel-Breeder		
Kennel Number	Primary Key	
Breeder		

Check Your Progress

Question 1. Let R be a relational of degree n. What is the maximum number of functional dependencies R can possibly satisfy (trivial as well as nontrivial)?

.....

.....

Question 2. List the set of all FDs satisfied (for all time) by shipments relation SPJ (defined in Question 1 of check your progress 2 in unit 1).

.....

.....

3. A relation NADDR is defined with attributes NAME (unique), STREET, CITY, STATE and ZIP. For any given zipcode, there is just one city and state. Also, for any given street, city, and state, there is just one zipcode. Give an irreducible set of FDs for this relation. What are the candidate keys? What are the normalised tables, based on these FDs.
-
-
-

3.11 SUMMARY

A **Functional Dependency (FD)** is a many to-to-one relationship between two sets of attribute of a given relation. Given a relation R, the FD $A \rightarrow B$ (where A and B are subsets of the attributes of R) is said to hold in R if and only if, whenever two tuples of R have the same value for A, and they also have the same value for B.

The notion of functional dependency generalizes the notion of superkey.

Let R be a relation schema. A subset K of R is a superkey of R no two tuples in any legal relation r(R) may have the same value on attribute set K.

We discussed Single valued normalisation and discussed the concepts of **first, second, third, and Boyce/Codd normal forms**. The purpose of normalisation is to avoid redundancy, and hence to avoid certain anomalies.

We learned that the desirable properties of a decomposition of a base relation to its normal forms, are that the decomposition should be attribute preserving, dependency preserving and lossless.

We also touched multivalued dependencies, although not in detail. We learned the various rules of data normalisation, which help to normalise the base relation very cleanly. Those rules are eliminating repeating groups, eliminate redundant data, eliminate columns not dependent on key, isolate independent multiple relationship and isolate semantically related multiple relationships.

3.12 MODEL ANSWERS

Check Your Progress

- Question 1. An FD is basically a statement of the form $A \rightarrow B$ where A and B are each subsets of the set of attributes of R. Since a set of n elements has 2^n possible subsets, each of A and B has 2^n possible values, and hence an upper limit on the number of possible FDs is 2^{2^n} .
- Question 2. The complete set of FDs - for relation SPJ is as follows:
- { S#, P#, QUANTITY } \rightarrow { S#, P#, QUANTITY }
 - { S#, P#, QUANTITY } \rightarrow { S#, P# }
 - { S#, P#, QUANTITY } \rightarrow { P#, QUANTITY }
 - { S#, P#, QUANTITY } \rightarrow { S#, QUANTITY }
 - { S#, P#, QUANTITY } \rightarrow { S# }
 - { S#, P#, QUANTITY } \rightarrow { P }

~~{ S#, P#, QUANTITY } → { QUANTITY }~~
~~{ S#, P#, QUANTITY } → { }~~

{ S#, P# } → { S#, P#, QUANTITY }
 { S#, P# } → { S#, P# }
 { S#, P# } → { P#, QUANTITY }
 { S#, P# } → { S#, QUANTITY }
 { S#, P# } → { S#, }
 { S#, P# } → { P }
 { S#, P# } → { QUANTITY }
 { S#, P# } → { }

{ P#, QUANTITY } → { P#, QUANTITY }
 { P#, QUANTITY } → { P#, }
 { P#, QUANTITY } → { QUANTITY }
 { P#, QUANTITY } → { }

{ S#, QUANTITY } → { S#, QUANTITY }
 { S#, QUANTITY } → { S#, }
 { S#, QUANTITY } → { QUANTITY }
 { S#, QUANTITY } → { }

{ S# } → { S# }
 { S# } → { }

{ P# } → { P# }
 { P# } → { }

{ QUANTITY } → { QUANTITY }
 { QUANTITY } → { }

{ } → { }

Question 3. Abbreviating NAME, STREET, CITY, STATE, and ZIP as N, R, C, T, and Z, respectively, we have:

N → RCT
 RCT → Z
 Z → CT

An obviously equivalent irreducible set is:

N → R
 N → C
 N → T
 RCT → Z
 Z → C
 Z → T

The only candidate key is N, as we can find any one of N, R, C, T, Z from N.

Let us first use the FD.

RCT → Z

We will get

R1 (Name, Street, City, State), R2 (Street, City, State, Zip) (3NF)

FDs $N \rightarrow RCT$

$RCT \rightarrow Z, Z \rightarrow CT$

We may decompose R2 further using $Z \rightarrow CT$

R21 (Street, Zip)

R22 (Zip, City, State)

but here the FD $RCT \rightarrow Z$ will be lost and is enforceable only on the join of the two relations. Although R21 and R22 will be in BCNF but it is not recommended and is a bad decomposition. Therefore, sometimes we may restrict ourselves to just 3NF.



Block

2

PRACTICAL ON RDBMS

1.0	
BLOCK INTRODUCTION	3
1.1	
BLOCK OBJECTIVES	3
1.2	
REVIEW QUESTIONS	4
1.3	
ENTITY-RELATIONSHIP DIAGRAM	6
1.4	
FUNCTIONAL DEPENDENCY AND NORMALISATION	12
1.5	
STRUCTURED QUERY LANGUAGE (SQL)	21
1.6	
MICROSOFT-ACCESS	32
1.7	
VIEW AND SECURITY USING SQL	50
1.8	
SUMMARY	60

BLOCK 2: PRACTICAL ON RELATIONAL DATABASE MANAGEMENT SYSTEMS.

Structure

- 1.0 Block Introduction
- 1.1 Block Objectives
- 1.2 Review Questions
 - 1.2.1 DBMS and File Oriented Approach
 - 1.2.2 Relational Databases and Integrity Constraints
- 1.3 Entity-Relationship Diagram
- 1.4 Functional Dependency and Normalisation
 - 1.4.1 Functional Dependency
 - 1.4.2 Normalisation
- 1.5 Structured Query Language (SQL)
- 1.6 Microsoft-Access
- 1.7 Views and Security Using SQL
- 1.8 Summary

1.0 BLOCK INTRODUCTION

In the previous block our emphasis was to present the concepts relating to RDBMS design. This block is an attempt to provide you with some practical exercises. This block is going to differ from the other blocks, as it has not been divided into units; however, we have tried to keep separate exercises for separate topics. Please note that an exercise for an advanced topic will involve use of all the application skills acquired by you in the earlier exercises, therefore, you must proceed forward in the block only if you have completed the previous exercises, for your own benefits.

This block covers wide range of solved and unsolved exercises on the basic concepts, normalisation, ER diagram, Database implementation using SQL and a RDBMS package. You must try out these exercises using any RDBMS package. However, for this course because of wide availability the practical may be tried out at least on MS-ACCESS platform. The next block has been kept for the package related activities on MS-ACCESS, so you must use it while implementing these exercises.

We have tried to give you basic competence, however, you are requested to attempt more problems from the reference books:

- 1.3 C.J. Date, *Database Concepts*, Addison Wesley Publications.
- 1.4 Korth A. et al, *Database Concepts*, McGraw Hill Publications.

1.1 BLOCK OBJECTIVES

After doing all the exercises and similar type exercises, you should be able to:

- Make FR diagram for a defined problem;
- Define the organized table from the ER diagram;

- Write various SQL based queries;
- Implement the table design into an RDBMS package;
- Make simple reports and forms.

1.2 REVIEW QUESTIONS

Please attempt the following questions to begin with:

- 1) A database system is fully relational if it supports _____ and _____.
- 2) A Relation resembles a _____ a tuple resembles a _____ and an attribute resembles a _____.
- 3) A candidate key, which is not a primary key is known as a _____ key.
- 4) Primitive operations are union, difference, product, selection and projection. The definition of A intersects B will be _____.
- 5) Which one is not a traditional set operator defined on relational algebra?
 - (i) Union
 - (ii) Intersection
 - (iii) Difference
 - (iv) Join
- 6) Write True/False
 - a) Any Binary relation is in 3NF.
 - b) Any Binary relation is in BCNF.
 - c) Any Binary relation is in 4NF.
 - d) Relation R (A, B, C) is equal to the join of its projections R1 (A, B) and R2 (A,C) iff the FD $B \rightarrow C$ holds in R.
 - e) If $R.A \rightarrow R.B$ and $R.B \rightarrow R.C$ then $R.A \rightarrow R.C$.
 - f) If $R.A \rightarrow R.B$ and $R.A \rightarrow R.C$ then $R.A \rightarrow R.(B, C)$.
 - g) If $R.B \rightarrow R.A$ and $R.C \rightarrow R.A$ then $R.(B, C) \rightarrow R.A$.
 - h) If $R.(B, C) \rightarrow R, A$ then $R.B \rightarrow R.A$ and $R.C \rightarrow R.A$.
 - i) Any relation can be non-loss-decomposed into an equivalent collection of 4NF relations.

Model Answers

- 1) Relational databases and A language as powerful as relational algebra
- 2) File, Record, Field
- 3) Alternate
- 4) A Minus (A minus B)
- 5) (iv)
- 6) a) True

- b) True
- c) True
- d) False
- e) True
- f) True
- g) True
- h) False
- i) True

Exercise 0

The basic objective of this exercise is to make sure that you have gone through the earlier block. Please try to answer questions in section 1 and section 2 yourself using Block 1 of this course.

1.2.1 DBMS and File Oriented Approach

Question 1. What are the five main differences between a file-processing system and a DBMS?

.....
.....
.....
.....

Question 2. What are two major disadvantages of a database system?

.....
.....
.....
.....

1.2.2 Relational Databases and Integrity Constraints

Question 1. Define a relational, Domain, Attribute, Primary key.

.....
.....
.....
.....

- Question 2. What are candidate and alternate key?
.....
.....
- Question 3. Describe the two components of a relation in a relational database.
.....
.....
- Question 4. Define a relational database.
.....
.....
- Question 5. List two reasons why null values may be introduced into the database.
.....
.....
- Question 6. What is referential integrity?
.....
.....

1.3 ENTITY – RELATIONSHIP DIAGRAM

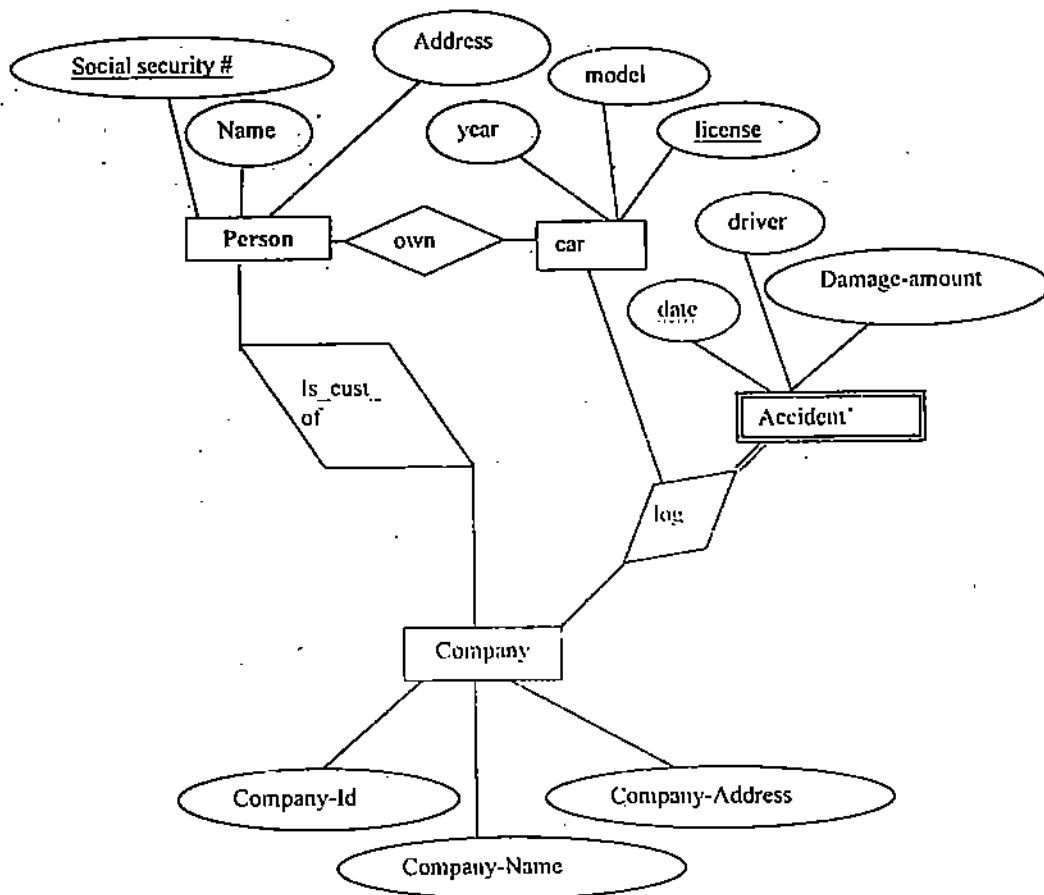
- Question 1. (i) Construct an E-R diagram for a car-insurance company that has a set of customers, each of whom owns one or more cars. Each car has associated with it zero to any number of recorded accidents.

(ii) Construct appropriate tables.
- Question 2. Consider an E-R diagram in which the same entity set appears several times. Why is allowing this redundancy a bad practice that one should avoid whenever possible?
- Question 3. The room booking side of a small hotel is to be computerized. The hotel has a number of rooms. Each room has a basic (double) price and a supplementary price for extra children. These prices also depend on the time of year- it is more expensive at Christmas, during the summer and around bank holidays, for example. There are 3 seasonal bands. The system must enable the hotel proprietor to answer phone calls from prospective clients (for example, rooms available now and in the future, with costs), make provisional bookings, do mailings of previous clients, prepare clients' bill (ignore extras such as papers, drinks etc). Recognise various entities and relationship among them. Construct an E-R diagram for the above.

Question 4. A student uses a particular computing system to do the computations for a given course using a limited hour account code. Find the appropriate relationship. Draw the E-R diagram.

Model Answers

Answer 1. (i)



(ii) Tables:

Person

Social Security#	Character	10	Primary Key
Name	Character	25	
Address-house-no	Character	20	
Address-street	Character	20	
Address-city	Character	20	
Address-state	Character	20	
Address-pincode	Character	6	

Company

Company-Id	Character	10	Primary Key
Company-name	Character	20	
Company-address	Define as per requirements		

Car

License	Character	10	Primary Key
Model	Character	10	
Year	Numeric		

Own (Assuming many to many relationship)

Social security #	Character	10	Part of Primary Key. Also a Foreign Key referencing Person Table.
License	Character	10	Part of Primary Key. Also a Foreign Key referencing Car Table.

Is-Customer-of

Social security #	Character	10	Part of Primary Key. Also a Foreign Key referencing Person Table.
Company-Id	Character	10	Part of Primary Key. Also a Foreign Key referencing Company Table.

Accident-log

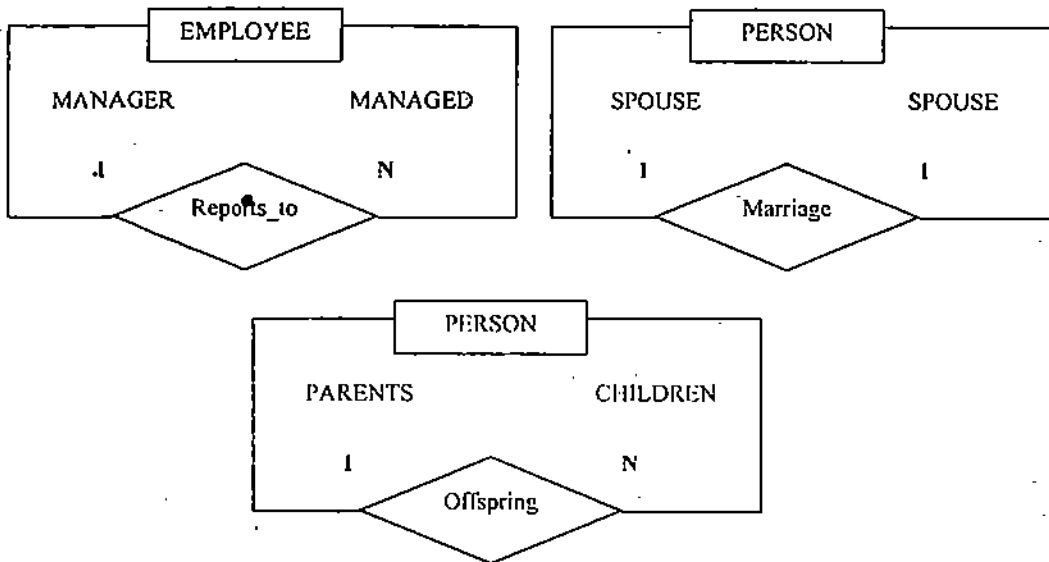
Company-Id	Character	10	Part of Primary Key. Also a Foreign Key referencing Company Table.
License	Character	10	Part of Primary Key. Also a Foreign Key referencing Car Table.

Adate	Date	
Driver	Character	25
Damage-amount	Numeric	

assumed discriminator in weak entity, part of key.

Practical on RDBMS

Answer 2. Here are given three examples for an E-R diagram in which the same entity set appears several times



One of the problems with such relationships is that to answer a query you may need to join same tables again and again. However, they may be necessary in some cases.

Answer 3. We are obviously concerned with entities such as CLIENT, ROOM, and BOOKING. A first attempt:

ROOM { RoomNumber, NumberOfDoubles, NumberOfSingles, WashingFacilities, Season, BasicPrice, SupplementaryPrice } are the attributes of ROOM.

Season = [low|middle|high] i.e., The value of season could be either low, middle or high.

CLIENT { ClientName + ClientAddress, ClientTelephoneNumber }

BOOKING { RoomNumber + DateOfArrival, ClientName, ClientAddress, NumberOfGuests, ExpectedStay, Deposit }

The underlined attributes are the primary key of the Entity.

Now there exist attributes in ROOM, which cannot be found just from the key. To find the price of a room, you need both to know the RoomNumber and the Season

We solve this problem by creating a new entity ROOM-PRICE, which allows us to find the prices for any room in any season. These attributes are removed from Room:

ROOM { RoomNumber, NumberOfDoubles, NumberOfSingles, WashingFacilities }.

ROOM-PRICE { RoomNumber + Season, BasicPrice, SupplementaryPrice }

Season = ["low"|"middle"|"high"]

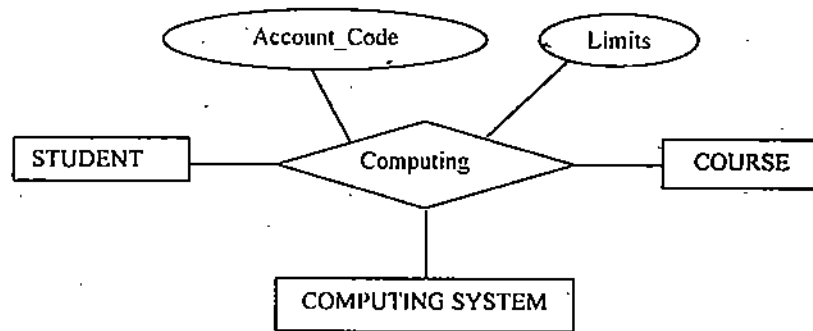
What about current occupancy of the room?

As some clients will book and then occupy, some will occupy without booking, some will book without occupying.....

ROOM-OCCUPANCY { RoomNumber + DateArrived, NameAndAddress, ExpectedLeavingDate }

Can we find the Booking from the Room-Occupancy, and if not what else do we need?

Answer 4.



Exercise 1

Question 1. Construct an E-R diagram for a hospital with a set of patients and a set of medical doctors. Associate with each patient a log of the various tests and examinations conducted.

Question 2. Explain the difference between a weak entity set and a strong entity set.

.....

.....

Question 3. Consider the entity EMPLOYEE with following attributes:

- Emp_ID
- Employee_Name
- Address
- Phone
- Dependent_Name
- Relationship_to_Employee
- Skill
- Designation
- Designation_Start_Date
- Salary
- Salary_Start_Date

Using the EMPLOYEE entity, convert each of the one-to-many association into a weak entity and a relationship. Identify the discriminator of each weak entity and the attributes of each relationship.

.....

.....

.....

.....

Question 4. Give examples of:

- A many-to-many relationship in which one of the participant is a weak entity.
- A many-to-many relationship in which one of the participant is another relationship.
- An existence dependency.
- A weak entity.
- An entity with composite attributes.
- An entity with multivalued attributes.

.....

.....

.....

.....

.....

Question 5. The people's Bank offers five types of accounts: ~~loan~~, checking, premium savings, daily interest saving, and money market. It operates a number of branches and a client of the bank can have any number of accounts. Accounts can be joint, i.e., more than one client may be able to operate a given account. Identify the entities of interest and show their attributes. What relationships exist among these entities? Draw the corresponding E-R diagram.

Question 6. A University decides to computerise its registration system. Identify the possible entities and relationships and Draw the E-R Diagram.

1.4 FUNCTIONAL DEPENDENCY AND NORMALISATION

1.4.1 Functional Dependency

Question 1. Find the FDs in following relations. Identify the problem and give the remedy.

Enrollment

Student-Name	Course	Phone No.	Department	Grade
Jones	353	237-4539	Comp. Science	A
Ng	329	427-7390	Chemistry	A
Jones	328	237-4539	Comp. Science	B
Martin	456	388-5183	Physics	C
Dulles	293	371-6259	Decision Science	B
Duke	491	823-7293	Mathematics	C
Duke	353	823-7293	Mathematics	B

Student-Name	Course	Phone No.	Department	Grade
Jones	491	237-4539	Comp. Science	C
Evan	353	842-1729	Comp. Science	A+
Baxter	379	839-0827	English	B

Question 2. Consider the following relation TIME_TABLE.

Professor	Course	Room	Day	Time
Mrs. Shah	Maths	20	Mon.	9:00 a.m.
Mr. Budhiraja	History	10	Mon	9.00 a.m.
Mrs. Shah	Maths	22	Tue	10:00 a.m.
Mrs. Mehta	Commerce	40	Wed	11:00 a.m.
Mr. Budhiraja	History	11	Fri	12:00 a.m.
Mr. Kumar	Maths	23	Wed	12:00 a.m.

Out of the FDs Course \rightarrow Prof, and Prof \rightarrow Course, which one is satisfied and which one not. Give reasons.

Question 3. Consider the relation STUDENT_INFORMATION.

Name	Course	Tel No.	Major	Prof	Grade
Jyoti	History	5579590	Maths	Mrs. Shah	A
Neeta	English	4254589	Hindi	Mrs. Vats	B
Jyoti	Maths	5579590	Maths	Mr. Kumar	B
Mahesh	Hindi	6239387	Physics	Mr. Venkatesh	A
Sumit	Computer Science	2231748	Commerce	Mr. Mishra	C
Vikas	Physical Chemistry	6346666	Chemistry	Mr. Chaubey	B
Vikas	Political Science	6346666	Chemistry	Mr. Pandey	In prog

Find the various FDs in above relation, which are satisfied.

Question 4. Consider the relation STUDENT_ADVISOR (Name, Department, Advisor) with the following functional dependencies:

Name \rightarrow Department

Name \rightarrow Advisor

Advisor \rightarrow Department

Decompose the relation STUDENT_ADVISOR into STUDENT_DEPARTMENT (Name, Department) DEPARTMENT_ADVISOR (Department, Advisor). Tell whether the decomposition is lossy or not.

Question 5. Here are two sets of FDs for a relation R {A, B, C, D, E}. Are they equivalent?

- 1) A \rightarrow B AB \rightarrow C D \rightarrow AC D \rightarrow E
 2) A \rightarrow BC D \rightarrow AE

Model Answers

Answer 1: The relation has following FD

Student-Name \rightarrow Phone No.

Student-Name \rightarrow Department

Student-Name, Course \rightarrow Grade

The problem with the relation is that unless the student takes atleast one-course, we cannot enter data for the student. And the other problem is that a change in the Phone No. or Department can lead to inconsistencies in the database. To rectify the relation is decomposed as follows:

Student

Student-Name	Phone No.	Department
Jones	237-4539	Comp. Science
Ng	427-7390	Chemistry
Martin	388-5183	Physics
Dulles	371-6259	Decision Science
Duke	823-7293	Mathematics
Evan	842-1729	Comp. Science
Baxter	839-0827	English

Enroll:

Student-Name	Course	Grade
Jones	353	A
Ng	329	A
Jones	328	B
Martin	456	C
Dulles	293	B
Duke	491	C
Duke	353	B
Jones	491	C
Evan	353	A+
Baxter	379	B

Answer 2.

In order to verify whether a given FD $X \rightarrow Y$ is satisfied by a relation R on a relational scheme R or not, we find any two tuples with the same X value, the FD $X \rightarrow Y$ is satisfied in R, if for the same value of X the values of Y in tuples must be same.

Thus, here the FD Prof \rightarrow Course is satisfied, as for any two same value for Prof, the value of Course will always be same, example, Mr. Budhiraja is teaching history only, and Mrs. Shah is teaching mathematics only.

But the FD Course \rightarrow Prof is not satisfied, as Maths could be taught by Mrs. Shah as well as by Mr. Kumar, thus, for the same value of Course the value of Prof is not same.

Answer 3. The following function dependencies are satisfied in the above relation.

Name \rightarrow Tel No

Name \rightarrow Major

Name, Course \rightarrow Grade

Course \rightarrow Prof (The present instance of table does indicate this FD as well Prof \rightarrow Course FD, one needs more data to find which of the two actually hold or ask questions as per example 2 above.)

Answer 4. We decompose the STUDENT_ADVISOR into two relations:

STUDENT_DEPARTMENT and DEPARTMENT_ADVISOR. We join the both of the relations STUDENT_DEPARTMENT and DEPARTMENT_ADVISOR into a new relation NEW_STUDENT_ADVISOR.

We can clearly see that the new relation NEW_STUDENT_ADVISOR is lossy.

STUDENT_ADVISOR

Name	Department	Advisor
Jyoti	Maths	Mrs. Shah
Neeta	Hindi	Mrs. Vats
Mahesh	Physics	Mr. Venkatesh
Sumit	Commerce	Mr. Mishra
Vikas	Chemistry	Mr. Chaubey

STUDENT_DEPARTMENT

Name	Department
Jyoti	Maths
Neeta	Hindi
Mahesh	Maths
Sumit	Physics
Vikas	Commerce
Mr. Chaubey	Chemistry
Mr. Pandey	Chemistry

DEPARTMENT_ADVISOR

Advisor	Department
Mrs. Shah	Maths
Mrs. Vats	Hindi
Mr. Kumar	Maths
Mr. Venkatesh	Physics
Mr. Mishra	Commerce

NEW_STUDENT_ADVISOR

Name	Department	Advisor
Jyoti	Maths	Mrs. Shah
Jyoti	Maths	Mr. Kumar
Neeta	Hindi	Mrs. Vats
Mahesh	Physics	Mr. Venkatesh
Sumit	Commerce	Mr. Mishra
Vikas	Chemistry	Mr. Chaubey
Vikas	Chemistry	Mr. Pandey

Please note that Student_Advisor relation suffers from Insertion anomaly. The information that Mr. Kumar is an advisor of Maths department cannot be stored in the table as he is not advisor to any student at present. This information is shown in Department_Advisor. On taking join of Department_Advisor and Student_Department one will get as many tuples per student as the total number of advisors in that department. Mathematically, the joining attribute of the two tables is Department. The decomposition would have been lossless if any of the following had hold: Department \rightarrow Name OR Department \rightarrow Advisor. Both of them are not in the FDs, thus decomposition is lossy.

Please note that this decomposition is not preserving the FDs also as the functional dependency Name \rightarrow Advisor is lost. Please find out which normal form the relation is presently in. If we decompose the relation as Student_Advisor and Advisor_Department, the decomposition will be loss-less. Please note in the said case the dependency Name \rightarrow Department will be maintained as transitive dependencies:

Name \rightarrow Advisor (in Name_Advisor)
and Advisor \rightarrow Department (in Advisor_Department)
 \Rightarrow Name \rightarrow Department

Answer 5. They are equivalent. Let us consider the FDs of the first set as follows:

- 1) A \rightarrow B
- 2) AB \rightarrow C
- 3) D \rightarrow AC
- 4) D \rightarrow E

Please note that if A \rightarrow B, then for any given value of A, B can be determined, which implies that in FD AB \rightarrow C, only by knowing A we can determine B and, therefore, C also.

Thus instead of FD AB \rightarrow C; A \rightarrow C is a better version as A \rightarrow B exist.

The FD D \rightarrow AC also means, D \rightarrow A as well as D \rightarrow C, since A \rightarrow C has been just derived therefore, D \rightarrow A and A \rightarrow C implies D \rightarrow C by transitivity and so can be dropped, leaving the FD.

D \rightarrow A

The first set of FDs is thus equivalent to the following irreducible set:

$A \rightarrow B$
 $A \rightarrow C$
 $D \rightarrow A$
 $D \rightarrow E$

The second set of FDs

$A \rightarrow BC$
 $D \rightarrow AE$ is clearly also equivalent to this irreducible set. Thus, the two given sets are equivalent.

Exercise 2

Question 1: Given a relation $R(x, y, z, t)$ and a set of functional dependence

$X \rightarrow y$
 $x \rightarrow z$
 $x \rightarrow t$

Is the decomposition of relation in $R_1(x, y)$, $R_2(x, z)$ and $R_3(x, t)$ loss-less? Is it necessary to decompose the table? Suggest situations where you would like to have such decomposition.

.....
.....

Question 2: Remove the redundant FDs from the following set:

$A \rightarrow BC$
 $AB \rightarrow CD$
 $B \rightarrow D$
 $D \rightarrow E$
 $BC \rightarrow D$

What are the possible candidate keys?

.....
.....
.....

Question 3: In a given instance of a relation, how will you ascertain whether a given FD exists? Explain with examples.

.....
.....
.....
.....

Question 4. What is non-key?

.....
.....

Question 5. Define Second-Normal Form.

.....
.....

Question 6. Define the Third Normal Form and the Boyce/Codd Normal form with examples.

.....
.....
.....
.....

Question 7. What are Multilevel Dependencies?

.....
.....
.....
.....

1.4.2 Normalisation

Consider the following relation STUDENT. Normalise the following relation:

Table 4.1

Student-Id	FirstName	LastName	Major
123-45-6789	Jack	Jones	Library and Information Science
222-33-4444	Lynn	Lee	Library and Information Science
987-65-4321	Mary	Ruiz	Pre-Medicine
123-54-3210	Lynn	Smith	Pre-Law
111-33-5555	Jane	Jones	Library and Information Science

You can easily verify for yourself that above relation STUDENT satisfies the definition of 1NF: viz., it has no duplicated rows; each cell is single-valued (i.e., there are no repeating groups or arrays); and all the entries in a given column are of the same kind:

In Table STUDENT, we can see that the key, Student-Id, functionally determines the other attributes; i.e., a given Student-Id implies (determines) a particular value for each of the attributes FirstName, LastName, and Major (assuming, at least for the moment, that a student is allowed to have only one major). In the arrow notation:

Student-Id → FirstName

Student-Id → LastName

Student-Id → Major.

A key attribute will, by the definition of key, uniquely determine the values of the other attributes in a table: i.e., all non-key attributes in a table will be functionally dependent on the key. But there may be non-key attributes in a table that determine other attributes in that table. Consider the following table:

Table 4.2

FirstName	LastName	Major	Level
Jack	Jones	LIS	Graduate
Lynn	Lee	LIS	Graduate
Mary	Ruiz	Pre-Medicine	Undergraduate
Lynn	Smith	Pre-Law	Undergraduate
Jane	Jones	LIS	Graduate

In Table 4.2 the Level attribute can be said to be functionally dependent on the Major attribute. Thus, we have an example of an attribute that is functionally dependent on a non-key attribute. This statement is true in the table *per se*, and that is all that the definition of functional dependence requires; but the statement also reflects the real-world fact that Library and Information Science (LIS) is a major that is open only to graduate students and that Pre-Medicine and Pre-Law are majors that are open only to undergraduate students.

Table 4.2 has another interesting aspect. Its key is a composite key, consisting of the paired attributes, FirstName and LastName. The Level attribute is functionally dependent on this composite key, of course; but, in addition, Level can be seen to be dependent on only the attribute LastName. (This is true because each value of Level is paired with a distinct value of LastName. In contrast, there are two occurrences of the value Lynn for the attribute FirstName, and the two Lynns are paired with different values of Level, so Level is not functionally dependent on FirstName.)

Problems in Table 4.2-

Thus this table fails to qualify as a 2nd Normal Form table, since the definition of 2NF requires that all non-key attributes be dependent on all of the key. (Admittedly, this example of a partial dependency is artificially contrived, but nevertheless it illustrates the problem of partial dependency.)

We can turn Table 4.2 into a table in 2NF in an easy way, by adding a column for the Student-Id, which will then be the natural thing to use as the key.

Table 4.3

Student-Id	FirstName	LastName	Major	Level
123-45-6789	Jack	Jones	LIS	Graduate
222-33-4444	Lynn	Lee	LIS	Graduate
987-65-4321	Mary	Ruiz	Pre-Medicine	Undergraduate
123-54-3210	Lynn	Smith	Pre-Law	Undergraduate
111-33-5555	Jane	Jones	LIS	Graduate

With the Student-Id defined as the key, Table 4.3 is in 2NF, as you can easily verify. This illustrates the fact that any table that is in 1NF and has a single-attribute (i.e., a non-composite) key is automatically also in 2NF.

Problems in Table 4.3-

1. It contains some repeated information about the LIS-Graduate pairing.
2. Consider what would happen to our knowledge (at least, as explicitly contained in a table) of the level of the major, Pre-Medicine, if Mary Ruiz left Enormous State University. With the deletion of the row for Ms. Ruiz, we would lose the information that Pre-Medicine is an Undergraduate major. This is an example of a **deletion anomaly**. We may possess the real-world information that Pre-Medicine is an Undergraduate major, but no such information is explicitly contained in a table in our database.
3. Suppose that a new student wants to enroll in ESU: e.g., suppose Jane Doe wants to major in Public Affairs. From the information in Table 4.1 we cannot tell whether Public Affairs is an Undergraduate or a Graduate major; in fact, we do not even know whether Public Affairs is an established major at ESU. We do not know whether it is permissible to insert the value, Public Affairs, as a value of the attribute, Major, or what to insert for the attribute, Level, if we were to assume that Public Affairs is a valid value for Major. The point is that while we may possess real-world information about whether Public Affairs is a major at ESU and what its level is, this information is not explicitly contained in any table that we have thus far mentioned as part of our database. This is **insertion anomaly**.

Although Table 4.3 is in 2NF, it is still open to the problems of insertion and deletion anomalies, as the discussion in the preceding section shows. The reason is that Table 4.3 deals with more than a single theme.

What can we do to turn it into a set of tables that are, or at least come closer to being, single-theme tables?

A reasonable way to proceed is to note that Table 4.3 deals with both information about students (their names and STUDENT-IDs) and information about majors and levels. This should strike you as two different themes.

Presented as follows is one possible set of single-theme tables dealing with the information in Table 4.3. One question which has also been addressed is: Can a student have more than one majors? If the answer is not then we can represent the information in just two relations R1 (Student-Id, FirstName, LastName, major) and R2 (Major, Level). But if the answer to the question is no (please note in that case key in table 4.3 will be (Student-Id + Major) then we need to decompose the relation in following three tables:

Table 4.4

Student-Id	FirstName	LastName
123-45-6789	Jack	Jones
222-33-4444	Lynn	Lee
987-65-4321	Mary	Ruiz
123-45-4321	Lynn	Smith
111-33-5555	Jane	Jones
999-88-7777	Newton	Gingpoor

Table 4.5

Major	Level
LIS	Graduate
Pre-Medicine	Under-graduate
Pre-Law	Under-graduate
Public Affairs	Graduate

Table 4.6

Major	Student-Id
LIS	123-456789
LIS	222-33-4444
Medicine	987-68-4321
Pre-Law	123-54-3210
LIS	111-33-5555

The three preceding tables should strike you as providing a better arrangement of the information in Table 4.3.

For one thing, this arrangement puts the information about the students into a smaller table, Table 4.4, which happily fails to contain redundant information about the LIS-Graduate pairing. For another thing, this arrangement permits us to enter information about students (e.g., Newton Gingpoor) who have not yet identified themselves as pursuing a particular major. For still another thing, it puts the information about the Major-Level pairings into a separate table,

Table 4.5, which can easily be expanded to include information (e.g., that the Public Affairs major is at the Graduate level) about majors for which, at the moment, there may be no students registered.

Table 4.6 provides the needed link between individual students and their majors (note that Newton Gingpoor's STUDENT-ID is not in this Table 4.6, which tells us that he has not yet selected a major).

Tables 4.4 to 4.6 are single-theme tables and are in 3NF, as you can easily verify.

Please note that the example covered here just try to reflect how you may do a design without worrying too much about the definitions of normalisation. However, for complete details on FDs, MVDs and normalisation please refer to block1 unit 3 of this course where overall process is explained through examples.

We are not giving any specific exercises in this section as normalisation is part of overall design process, therefore, you must normalize all the designs, which you will create as per further exercises.

1.5 STRUCTURED QUERY LANGUAGE (SQL)

You have already gone through SQL in CS-06, we present an example here.

Consider the supplier relations.

S

S# (Supplier No.)	SNAME (Supplier Name)	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Adams	30	Athens
S5	Clark	20	London

SP

S#	P# (Part No.)	Quantity
S1	P1	300
S1	P2	200
S2	P1	100
S2	P2	400
S3	P2	200
S4	P2	200

- Question 1. Get supplier numbers for suppliers with status > 20 and city is Paris.
.....
- Question 2. Get Supplier Numbers and status for suppliers in Paris, in descending order of status.
.....
- Question 3. Get all pairs of supplier numbers such that the two suppliers are located in the same city. (Hint: It is retrieval involving join of a table with itself.)
.....
- Question 4. Get unique supplier names for suppliers who supply part P2.
.....
- Question 5. Give the same query above by using the operator IN.
.....
- Question 6. Get part numbers supplied by more than one supplier. (Hint: It is retrieval with a sub-query, with interblock reference and same table involved in both blocks).
.....
- Question 7. Get supplier numbers for suppliers who are located in the same city as supplier S1. (Hint: Retrieval with sub query and unqualified comparison operator).
.....
- Question 8. Get supplier names for suppliers who supply part P1. (Hint: Retrieval using EXISTS)
.....

Question 9. Get part numbers for parts whose quantity is greater than 200 or are currently supplied by S2. (Hint: Use a retrieval using union).

.....

Question 10. Suppose for the supplier S5 the value for status is NULL instead of 20. Get supplier numbers for suppliers greater than 25 or is NULL. (Hint: Retrieval using NULL).

.....

Question 11. Get the number of Suppliers, who are supplying at least one part. (Hint: This query is using the built-in function count).

.....

Question 12. For each part supplied, get the part no. and the total quantity supplied for that part. (Hint: The query using GROUP BY).

.....

Question 13. Get part numbers for all parts supplied by more than one supplier. (Hint: It is GROUP BY with HAVING).

.....

Question 14. For all parts such that the total quantity supplied is greater than 300 (exclude from the total all shipments for which quantity is less than or equal to 200), get the part no. and the maximum quantity of the part supplied, and order the result by descending part no. within those maximum quantity values which are in ascending order.

.....

Question 15. Double the status of all suppliers in London. (Hint: UPDATE Operation).

.....

Question 16. Let us consider the table TEMP has one column, called P#. Enter into TEMP part numbers for all parts supplied by S2.

.....

Question 17. Add part P7.

.....

Question 18. Delete all the suppliers in London and also the supplies concerned.

.....

Question 19. Consider the following Relational database.

employees (eno, ename, address, basic salary)

projects (pno, pname, nos-of-staff-alotted)

workin (pno,eno,pjob)

Two queries regarding the data in the above database have been formulated in SQL. Describe the queries in English sentences.

```
(i) SELECT ename
     FROM employees
     WHERE eno IN ( SELECT eno
                   FROM workin
                   GROUP BY eno
                   HAVING COUNT (*) =
                   (SELECT COUNT (*) FROM projects));
```

```
(ii) SELECT pname
      FROM projects
      WHERE pno IN ( SELECT pno
                    FROM projects
                    MINUS
                    (SELECT DISTINCT pno
                     FROM workin
                     GROUP BY eno));
```

.....

.....

.....

.....

.....

.....

.....

Model Answers

Please note:

In certain Database Management System S# or P# may not be accepted kindly use sno, pno instead.

```
Answer 1.  SELECT S#
           FROM S
           WHERE CITY = 'PARIS'
           AND STATUS > 20
```

Result:

S#
S3

```
Answer 2.  SELECT S#, STATUS
           FROM S
           WHERE CITY = 'PARIS'
           ORDER BY STATUS DESC
```

Result:

S#	STATUS
S3	30
S2	10

Answer 3.
SELECT FIRST.S# , SECOND.S#
FROM S FIRST , S SECOND
WHERE FIRST.CITY = SECOND.CITY AND FIRST.S# < SECOND.S#

Please note the condition after AND is needed to make sure that you do not get tuples where Supplier number is same; and do not get the tuples which are repeated in reverse order. For example, S1 S5's reverse order is S5 S1. You may try the query by not giving clause after AND also, you will find the difference.

Result:

S#	S#
S1	S5
S2	S3

Answer 4.
SELECT DISTINCT SNAME
FROM S , SP
WHERE S.S# = SP.S#
AND SP.P# = 'P2'

Result:

SNAME
Smith
Jones
Blake
Adams

Or,
SELECT SNAME
FROM S
WHERE S# = ANY (SELECT S#
FROM SP
WHERE P# = 'P2')

In this query, we are making a set of supplier numbers using the sub query "who supply a part P2".

Answer 5.
SELECT SNAME
FROM S
WHERE S# IN
(SELECT S#
FROM SP
WHERE P# = 'P2')

Answer 6:

```
SELECT DISTINCT P#
FROM SP SPX
WHERE P# IN ( SELECT P#
              FROM SP
              WHERE SP.P# = SPX.P# AND SP.S# < SPX.S#)
```

Result:

P#
P1
P2

Answer 7.

```
SELECT S#
FROM S
WHERE CITY = (SELECT CITY
              FROM S
              WHERE S# = 'S1')
```

Please note that this query will work as S# is the key to relation S and will result in one city for Supplier S1. It may create problem if value of city for S1 is NULL.

Result:

S#
S1
S5

Answer 8.

```
SELECT SNAME
FROM S
WHERE EXISTS (SELECT *
              FROM SP
              WHERE SP.S# = S.S#
              AND P# = 'P1')
```

Result:

SNAME
SMITH
JONES

Answer 9.

```
SELECT P#
FROM SP
WHERE QUANTITY > 200 UNION SELECT P#
                             FROM SP
                             WHERE S# = 'S2'
```

Result:

Practical on
RDBMS

P#
P1
P2

Answer 10. SELECT S#
FROM S
WHERE STATUS > 25 OR STATUS IS NULL

Result:

S#
S3
S4
S5

Answer 11. SELECT COUNT (DISTINCT S#)
FROM SP

Result: 4

Answer 12. SELECT P#, SUM(QUANTITY)
FROM SP
GROUP BY P#

Result:

P#	
P1	400
P2	1000

Answer 13. SELECT P#
FROM SP
GROUP BY P#
HAVING COUNT (*) > 1

Please note it is almost the same query as we had in question 6.

Answer 14. SELECT P#, MAX(QUANTITY)
FROM SP
WHERE QUANTITY > 200
GROUP BY P#
HAVING SUM(QUANTITY) > 300
ORDER BY 2, P# DESC

This is a very interesting query, please try various ascending, descending combinations.

Answer 15: UPDATE S
SET STATUS = 2 * STATUS
WHERE CITY = 'LONDON'

Answer 16. INSERT INTO TEMP
SELECT P#
FROM SP
WHERE S# = 'S2'

Answer 17. INSERT INTO SP(S#,P#,QUANTITY)
VALUES ('S5','P7',100)

Note: Part P7 can be added only if one Supplier is supplying it; therefore, we assume that Supplier S5 is supplying it.

Answer 18. To make sure that on deletion of few tuples from S will cause Cascading deletion from SP, we need to set up Referential Integrity constraint. If this constraint is set between SP and S, then the deletion of record in S will cause Cascading deletion of records in SP.

Thus, in such cases the following command will be sufficient to perform the required deletion.

```
DELETE FROM S
WHERE S# IN ( SELECT S#
              FROM S
              WHERE CITY = 'LONDON')
```

Answer 19. (i) Give names of employees who are working on all projects.
(ii) Give names of the projects, which are currently not being worked upon

Exercise 3

Question 1. An orchestra database consists of the following relations:

CONDUCTS (Conductor, Composition)

REQUIRES (Composition, Instrument)

PLAYS (Player, Instrument)

LIKES (Player, Composition)

Give the query in SQL,

1) List the players and their instruments who can be part of the orchestra when Latifa Melody conducts.

2) From the list of players on previous page, identify those who would like the composition they are to play.

Practical on
RDBMS

.....
.....
.....
.....

Question 2. Consider the following relational scheme:

COURSES (cno, cname)

STUDENTS (roll no, Sname, age, year)

REGISTERED FOR (cno, roll no)

The underlined attributes indicate the primary key for relations. The "year" attributed for STUDENTS relation indicates the year in which the student is currently studying (First year, second year etc.)

Write a SQL Query to find the age and year of youngest student in each year.

.....
.....
.....
.....

Question 3. Consider the Car-insurance database given by Q1 in Section E-R Diagrams. Construct the tables and enter meaningful data and create following SQL Queries for this relational database.

- 1) Find the total number of people whose cars were involved in accidents in 1998.
- 2) Find the total number of accidents in which the cars belonging to "Ram Kumar" were-involved.
- 3) Add a new customer to the database.
- 4) Delete the Mercedes belonging to "Ram Kumar."
- 5) Add a new accident record for the Toyota belonging to "Rashmi Sinha."

.....
.....
.....
.....
.....

Question 4. Consider the DIRECTOR table given below:

Director

Dir_id	Dir_lname	Dir_fname	Dir_phone	Dir_address	Dir_city	Dir_state	Dir_zip
D001	Lawrence	Marlin	415658-9932	6223, bateman	Berkely	CA	94705
D002	Laker	Larry	415986-7020	309, 63 rd floor, St # 441	Oakland	CA	94618
D003	Nichols	Mike	415548-7723	589 Darwin Ln.	Berkely	CA	94705
D004	Schaeffer	Eric	801826-0742	67, Seventh Avenue	Salt lake city	UT	84152
D005	Frears	Stefen	801826-0752	3 balding Place	Salt lake city	UT	84152

Some sample records are given in the table above.

- a) Add more records in the table using SQL.
- b) Perform following queries:
 - Display all the rows in the director table.
 - List the name of directors who live in state CA.
 - List the id's of all directors. To improve readability change the headings suitably.
 - List the name of directors who have id's between 'D003' and 'D005' (both inclusive).
 - List the names of all directors whose phone numbers start with 41.

.....
.....
.....
.....
.....

Question 5. For the PROJECT, EMPLOYEE, and ASSIGNED_TO relations-

PROJECT (Project#, Project_Name, Chief_Architect)

EMPLOYEE (Emp#, EmpName)

ASSIGNED_TO (Project#, Emp#)

express the following queries in SQL:

- 1) Get Emp# of employees working on project numbered COMP100.
- 2) Get details of employees (name and number) working on project COMP100.
- 3) Get details of employees working on all database projects.
- 4) Get details of employees working on both COMP100 and COMP101.

- 5) Get employee numbers of employees who work on at least all those projects that employee 105 works on.
- 6) Get employee numbers of employees who do not work on project COMP200.
- 7) Get employee numbers of employees who work on all projects.
- 8) Get employee numbers of employees who work on at least one project that employee 105 works on.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Question 6. Consider the employee database given below. Give an expression in SQL for each of the following queries:

EMPLOYEE (Employee-Name, Street, City)

WORKS (Employee-Name, Company-Name, Salary)

COMPANY (Company-Name, City)

MANAGES (Employee-Name, Manager-Name)

- 1) Find the names of all employees who work for First Bank Corporation.
- 2) Find the names and cities of residence of all employees who work for the First Bank Corporation.
- 3) Find the names, street address, and cities of residence of all employees who work for First Bank Corporation and earn more than Rs.10,000.
- 4) Find the employees in the database who live in the same cities as the companies for which they work.
- 5) Find all employees in the database who live in the same cities and on the same streets as do their managers.
- 6) Find all employees in the database who do not work for First Bank Corporation.

- 7) Find all employees in the database who earn more than every employee of Small Bank Corporation.
- 8) Assume that the companies may be located in several cities. Find all companies located in every city in which Small Bank Corporation is located.
- 9) Find all employees who earn more than the average salary of all employees of their company.
- 10) Find the company that has the most employees.
- 11) Find the company that has the smallest payroll.
- 12) Find those companies whose employees earn a higher salary, on average, than the average salary at First Bank Corporation.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

1.6 MICROSOFT-ACCESS

- Question 1. Design a package known as "Library Book Collection System" that will make accessing books and author details in a library easier and more efficient. It should make use of simple forms and queries to generate reports on authors and books.
- Question 2. Consider a database for EMPLOYEE and their dependents and show the steps of creating tables, giving queries, making forms and reports etc. in MS-ACCESS.

Model Answers

Answer 1. In the beginning we need to know that what are our goals in a "Library Book Collection System", i.e., what should be the functionality in the system.

In broad sense, our "Library Book Collection System" should be able to provide:

- Detailed Report on all authors and books
- List of books on same topics or subjects.
- Displaying all the books written by an author.

Please draw the E-R diagrams of this system yourself assuming at least three entities: Book, Author, and Publisher.

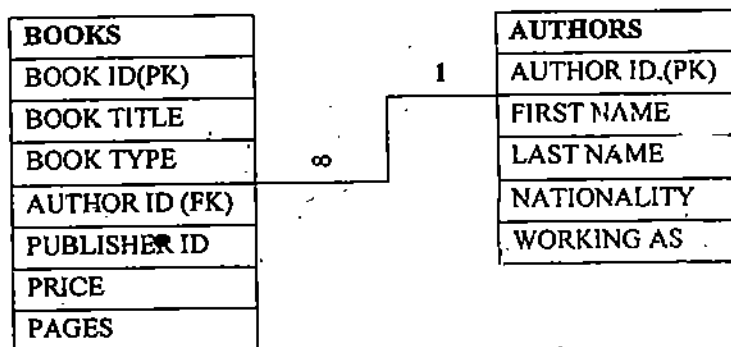
However, the objective of this example is to show how tables and forms may be created, so we are assuming a very elementary design of the database, as follow:

Database Name: Book Collection

Number of tables: 2

Name of tables: 1) Books and, 2) Authors

Now let draw schematic diagram to show table Relationships using primary/foreign key.



Assumption: One book is written by only one author. (You must attempt to remove this assumption and in that case your design will be different. In fact it will of three tables.)

PK: PRIMARY KEY

FK ; FOREIGN KEY

FORM DESIGN

Form Name	Form Type	Description	Table/Query used
BOOK DETAILS	Columnar	Used to enter data in the BOOKS table	BOOKS & AUTHOR
AUTHOR DETAIL	Columnar	Used to enter data in the AUTHORS table	AUTHORS

The form would like as follows:

AUTHOR DETAILS

AUTHOR ID	<input type="text"/>
FIRST NAME	<input type="text"/>
LAST NAME	<input type="text"/>
NATIONALITY	<input type="text"/>
WORKING AS	<input type="text"/>
PHOTOGRAPH	<input type="text"/>

BOOK DETAILS

BOOK	<input type="text"/>
BOOK TITLE	<input type="text"/>
BOOK TYPE	<input type="text"/>
AUTHOR-ID/NAME	<input type="text"/>
PUBLISHER-ID	<input type="text"/>
PRICE	<input type="text"/>
PAGES	<input type="text"/>

BOOK ID	TITLE	TITLE TYPE	PUB. ID	PRICE	PAGES

Validations to be performed:

- 1) BOOK ID in the BOOKS table to be unique.

- 2) AUTHOR ID in the AUTHORS table to be unique.
- 3) BOOK TYPE to be "General", "Technical", "Fiction", "Science" etc.
- 4) PRICE > 0.
- 5) TOTAL PAGES > 0
- 6) None of the fields to contain NULL values.
- 7) AUTHOR ID in BOOKS table must exist in AUTHORS table (Referential Integrity).

QUERIES USED

Query Name	Query Type	Description	Table used
AUTHORS	Simple	Displays details of all authors	AUTHORS
BOOKS	Simple	Displays details of all books	BOOKS
BOOK AND AUTHORS	Simple	Displays both book details and their corresponding author details	AUTHORS AND DETAILS

REPORT OUTLINE

Report Name	Report Type	Description	Tables/Query used
AUTHORS	AUTO REPORT COLUMNAR	Displays details of all authors	AUTHORS
BOOKS AND AUTHORS	MULTIPLE TYPE REPORT	Displays details of all authors and books	AUTHORS AND BOOKS

REPORTS

AUTHORS USED

AUTHORS	
AUTHOR ID	1
FIRST NAME	Hari Shankar
LAST NAME	Parsai
NATIONALITY	Indian
WORKING AS	Artist
PHOTOGRAPH	

AUTHORS AND BOOKS

<u>AUTHOR ID</u>	1		
<u>NAME</u>	Hari Shankar Parsai		
<u>NATIONALITY</u>	Indian		
<u>WORKING AS</u>	Artist		
 <u>BOOKS BY HIM:</u>			
<u>BOOK TYPE</u>	Literature (Satire)		
<u>BOOK ID</u>	<u>BOOK TITLE</u>	<u>TOTAL PAGES</u>	<u>PRICE</u>
1	Tab ki baat aur thi	100	50/-
2	Kahani Sangrah	500	100/-

The above is just a sample design; you may create your own designs based on the needs. Try implementing above design in MS-Access after going through answer 2 given here.

Answer 2. Let us consider the table Employee and dependents with the description as follows. See the steps for creating the table in the third block of your course i.e., the block for MS-ACCESS. You must design more complex database after going through that block.

As we know any table in MS-Access has three columns viz., FIELD NAME, DATA TYPE, DESCRIPTION.

Let us consider the case of EMPLOYEE table. The columns of EMPLOYEE table will have following fields:

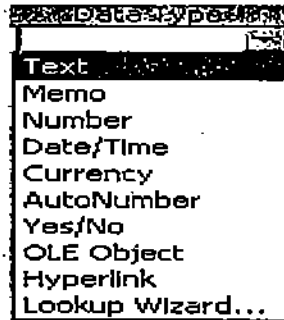
- Employee Name
- Employee Code
- Employee Salary
- Employee Date of joining

Let us consider the case of DEPENDENTS table. The columns of DEPENDENTS table will have following fields:

- Field Name
- Employee Code
- Dependent Name
- Relation with Employee

Datatype

There are different datatypes provided by the MS-Access, as:



For above example, we will choose the following data types for the various attributes.

EMPLOYEE

Field Name	Data Type
Employee Name	text
Employee Code	Text
Employee Salary	number
Employee Date of joining	date/time

DEPENDENTS

Field Name	Data Type
Employee Code	Text
Dependent Name	Text
Relation with Employee	Text

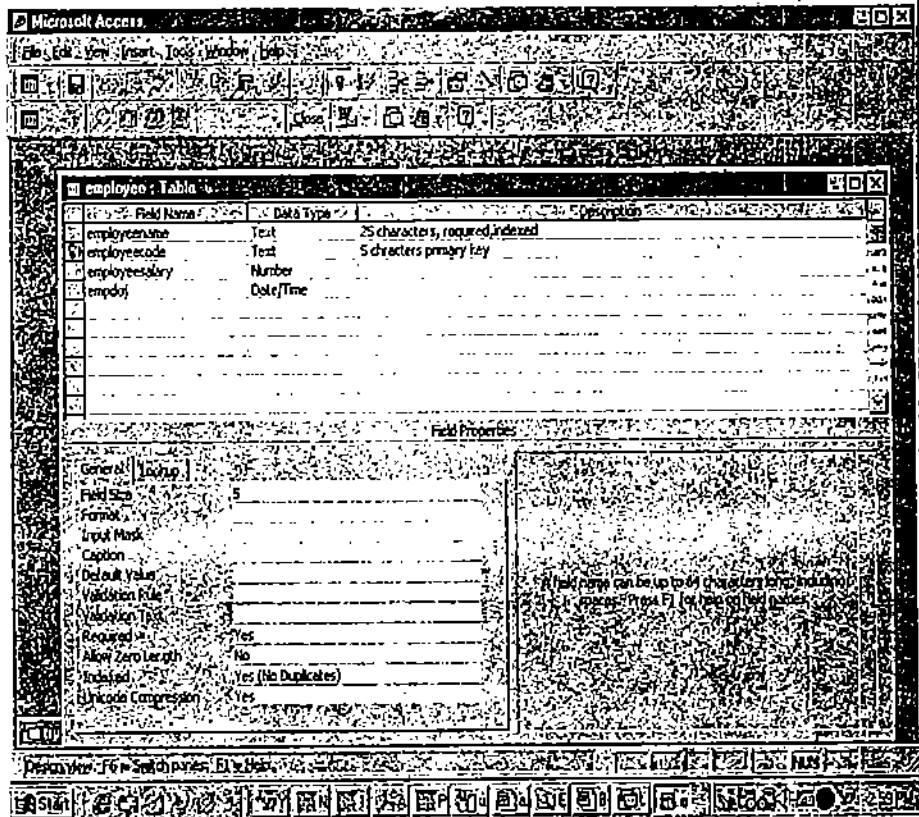
Description Field

It contains the description of the field for example for the Employee name the description will be "this field is for the name of the Employee of the Organisation".

Creating a Table

The easiest way to create most common tables for business or personal use is with the table Wizard however you can also create a table without the Table Wizard. Here we will be considering creating a table without using the wizard. The steps are:

- 1) In the Database window, click the Table button (or choose Tables from the View menu).
- 2) Choose the New button. Microsoft Access displays the New Table dialog box.
- 3) Choose the New Table button. Microsoft Access opens a blank Table window in Design view.



The above figure shows the creation of the Employee table in the same way you should create the Dependent Table.

When you design or modify a table, you specify the fields that you want it to contain in the upper portion of the Table window. In the lower portion you can enhance your table by setting properties for each field.

Setting a Primary Key

Microsoft Access works most efficiently if you specify a primary key. The primary key of a table consists of one or more fields that uniquely identify each record you store in the table. A primary key is often an ID number or code, since this type of value is always different for each record.

Steps to set the Primary key:

- 1) Click the field that you want to use for the primary key. To create a multiple-field primary key, hold down the CTRL key and click the field selector to the left of each field that you want to include.
- 2) From the Edit menu, choose Set Primary Key (or click the set primary key button on the toolbar).
- 3) Microsoft Access adds a key indicator to the left of the field or fields you specify as the primary key.

In the Employee Table the Primary key is the Employee code and in the Dependent Table the Primary key is a multiple-field primary key, which includes Employee code and Dependent name

Saving the Table

After you add fields to a table, you must save the table design before you can add any records. After you save the Table, its title appears in the list of tables in the Database window.

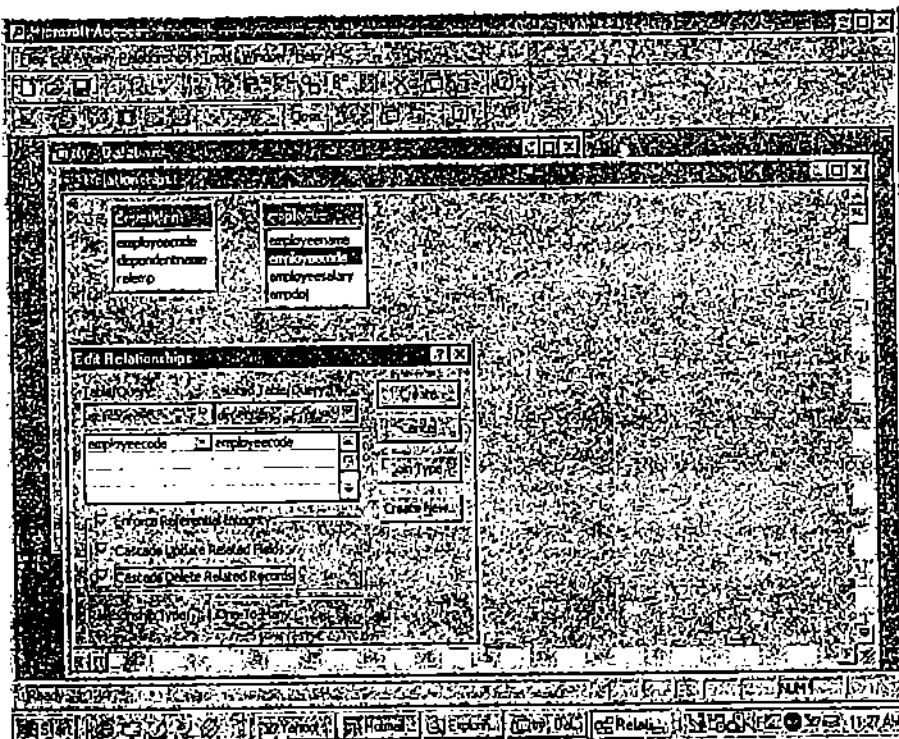
Steps to save and name a Table:

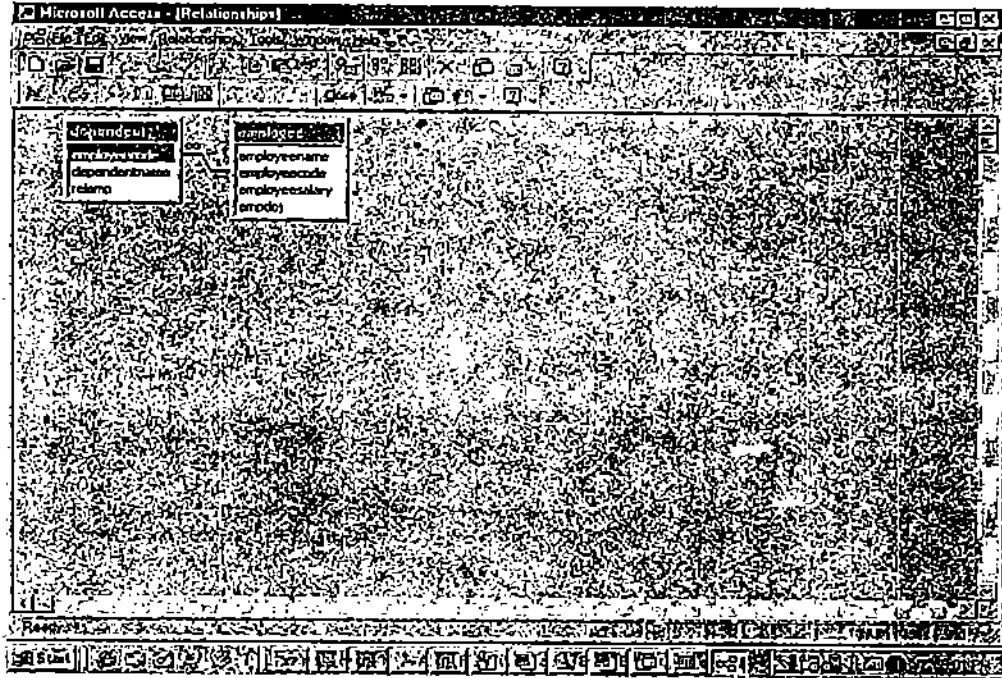
- 1) From the File menu, choose Save.
- 2) If you are saving the table for the first time, type a name for the table, and then choose OK.

Setting of Relationship

Steps for setting of relationship:

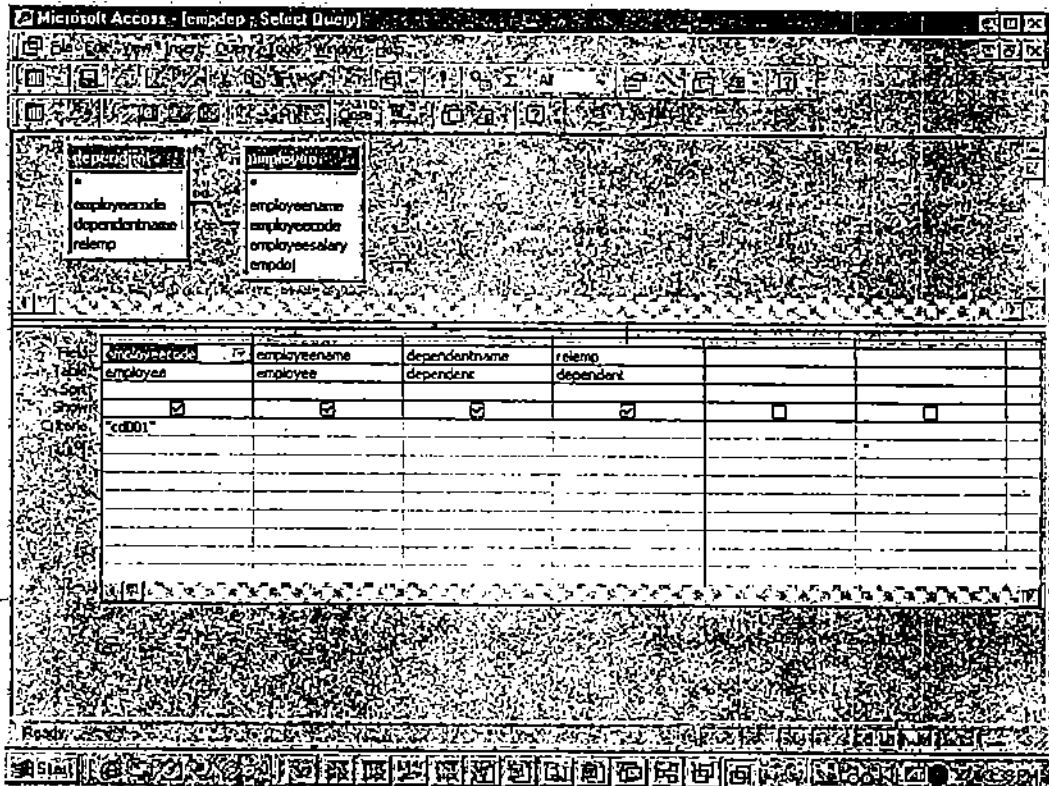
- 1) Employee code of EMPLOYEE should be linked to employee code of DEPENDENT.
- 2) Here we would like to enforce referential integrity with Cascade, Delete and Update related records because if some employee leaves the Company then his Dependent records should also be deleted.





Creating a Query

Here we will be creating a Query to display the details of an employee including his dependents when the employee code is given.



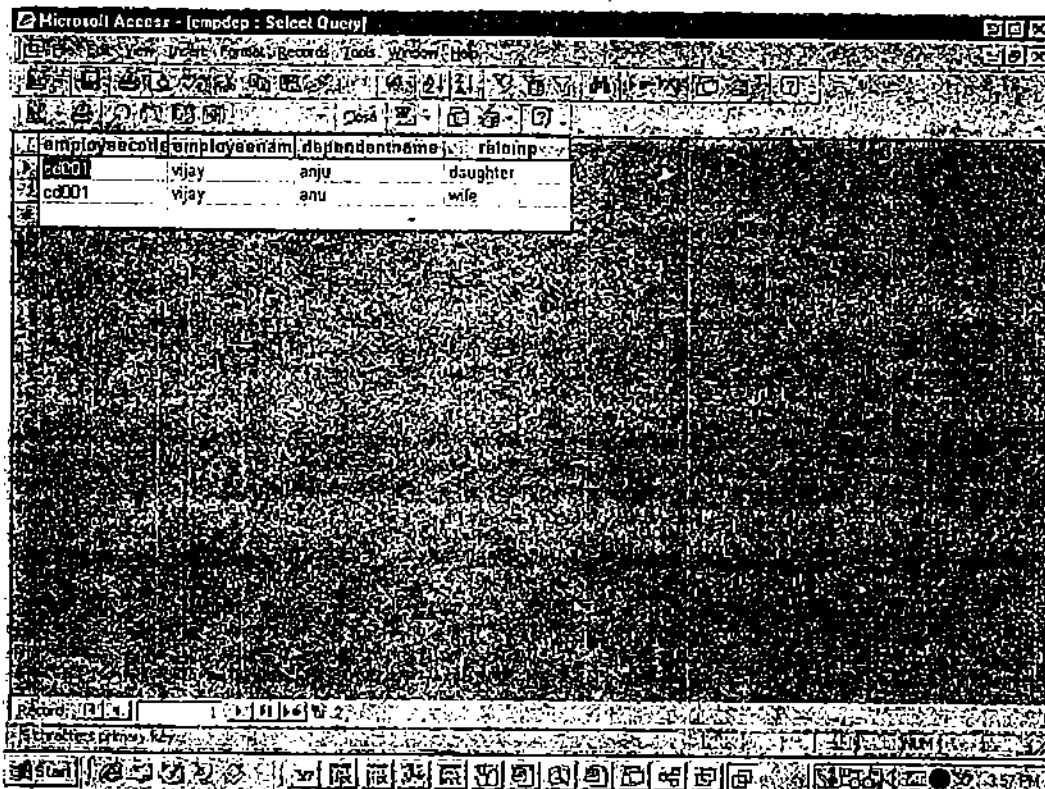
To Create a Query without a Query Wizard-

- 1) In the Database window, click the Query button.
- 2) Choose the New button
- 3) Microsoft Access displays the New Query dialog box.
- 4) Choose the New Query button.
- 5) Microsoft Access opens a Select Query window and displays the Add Table dialog box, which displays the tables and queries in your database.
- 6) Select the table that contains the data you want to add to your query:
- 7) In the Query window, Microsoft Access displays a field list for each table you select.
- 8) Choose the close button.

Saving a Query

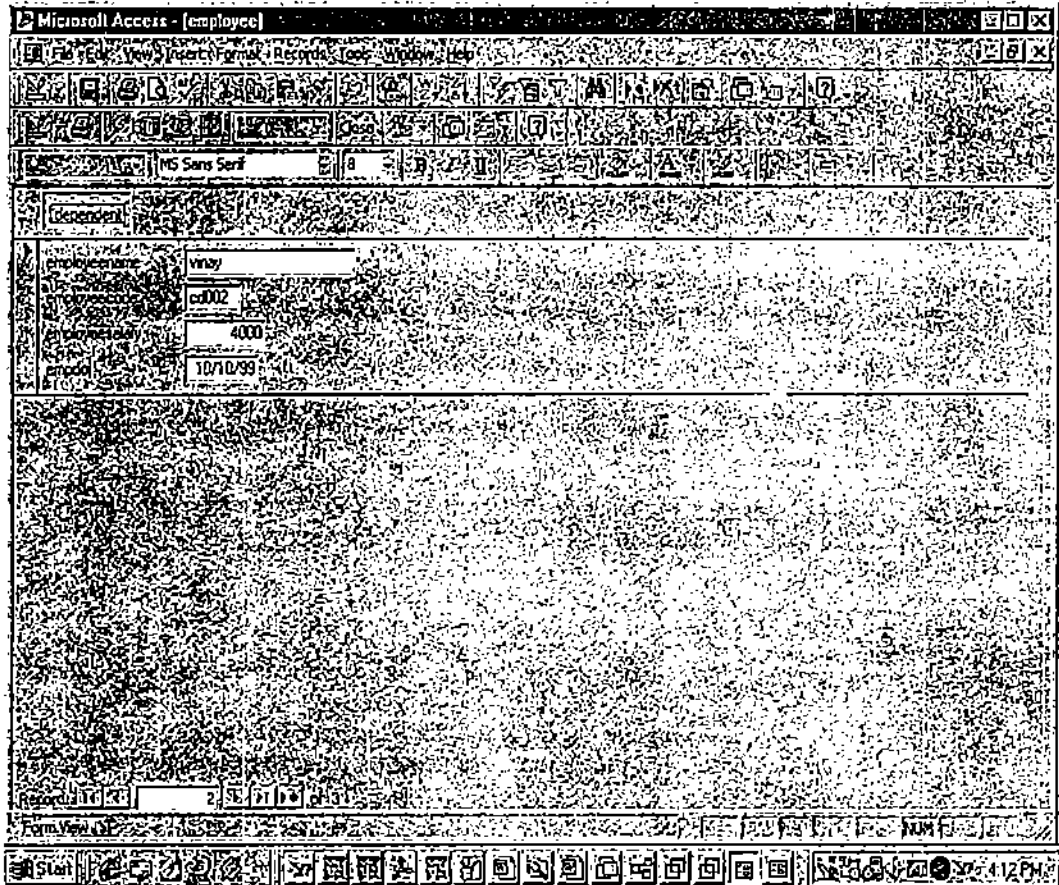
- 1) From the File Menu, choose Save (or click the Save button on the toolbar).
- 2) If you are saving the query for the first time, type the name for the query, and then choose OK. A query name can contain upto 64 characters and can include spaces.

The above Query should generate the following output:



employeecode	employaenam	dependantname	relationp
cd001	vijay	anju	daughter
cd001	vijay	anu	wife

Creating a Form



To create a Form without a Form Wizard-

- 1) In the Database window, click the Form button (or choose Forms from the View menu).
- 2) Choose the New Button (or choose New from the File menu, and then choose Form).
- 3) Microsoft Access displays the New Form dialog box.
- 4) From the Select A Table/Query list box, select the table or query that contains the data you want to display on the form. (If the form won't contain data, don't make a selection in this list.)
- 5) Choose the Blank Form button.
- 6) Microsoft Access displays the form in Design view.

Here we have created a form employee, please note the button dependent in that form, after entering data of employee, you can enter data of dependents by clicking that button.

employeeid	dependentname	relation
cd001	anu	daughter
cd001	anu	wife
cd002	ajay	son
cd002	raj	wife
cd003	akansha	sister
cd003	anu	brother

Creating a Report

Using a Report Wizard to help you build a report is a fast and foolproof way to create a basic report. You can use the basic report as is or refine it to get exactly the report you want.

To Create a form with the AutoReport Wizard-

- 1) In the Database window, click the Table or Query button.
- 2) Select the Table or Query on which you want to base the
- 3) Click the Auto Report button on the toolbar.
- 4) Microsoft Access displays the form.

The Reports generated for the Employee and Dependent tables are:

Microsoft Access Help

dependent

<i>employeecode</i>	<i>dependentname</i>	<i>relship</i>
cd001	arju	daughter
cd001	aru	wife
cd002	ejoy	son
cd002	rani	wife
cd003	aruj	brother
cd003	akansha	sister

Microsoft Access - {employee}

employee

<i>employeename</i>	<i>employeecode</i>	<i>employeesalary</i>	<i>empid</i>
vjey	cd001	2000	12/12/99
viney	cd002	4000	10/10/99
deeksha	cd003	5000	10/12/99

You can also create a report using more than one table.

The screenshot shows a Microsoft Access window titled 'employee & dependents'. The main area displays a table named 'employee dependents' with the following data:

<i>employeecode</i>	<i>employeename</i>	<i>dependantname</i>	<i>relship</i>
cd001	vjay	anu	wife
		anju	daughter
cd002	vinoj	rani	wife
		ojay	son
cd003	deepsha	akansha	daughter
		anil	brother

Exercise 4

Instructions: You have to perform following exercises in MS-Access.

Question 1. Invoicing is one of the most important activity in Sales management. The selling routine is not considered complete until money due is realised from the customer. To get payment from a customer for goods delivered, the customer will have to be sent an invoice. Once an order has been fulfilled and the goods have been dispatched to the customer, the next step is to prepare and send the invoice. Computerise the whole process.

- 1) Create the appropriate database and normalised tables. Tables could have following fields:
 - Order Number
 - Invoice Number
 - Invoice Date
 - Order Date
 - Product Code
 - Customer Name
 - Customer Address
 - Description
 - Quantity dispatched

- Price per Unit
- Invoice Amount
- Sales Tax Amount
- Total Amount
- Order Number
- Pending.....etc.

2) While creating the structure for the invoice table, the following validations should be taken care of:

- The invoice number must be unique
- An invoice can only be generated if the corresponding order number exists
- The date of the invoice should be current date.

3) Apart from validations, the following tasks should be performed during data entry.

Data entry from the tables has to be done using Forms, which can be created through the Form wizard.

4) In the form,

- The invoice amount has to be calculated and displayed.
- The sales tax at the rate of 8% of the invoice amount needs to be calculated.
- The total amount due from the customer has to be calculated.

5) Records must be added to the tables after creating a query, linking the various tables (decided by you) on the basis of the order number.

Question 2. 1) Create a new database for any banking Enterprise and name it suitably. The entities you have to consider are following (You can add more entities):

- Employee
- Customer
- Account-Number
- Account Type (Savings, Current etc.)
- Loan
- Payment
- Branch

- 2) Identify various attributes of the entities above. Identify all the relationships between these entities.
- 3) First create one single table for the complete banking Enterprise. Identify Key attributes.
- 4) Normalize the table to various normal forms. In each table identify primary key, foreign keys etc. if possible. Apply both the integrity constraints on the database.
- 5) Create a suitable form through which we can enter the data for various entities. Associate this form to various tables. More than one form can also be made.
- 6) At least fill 10 records for each table.
- 7) Give the following Queries and print the report in a suitable format with suitable Headings for each column:
 - Find the names and cities of residence of all employees who work for this Banking Enterprise.
 - Find the names, Street Address, and cities of residence of all employees who work for this Banking Enterprise and earn more than Rs. 10,000 per month.
 - Find the names of all customers of this banking Enterprise.
 - Find the names of the customers who have taken loan from the banking Enterprise and also print the loan amount in the report.
 - For the customers who have taken loan, how much amount of loan has been paid back and how much amount is pending.
- 8) On last day of the month add 2% interest on the minimum amount present in the account after the date 10th of that month. Generate this statistics by using in-built mathematical functions present in MS-Access, e.g.. Count, sum, min etc.... Record this information as one of the field in suitable table.
- 9) Generate a report in the format of passbook with following fields.

Date	Mode (Cheque, cash etc.)	Interest to date	Amount debited	Amount credited	Balance to date

11) Add various constraints like,

- A person cannot have more than two accounts in this banking enterprise.
- A person cannot apply for the second loan until he has already paid the first loan completely etc.

Question 3. There is a way in MS-Access through which we can run the SQL. Find it and give all the queries in SQL section in MS-Access and find the results of various queries.

Question 4. Design and implement the database for a University in MS-Access, keeping following in mind.

1) The entities you have to consider are following (You can add more entities):

- Student
- Course
- Registration
- Instructors
- Result
- Scholarship

2) Identify various attributes of the entities above. Identify all the relationships between these entities.

3) First create one single table for the University. Identify Key attributes.

4) Normalize the table to various normal forms. In each table identify primary key, foreign keys etc. if possible. Apply both the integrity constraints on the database.

5) Create a suitable form through which we can enter the data for various entities. Associate this form to various tables. More than one form can also be made.

6) At least fill 20 records for each table.

7) Give the following Queries and print the report in a suitable format with suitable Headings for each column.

- Find the various courses available in University.
- Find the minimum percentage required to enter in the courses like Computer Science, Mathematics, and Physics etc.

- Find the number of students registered in the course Computer course etc.
- Find the toppers of all courses in year 1998.
- Find various Instructors for various Courses.
- Find the Number of students in the third year of course Computer Science in year 1999.
- Find names of students getting Scholarship, Find their marks also, and year of registration, etc.

8) Generate a report in the following format. (You can add more fields)

Name of Student	Year of Registration	Courses taken	Instructors	Marks in various subject he has taken	Position

9) Taking this database as the backend, create a Graphical User Interface on some other language like Microsoft Visual Basic or PowerBuilder etc., hence exploit ODBC.

10) Add various constraints like,

- A student has to select three courses.
- If students fail in one or more course then he cannot be promoted to next year, thus it is mandatory to clear all the courses before entering next semester or year etc.

Note: For the following questions do proper problem analysis. Draw dataflow diagrams and/or E-R diagrams. Design suitable forms having proper validation checks, reports and queries. You may use menus (if needed). Implement the above in MS-Access. You may also use a front-end (may be Visual Basic) if you need.

Question 5. Design and implement database for a Library in MS-Access.

Question 6. Design and implement database for the hospital in MS-Access.

Question 7. Design and implement database for a Shop (say Electronics Shop) in MS-Access. You can select the any type of shop you want.

1.7 VIEWS AND SECURITY USING SQL

Question 1. Imagine you work for Sandwich shops, and each of the stores in Delhi report to you all their data ... sandwiches sold, of each type, money made that day, supplies used, which supplies are unusually low (each store gets a certain amount of each supply ... Bread, Butter etc. each day, but they may need more the next day if they used a lot of it that day), etc. Each store manager can insert and change data about their store, but shouldn't be able to read data about the other stores. The person in charge of deliveries should be able to look at all the stores to see which supplies need to be delivered. The accountant could look at anything, but is only interested in the money taken in and spent (to buy supplies, for salaries, etc).

- 1) The Shop just hired you as their data administrator. Do you want to put this in a database system or not? Give reasons. Feel free to make up additional details, which would be likely to occur in a retail store situation like this.

.....

.....

.....

.....

- 2) Let us suppose that you decide to use the database DBMS. How many users views would you create at the external level, and why? (The reason to create a user view is either security or ease of use.)

.....

.....

.....

- 3) The data dictionary contains at least all the relations and views that are in your database, and which users can access each view. State a few reasons why a dictionary is useful to the database system, and a few reasons why it is useful to you as the data administrator.

.....

.....

.....

.....

Question 2. Consider the following relations:

- 1) **Staff** (sno, fname, name, tel#, position, sex, salary, bno) where Staff numbers (sno) are unique and each staff member has one first name, last name, position, sex, salary, and works for one branch. A staff member may have several telephone numbers that you can reach them at.
- 2) **Branch** (bno, street, city, tel#, fax#) where Branch numbers (bno) are unique, and each branch is at one place with one main tel# and one fax#.

- 3) **Property_for_rent** (pno, street, city, type, rooms, rent, sno, bno) where Property numbers (pno) are unique, each property is in one place and is of a particular type (apt. house, duplex, etc), with a certain number of rooms, is listed by one branch and one staff member.
- 4) **Renter** (rno, fname, lname, street, city, tel#, pref_type, max_rent, bno). What semantics will you use if you say that the primary key of renter is rno, pref_type?
- 5) **Viewing** (mo, pno, date, comment) where a person looks at a particular property (say room or apartment etc.) on a particular date and might or might not make a comment.

- a) Give the semantics; write down the candidate keys for the above relations. If you think that the given information is not enough then write down any additional assumptions that you need to make. Although the candidate keys are given with the relations, but, write down the real world assumptions to make that a candidate key.

.....

.....

.....

.....

- b) If there were 1000 different renters, 5000 different properties, 365 dates (lets just consider 1 year), and 20 different possible comments (things like "too small", "ugly", "wrong part of town", etc.) what would be the maximum cardinality of viewing? What would be a more realistic maximum estimate, and how would you determine that?

.....

.....

.....

.....

- c) Write the following queries using SQL for the above relations (You need to attempt this question yourself based on section 6).

- (i) Find those property pno, which are 6 room houses and are in the same city as the branch that listed the property.

- (ii) Find the renters last names who live in Delhi and viewed a property located in Delhi and the staff member that showed the property had a last name of Sharma.

- (iii) Find the property pno that is in Delhi, but no one that viewed the property lives in Delhi.

- (iv) Find those property pno and streets of 6 room houses that are in Delhi and everyone that viewed the property lives in Delhi.

- (v) Find those property pno and streets of 6 room houses that are in Delhi and every renter living in Delhi has viewed the property. (What is the difference in the answers between (iv) and (v)? Figure that out first before trying to write the query.) Any logically correct answer for these is fine.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Question 3. Write SQL queries for the following English queries. Please change the field names yourself such that they do not conflict with key words if needed.

- a) Find those renters who looked at properties in Delhi, where their max_rent is larger than the average apt rent in Delhi. Return their last name, their mo, and the difference between their max_rent and the average apt rent in Delhi.

.....

.....

.....

- b) Find those renters last names that viewed all the 3 room properties for rent in Delhi. (Hint: This is division.)

.....

.....

- c) Find those properties pno and addresses that were viewed in January by all the renters living in Delhi whose max_rent was < 600. (Hint: This is division.)

.....

.....

- d) Create a view that produces the average apt rent per city. It should return the average rent and the city. Show that it works.

.....

.....

- e) Use the view created in d) that returns the pno of the apartments whose rent is exactly the average apartment rent for the city it is in.

.....
.....

- f) You could have directly substituted the code for the view into your query for e), and avoid creating the view. When do you want to actually make it a view?

.....
.....

- g) Give one of your friends, permission to insert into and read your "property_for_rent" table. Also see if you can give them permission to insert into and read your view you created in part c). Attempt to read, insert into, and delete from your friend's table, and read and insert into your friend's view. Explain what happens.

.....
.....

- h) Create a view with all the attributes of "property_for_rent" except the "rent" field. Try to insert into that view. Try to delete from that view. Explain what happens.

.....
.....

- i) Explain how, if you wanted to implement security issues, you could allow certain staff members access to only the properties in Delhi. (It is sufficient to write this in English, you don't have to code it up in SQL).

.....
.....

Model Answers

- Answer 1. 1) I would like to put the shop data in a DBMS. In order to arrive at this decision, here are the assumptions I made, the disadvantages of file-based approach I considered and the advantages of DBMS I appreciated.

Assumptions:

- (i) Delivery person takes money from the accountant everyday and spends to buy supplies.
- (ii) Salaries are managed by store manager.

Data:

- (i) Raw supplies received on a day (bread, cheese, sandwich bread etc.)

- (ii) Supplies used
- (iii) No of sandwiches of each type sold (vegetable sandwich, cheese sandwich etc.)
- (iv) Money made on a day
- (v) Which supplies are low and how much extra of each item is needed
- (vi) Salaries given out that day
- (vii) Money taken/spent for buying supply items
- (viii) Total money made on a day by selling sandwiches
- (ix) Total money given away as salaries.

Let's identify store manager as P1, delivery person as P2 and accountant as P3. Table 1 gives the data (listed above) that each one is concerned about. The numbers 1, 2, 3, 4, 5, 6 etc. denote the data by their respective numbers above e.g., Raw supplies are denoted by 1, Supplies used is denoted by 2 etc.

Table 1.

Person	Concerned Data
P1	1,2,3,4,5,6
P2	1,2,5,7
P3	1,2,3,4,5,6,7,8,9. Mostly concerned about 4, 6, 7, 8, 9.

So, if we take the file system based approach and have separate files (containing the Concerned Data) for each instance of P1 and each of P2 and P3, we have the following disadvantages:

- (i) **Data Duplication:** It's very evident from the above table that there would be a lot of data duplication across the files.
- (ii) **Separation of data:** Each store manager has a set of data. The delivery person needs an access to the data of all the stores in order to decide on how much of each item he needs to buy and supply to each store that day. For this he needs access to the individual files. Data is not isolated, since it is all reachable without extensive effort.
- (iii) **Data dependence:** Items 1, 3, and 5 are related, as are items 6 and 9. In order for the values to be correct, updates to one field need to check that the other values are still correct.
- (iv) **Incompatible file formats:** Depending on who creates the files, incompatible file formats can complicate how to maintain and update these files. If one person created them all, we don't have a problem.

Consider a DBMS based approach and store data in the form shown in Table 2 follows:

Table 2.

Person	Concerned Data
P1	store Id, 1, 2, 3, 4, 5, 6
P3	All store Ids, 7, 8, 9

The DBMS approach shall have the following advantages--

- (i) **Controlled redundancy:** Store ids are the only replicated information.
- (ii) Since store ids are unlikely to change, we don't have problems in updating. Items 6 and 9 are in separate relations, but the database can keep those correct automatically by the DBA setting it up properly-- another advantage of using the database system.
- (iii) Could give more information from same amount of data. item 5 might be omitted by setting up an automatic query using information from items 1 and 3.
- (iv) **Enables sharing of data:** Each manager is able to look at his individual store, but the accountant can have as much information as desired. Database systems have more flexible sharing abilities.
- (v) **Improved data integrity:** For example, a typical integrity constraint can be that the money given out on a day should be strictly less than the money made on that day. Another constraint can be that the money made on a day at a store is consistent with the number of sandwiches sold.
- (vi) **Improved security:** The manager of a store will not be allowed to view data about other stores. DBMS approach supports the concept of views to support such security.
- (vii) **Backup and Recovery:** Knowing how much of each supply was already sent out and knowing the salary history is something that we would like to keep even in case of machine failure.
- (viii) The other advantages are less obvious, if applicable at all; maybe improved accessibility if the accountant wants to do other kind of monitoring, it might be easier to write queries if the info were in a DBMS, but concurrency and economy of scale aren't really applicable, since there isn't a whole lot of traffic going on anyway, and we don't have huge volumes of data.

Hence a DBMS based approach is suitable for storing the subway data.

2) The situation requires several views:

- One view for store manager - This view is to ensure security. No store manager will have access to information about other stores.
- One view for delivery person - This view will have access to the data 1,2,5 of all stores based on store Id and 7. This view ensures security min that it doesn't allow the delivery person to view data 8,9. The view also ensures

case of use by providing only required data (centrally) to the delivery person.

- One view for accountant - This view gives access to the complete data. But, in order to support ease of use, it will also have an option of filtering out some information and show data 4,6,7,8,9.

3) The data dictionary (also called system catalog) for the SHOP database contains:

- names, types and sizes of data items (The data items include the data that has been listed in part (a))
- Integrity constraints (total money made, money spent etc.)
- Names of authorized users and their views and authentication information (store managers, accountant, delivery person)

The information stored in the system catalog helps the DBMS in the following ways:

- (i) Knowing the data types, names of data, and location
- (ii) Checking authorization when a user accesses data.
- (iii) Data integrity is taken care of internally (makes sure updates keep data consistent).

The information stored in the system catalog helps the Data Administrator in the following ways:

- (i) Knowing who has what authorization for which views
- (ii) Knowing the structure and organization of the DB when extending or reorganizing it.
- (iii) More easily checking errors by referring to this automatic documentation to see what could possibly be set up incorrectly, etc.

Answer 2. The given relations are:

Staff (sno, fname, lname, tel #, position, sex, salary, bno)

branch (bno, street, city, tel #, fax #)

property_for_rent (pno, street, city, type, rooms, rent, sno, bno)

renter (rno, fname, lname, street, city, tel #, pref_type, max_rent, bno)

viewing (rno, pno, date, comment)

Semantics for the renter:

Renter numbers are unique to a renter. Each renter may have any number of preferences. Each renter has only one first name, last name, max_rent, tel#, branch he approached, and address.

a) Candidate keys:

(i) staff

- sno, tel#. If you assumed a tel# goes to only one staff member, then the candidate key would be just tel#

(ii) branch

- bno. there's a unique branch number for each branch.
- tel #. Assuming a phone# goes to only one branch.
- fax #. Same assumptions as on tel#
- street, city. If there's only one branch per address.

(iii) property_for_rent

- pno. There's a unique number for each property.
- street, city. Only one property per address. This may not be true, depending on what is included in the address.

(iv) renter

- rno, pref_type

The assumptions made for making the above attributes a primary key are:

- More than one renter can have same names, addresses, branch numbers, max_rent specifications and/or a combination of these.
- Renter has many preferences on the type of property.

(v) viewing

- rno, pno

One renter can view many properties and one property can be viewed by many renters. Assuming that one renter views one property only once, a combination of rno and pno forms a primary key for viewing.

If a renter can make several different comments about the same property, add it in to the key, but then you have to not allow null in the comment field, so you would have to add a legal value like "none".

- b) • In the worst case, if we assume that each renter sees each property each day and makes all possible comments, then the cardinality will be—

$$\text{no. of renters} * \text{no. of properties} * \text{no. of dates} * \text{no. of comments} = 1000 * 5000 * 365 * 20 = 36,500,000,000.$$

- Using my candidate key above, we get—

$$\text{no. of renters} * \text{no. of properties} = 1000 * 5000 = 5,000,000$$

And even this is big. A person probably won't look at more than 100 properties maximum, so a more realistic estimate would be 100,000. (1000 renters time 100 properties each).

Answer 3.

```
(a) SELECT lname, rno, max_rent - a.avgrent
FROM (SELECT avg(rent) AS avgrent
FROM property_for_rent
WHERE city = 'Delhi' AND type = 'apt' ) a,
(SELECT lname, r.no, max_rent
FROM renter r, viewing v, property_for_rent p
WHERE r.rno=v.rno and v.pno=p.pno AND p.city='Delhi'
GROUP BY lname, r.rno, max_rent)
WHERE max_rent > a.avgrent;
```

Alternate to above solution is:

```
SELECT r.lname, r.rno, max_rent-a.avgrent
FROM (SELECT avg(rent) as avgrent
FROM property_for_rent
WHERE city='Delhi' AND type = 'apt' ) a, renter r, viewing v,
property_for_rent p
WHERE r.rno=v.rno AND v.pno=p.pno AND p.city='Delhi' AND r.max_rent
> a.avgrent;
```

```
(b) SELECT lname
FROM renter
WHERE rno IN (SELECT rno
FROM viewing v
WHERE pno IN (SELECT v.pno
FROM viewing v, property_for_rent p
WHERE p.pno = v.pno
AND rooms = 3 AND city = 'Delhi')
GROUP BY rno
HAVING count(distinct pno) = some ( SELECT
count(distinct v.pno)
FROM viewing v, property_for_rent p
WHERE p.pno = v.pno AND rooms= 3 AND city
= 'Delhi'));
```

```
(c) SELECT pno, street
FROM property_for_rent
WHERE pno in (SELECT pno
FROM viewing v
WHERE date = 'Jan' AND rno IN (SELECT r.rno
FROM renter r, viewing v
WHERE r.max_rent < 600 AND city =
'Delhi')
GROUP BY pno
HAVING count(distinct rno) = some (SELECT count(distinct
r.rno)
FROM renter r, viewing v
WHERE r.max_rent < 600 AND city =
'Delhi'));
```

```
(d) CREATE VIEW average_rent_city AS
(SELECT avg(rent) AS avg_rent, city
FROM property_for_rent
WHERE type = 'apt'
GROUP BY city);
```

(e) `SELECT pno
FROM average_rent_city a, property_for_rent p
WHERE p.rent = a.avg_rent AND p.city = a.city;`

(f) Views are created only when a certain set of data are to be used often, if we just wanted the relation for that query, we would have just defined it in the from statement itself (as we did in part a) of this question).

(g) `GRANT SELECT, INSERT ON property_for_rent TO Ravi (say);` This Grant will be successful.

`GRANT SELECT, INSERT ON average_rent_city TO Ravi;` In this case also Grant will be successful.

`SELECT * FROM Ravi.property_for_rent;` Here also respective rows will be selected.

`INSERT INTO Ravi.property_for_rent values ('pd2','Khel gaon marg','Delhi','apt', 4,400,'s21','b21');` 1 row will be created.

`INSERT INTO Ravi.average_rent_city VALUES (1200,'New York');` Here we will get error as data manipulation operation not legal on this view.

`DELETE FROM Ravi.property_for_rent WHERE pno='pd2';` ERROR table or view does not exist as Ravi can insert and read property_for_rent since the permissions were given to Ravi as you, but he cant delete from it because that permission wasn't given.

You can read the view, but you can't insert into it because it makes no sense to do so, given an average rent for Bombay, you cant insert any meaningful tuples in the relation the view is based on.

(h) `CREATE VIEW pfrrent AS (SELECT pno, street, city, type, rooms, sno, bno
FROM property_for_rent);` View gets created.

`INSERT INTO pfrrent VALUES ('pfi', NULL, NULL, NULL, 3, NULL, 2
NULL);` 1 row is created.

`DELETE FROM pfrrent WHERE pno = 'pfi';` 1 row deleted. You can do this since this view is based on only 1 relation, and the missing "rent" field is just filled with null (as are a number of other fields that we actually put NULL in the tuple we inserted).

(i) Please attempt this question by creating a view having a condition.

Exercise 5

Please design views for the problems given in the previous exercise keeping in mind the various users and their expected security levels. Write SQL commands for creation of views and granting and/or revoking of access rights.

1.8 SUMMARY

In this block, we have discussed about various types of problems relating to relational database design and implementation. We have also presented some exercises, which you must try to attempt during your lab hours. Please make sure that before you go to the lab, you are ready with the basic design of your database, queries, forms (having proper validation checks) and reports. The problems presented in the block are just the sample problems, however, you must try attempting additional problems, specially related to the concepts of transactions and web interfaces using a front-end.

The basic topics which are covered in the block include use of ER diagrams, functional dependency and its use for normalisation, a practical example of normalisation, use of SQL for query, you must do more exercises on the topic as SQL is one of the standard platform for all the Commercial Database management systems. One of the prime issues of the block is to implement the designed database into an RDBMS package. Presently you will be using MS-Access for such implementation.

One of the questions you must be asking is why we have selected MS-Access and its version? Well the answer is, it is a package, which helps you as a ladder for higher commercial DBMSs. Please note the issue is not the version or a commercial RDBMS in which you implement the design. The main issue is whether you are adaptable to do so. MS-Access version, which you may be using, may be different, but you must try to understand the design and try to implement using your version.

Finally in the block, we have presented views and security issues using SQL. You must try to find out details of security features of RDBMS package, which you are using.



UTTAR PRADESH
JAJASHI TANDON OPEN UNIVERSITY

BCA-12 RDBMS LAB

CC-08 adopted as CS-67/Block-3

Block

3

RDBMS LAB INTRODUCTION TO MS ACCESS

UNIT 1

INTRODUCTION MICROSOFT ACCESS 5

UNIT 2

MICROSOFT ACCESS BASICS 11

UNIT 3

WORKING WITH DATABASE 18

UNIT 4

CREATING A TABLE 25

UNIT 5

FINDING DATA 40

UNIT 6

CREATING A QUERY 48

UNIT 7

CREATING A FORM 59

UNIT 8

CUSTOMISING YOUR FORM 67

UNIT 9

SHOWING DATA FROM MORE THAN ONE TABLE ON A FORM 75

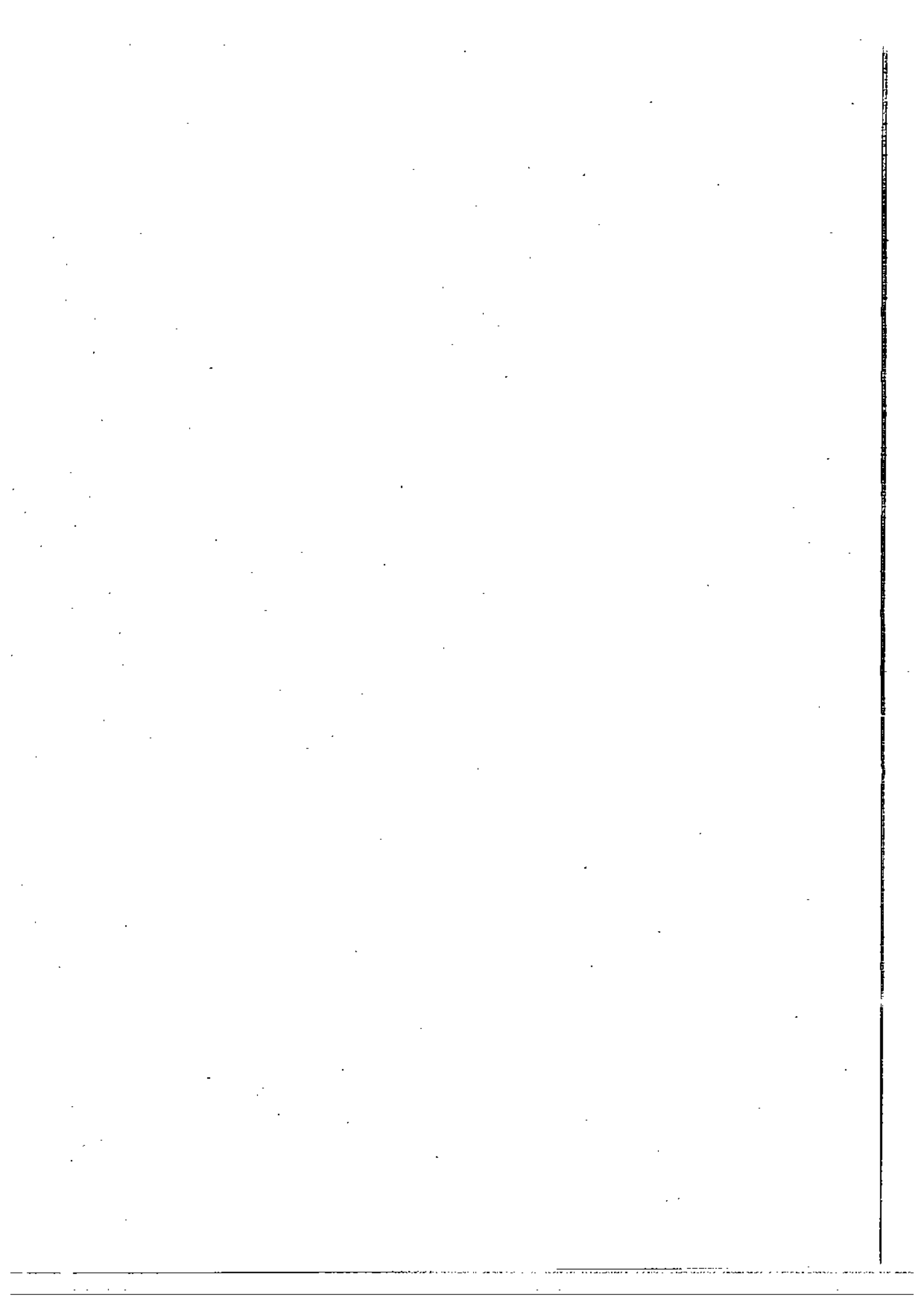
UNIT 10

CREATING REPOPTS AND MAILING LABELS 82

BLOCK INTRODUCTION

This block provides details of an RDBMS package. Please note whenever, you want to shift from one RDBMS package to other you just need to go through the changed reference manual. You must carry the concepts given in block 1 and block 2, as they will be required for any relational database management system application development.

The version of MS Access used in this block may be an old version to use at some places, but basic essence of commands and creations of RDBMS are not going to change. Microsoft Access is good launching platform for commercial RDBMS development in Oracle, Ingres, SQL Server, and Sybase etc.



UNIT 1 INTRODUCING MICROSOFT ACCESS

Structure

- 1.0 Introduction
- 1.1 Objectives
- 1.2 What is a Database Management System (DBMS)
- 1.3 What is a Microsoft Access Database?
- 1.4 Tables and Queries
 - 1.4.1 A Table – Data
 - 1.4.2 A Query – A Question and an Answer
- 1.5 Forms and Reports
 - 1.5.1 Forms and Reports
 - 1.5.2 A Report – Polished Results in Print
- 1.6 Summary

1.0 INTRODUCTION

This unit gives you an introduction, as to what an RDBMS is, and what is the difference between MS-Access, an RDBMS and other RDBMS packages. The basic components of MS-Access will also be discussed in this unit.

1.1 OBJECTIVES

After going through this unit you will be able to define a:

- Database Management System;
- Microsoft Access Database;
- Table;
- Query;
- Form;
- Report.

1.2 WHAT IS A DATABASE MANAGEMENT SYSTEM (DBMS)?

Before we look at the Access software and its capabilities, let us try understanding what databases are, and what is meant by Database Management System, or DBMS, as it is popularly called.

For a moment, just go back to your school days, when you used to maintain different *copies* of your 'Home Work Assignment' and 'Class Assignment'. In those copies on the first page you used to make the 'Index', which contained the headings as Serial No., Chapter, Date, and Remarks. And under those headings, the details of all the 'Assignments' we used to store. Why we stored these details? What was that? Was it a database?

Index. You mean to say that 'Index' was a database? YES. A database is a collection of data related to a particular topic. Employee records in a file cabinet, a stamp collection in an album - each of these collections is nothing but a database. Database, typically consists of a heading that describes the type of information it contains, and each row contains some information. In database terminology, the columns are called *fields* and the rows are called *records*. This kind of organisation in a database is called a *table*.

A database management system (DBMS) is a system that stores and retrieves information in a database. Data management involves creating, modifying, deleting and adding data in files and using this data to generate reports or answer adhoc queries. The software that allows us to perform these functions easily, is called a DBMS.

1.3 WHAT IS A MICROSOFT ACCESS DATABASE?

Since its introduction in the late eighties, the number of installations of windows has increased tremendously. With it has increased the demand for applications on windows. To meet this demand, Microsoft Corporation introduced a relational database management system for the windows platform in 1992 called *Access*.

Microsoft Access is a relational database management system (RDBMS). Microsoft Access, is also a database like any other database. The question arises, why one should go in for MS-Access, why not any other one; like Foxbase or Dbase? What is so different about MS-Access?. Lets find out the answer.

Let us introduce you to Soft Garments, wholesalers for shirts, trousers, and T-shirts. They purchase from various manufacturers and still wholesalers. The company has four departments - Sales, Accounts, Stores and Payroll. There are around 2000 employees working under the organisation. The company wants to maintain a database which will store the details and the entire information about all the employees. They want to store the Employee Code, Employee Name, Date of Birth, Date of Joining, Designation, Department and Photographs of the Employees.

Can we make such kind of a database in Foxbase, or in Dbase? May be not. We can store all the information about the employee except the photograph.

But in MS-Access its possible to have a photograph of the employee also on screen, apart from all the other details.

Introducing
Microsoft
Access

That's it !. It means that the only difference between any other database and MS-Access is that in MS-Access, you can store pictures in Access but not in other databases?

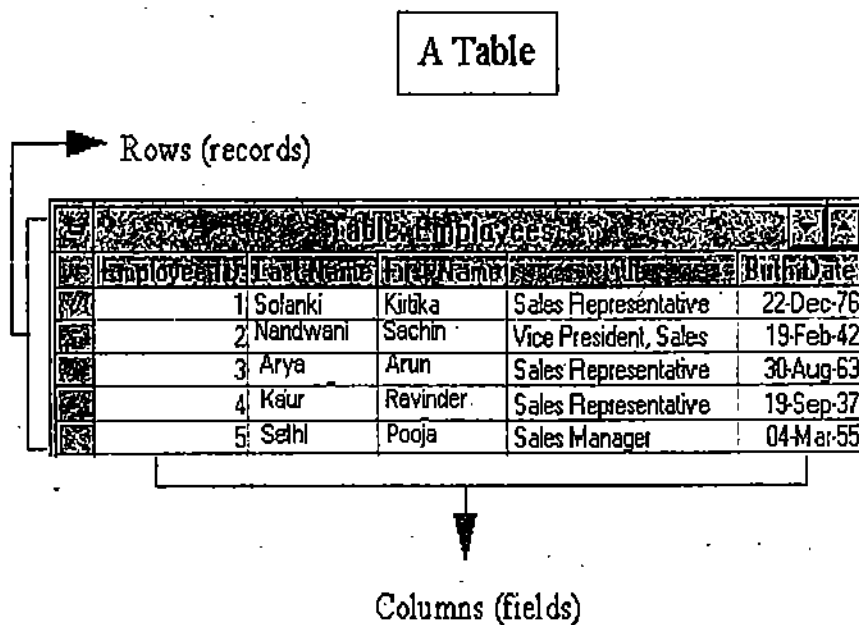
HOLD ON. Its just one of the advantages of using MS-Access, a lot more is still to come.

1.4 TABLES AND QUERIES

1.4.1 A Table - Data

A Microsoft Access database is a collection of database files, which are also known as Tables. And each database (a table) is a collection of records, and a record is a collection of fields.

Now, if the *Soft Garments* wants to store the employee details, they will have to make a table, which will be a part of some database. The information about one employee will make one record of that table, and the information will be stored under fields, fields as *employee code* and *employee name* and others.



Each record in a table contains the same set of fields and each field contains the same type of information for each record.

1.4.2 A Query - A Question and an Answer

Assume, that you are a senior executive in the *Soft Garments* and heading the payroll department. One day the manager of the company calls you, and he wants to know that how many employees are in 'A' grade; how many are in 'B' grade and how many are in 'C' grade. Will you be in a

position to answer that Query, right at that moment? May be Yes, May be No. Keeping a track of 2000 employees, is quite difficult.

Not to worry. Manager had a query, he asked you. If you don't know the answer, you can use the 'Query' of your database.

In MS-Access, you have something called Query. A Query is a question you ask about the data in your database. The data that answers the question can be from a single table or several - the query brings the information together.

A Query

Employee ID	Last Name	First Name	Title
1	Solanki	Kirtika	Sales Representative
2	Fuller	Andrew	Vice President, Sales
3	Leverling	Janet	Sales Representative

You create a query that describes the set of records you want. When you use the query to access the data, you automatically get current data from the tables.

1.5 FORMS AND REPORTS

1.5.1 A Form - Information on the Screen

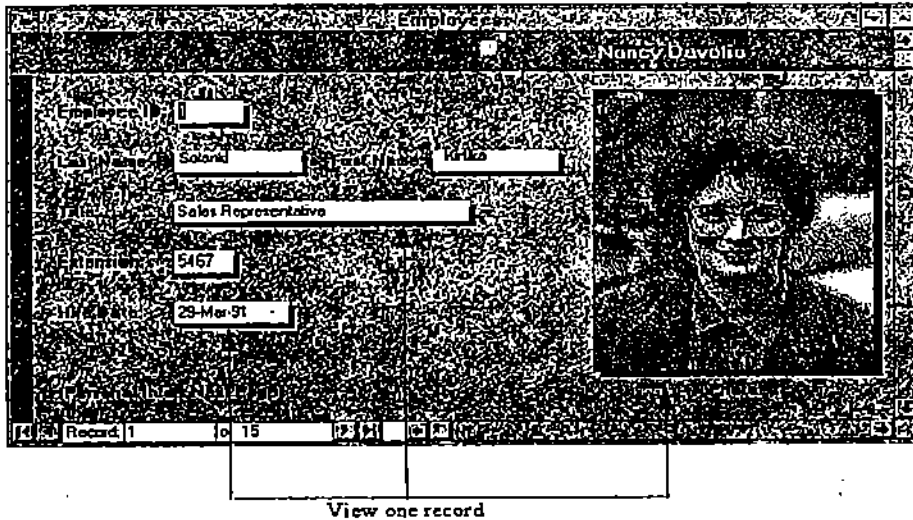
There are two ways in which you can view the data, which is stored in a table. Those ways are:

First Way :-



Employee ID	Last Name	First Name	Title	BirthDate
1	Solanki	Kirtika	Sales Representative	22-Dec-76

Second Way :-



View one record

Which way of viewing the data on screen you would prefer? No doubt, the second one. A query will give you an output, which will be displayed, the way it has been displayed in 1st figure in this subsection. If you want to get it displayed, in the second way, you have something called Forms.

A form is a customized way of viewing, entering and editing records in a database. You can specify how data is to be displayed when you design the form. Form can be created to resemble more closely the way data would be entered on paper form so that the user feels familiar with the operation.

1.5.2 A Report - Polished Results in Print

Forms and Queries present the data on screen. Reports are used to present data on printed-paper. It provides a way to retrieve and present data as meaningful information, which might include totals and grand totals, which have to be shown, across an entire set of records.

A Report

Sales Summaries		
20-Nov-96		
Employee Sales:	Employee Name:	Employee Sales:
	Pooja Sethi	\$47,674
	Sachin Nandwani	\$59,687
	Anu Solanki	\$93,932
Category Sales:	Category Name:	Category Sales:
	Beverages	\$127,189
	Condiments	\$56,462
	Confections	\$88,900

Check Your Progress

Question 1. Match the following:

- | | |
|-------------|--|
| a) Database | i) Data is stored in rows and columns. |
| b) Table | ii) Collection of tables. |
| c) Query | iii) Print records in custom layout. |
| d) Report | iv) Data based on specific criteria. |

Question 2. Define the following terms:

- Database
- DBMS
- RDBMS.

1.6 SUMMARY

DBMS	A collection of data that's related to a particular topic.
RDBMS	A relational database is a database, which consists of tables of related information that are linked together based on a key field.
Table	A object that stores data in rows (records) and columns (fields).
Query	Data based on a specific criterion.
Form	To enter, change, and view records of data in a custom layout.
Report	To print records in a custom layout.

UNIT 2 MICROSOFT ACCESS BASICS

Structure

- 2.0 Introduction
- 2.1 Objectives
- 2.2 Starting and Quitting Microsoft Access .
- 2.3 Opening a Database
- 2.4 The Database Window
- 2.5 Objects of the Access Database
- 2.6 Summary

2.0 INTRODUCTION

In the last unit we got introduced to a Database Window. In this unit, you will learn how to open an existing database, and what all objects are there in an Access Database.

2.1 OBJECTIVES

After going through this unit you will be able to:

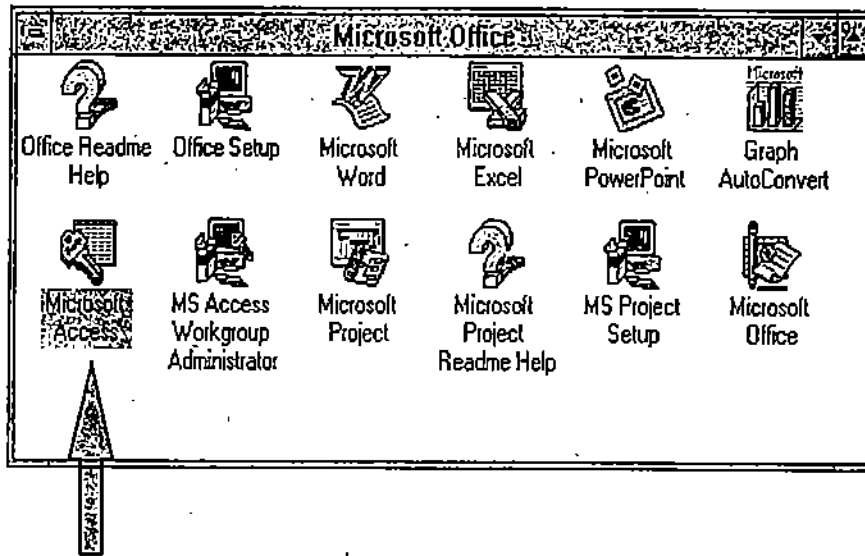
- Start and Quit Microsoft Access;
- Open a Database;
- Define The Database window;
- Define Objects of the Access Database.

2.2 STARTING AND QUITTING MICROSOFT ACCESS

You can start and quit Microsoft Access in the same way as you do other windows based applications.

To Start Microsoft Access

1. Open the program group that contains the Microsoft Access icon.

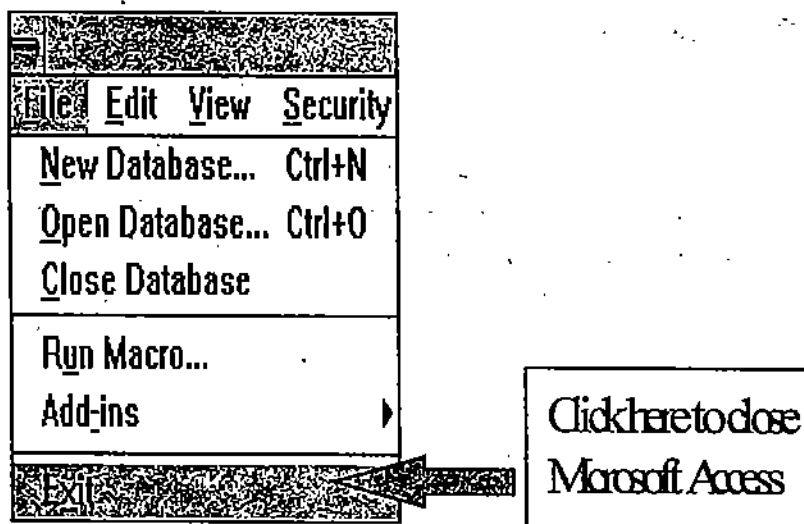


Click on this
Icon

2. Double click the Microsoft Access icon. Microsoft Access starts and displays Microsoft Access window, where you can create or open a database.

To Quit Microsoft Access

- Choose Exit from File menu.

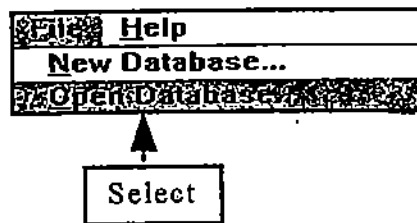


2.3 OPENING A DATABASE

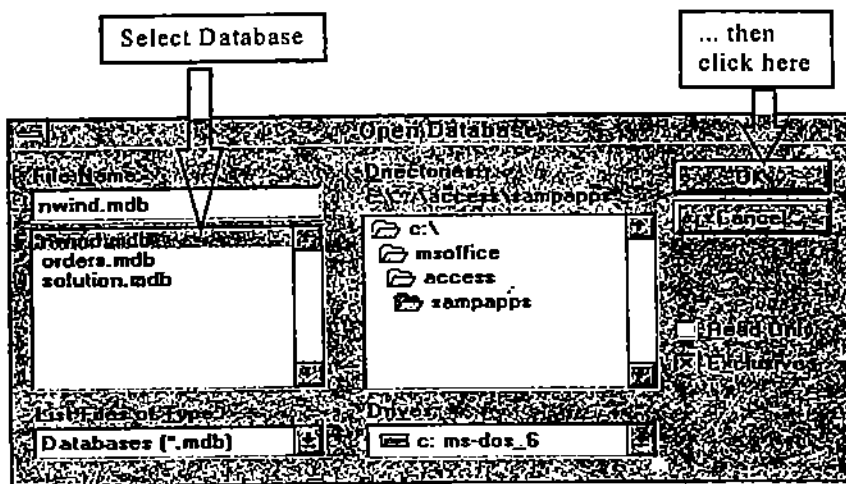
A Microsoft Access database is a collection of objects. A database file contains the tables, queries, and forms, reports that help you to use information in the database.

To Open a Database

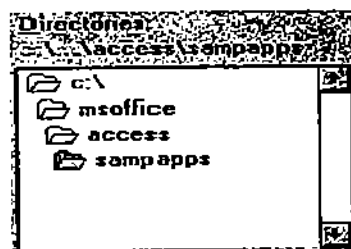
1. Choose *Open Database* from the *File* menu.



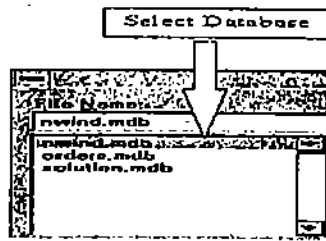
It will show you the following Open Database Window.



2. Select directory from directories list that contain database file.



3. Select Database from File Name list box.



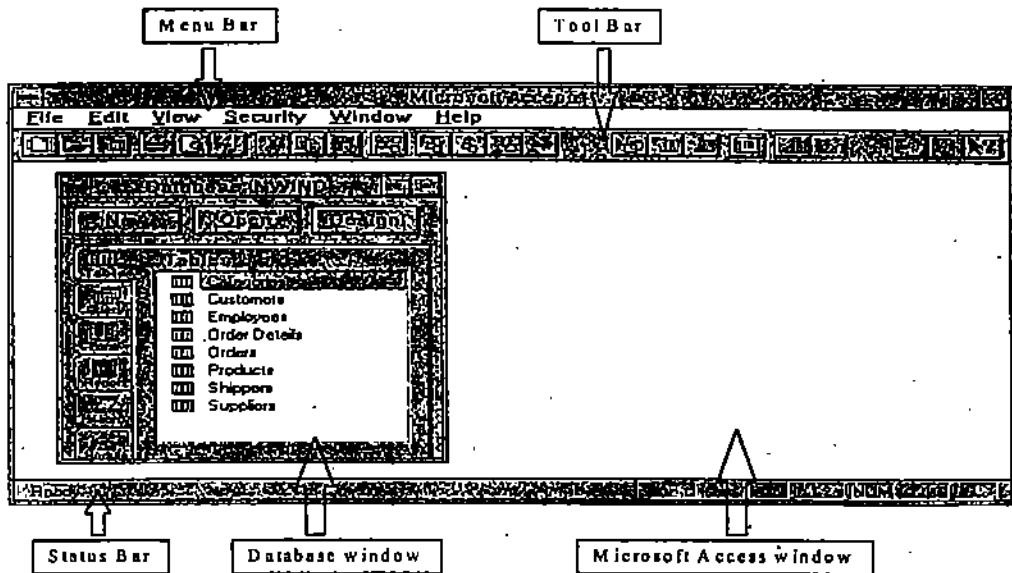
4. Click on OK to displays Microsoft Access Database window.



As soon as you click on OK, a *Database window* will be displayed to you (as shown in the picture given below). The Database Window displays a list of the tables in the database that you have made.

2.4 THE DATABASE WINDOW

When you open a database, Microsoft Access displays its Database Window in the Microsoft Access window. From Access window you can create and use any object in your database and others features of the Microsoft Access.



Title bar located at the top of the screen, displays the name of the program.

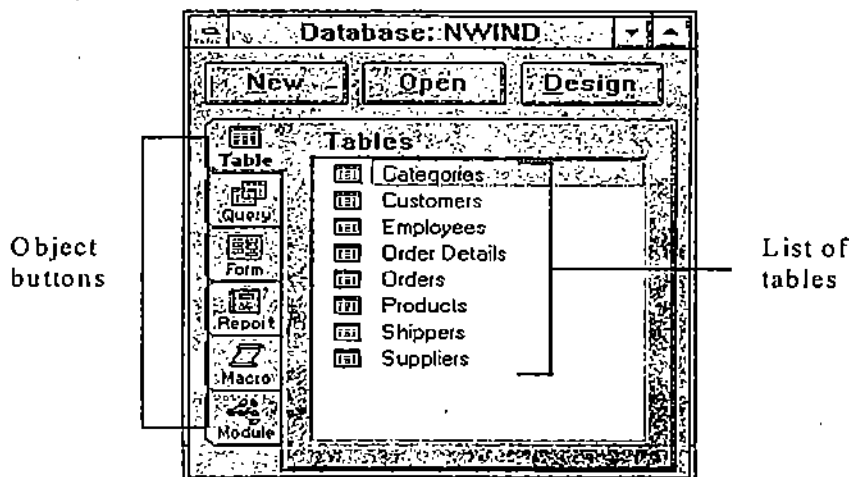
Menu bar located below the title bar, lists the options.

Tool Bars generally located below the menu bar, provides quick access to most frequently used commands and utilities. User can drag the toolbars and place to the position of his convenience.

Status Bar a horizontal bar at the bottom of the screen that displays information about commands, toolbar buttons and other options.

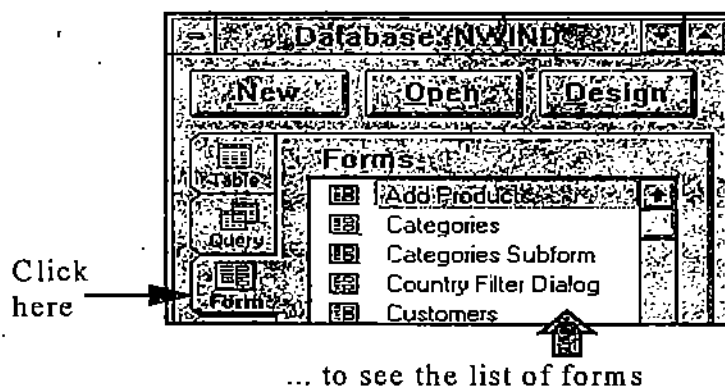
2.5 OBJECTS OF THE ACCESS DATABASE

Tables, forms, queries, reports, Macros and modules are objects of the Access Database. The object buttons in the Database window provide direct access to every object in your database. All these objects will be discussed, in detail, later.



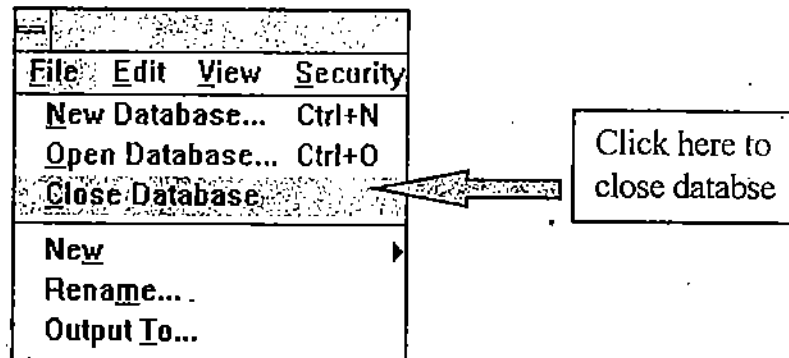
To view which all forms you have made you can:

Click the *Form* button. Microsoft Access displays the list of forms stored in the database. Click the table button to see again a list of tables, and similar for all the other objects also.



To Close a Database

- Select Close Database from the File menu.



Check Your Progress

- Question 1. Is the sequence of steps correct to open a database? If not, give the correct sequence.
- Select database from the file name box.
 - Click Ok to display Access Database Window.
 - Choose open from the file menu.
 - Select directory from directory list.
- Question 2. What is the extension of an Access database?

2.6 SUMMARY

Access is started by double clicking on the Access icon.

You quit from the Access by selecting Exit from the File menu.

You open a existing database by selecting Open from the File menu to display a dialog box, and select the database file name.

The file extension .mdb designates the file as an Access database.

The database window provides access to any tables, queries, forms or other objects that you create.

Everything in Access is an object. Access has the following objects

Microsoft
Access
Basics

- Tables
- Queries
- Forms
- Reports
- Macros
- Modules.

UNIT 3 WORKING WITH DATABASE

Structure

- 3.0 Introduction
- 3.1 Objectives
- 3.2 Creating a Microsoft Access Database
- 3.3 Creating Objects
 - 3.3.1 Create or Modify an Object
 - 3.3.2 Use Access Wizards to Create Objects
- 3.4 Set Toolbars to Your Working Style
- 3.5 Summary

3.0 INTRODUCTION

Now, you know how to open an existing database, and what all objects are there in a database. But, we still don't know how to create the objects (forms, tables, queries etc.) and a new database. In this unit, you will be covering all this.

3.1 OBJECTIVES

After going through this unit you shall be able to:

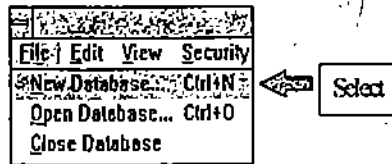
- Create a Microsoft Access Database;
- Create Objects;
- Set Toolbars to your Working Style.

3.2 CREATING A MICROSOFT ACCESS DATABASE

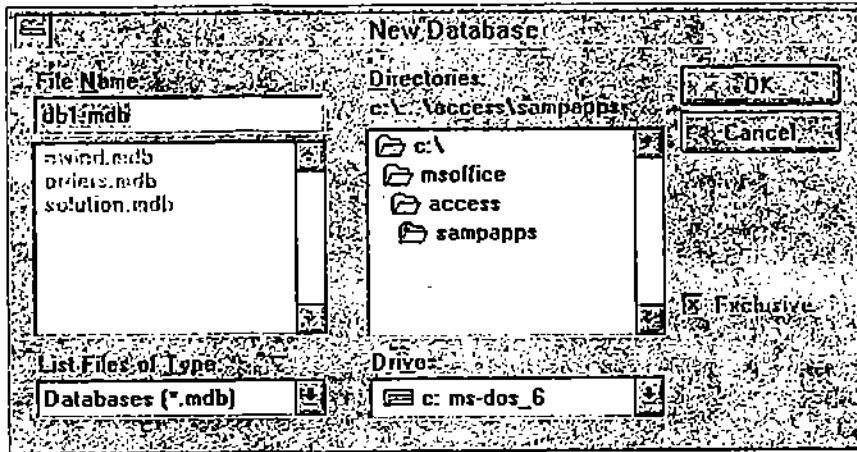
When you create a Microsoft Access database, you create one file that contains all the tables in your database as well as queries, forms, reports, and other objects that help you, use the information you had stored in the database.

To create a Microsoft Access Database

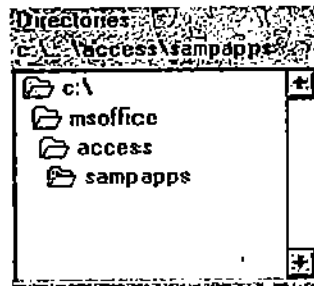
1. Select *New Database* from the *File* menu.



2. It will display you a Dialog Box as:



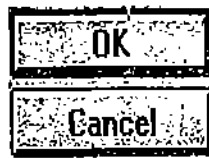
3. From this dialog box select the directory in which you want to create your database.



4. Enter the name of database in the *File name* box. A database name can contain upto eight characters but cannot contain spaces. No need to give extension because Microsoft Access automatically adds a .MDB extension to the database name.



5. Click on *OK* to creates an empty database file.



3.3 CREATING OBJECTS

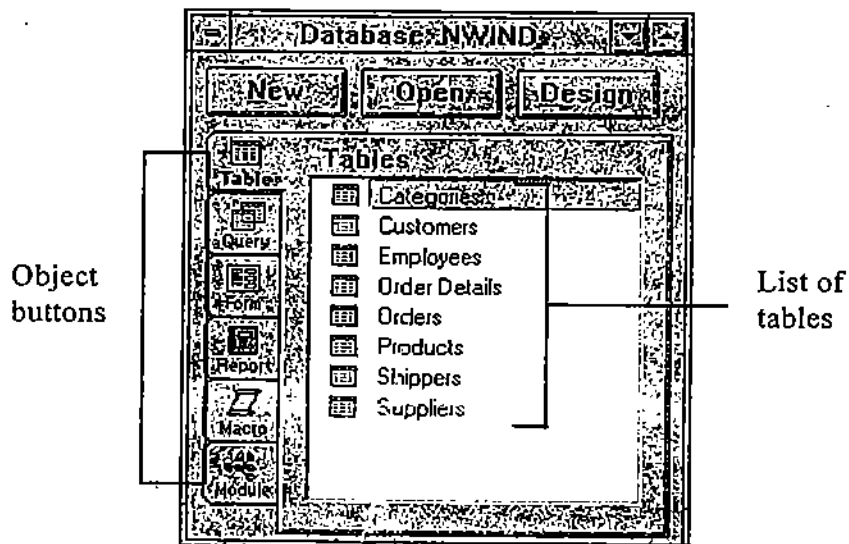
In the previous sessions we have studied, that a database contains different types of objects. Now we know how to create a database, the next step is to know, the creation of those objects. Creating tables is the first step to build a database. It depends on your data and your need as to how many tables you ought to create. After you have created tables, you can create queries, forms, reports and other objects to get desired information from the database.

3.3.1 Create or Modify an Object

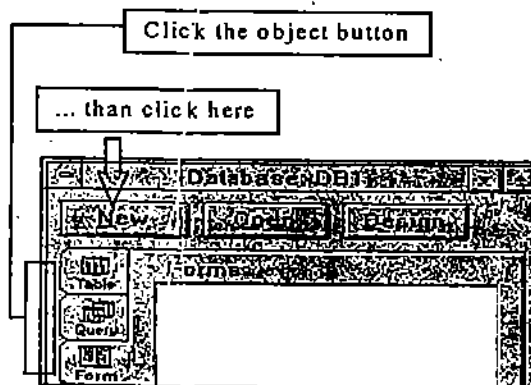
You can create or modify an object of any type from the Database window.

To Create a Microsoft Access Object

1. Select the object type to create, from the database window.

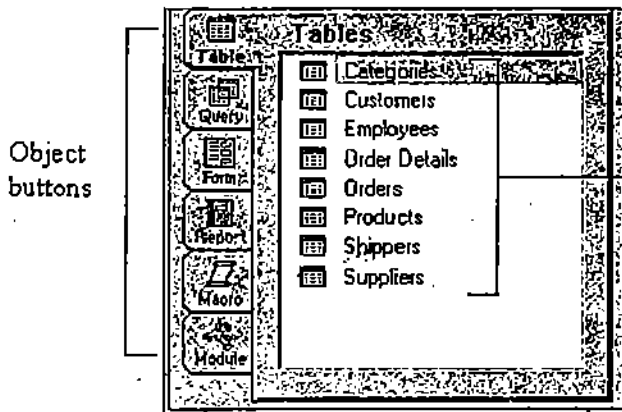


2. Click the *New* button.

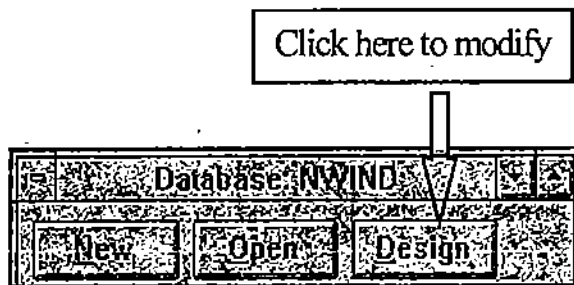


To Modify the Design of an Object

1. Select the object type to modify, from the Database window.

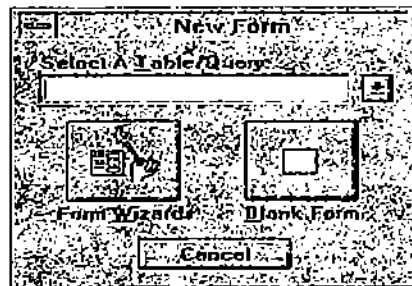


2. Select the object name from the list, to modify.
3. Click the Design button to display object window in design view.



3.3.2 Use Access Wizards to Create Objects

When you create a new table, query, form, or report, Microsoft Access displays a dialog box to ask, you want to build object yourself or use access wizard to build it.

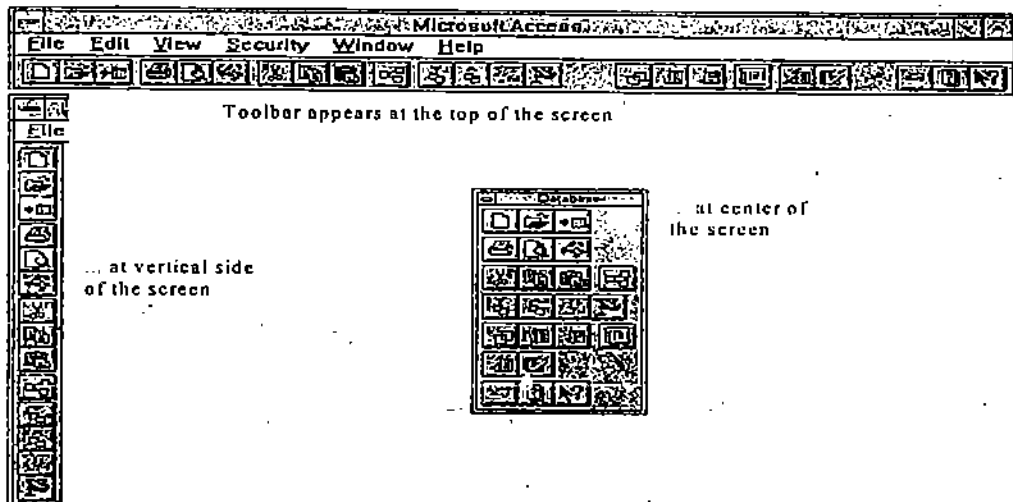


An access wizard is like a database expert who prompts you with questions about the object and then build the objects based on your answers. How to create Objects with the help of Wizards will be covered in the coming sessions.

3.4 SET TOOLBARS TO YOUR WORKING STYLE

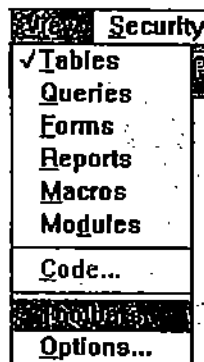
Microsoft Access provides a wide variety of graphical tools you can use to create and modify objects in your database. When you start Microsoft Access, it displays tools for opening a database and creating a database only. After you open a database, some new toolbars get added to the existing ones. The toolbars get or loose focus as and when you open any object (forms, tables, queries etc.) in *Design, Open, or New view*.

Initially, the toolbar appears at the top of the Microsoft Access window and tools are arranged in a single row. However you can pull the toolbar into vertical side of window, bottom of the window and middle of window and change its shape.

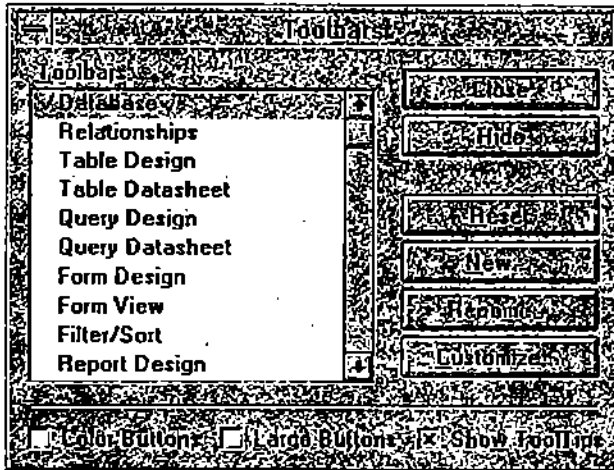


To Customize Toolbars

1. Select Toolbars from *View* menu to display toolbars dialog box.

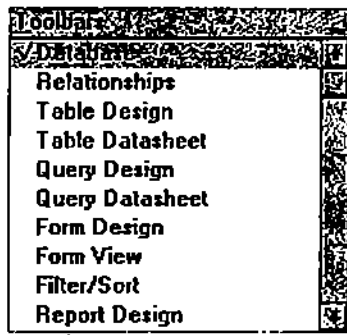


It displays built-in toolbars.



2. In Toolbars dialog box, you can:

- Hide or show the toolbars you have selected from the *Toolbars* list.



- Click on *Color buttons* check box to turns colours On or Off for all toolbar buttons.



- Click *Large buttons* check box to enlarge toolbar buttons, or return them to their original size.



- Click on *Close button* to close the dialog box.



Check Your Progress

- Question 1. What is the difference between a Datasheet view and Design view?
- Question 2. How to add new tools in your existing toolbars? Write the steps.

3.5 SUMMARY

To create a new database, select New from the File menu to display a dialog box, which asks you for the name of the new database and where you want the file to be stored.

Click on New button to create an object.

Click on Design button to modify an object.

An access wizard is like a database expert who prompts you with questions about the object and then builds the object based on your answers.

Toolbar provides a short-cut way to create and modify object in the database.

UNIT 4 CREATING A TABLE

Structure

- 4.0 Introduction
- 4.1 Objectives
- 4.2 Plan Fields and Data Types
- 4.3 Creating a Table
- 4.4 Set Field Properties
- 4.5 Save and Close a Table
- 4.6 Add and Save Records
- 4.7 Edit Records and Close a Table
- 4.8 Modify Fields in a Table
 - 4.8.1 Edit a Field
 - 4.8.2 Move a Field
 - 4.8.3 Delete a Field
 - 4.8.4 Insert a Field
- 4.9 Modify Columns and Rows in Datasheet
 - 4.9.1 Change the Width of a Column
 - 4.9.2 Change the Row Height
 - 4.9.3 Move a Column
 - 4.9.4 Save Datasheet Layout
- 4.10 Attach Validation Rule to a Field
- 4.11 Summary

4.0 INTRODUCTION

A table is a collection of data stored about a particular subject. The data in a table is presented in columns and rows. How to create the basic structure of a table, how to add rows(records) and how to edit them, you will learning in this session.

4.1 OBJECTIVES

After going through this unit you shall be able to:

- Create a table;
- Set field properties;
- Add and save records;
- Edit records and modify fields in a table;
- Modify columns and rows in Datasheet;
- Attach validation rule to a field.

4.2 PLAN FIELDS AND DATA TYPES

As we have already discussed in the previous sessions that a Microsoft Access database consists of *tables*. Your first step in designing a database is to make table structure. Each table in your database should fact on a single subject, for example customer or invoice or (*Employees of Soft Garments*).

Before starting with the designing of a table, one should be very clear about the data he is going to store in the table, based on which you will be making your table structure. For example, if in a table you are storing the details of an employee, you should know about, what details you will be storing about each employee (Employee Number, Name, Salary, Date of Joining etc.). In Database terminology we can say what all fields do you require for your table, and those fields will be of what type (Numeric, Character, or Date). Microsoft Access uses the data type to decide how much storage space to give to a field and to ensure that the right kind of data is entered in the field, for example you cannot enter text in number field.

It's important to choose the right data type for a field before you start entering data in the table. You can change a data type of a field that already contains data, but if the data types are not compatible, you might loose some data.

Design Fields a Table and there Data Types

Lets try making a table. Assume we want to store the details of an invoice in our table. The data, which we may require to store, can be:- Invoice Number, Invoice Amount, Invoice Date, and Customer name.

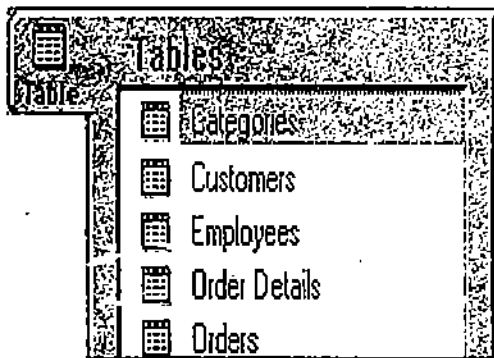
- The *Invoice no* and *Invoice amt.* fields will have numeric type data in them. So it can be given a data type as 'Numeric'.
- The *Invoice date* field will be storing the date of the invoice, and will have a data type as 'date/time'.
- The *Customer name* field will be storing the name of a customer; hence the data type can be of 'Text' type, which is used for storing alphanumeric data.

4.3 CREATE A TABLE

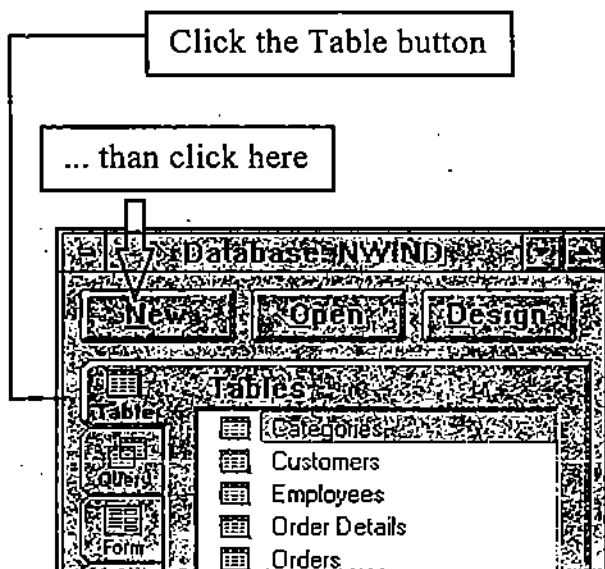
When you first create a table, it is an empty container for data. You design the table to contain the specific type of data you want to store, such as names and addresses.

To create a table

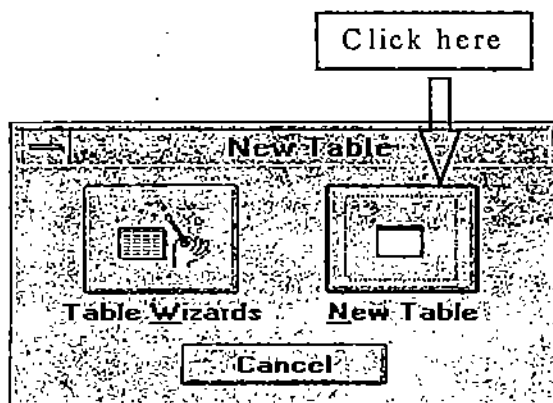
1. Click the *Table* button in Datasheet window.



2. Click the *New* button to display the New Table dialog box.



3. Click the *New Table* button to opens table window in design view.

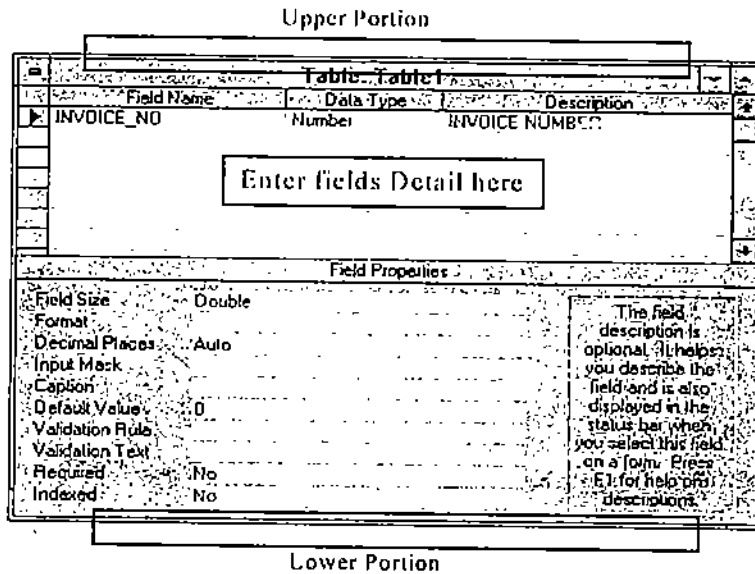


To Add Fields

In front of us, now we have a window where we can specify what kind of fields we require in our table, and what kind of data they will be storing. The creation of Table Structure begins from here. The following window depicts you the table in *Design View*. For creating the structure:

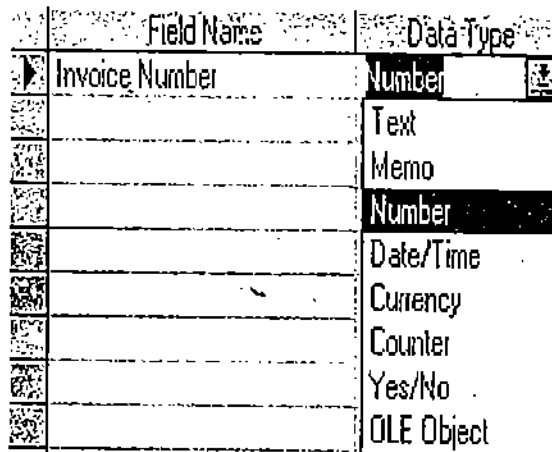
1. Enter the first field name i.e. 'Invoice Number' in *Field Name* box.

Field name can consist of up to 64 characters



This *Table* window has two portions, upper portion and lower portion. The upper portion contains field name, data type and description of field. The lower portion contains field properties like size, formatting, validation rule and indexing. (Covered a little later)

2. Press Tab key to go in *Data type* box and select Data type, for example Number.



3. Press Tab key to go in *Description* box and type it, this description appears in the status bar when you are entering data in the field.

Data Type	Description
Number	INVOICE NUMBER

Creating
a Table

Press Tab key to go in next field

- Repeat step 1 to step 3 to add other fields.

4.4 SET FIELD PROPERTIES

We have made the Invoice table for storing the details of an invoice. Somebody will enter the details. For checking whether that person enter the details correctly or not, you can either sit with him and crosscheck each and every entry he makes, or you can leave this job for your Microsoft Access. You can control the appearance of data, specify default values and speed up searching and sorting by setting field properties in table's design view.

Field property

Field size

Suppose you don't want that the Customer Name should not be more than 25 characters, you can adjust the size of a text field or limit the range of allowable values in Number field.

Data Type	Contains
Text	upto 255 characters
Number	

Field size property

Byte	0 to 255	
Integer	-32,768 to 32,768	
Long integer	-2,147,483,648 to 2,147,483,647	
Single	38	38
	$-3.4 * 10$ to $3.4 * 10$	
Double	308	308
	$-1.797 * 10$ to $1.797 * 10$	

Format

You can display the Invoice Date in any of the following formats. The Formats are:

Number Format

General Number	3456.789
Currency	£3,456.79
Fixed	3456.79
Standard	3,456.79
Percent	123.00%
Scientific	3.46E+03

Date Format

General Date	19/06/94 17:34:23
Long Date	19 June 1994
Medium Date	19 Jun-94
Short Date	19/06/94
Long Time	17:34:23
Medium Time	05:34 PM
Short Time	17:34

Decimal places

Display a certain number of places after the decimal point when using a format for a number or currency field.

Default value.

Suppose if you want that if the Invoice date is not entered by the user, the current date should be taken as the invoice date, you can use a Default Value. It automatically fills in a certain value for a field in new records.

Indexed

Indexing data on this field (by default is No)

To Set a Field Property

1. Select field in the upper portion of the table window in design view.

Field Name	Data Type	Field Properties
Invoice Number	Number	Invoice Number

2. Set field properties in the lower portion of the table window.

Field Properties	
Field Size	Double
Format	
Decimal Places	None
Input Mask	
Caption	
Default Value	0
Validation Rule	
Validation Text	
Required	No
Indexed	No

4.5 SAVE AND CLOSE A TABLE

After you add fields and set properties to a table, you must save the table design before you can add any records.

To Save and Name a table

1. Select *Save* from the *File* menu.

File	
New...	Ctrl+N
Open...	Ctrl+O
Close	
Save	Ctrl+S
Save As...	
Save All	

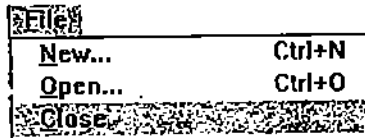
2. If you are saving the table for the first time, type a name (for example Invoice) for the table and then click OK.



Table name can upto 64 characters.

To Close a Table

1. Select *Close* from *File* menu.



4.6 ADD AND SAVE RECORDS

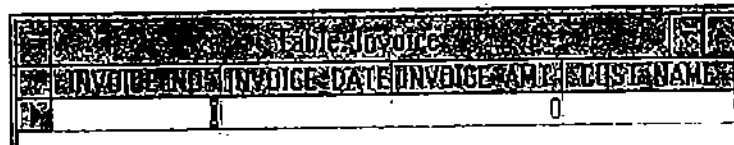
After finish designing, you can add records to a table.

To add records

1. Select Table to add records from the Database Window.
2. Click the *Open* button from Database window to open table in Datasheet view.



3. Enter a value in each field, pressing TAB key to move to the next field.



4. After you fill in all the fields, press TAB key to move to the new blank record.

To save records

When you move to the next record, Microsoft Access saves the record you added to the datasheet. When you finish adding records just close the datasheet, you don't have to save your work.

Creating
a Table

4.7 EDIT RECORDS AND CLOSE A TABLE

When you open a datasheet, the first field of first record is selected. If you begin typing while data is selected, the characters you type replace the selected data.

To Edit the value in a field

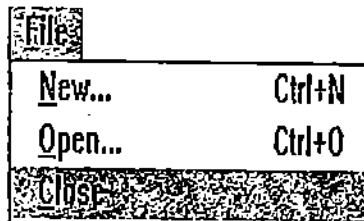
1. Use the mouse to select the contents of the field you want to modify.

Employee ID	Last Name	First Name	Title
1	Solanki	Kirtika	Sales Representative

2. Type the new value for the field. If you want to cancel all editing changes to a field, press Esc key.

To Close a table

1. Select *Close* from *File* menu.



4.8 MODIFY FIELDS IN A TABLE

If you change your mind about fields you have added, you can rearrange them, edit them, delete them or insert a new field.

4.8.1 Edit a Field

1. Select the field to edit.

Invoice Number	Number	Invoice Number

2. Edit name, data type or description of the field.
3. Modify the field properties in the lower portion of the table window.

Field Properties	
Field Size	Double
Format	
Decimal Places	Auto
Input Mask	
Caption	
Default Value	0
Validation Rule	
Validation Text	
Required	No
Indexed	No

4. Save it and close the table.

4.8.2 Move a Field

1. Select the field by clicking the field selector to the left of the field name.

Table: Invoice			
	Field Name	Data Type	Description
<input type="checkbox"/>	INVOICE_NO	Number	INVOICE NUMBER
<input type="checkbox"/>	INVOICE_DATE	Date/Time	INVOICE DATE
<input type="checkbox"/>	INVOICE_AMT	Number	INVOICE AMT
<input checked="" type="checkbox"/>	CUST_NAME	Text	CUSTOMER NAME

2. Click the field selector again and hold the mouse button and drag it to a new location.

Table: Invoice			
	Field Name	Data Type	Description
<input type="checkbox"/>	INVOICE_NO	Number	INVOICE NUMBER
<input checked="" type="checkbox"/>	CUST_NAME	Text	CUSTOMER NAME
<input type="checkbox"/>	INVOICE_DATE	Date/Time	INVOICE DATE
<input type="checkbox"/>	INVOICE_AMT	Number	INVOICE AMT

3. Save it and close the table.

4.8.3 Delete a Field

1. Select the field by clicking the field selector to left of the field name.

Table: Invoice			
	Field Name	Data Type	Description
<input type="checkbox"/>	INVOICE_NO	Number	INVOICE NUMBER
<input checked="" type="checkbox"/>	CUST_NAME	Text	CUSTOMER NAME
<input type="checkbox"/>	INVOICE_DATE	Date/Time	INVOICE DATE
<input type="checkbox"/>	INVOICE_AMT	Number	INVOICE AMT

2. Press DEL key or select delete row from the *Edit* menu.

Edit	
Cut	Ctrl+X
Copy	Ctrl+C
Delete	Del
Select All	Ctrl+A
Insert Row	
Delete Row	
Set Primary Key	

3. Save it and close the table.

4.8.4 Insert a Field

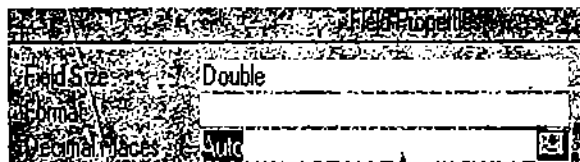
1. Select *Insert Row* from the *Edit* menu.

Edit	
Cut	Ctrl+X
Copy	Ctrl+C
Delete	Del
Select All	Ctrl+A
Insert Row	
Delete Row	
Set Primary Key	

2. It inserts an empty row before the current row.

INVOICE DATE	Date/Time	INVOICE DATE
INVOICE AMT	Number	INVOICE AMT

3. Enter field name and other information in empty row.
4. Set field properties.



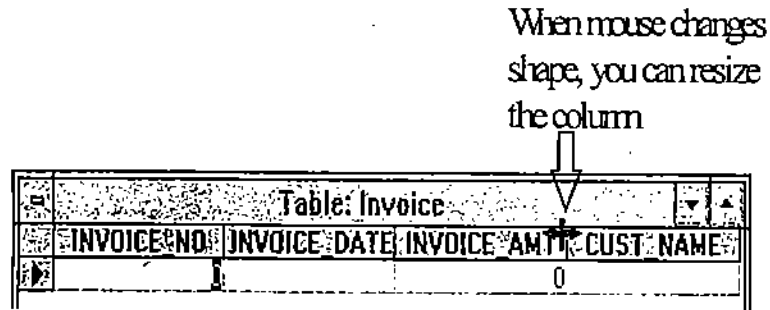
5. Save it and close the table.

4.9 MODIFY COLUMNS AND ROWS IN DATASHEET

If the columns in a datasheet don't fit the field values they display, you can change the width of each column and you can also rearrange the datasheet columns.

4.9.1 Change the Width of a Column

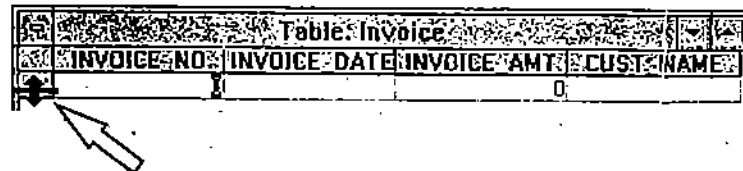
1. Position the mouse pointer at the right side of the field selector for the column you want to resize.



2. Drag the column border to the desired size or select column width from the Format menu and select Best Fit to fit the data it displays.

4.9.2 Change the Row Height

1. Position the pointer between two records selectors at the left side of the datasheet.



When mouse changes shape, you can change height of row

2. Drag the row to the desired size. All rows in the datasheet change to the new row height.

4.9.3 Move a Column

1. Select the column you want to move by clicking the field selector.

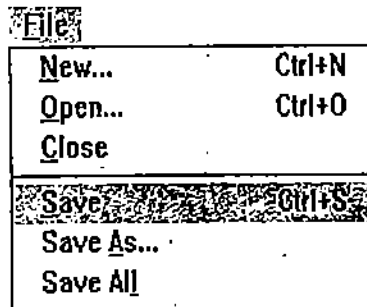
- Click the field selector again and drag the column to its new position. As you drag the column, its destination is indicated by a solid bar between columns.

Creating
a Table

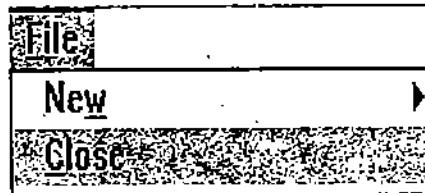
Table: Products				
Product ID	Supplier ID	Category ID	Product Name	English Name
1	1	1	Chai	Dharamsala Tea
2	1	1	Chang	Tibetan Barley Beer
3	2	2	Aniseed Syrup	Licorice Syrup
4	2	2	Chef Anton's Cajun Seasoning	Chef Anton's Cajun Seasoning
5	2	2	Chef Anton's Gumbo Mix	Chef Anton's Gumbo Mix
6	3	3	Grandma's Boysenberry Spread	Grandma's Boysenberry Spread
7	3	3	Uncle Bob's Organic Dried Pears	Uncle Bob's Organic Dried Pears

4.9.4 Save Datasheet Layout

- Select *Save* from the *File* menu.



- Select close from the File menu to close datasheet.



4.10 ATTACH VALIDATION RULE TO A FIELD

Microsoft Access automatically validates values based on a field's data type, for example, "You can't enter text in a numeric field. You can set more specific rules for data using validation rules. You set the validation rule, property for the field.

When you set the validation rule property, you specify requirements for data that's entered into a field. For example, if I want that invoice amount should not be less than zero, or customer name should not be left blank, I can specify Validation Rule for it.

If the Validation Rule is violated when you make an entry, and you want that some kind of message should be displayed to you, you can specify the text in Validation Text. Microsoft Access displays the text you specify when an entry in the field breaks the validation rule.

Some examples of validation rule:

Validation Rule	Validation Text
"Del" or "Cal" or "Bom"	You must enter "Del", "Cal" or "Bom"
<0	Please enter a non-zero value
>100	Value must be greater than 100
>=#01/01/96#	Date cannot be less than 01/01/96

To set validation rule

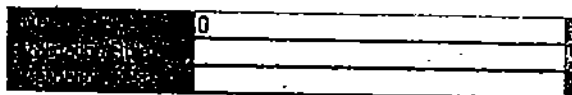
1. Open table in *Design View*.



2. Select field to attach validation rule.

Table: Invoice		
Field Name	Data Type	Description
INVOICE_NO	Number	INVOICE NUMBER
CUST_NAME	Text	CUSTOMER NAME
INVOICE_DATE	Date/Time	INVOICE DATE
INVOICE_AMT	Number	INVOICE AMT

3. Set the rule to the *Validation Rule* and *Validation Text* of the field properties.



4. Save it and close the table.

Check Your Progress

Question 1: Plan the fields and the data types for creating a table, which will store the details of a customer.

Question 2. Write down the steps for creating a table named 'TRANS', having the following fields.

- Trans_No, which is of Numeric type and is unique.
- Item_No, which will store AlphaNumeric data. (Field size should be 5)
- Trans_Date, which will store the transaction date. (Format:- 12'Jan 96)
- Qty to be issued

4.11 SUMMARY

Table provides a way to stored data.

To create a table, click on New button and than click on New table to display table in Design view, where you enter field name, field data types and set field properties.

You validate and format the data by set the field properties.

To add records, open table in Datasheet view.

To save a record, press TAB key to move to first field of the next record.

A validation rule specifies criteria to ensure that data entered into a cell valid.

UNIT 5 FINDING DATA

Structure

- 5.0 Introduction
- 5.1 Objectives
- 5.2 Find a Value
- 5.3 Find and Replace
- 5.4 Create and Apply a Filter
- 5.5 Specify Criteria
- 5.6 Sort Records
- 5.7 Summary

5.0 INTRODUCTION

We know, that table is used for storing whatever kind of data we want to store, and we store data, so that we can retrieve it whenever we require. In this unit we will come across one of the ways by which you can view data stored in a table based on some criteria.

5.1 OBJECTIVES

After going through this unit you shall be able to:

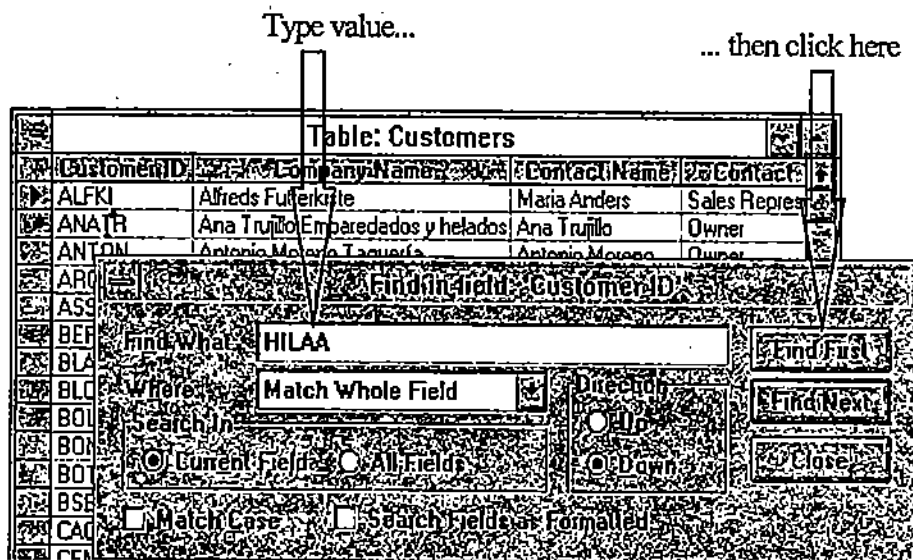
- Find a value;
- Find and replace;
- Create and apply a filter;
- Specify criteria;
- Sort records.

5.2 FIND A VALUE

Suppose you require the details of an invoice where Invoice Number is '4815'. One way of getting the details of this invoice is that you open the table in 'Open' mode, and browse through all the records one by one, and the other way is that you use the *Find* option. When you want to find a specific record or find certain values within the fields, you can use the *Find* option to go directly to a record. You can also use the *Find* option to navigate through records and find one record after another.

To find a specific value in a field

1. Select the field where you want to search.
2. Select *Find* from the *Edit* Menu.



3. In the *Find What* Box, type the value you want to find.



4. Click the *Find first* button to move to the record (if it exists).



5. Click the *Find Next* Button to find the next occurrence of the specified value. At the end, click the close button to close the dialog box.

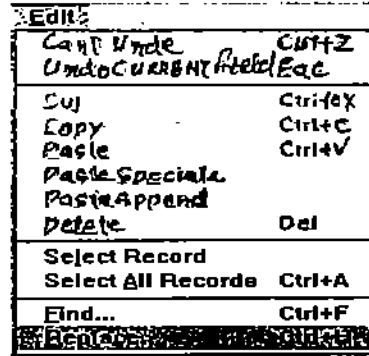


5.3 FIND AND REPLACE

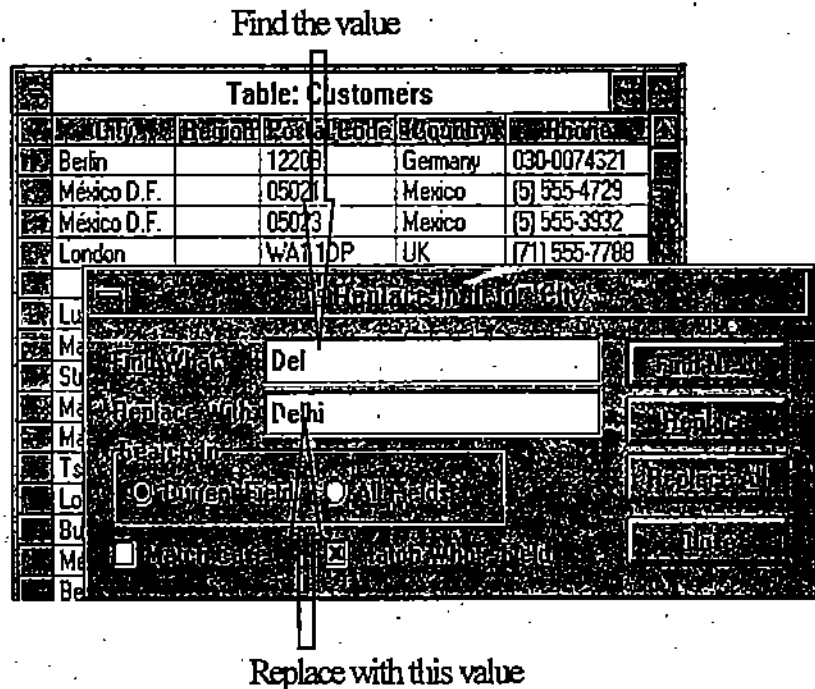
Sometimes it happens that you require making the same change to your data in several places. If you want Microsoft Access to find occurrences of specific text and replace them with different text, you can use the *Replace* command. With the *Replace* command, you can look at each occurrence of the specified text before you make the change or you can make global changes.

To Find and Replace all occurrences of specified text

1. Select the Field where you want to search and replace (in the open view).
2. Select *Replace* from the *Edit* Menu.



It will show you the Replace dialog box: -



3. Type the text in the find what box.



4. Type *Replace Text* in the *Replace With* Box.



5. Click the *Replace All* button.

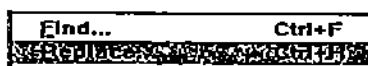


6. Click the *Close* button to close the dialog box.



To find and replace occurrences of specified text one at a time

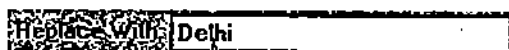
1. Select the field where you want to Search and Replace (in the open view mode).
2. Select *Replace* from the *Edit* Menu.



3. Type the text in the *Find What* box to find it.



4. Type *Replace Text* in the *Replace With* Box.



5. Click the *Find Next* Button to move to the next occurrence of the text you specify and select it.



6. To replace the specified text and move to the next occurrence, click the replace button or if you want to leave the selected text click the find next button.
7. When you finish replacing, click the close button to close the dialog box.

5.4 CREATE AND APPLY A FILTER

Microsoft Access provides two ways to create a custom view of the data in tables; you can create a query or you can create a filter for a table. A filter is like a simple query, except that it applies only to an open table.

A filter is best for temporarily changing the set of records you are viewing.

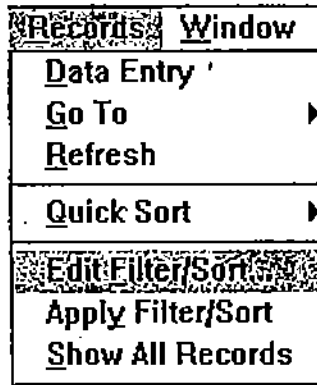
To create a filter, you tell Microsoft Access, which records you want to see, and the order in which you want to see them. You do this by specifying the criteria and the sort order in the filter window. Then, you apply the filter to view the subset of records in the table.

To Filter

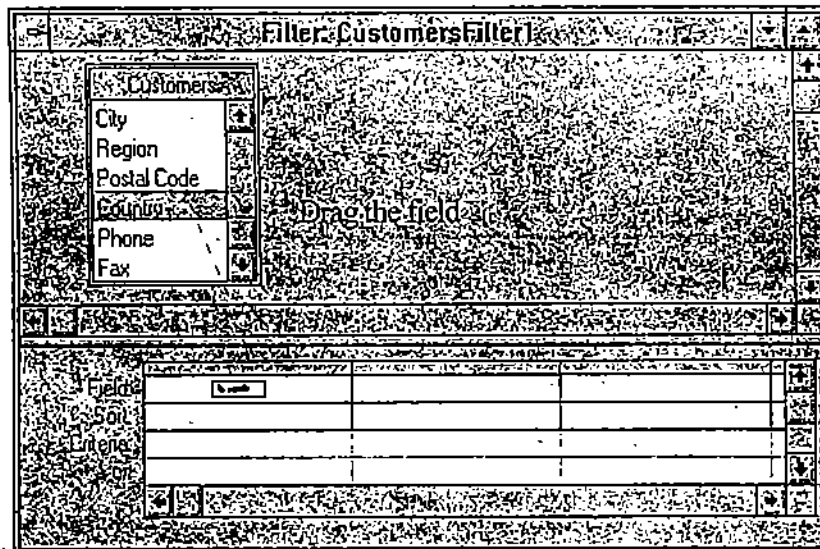
1. Open table in data sheet view.

Table: Employees					
Employee ID	Last Name	First Name	Title	Birth Date	
1	Solanki	Kirika	Sales Representative	22-Dec-76	
2	Fuller	Andrew	Vice President, Sales	19-Feb-42	
3	Leverling	Janet	Sales Representative	30-Aug-63	
4	Peacock	Margaret	Sales Representative	19-Sep-37	
5	Buchanan	Steven	Sales Manager	04-Mar-55	

2. Select *Edit Filter/Sort* from the *Records* Menu to open Filter window.



3. Drag one or more fields to the field row of the grid in the lower portion of the filter window

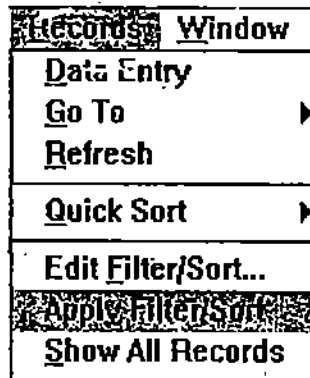


4. Specify criteria, sort order or both under the field in the filter window.

Finding
Data

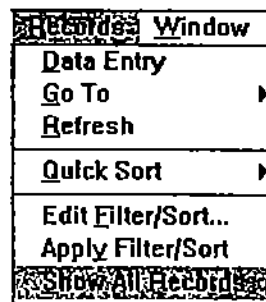
Field	Country	City
Sort		Ascending
Criteria	"UK"	

5. Select *Apply Filter/Sort* from the *Records* menu to display some filtered records in the table.



To remove a filter

- Select *Show All Records* from the *Records* Menu.



5.5 SPECIFY CRITERIA

You specify criteria if you want to see some specific records, not all. For example, you don't know the invoice number of an invoice, but you know that that invoice was raised for a customer named 'Deepak'. You can view all those records where the customer name is 'Deepak' by specifying criteria. When you Specify criteria for a field in a filter, you can use an expression. An expression is a combination of operators (such as < or >) and values (text or numbers) that Microsoft Access evaluates. When used for specifying criteria in a filter, an expression defines the set of records you want to view.

Some examples of criteria:

"Del"

Value is "Del"

100

Numeric value is 100

<50

Numeric value is < 50

>=#01/01/96#

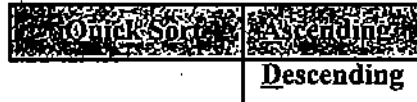
Date is on or after 01/01/96

5.6 SORT RECORDS

You can sort records in a table in a different order than they are usually displayed by using the *Quick Sort* command. You can sort records for display in either ascending or descending order.

To Sort Records in a table

1. Select the column in a data sheet to sort.
2. Select *QuickSort* from the Records Menu and then select *Ascending* or *Descending*.



Check Your Progress

FLIGHT

Flight_No	Text
Destination	Text
Fare	Numeric
Departure_Time	Date/Time

With reference to the FLIGHT table:

- Question 1. Write down the steps for changing the field contents for the field 'Destination' from "DEL" to "DELHI" and from "CAL" to "CALCUTTA" for all the records.
- Question 2. Create a Filter which will display only those records where Destination is equal to "DELHI" and Fare is greater than 10,000. (Just the steps).

5.7 SUMMARY

Page 6
15.1

Find command in the Edit menu provides a way to search a specific value.

Replace command in Edit menu provides a way to search and replace with a value.

To display specific records by create and apply the filter.

You can see the records in ascending or descending order by Quick sort command from the Records menu.

UNIT 6 CREATING A QUERY

Structure

- 6.0 Introduction
- 6.1 Objectives
- 6.2 Create a Query
- 6.3 The Query Window
- 6.4 Join Tables
- 6.5 Select Fields
- 6.6 Specify Criteria
- 6.7 Sort Records
- 6.8 Calculate Totals
- 6.9 Modify a Query
- 6.10 Save a Query
- 6.11 Summary

6.0 INTRODUCTION

Applying filter was one of the ways of viewing the data based on a criteria. Another way by which you can view data is Query. In this session, you will be creating Queries, and you will learn how to sort records.

6.1 OBJECTIVES

After going through this unit you shall be able to:

- Create a query;
- Join tables;
- Select fields;
- Specify criteria;
- Sort records;
- Calculate totals;
- Modify a query;
- Save a query.

6.2 CREATE A QUERY

With a query, you ask a question about the data stored in your tables. The way you design your query tells Microsoft Access exactly what data to retrieve. Microsoft Access gives you a great deal of flexibility in designing queries. With queries you can:

ProductID	Product Name	SupplierID
17	Alice Mutton	7
3	Aniseed Syrup	1
40	Boston Crab Meat	19
60	Camembert Pierrot	28
18	Carnarvon Tigers	7
1	Chai	1
2	Chang	1

↓
Data from more
than one table

Choose Fields You don't have to include all of tables fields in your query.

Choose Records You can specify criteria for displaying specific records.

Sort Records You can view records in a specific order.

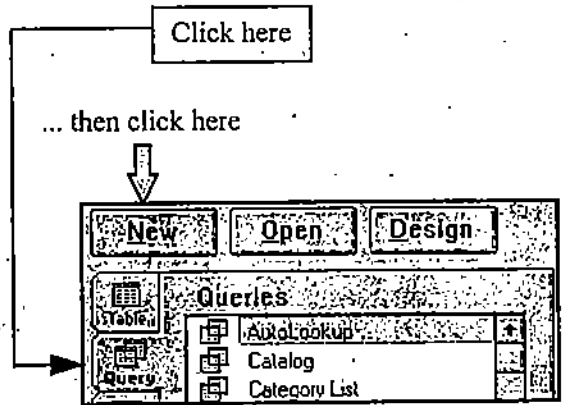
Ask Questions about data in several tables You can use a query to answer a question about data from more than one table and see the results in a single data sheet.

Perform Calculations You can create new fields that contain the results of a calculation, called calculated fields.

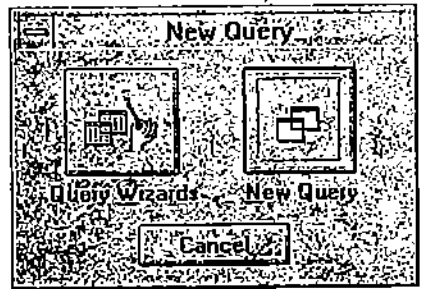
Make changes to Data in Tables You can update, delete, or append a group of records all at once using action queries.

To Create a Query

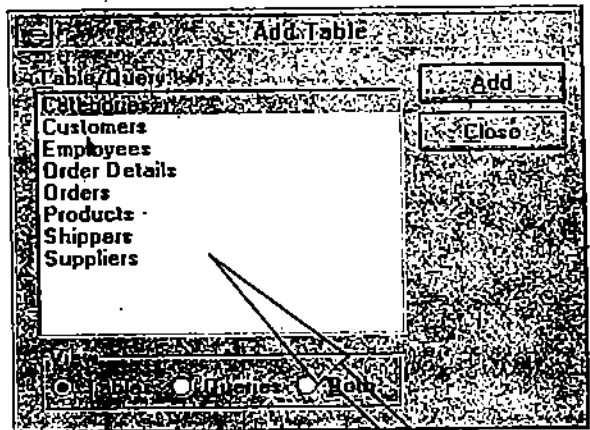
1. Click the *Query* button in the database window.
2. Click the *New* Button to display the new query dialog box.



3. Click the *New Query* Button to open a select query window and displays the add table dialog box, which displays the tables and queries in your database.



4. Select the table and click on *Add*, to display a field list for each table.

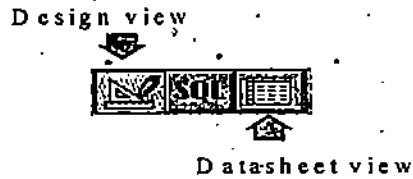


Tables in the Database

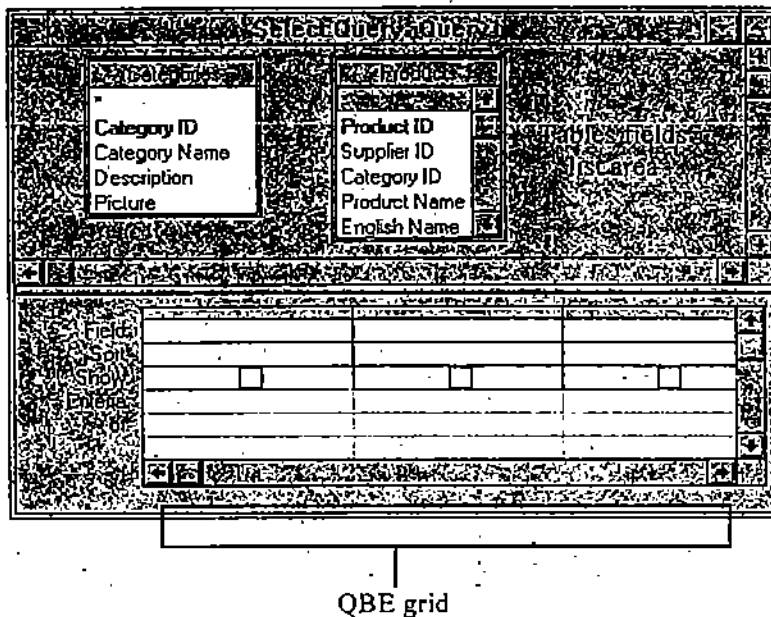
5. Click the close button.

6.3 THE QUERY WINDOW

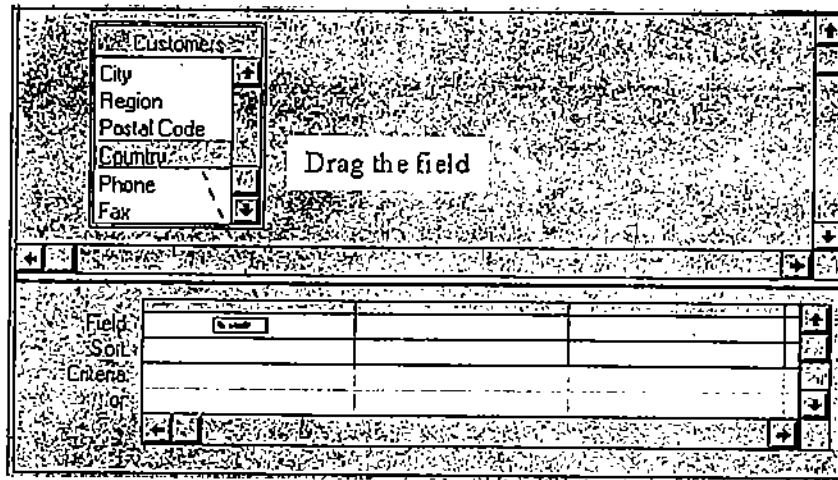
As soon as close the 'Add Table' Window, you will see a 'Query Window'. The query window has three views - Design view, DataSheet view and SQL view.



- **Design View:** To create a query or change the design of an existing one. In design view, you use graphical query tools to create a query.
- **DataSheet View:** To see the data retrieved by query.
- **SQL View:** To enter SQL (structured query language) statements to create or change a query.



When you use Design View to create query, you use a tool called Graphical Query by Example (QBE). With Graphical QBE, you create queries by dragging fields from the field list in the upper portion of the query window to the QBE grid in the lower portion of the window.



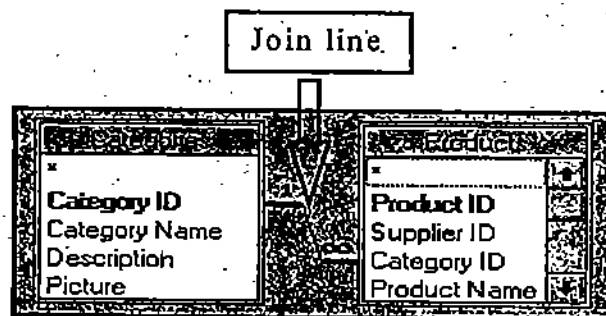
In the QBE grid, each column contains information about a field included in the query.

6.4 JOIN TABLES

Suppose you have two tables ITEM and TRANS. ITEM table contains Item_code, Description, QOH, and Rate. And TRANS table contains Trans_no, Item_code, Qty, and Date_of_Trans. Now if you want to make a query, which contains *Trans_no, Item_code, Description and Date_of_Trans*, you will have to join two tables. To create a query that uses more than one table; you add the tables you want and make sure the tables are joined to each other. Joining the tables tells Microsoft Access how to connect the information in the tables. You can join the tables by drawing the join lines between tables, although in many cases Microsoft Access creates join lines automatically. In most cases, a join line tells Microsoft Access : "Select the records from the both tables that have the same values in the fields that are joined." This is an inner join. The fields join in this way are called join fields.

To Join two tables in the query window

- Select a field in one table and drag it to the equivalent field in the other table. It draws a join line from one table to another.



To Delete a join between two tables

Creating
a Query

- Select the join line and press Del key.

6.5 SELECT FIELDS

After you have added tables to your query, now you can select the fields to include in the query. The fields shown in the field list belong to the tables or queries you selected in the add table dialog box. The fields you select determine the data you see when you look at the query's Dynaset (records displayed as output) in DataSheet view.

The field list in the upper portion of the query window displays all the fields available for your query. If you add more than one table, you see a field list for each table.

To add a field to a query

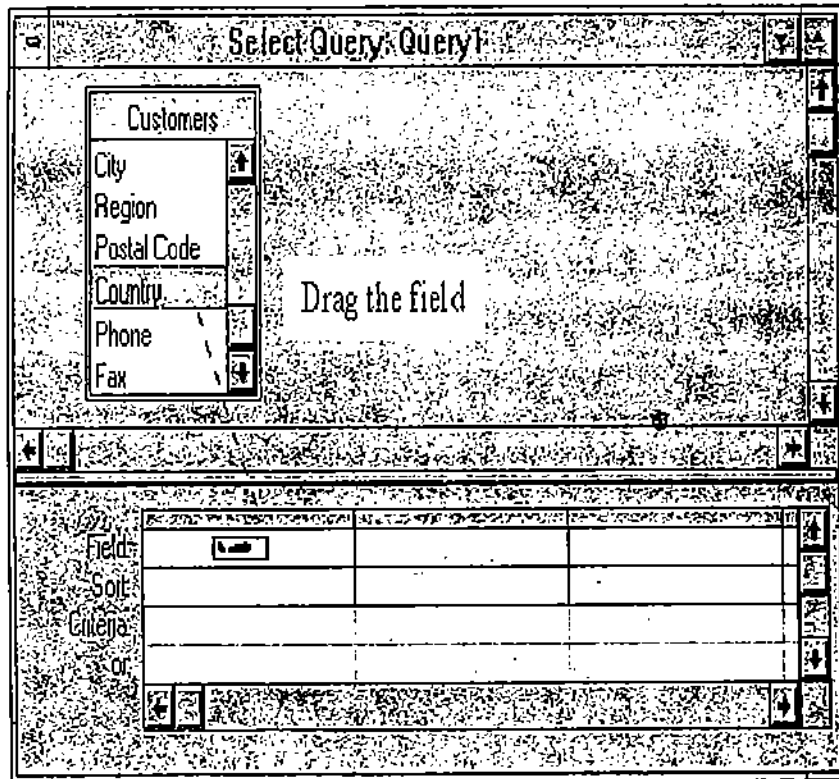
- Drag the field from the field list to a cell in the field row of the QBE grid.
- Repeat this procedure until all the fields you want in your query are shown in the QBE grid.

6.6 SPECIFY CRITERIA

To limit your query's Dynaset to certain records, you specify criteria for the query. When you specify criteria for a query, you use an expression. An expression tells Microsoft Access which records to include in the query's Dynaset.

To specify criteria for a field

1. In the QBE grid, select the criteria cell in which you want to define an expression.



2. Type the expression in the criteria box.

Field	Sort	Criteria
Country		"UK"
City	Ascending	

Some Examples of Expression:

Criteria	Retrieves Records where
100	Value is 100
>15000	Value is greater than 15000
<#01/01/96#	Date value is before 01/01/96
"USA"	Value is USA

To check your results

Creating
a Query

- Select DataSheet from the view menu.

6.7 SORT RECORDS

When you run a query and view the Dynaset, it's often easier to interpret the data if you have sorted it. When you sort records, you can place them in alphabetical order or in numeric sequence.

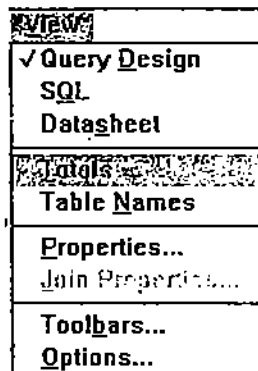
Creating a query that sorts data is useful when you want to display sorted data in a form or a report. You create a query that sorts the data and then use the query as the source of data for the form or report.

6.8 CALCULATE TOTALS

Often the most interesting information you can see is the total, average or some other type of tally over a set of records.

To calculate totals

1. Select Totals from the View menu to display the totals row in the QBE grid. It automatically fills "Group By" in each box.



2. Select the field to total on it.
3. Select sum from the list of total cell.

Field	Category Name	Unit Price
Total	Group By	Sum
Sort		
Show	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Criteria		
or		

4. Select DataSheet from the View menu to see the results.

View
<input checked="" type="checkbox"/> Table Design <input type="checkbox"/> Datasheet
Indexes... Table Properties...
Toolbars... Options...


6.9 MODIFY A QUERY

The query window is designed for flexibility and ease of use. Because of the windows graphical features, you can easily move, delete columns in your query.

To move a column in a query

1. Click the field selector (column heading) of the column in design view.
2. Click the field selector again, hold down the mouse button and drag the column to its new location.

Select this column and drag it to new location



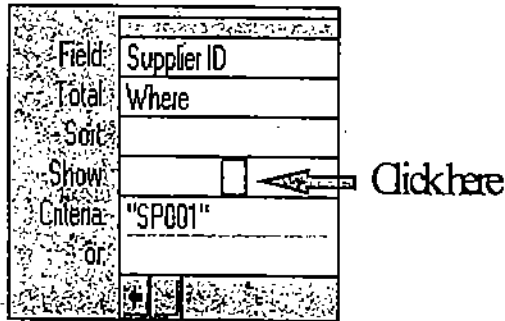
Field	Supplier ID	Category Name	Unit Price
Total	Where	Group By	Sum
Sort			
Show	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Criteria	"SP001"		
or			

To Delete a column in a query

1. Click the field selector (column heading) of the column in design view.
2. Press Del key.

To Exclude a field from the query's DataSheet

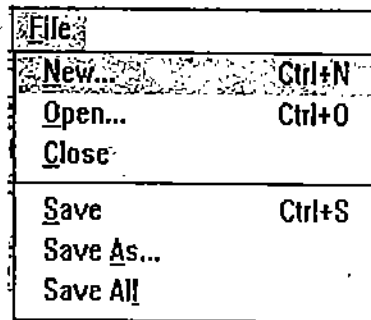
- Clear the field's show box by clicking it.



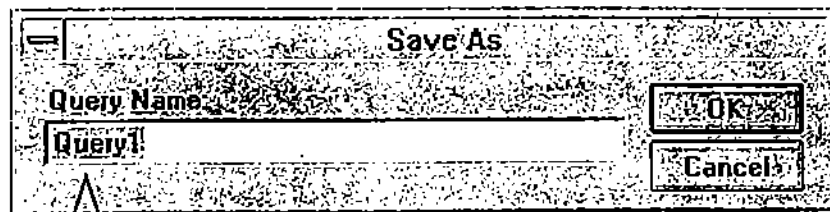
6.10 SAVE A QUERY

To save a query

1. Select Save from the File menu to display Save as dialog box (if you are saving a query first time).



2. Type name in query name-box.
3. Click OK to save query in the database.



Type query name

Check Your Progress

- Question 1. How many kinds of Queries are there in MS-Access?
- Question 2. What is the difference between all of the Queries provided by MS-Access?
- Question 3. What is Query Wizard used for?
- Question 4. Can we make all kinds of Queries with the help of Query Wizard?

6.11 SUMMARY

A query is a conditional selection of about the data.

A new query can be created by clicking on the Query button in the database window.

In the query window, you can design a query using a feature called Graphical Query By Example (QBE).

You specify criteria in QBE to display specific records.

Multiple tables can be added in a query.

To display records from more than one table, join the tables by joining lines.

UNIT 7 CREATING A FORM

Structure

- 7.0 Introduction
- 7.1 Objectives
- 7.2 Create a Form with a Form Wizard
- 7.3 View Records in a Form
- 7.4 Add, Delete and Save Records
- 7.5 Save and Close a Form
- 7.6 Summary

7.0 INTRODUCTION

You can view the records from a table using a Query or by applying a Filter. But both the ways of viewing records display you the data in raw form. To view the data in a customized way, you have something known as 'Forms'. In this session you will be able to create forms using *Form Wizards*.

The screenshot shows a Microsoft Access form window titled "Products". The form displays the following data for a record:

Product ID:	11
Supplier:	Sunil Kumar
Category:	Beverages
Product Name:	Chai
English Name:	Dharamsala Tea
Quantity Per Unit:	10 boxes x 20 bags
Unit Price:	\$18.00
Units In Stock:	39
Units On Order:	0
Reorder Level:	10
Discontinued:	<input type="checkbox"/>

At the bottom of the form, there is a status bar that reads "Record: 1 of 77".

A Form provides an easy way to view data, you can view all the values for one record in form view or you can switch to the datasheet view of the form to see all the records for that form. Using a form is also an efficient way to enter data. It can save you time and prevent typing errors.

A Form offers the most convenient layout for entering, changing and viewing records in your database. The form design tools in Microsoft Access help you design easy-to-use forms that present your data in an attractive format with special fonts and other graphical effects such as colour and shading.

7.1 OBJECTIVES

After going through this unit you shall be able to:

- Create a Form with a Form Wizard;
- View Records in a Form;
- Add, Delete and Save Records;
- Save and Close a Form.

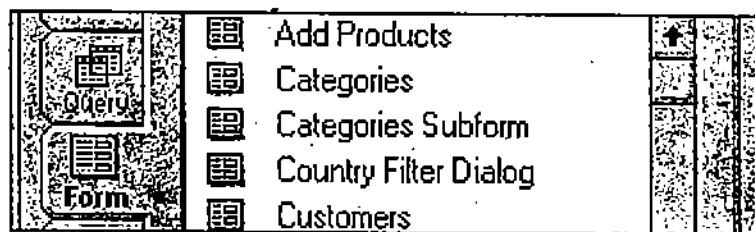
7.2 CREATE A FORM WITH A FORM WIZARD

You can create forms with or without the aid of form wizards. Form wizards speed up the process of creating forms because they do all the basic work for you. The autoform wizard produces a simple preformatted form without prompting you for information. The other form wizards prompt you with questions and then create a form based on your answers. Even if you have created many forms, you may want to use a form wizard to quickly create a form that you can use as is or refine to your specifications.

When you use a form to enter or display data in your database, you must connect the form to the table or query that is its source of data. If all the data is in one table, base the form on that table. If the data is in more than one table, base the form on a query.

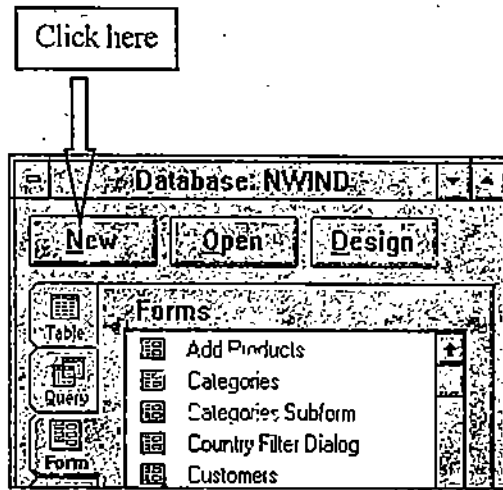
To Select a Form by Using a Form Wizard

1. Click *Form* button in the Database window.

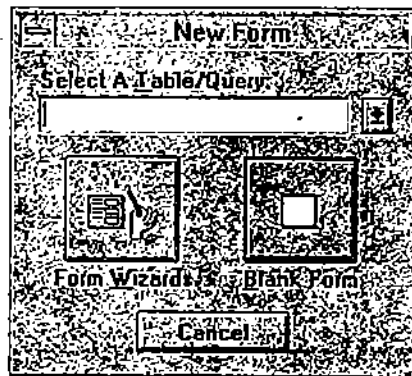


Click here

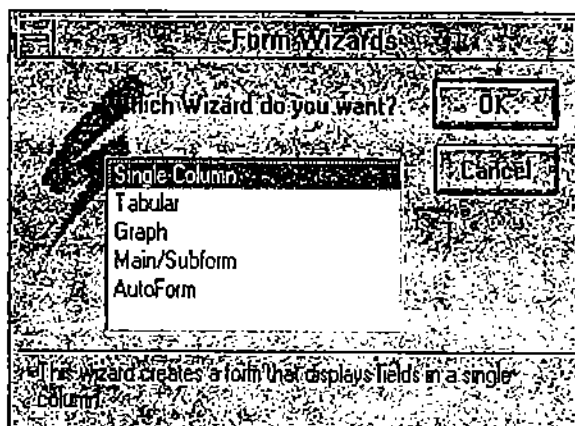
2. Click the *New* button to displays a New Form dialog box.



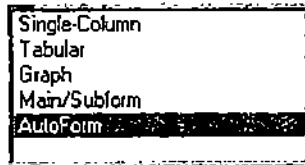
3. Select a *Table/Query* in respective list box.



4. Click the *Form wizard* button to display the first form wizard dialog box.



5. Select *AutoForm*.



6. Click *OK* to create and open the form. The form shows the first record in your table.

A screenshot of a Microsoft Access form titled 'ITEM MASTER'. The form contains several text boxes with labels: 'ITEM CODE' (value: 3), 'DESCRIPTION' (value: COMB), 'QOH' (value: 2), 'RATE' (value: £23.00), 'BOQ' (value: 5), and 'RSN STATUS' (value: F). At the bottom, there is a status bar that reads 'Record: 1 of 2'.

7.3 VIEW RECORDS IN A FORM

Now you have a form that you can use to view, change, add, and delete records in the table. The objects on the form are called controls. These controls are used to change and view data. The controls on your form are:

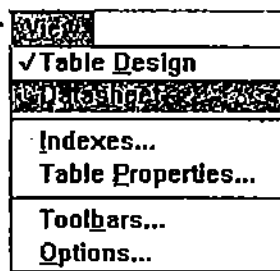
A screenshot of a form with several text boxes, each with a label to its left. The labels and their corresponding values are: 'Product Name' (Chai), 'English Name' (Dharamala Tea), 'Quantity Per Unit' (10 boxes x 20 bags), 'Unit Price' (\$18.00), 'Units in Stock' (39), 'Units On Order' (0), and 'Reorder Level' (10). A white rectangular box is drawn below the 'Reorder Level' text box.

Text boxes with attached labels

- A *label*, which displays text.
- A *text box* provides an area to display or type text that are stored in your database.

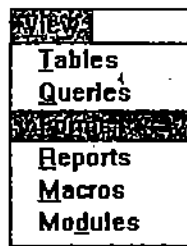
To Switch to Datasheet View

- Select Datasheet from the view menu, to displays form's data in datasheet view.



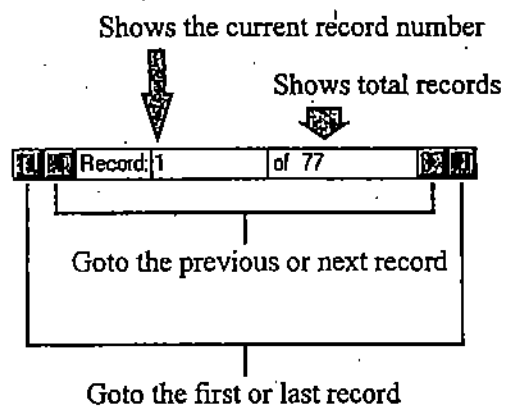
To Switch to Form View

- Select Form from the View menu to displays a record in form view.



To Move from Record to Record on a Form

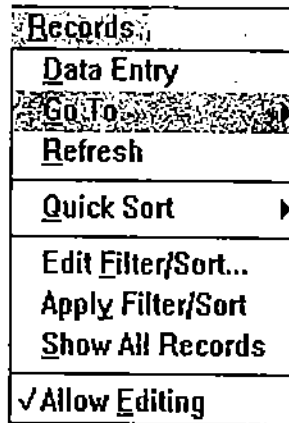
- Use the navigation buttons to go to the first, last, previous or next record.



7.4 ADD, DELETE AND SAVE RECORDS

To add a new record

1. Select GoTo and then select New from the Records menu to add a new record.



2. Type the value in the first text box.

ID	ITEM CODE	TRANS TYPE	QTY
1	C1	I	3
2	S1	R	2
(Counter)			0

3. Press TAB key to move to next field.
4. Repeat step 1 to step 3 to enter other information.

To save a record

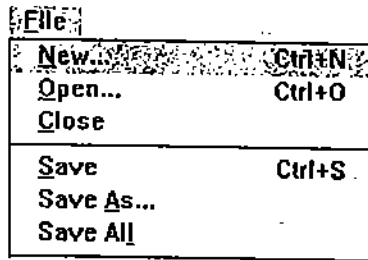
- Press TAB key to move to the next record.

As you as you leave the record, Microsoft Access saves it in the table. You don't need to do anything else to save the data.

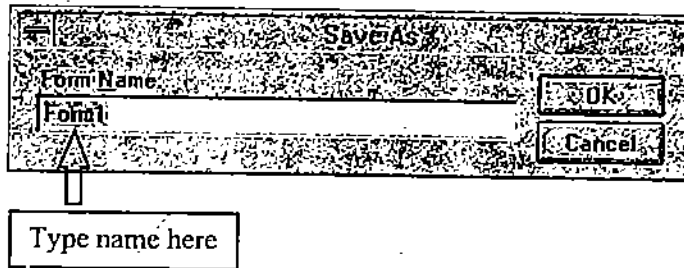
7.5 SAVE AND CLOSE A FORM

To save a form

1. Select Save from the File menu to displays Save As dialog box.

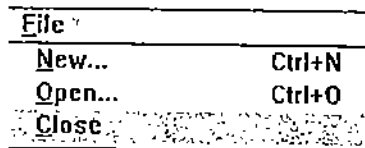


2. Type form name and click Ok to save it.



To close a form

- Select Close from the File menu to close the form.



Check Your Progress

Question 1. How many Views does a Form Whizard have

Question 2. What is the difference between...

Question 3. What are Links used for?

7.6 SUMMARY

Microsoft Access forms can be used to enter, change and view data.

The form window has three views: Form view, Datasheet view and Design view.

You can change and view data in the form view, usually see all the fields for one record at a time.

You can change and view data in the datasheet view, using a datasheet.

You can change the form structure in the design view.

UNIT 8 CUSTOMISING YOUR FORM

Structure

- 8.0 Introduction
- 8.1 Objectives
- 8.2 Change a Form's Design
- 8.3 Select and Resize Controls
- 8.4 Move and Delete Controls
- 8.5 Change Fonts, Size and Colour of Text
- 8.6 Summary

8.0 INTRODUCTION

In the last session, you had learnt how to create a form using *Form Wizard*.

In this session you will learn how to customize your forms.

8.1 OBJECTIVES

After going through this unit you shall be able to:

- Change a Form's Design;
- Select and Resize Controls;
- Move and Delete Records;
- Change Fonts, Size and Colour of Text.

8.2 CHANGE A FORM'S DESIGN

You have used form to add, delete, and modify records. Now you want to do some changes in the form's design. You can change the appearance of a form in the form's Design view. You can open your form in Design view from the Database window.

Change this form ...

The screenshot shows a form window titled "Products" with a data entry view. The fields and their values are:

Product ID:	1
Supplier:	Exotic Liquids
Category:	Beverages
Product Name:	Chai
English Name:	Dharamsala Tea

... to look like this

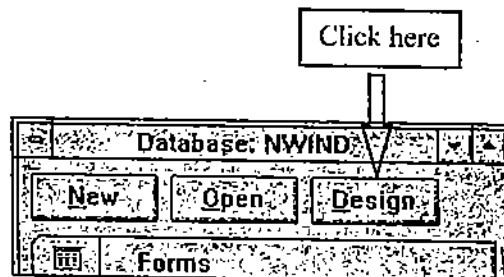
The screenshot shows the "Products" form in Design View. The form is divided into sections for data entry and display. The fields and their values are:

Product ID:	1	Product Name:	Chai		
English Name:	Dharamsala Tea	Quantity Per Unit:	10 boxes x 20 bags	Unit Price:	\$18.00
Units In Stock:	39	Units On Order:	0	Reorder Level:	10

At the bottom, a status bar shows "Record: 1 of 77".

To Open a Form in Design View

1. Click the *Form* button in the Database window.
2. Select form from the forms list.
3. Click the *Design* button to opens Form in Design view.

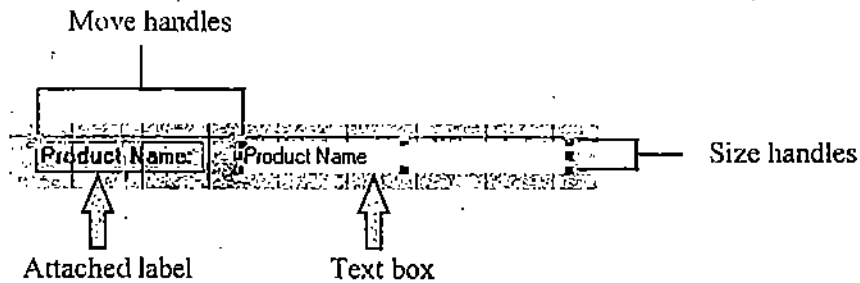


8.3 SELECT AND RESIZE CONTROLS

Controls are the objects on your form that display data from the field or label to display text. In Design view, you can select and resize the controls on a form.

To select a control

- Click the text box, to displays size and move handles around the control.

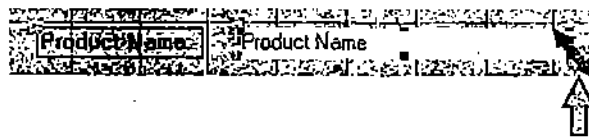


Use to select a control

- Drag the handles on the top and bottom to size the text box vertically.
- Drag the handles on the left and right sides to size the text box horizontally.
- Drag the handles in the corners to size the text box both vertically and horizontally.
- In controls with attached labels, use the move handles to move each part separately.

To resize a control

1. Position the pointer at the corner of the text box.
2. Drag the border to resize the control.



When pointer look like this, you can resize the control

You can also select more than one control.

To select more than one control

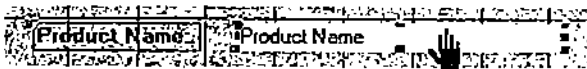
- Position the pointer and then drag diagonally through all the controls you want to select. It draws a rectangle around the controls. When you release the mouse button, all the controls in or touching the rectangle are selected.

8.4 MOVE AND DELETE CONTROLS

All the text box controls on your form have attached labels. You can move a control and its attached label together, or you can move them separately.

To move a control

1. Select the control to move.
2. Position the pointer anywhere on the control and hold down the mouse button.



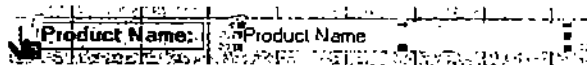
When pointer look like this, you can move the control

3. Drag the control (it moves text box and its attached label together.).
4. When the control is positioned where you want it, release the mouse button.

When you move a text box, its attached label moves with it. You can also move a control or its attached label separately.

To move the attached label separately

1. Select the control.
2. Position the pointer at the left top corner of the label and hold down the mouse button.

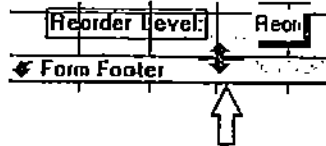


When pointer look like this, you can move the label separately

3. Move the label around the text box to display.
4. When the label is positioned where you want it, release the mouse button.

To lengthen a section

1. Position the pointer on the border between the detail section and the form footer.



When pointer look like this, you can change the length of the section

2. Drag the border down to increase the length of the section.

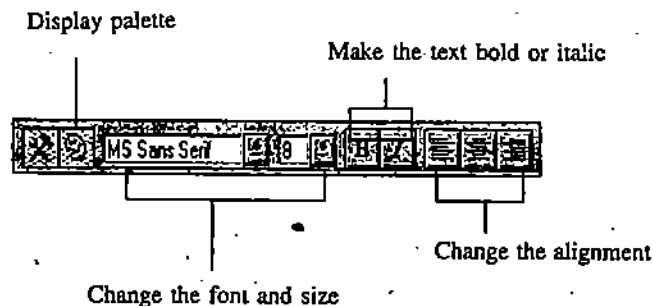
To delete a control

1. Select the control to delete.
2. Press DEL key.

It deletes text box as well as delete the attached label.

8.5 CHANGE FONTS, SIZE AND COLOUR OF TEXT

Microsoft Access provides so many choices for the appearance of controls on forms. You can change the size of the text in the label, change the font of the text and can make the text bold or italic. You can also fill colour to background of the text and text colour.



To make text bold or italic

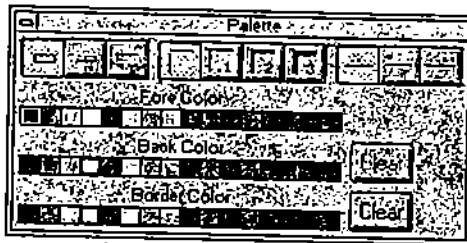
1. Select the label, to enables additional tools in the toolbar for changing the appearance of the text.
2. Click the bold or italic button to changes the text.

To change the font and size of a label

1. Select the label.
2. Select font from the fonts list to change the appearance of the text.
3. Select size from the size list to resize the label text.
4. Using the mouse pointer to resize the control.

To change the colours of the text

1. Select the label.
2. Click the palette button on the toolbar to displays the palette.



3. Select the Fore color or Back color or Border color, to change the text colour, fill the colour, and change the border colour respectively.
4. Click the palette button on the toolbar to close the palette.

Check Your Progress

- Question 1. Microsoft Access divides the form into how many sections in Design view?
- Question 2. Write the steps for changing the Font of Label and Text Box.

8.6 SUMMARY

To change the structure of the form in the design view, click on the Design button.

Form header contains the heading label of the form. It appears once at the top of the window.

Detail section contains the fields from the table to viewing data. The detail section is repeated for each record.

Form footer appears once at the bottom of window.

A label displays instructions or descriptive text.

You can enter or display data by the text box.

To change the fonts and size, select Font or Size from the respective list of the Formatting toolbar.

Select color from the palette to change the color of the label or text box.

UNIT 9 SHOWING DATA FROM MORE THAN ONE TABLE ON A FORM

Structure

- 9.0 Introduction
- 9.1 Objectives
- 9.2 Create a Form that Contains a Sub Form
- 9.3 Use a Query to Include Fields from More Than One Table
- 9.4 Summary

9.0 INTRODUCTION

In the previous two sessions, you have learnt how to create customized forms, using one table. This unit covers sub forms. You will learn how to create forms using two tables or a Query.

9.1 OBJECTIVES

After going through this unit you shall be able to:

- Create a form that contains a sub form;
- Use a query to include fields from more than one table.

9.2 CREATE A FORM THAT CONTAINS A SUB FORM

Using a subform is one way to include information from more than one table in a form. A subform is a form within a form. In Microsoft Access, the primary form is called the main form and a form within the main form is called the subform.

When you use a subform, it's easy to see relationship between records from two or more tables. The main form and subform in this type of form are linked, so that the subform displays only records that are related to those in the main form.

Using the Main/Subform Form wizard is usually the quickest, most convenient way to design a main form and its subform. Even if you want to customize the main form and subform extensively, you can start out by using the wizard to create a quick prototype of a form that you can change later.

When you create a Form/Subform using the wizard, you are free to view data in a subform in either Datasheet view or Form view.

Categories

Category ID: Main form

Category Name:

Description:

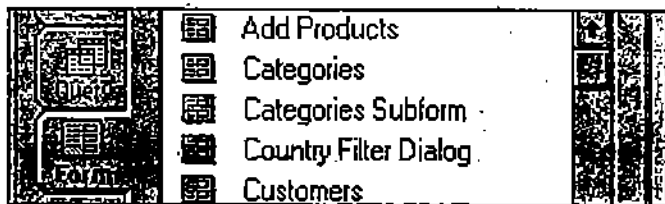
Product ID	Product Name	Unit Price
1	Chai	\$18.00
2	Chang	\$19.00
24	Guaraná Fantástica	\$4.50
34	Sasquatch Ale	\$14.00
35	Steeleye Stout	\$18.00
38	Côte de Blaye	\$263.50
39	Chaitreuse verte	\$18.00
43	Ipoh Coffee	\$46.00

Record: 1 of 12

Record: 1 of 8

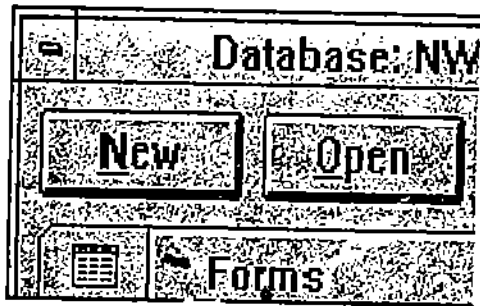
To create a main form and a subform using the Main/Subform form wizard:

1. Click the *Form* button in the Database window.

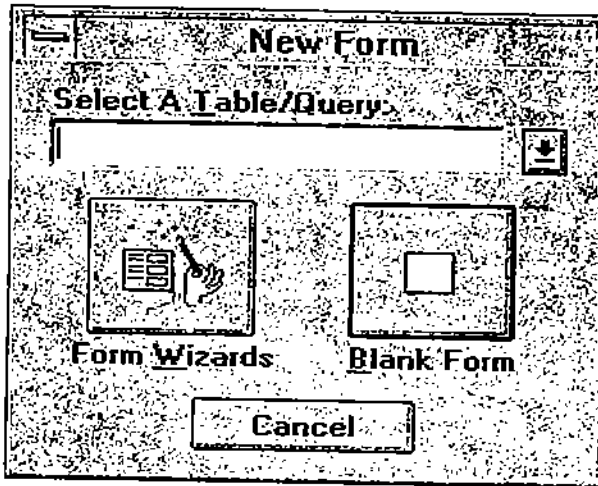


2. Click the *New* button to displays the New Form dialog box.

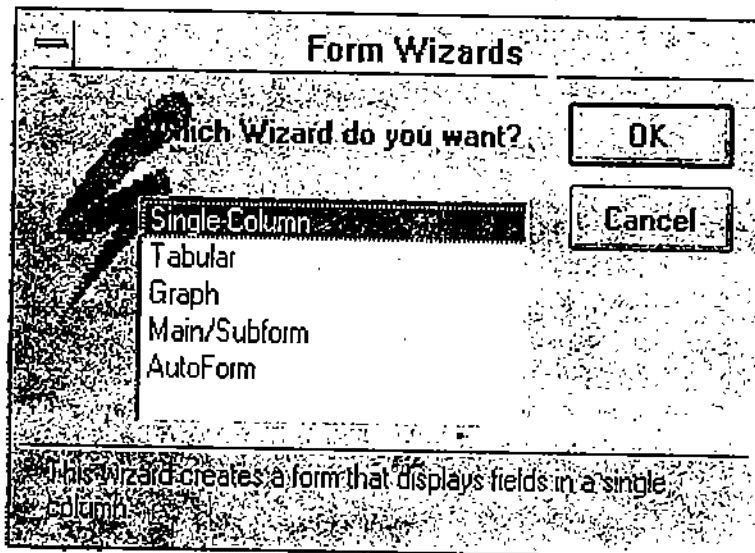
Showing
Data from
More Than
One Table on
A Form



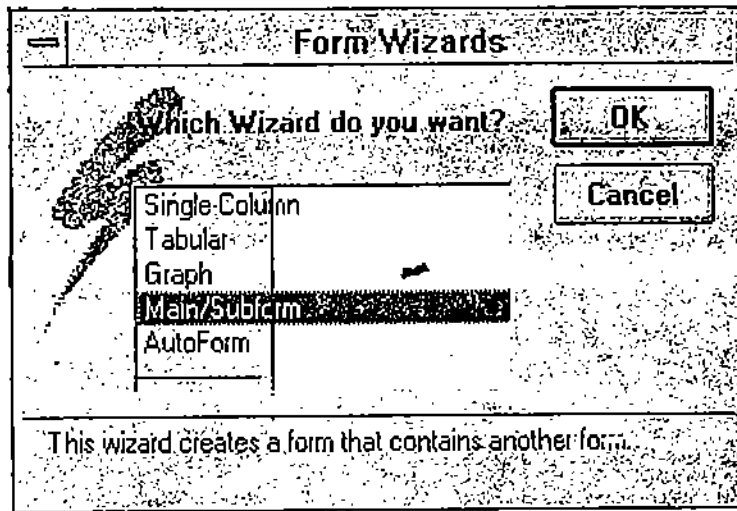
3. Select a *Table/Query* to display on the main form.



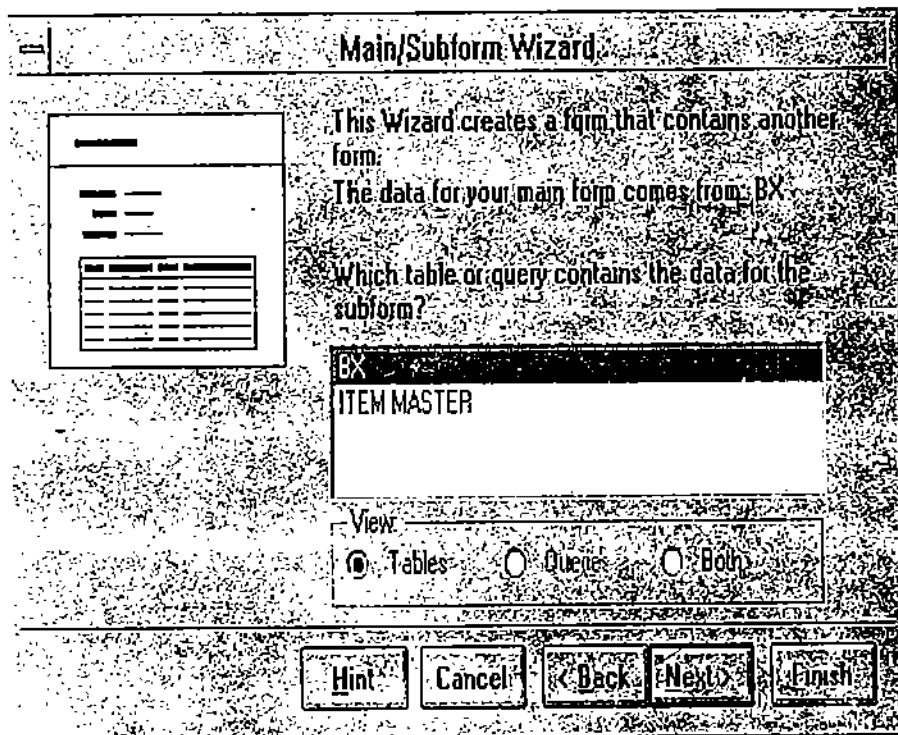
4. Click the *Form* wizards button to displays the First Form wizard dialog box.



5. Select the *Main/Subform* form wizard.



6. Select a *Table/Query* to display on the subform.



7. Follow the directions in Form wizard dialog box.

8. Click the *Finish* button in the last dialog box to entering or viewing data.

9.3 USE A QUERY TO INCLUDE FIELDS FROM MORE THAN ONE TABLE

A form or a subform can be based on a query. Consider using a query as the basis for a subform when you want the subform to display values from several tables or when you want to display limited or sorted information from one or more tables.

Product Name:	Alice Mutton
Unit Price:	\$39.00
Units In Stock:	0
Units On Order:	0
Contact Name:	Ian Devling
Phone:	(03) 444-2343

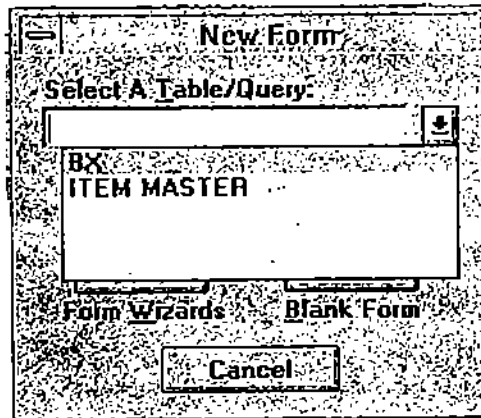
Record: 1 of 77

To Create a Query :(Discussed in the earlier sessions in detail)

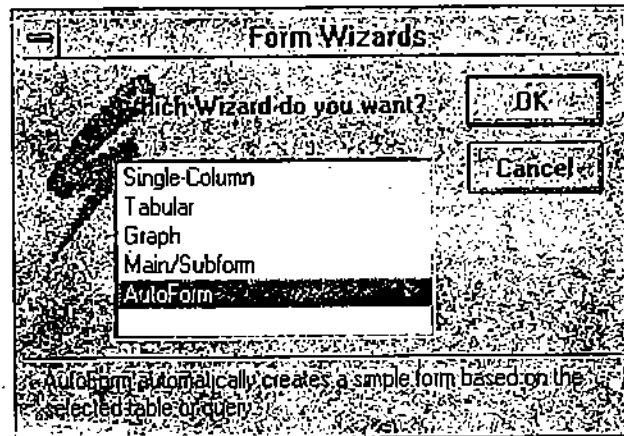
1. Click the *Query* button and click the *New* button to open the *New query* window.
2. Add the two tables, to display data in the form.
3. Connect the tables with join line.
4. Drag the fields from the field list to QBE grid.
5. Save the query.
6. Close the query.

To base a form on a query

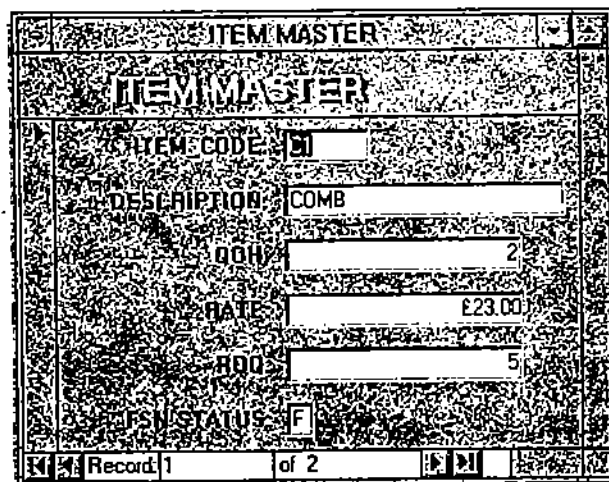
1. Click Form button and click New button to display New Form dialog box.
2. Select the query (you have just created) from the list.



3. Select autoform.



4. Click OK to creates and opens the form



Save it and close the form.

9.4 SUMMARY

Showing
Data from
More Than
One Table on
A Form

To show the set of records associated with a single record on the main form by the sub form.

To display data from more than one table, use a query as a base for the form.

UNIT 10 CREATING REPORTS AND MAILING LABELS

Structure

- 10.0 Introduction
- 10.1 Objectives
- 10.2 Use Reports to Present Data
- 10.3 Create a Report
- 10.4 Preview, Print and Save a Report
- 10.5 A Report in Design View
- 10.6 Create and Print Mailing Labels
- 10.7 Summary

10.0 INTRODUCTION

Reports are used to present data on a printed paper. It provides a way to retrieve and present data as meaningful information, which might include totals and grand totals also. In this unit, you will use reports to present data.

10.1 OBJECTIVES

After going through this unit you shall be able to:

- Use Reports to present data;
- Create a Report;
- Preview, Print and Save a Report;
- Create and Print Mailing Labels.

10.2 USE REPORTS TO PRESENT DATA

A Report is information you have organized and formatted to fit your specifications. A report gives you a way to retrieve and present data as meaningful information that you can use and distribute. Examples of reports are mailing labels, Invoices and sales summaries.

With Microsoft Access, you can design reports that present information the way you want. You can use many different design elements, such as text, data, pictures, lines, boxes and graphs to create the report.

Reports are also an efficient way to print information that you use regularly. You can create a design for a report and then save it, so that you can use it over and over again. Once you save the report design, it stays the same, but you get current data each time you print the report. If your reporting needs change, you can adjust the report design or create a similar report based on the original report.

Sales Summaries		
20-Nov-96		
Employee Sales:	Employee Name:	Employee Sales:
	Steven Buchanan	\$47,674
	Laura Callahan	\$59,687
	Nazzy Davolio	\$93,932
Category Sales:	Category Name:	Category Sales:
	Beverages	\$127,189
	Condiments	\$56,462
	Confections	\$88,500

You can create reports that:

- Organise and present data in groups.
- Calculate running totals, group totals, grand totals and percentages of totals.
- Include sub reports and graphs.
- Present your data in an attractive format with pictures, lines and special fonts.

The records appear in one large group

Select Query: Product List			
Category Name	Product Name	Product ID	Units
Beverages	Sesquatch Ale	34	
Beverages	Steeleye Stout	35	
Condiments	Aniseed Syrup	3	
Condiments	Chef Anton's Cajun Seasoning	4	

Report: List of Products by Category			
List of Products by Category			
20-Nov-96			
Category	Product	Product ID	Quantity
Beverages			
	Chai	1	1000
	Chang	2	24-1
	Charreusa verte	39	7500
	C8ta de Blaye	38	12-7
	Ipoh Coffin	43	16-5
	Lakkalik66ti	76	5000
	Laughing Lurbejack Lager	67	24-1
	Outback Lager	70	24-3
	Rhombus Klosterbier	75	24-0

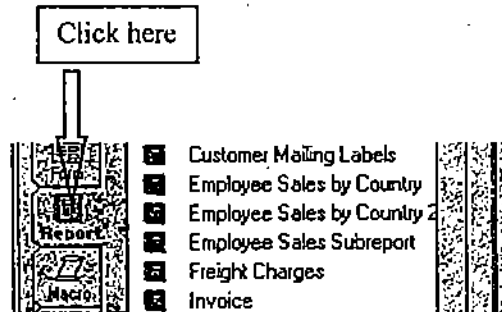
The records for each category appear as separate groups

10.3 CREATE A REPORT

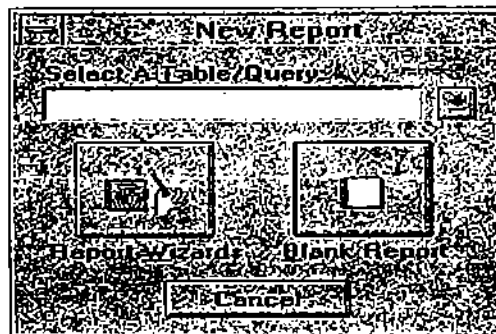
Your report will group the day wise sales for each month under the month's heading and show total sales at the end.

To create a report with grouped data

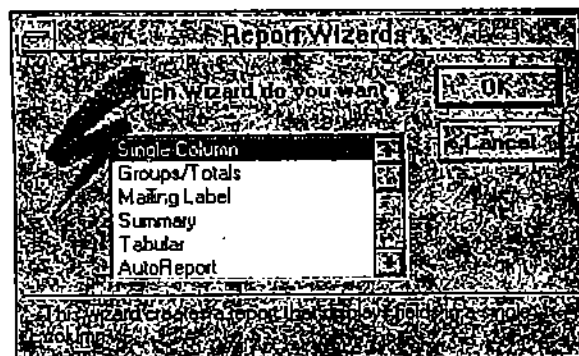
1. Click the *Report* button in the Database window.



2. Click the *New* button to displays the New Report dialog box.

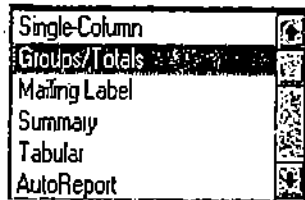


3. Select the tables and queries from the list.
4. Click the *Report* wizards button to displays the first report wizard dialog box.

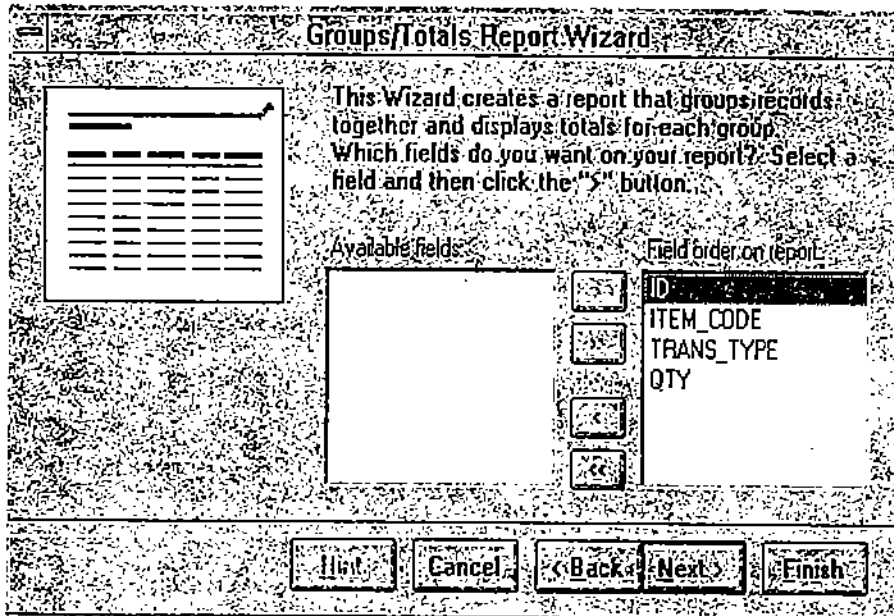


5. Make these choices as you go through the dialog boxes:

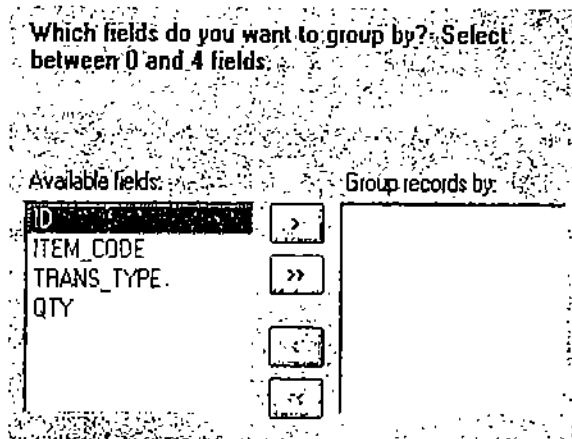
a) Select the *Groups/Totals* wizard.



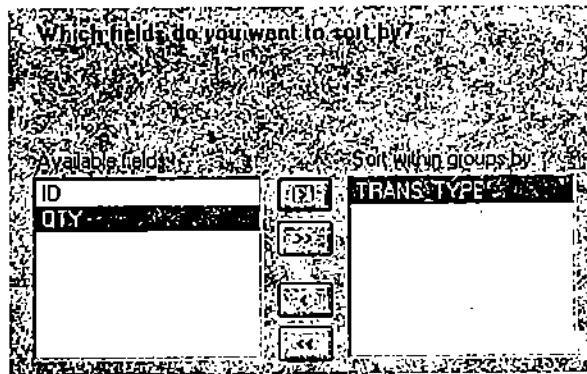
b) Add all the fields from the query to the report.



c) Group records by the *Item_code* field.



d) Sort records on the Trans_Type field.



e) Select the Executive style for your report.

f) Give the title to your report.



g) Click the Finish button to create and opens the report in Print Preview.

Microsoft Access

File Edit View Format Window Help

Report: List of Products by Category

List of Products by Category
26-Nov-96

Category	Products	Product ID:	Quantity Per Unit:
Beverage	Chai	1	10 boxes x 20 bags
	Chang	2	24 - 12 oz bottles
	Chartreuse verte	39	750 cc per bottle
	Côte de Blaye	38	12 - 75 cl bottles
	Ipoh Coffee	43	16 - 500 g tins
	Lakkaikosti	76	500 ml
	Laughing Lumberjack Lager	67	24 - 12 oz bottles
	Oatmeal Lager	70	24 - 355 ml bottles
	Rhône-Alpin Klausebier	75	24 - 0.5 l bottles

Page: 1

10.4 PREVIEW, PRINT AND SAVE A REPORT

Creating
Reports and
Mailing
Labels

After the Report wizard creates your report, Microsoft Access displays your report as it would appear in print.

To see a whole page in report

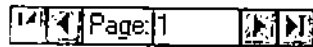
1. Position the pointer over the report in Print preview. The pointer changes to a magnifying glass.



2. Click the report to display a view of the whole page.
3. Click the report again to zoom back and view data.

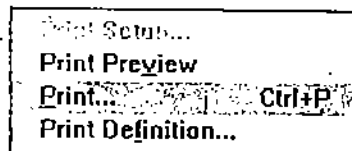
To scroll in one page and through pages

- Click the horizontal and vertical scroll bars to scroll in a page.
- Click the page buttons to scroll in other pages.

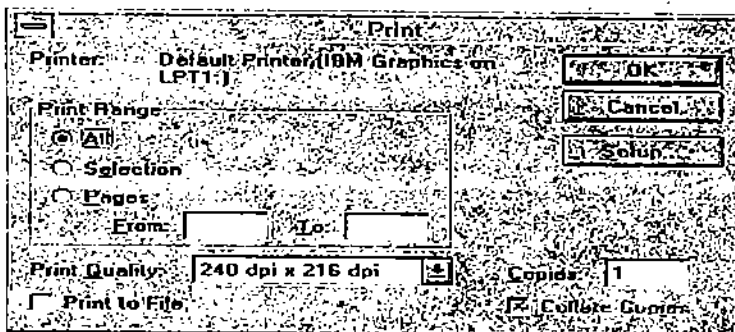


To print a report

1. Select Print from the File menu.

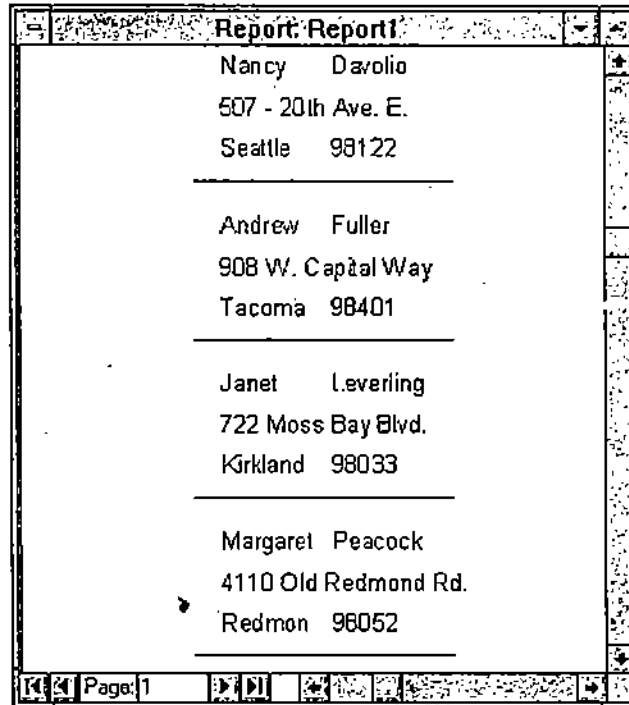


2. It will show you the following Print Dialog Box:



3. Select OK to print.

5. Select the Mailing Label Report from the wizard.
6. Follow the directions to make mailing labels.
7. Click the Finish button from the last wizard dialog box to creates mailing report.



To print the mailing labels

- Select Print from the File menu to display print dialog box.
- Click Ok to print labels.

10.7 SUMMARY

A report provides a way to retrieve and present data as meaningful information that you can use and distribute.

You can create reports with or without the help of Report Wizards.

You can use table or query as the base of the report.

You can create reports for:

- Organise and present data in groups.
- Calculate group totals, grand totals and percentage of totals.
- Present your data in an attractive format with pictures, lines and special fonts.