CONTENTS

DATABASE MANAGEMENT SYSTEM

SECTION A

Database Concepts: What is Database? Need of Database, Function of the Database; Types Database ; Relational Database Management System, Relational Model - Key Concept; Domain Constraint, Integrity Constraints; Foreign Key.

SECTION B

Database Development Process, Database Modeling & Database Design.
E-R Model, Attributes, Relationship, Logical Database Design, Normalization, First Form, Second Normal Form, Third Normal Form, Translating E-R Diagram to Relations, Physical Database Design.

SECTION C

Relational Algebra & SQL Relational Database Commands.
Data-types, Create Table, Drop Table, Alter Table, Insert Into, Delete From, Update, General Query Syntax (Select), Create View, Drop View, Set Operators - Union, Intersect, Minus, Functions, Group Functions, Join, Sub Queries.

SECTION D

Data Administration, Client / Server and Distributed Databases.
Data Administration Functions, Data Administration Tools - Repositories, CASE Tools, Concurrency Control, Database Security, Database Recovery.
Database Applications: Financial Systems, Marketing System, Foreign Trade, Inventory Information Systems.

CHAPTER 1

DATABASE CONCEPTS

LEARNING OBJECTIVES

After going through this chapter, you should appreciate the following:

- Primary Key
- What is Database?
- Need of Database.
- Function of the Database
- Types of Databases
- Relational Database Management System
- Relational Model - Key Concept
- Integrity Constraints
- Foreign Key

NOTES

PRIMARY KEY

A basic unit called in a database is called a Table that organizes the data. Rows are related. The columns in the table classify the nature of the data items that are related by virtue of row membership. The Name column identifies that whatever appears at this position in any row is a person's name. The Address column identifies that whatever appears in this position in an address. Membership in a row that has columns makes the data items in the row belong to a set.

In fact, a database table is a conceptual representation of the set theory we all struggled with in grade school mathematics. Columns also play a special classificatory role in a database; they identify the type of the data stored in them. Data type is the means by which data storage is interpreted.

You must remember the following:

- Data is stored in tables.
- Tables have rows that group related data items together and columns that classify the data as to its type and its role in the world.

Tables

A table is the place in the database where all the data is stored. Every piece of information that gets loaded into an Oracle database must be placed inside an Oracle table. In fact, all the information needed by an Oracle database to manage itself is stored in a series of tables that are commonly known as the data dictionary. Think of the data dictionary as table about tables. The data dictionary tables tell the database what kind of data is stored in the database, where it is located and how the database can work with it.

A table is made up of columns. Each column must be given a unique name within that table and assigned a data type (such as varchar2, date or number) with an associated width (which could be predetermined by the data type, as in date). Each table column can also be designated as null or not null. Not null means the column data is mandatory for that column. In other words, for rows of data to be entered into that table, all columns assigned not null destination must contain valid data values.

To enforce defined business rules (integrity constraints) on a table's data, Oracle9i allows you to associate integrity constraints and triggers for a table.

Attribute

Within a database for example, you have names. These names would be there in your office records but you can assign various attributes to it too. These attributes are the additional information you need to keep for example, your job title.

Tuple/Rows

Nobody calls them by the old names, i.e., Tuples, everybody now calls them rows. As mentioned above a database actually consists of columns and rows. A row is a record of the data in the database. For example, in a school database, a record may contain the information related to a student, like his enrolment number, name, address, class, section, phone number, etc. A row comprises of fields that contain data. This record in turn belongs to a table.

Field

It is the other side of the record, i.e., column. It specifies a particular data in a database. It also corresponds to the name given to it. Every database must consist of a column.

Data

NOTES

It is the raw information which is fed into the database. It could be in the form of information collected from various sources. Like in the school database, the information about the student is collected from the form which he has filled up.

Concept of String

A string is a simple concept: a bunch of things in a line, like houses, popcorn or pearls, numbers, or characters in a sentence. Strings are frequently encountered in managing information. Names are strings of characters, as in Sachin Tendulkar. Phone numbers are strings of numbers, dashes, and sometimes parentheses, as in a telephone number, (011)-2551 6754. Even number, such as 5516754 can be considered as either a number or a string of characters.

Strings can include any mixture of letters, numbers, spaces, and other symbols (such as punctuation marks and special characters) are called character strings, or just character for short.

There are two string data types in Oracle.

CHAR strings are always a fixed length. If you set a value to a string with a length less than that of a CHAR column, Oracle automatically pads the string with blanks. When you compare CHAR strings, Oracle compares the strings by padding them out to equal lengths with blanks. This means that if you compare "character" with "character" in CHAR columns, Oracle considers the strings to be the same.

VARCHAR2 data type is a varying length string. The VARCHAR datatype is synonymous with VARCHAR2, but this may change later, so you should avoid using VARCHAR.

Use CHAR for fixed-length character string fields and VARCHAR2 for all other character string fields.

Number Values

In Oracle NUMBER stores any type of number. For example,

```
NUMBER  ( MAX_LENGHT)
```

You may specify a NUMBER's data precision with the following syntax:

```
NUMBER  (precision, scale)
```

Subtypes: DEC, DECIMAL, DOUBLE PRECISION, INTEGER, INT, NUMERIC, REAL, SMALLINT, FLOAT PLS_INTEGER defines column that may contain integers with a sign, such as negative numbers.

Date values

This is there to keep track of date variable.

Data type and Data integrity

There are various type of data inserted into tables. There are two aspects to it. Once is to make sure that the data which is being entered is of the right type and then we

NOTES

have to make sure that the data which is being entered is accurate. First we would see how the integrity of the data matters.

Domain Integrity

As discussed the primary key of the table must have the unique values, which identify the each row. So, if any row is having the value NULL for the primary key. The rule of primary key violate that is the primary key must be unique. This is the reason to follow the entity integrity rule according to which primary key will not accept the null value. This rule state that if attribute A of relation R is a prime attribute of R, then A cannot accept NULL values.

Referential integrity

It is the assurance of consistent and accurate data within a database. Referential integrity simply means that the values of one column in a table depend upon the values of a column in another table. For instance, in order for a customer to have a record in the ORDERS table, there must first be a record for that customer in the CUSTOMERS table. In order for data to be in the EMPLOYEE_PAY table, there must first be a corresponding personnel record in the EMPLOYEES table.

Types of Keys

Keys are the ways of connecting tables. These help you in creating relation between them. There are 4 type of keys.

Candidate key

We all know that in a table, every row must be different. There should be at least one attribute that can uniquely identify the row. These attributes are called Candidate keys.

Alternate key/Surrogate

It is used in case there is no possibility of naming a primary key. Then in that case you assign a another key as the primary key.

Primary key

A primary key for a uniquely identifies each row in a table and cannot be null. Oracle tables must have only one primary key defined.

Foreign keys

In a table, a foreign key, normally a single field, directly references a primary key in another table to enforce referential integrity.

WHAT IS DATABASE

Database means base of the data. It is based on the information given by you. It can be catalogued, stored and used. For all this information, database is used. Any collection of related information grouped together as a single item is a database. A metal filling cabinet containing customer records, a card file of names and phone numbers, and notebook containing a listing of a store inventory, all are databases.

Any collection of related information grouped together as a single item is a database.

NOTES

However, a file cabinet or a notebook does not itself make a database. Containers, like cabinets, notebooks, or computer programs like FoxPro, are only aids in organizing information.

In Database, data is stored in the form of rows and columns.

Database uses the table format of **rows** and **columns** to store the information, as shown here. A database, or a FoxPro database file, may consist of one such table or several ones.

In the example shown next you will notice that each row contains a name, an address, a phone number and a customer number. *Each row is related to the others because they all contain the same types of information in the same places.*

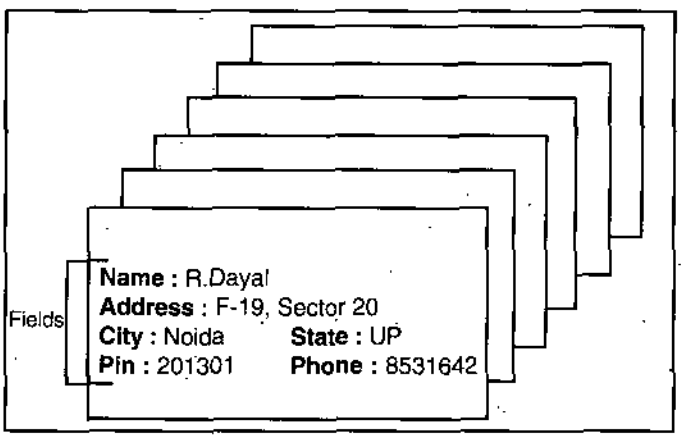
A mailing list, for example, containing a similar type of data about various people, is a good example of a database.

Name	Address	City	State	Pin	Ph. No.	Cust.No.
R.Dayal	F-19, Sec-20	Noida	UP	201301	8531642	0005
Sachin	G-478, Sakurpur	Delhi	Delhi	110034	2454312	0001
Rahul	12, Asaf Ali Road	Delhi	Delhi	110002	3255582	0002
Saurav	14, Asaf Ali Road	Delhi	Delhi	110002	3266698	0004
Anil	21, Kinari Bazar	Delhi	Delhi	110006	3280816	0006
Yuvraj	345, Surya Nagar	Ghazi	UP	201204	8876543	0003

Rows in a database table are called records, and columns are called fields.

The figure here illustrates this idea by comparing a simple one-table database to an address filling system kept on 3x5 file cards. Each **category** of information on a card is a field.

Fields can contain any type of information, as long as each field always contains the same type of information. In the card box, each record contains six fields: a name, address, city, PIN code, and phone number. Since every card in the box has the same type of information, the card box is a database.



A computerized database provides speed, compactness and flexibility. It is also easier to locate a particular record in computerized database.

Databases and database technology are having a major impact on the growing use of Computers. It is fair to say that databases play a critical role in almost all areas where computers are used, including business, electronic commerce, engineering, medicine, law, education, and library science, to name a few.

The preceding definition of database is quite general; for example, we may consider the collection of words that make up this page of text to be related data and hence to

NOTES

constitute a database. However, the common use of the term database is usually more restricted.

A database has the following implicit properties:

- A database represents some aspect of the real world, sometimes called the miniworld or the universe of discourse (UoD). Changes to the miniworld are reflected in the database.
- A database is a logically coherent collection of data with some inherent meaning. A random assortment of data cannot correctly be referred to as a database.
- A database is designed, built, and populated with data for a specific purpose. It has an intended group of users and some preconceived applications in which these users are interested.

We will use the word data as both singular and plural, as is common in database literature; context will determine whether it is singular or plural. In standard English, data is used only for plural; datum is used for singular.

Database Systems

A database can be of any size and of varying complexity. For example, the list of names and addresses referred to earlier may consist of only a few hundred records, each with a simple structure. On the other hand, the computerized catalog of a large library may contain half a million entries organized under different categories—by primary author's last name, by subject, by book title—with each category organized in alphabetic order.

A library database may contain name of book, author's last name, subject, book title in alphabetical order.

A database of even greater size and complexity is maintained by the *Internal Revenue Service* to keep track of the tax forms filed by taxpayers. If we assume that there are 100 million taxpayers and if each taxpayer files an average of five forms with approximately 400 characters of information per form, we would get a database of $100 \times 10^6 \times 400 \times 5$ characters (bytes) of information. If the revenue department keeps the past three returns for each taxpayer in addition to the current return, we would get a database of 8×10^{11} bytes (800 gigabytes). This huge amount of information must be organized and managed so that users can search for, retrieve, and update the data as needed.

A database may be generated and maintained manually or it may be computerized. For example, a library card catalog is a database that may be created and maintained manually. A computerized database may be created and maintained either by a group of application programs written specifically for that task or by a database management system. Of course, we are only concerned with computerized databases in this book.

NEED OF DATABASE

Information from the database, stored in the form of a computerized filling system, can be stored and retrieved quite easily. Tasks that would be time-consuming to accomplish manually, are easily done with the aid of a computer. *In principle, a database*

in a computer is not different from a database recorded on paper and filled in cabinets. But the computer does the tedious work like maintaining and searching through a database, and it does so quickly. A computerized database that can do all of this is known as a DataBase Management System, or DBMS for short.

NOTES

Storing massive amounts of information into written directories and filling cabinets can consume a great deal of space and time.

Manual database systems are usually not fool-proof. A telephone book, for example, is fine for finding telephone numbers, but if you have an address and not the name of the person who lives there, the telephone directory becomes useless for finding that person's telephone number. A similar problem plagues conventional office filling systems: if the information is organized by name and you want to find all the clients located in a particular area, you could be in for a tedious search. In addition, storing massive amounts of information into written directories and filling cabinets can consume a great deal of space.

A manual database can also be tedious to modify. For example, inserting a new phone number into a list and rearranging the list. Or, if the phone company were to assign a new area code, someone would have to search for all phone numbers having the old area code, and replace it with the new one. It is very easy to eliminate these type of problems, if you are using the computer for your database. **Advanatages of using the Database** can be listed as following:

- A computerized database provides speed; finding a phone number from among a thousand entries or putting the file in alphabetical order takes just seconds with the database management system.
- A computerized database is compact; a database with thousands of records can be stored on a single floppy CD.
- A computerized database is flexible: it has the ability to examine information from a number of angles, so you can search for a phone number by name, by address, or by pin code and then name.

Of course, these are just few of them.

A manual database can also be tedious to modify.

FUNCTION OF THE DATABASE

Computer programmers who have written programs for other applications know how much a database programming differs from the traditional programming. Here the database programming is quite different from the traditional approach of programming with files. In **traditional file processing**, each user defines and implements the files needed for a specific software application as part of programming the application.

Programming for database is quite different from traditional programming.

For example, for one user, the grade reporting office, may keep a file on students and their grades. Programs to print a student's transcript and to enter new grades into the file are implemented as part of the application. For second user, the accounting office,

NOTES

may keep track of students fees and their payments. Although both users are interested in data about students, each user maintains separate files—and programs to manipulate these files—because each requires some data not available from the other user's files. This redundancy in defining and storing data results in wasted storage space and in redundant efforts to maintain common data up to date. In the database approach, a single system of data is maintained that is defined once and then is accessed by various users. The main characteristics of the database approach versus the file-processing approach are the following:

- Self-describing nature of a database system
- Insulation between programs and data, and data abstraction
- Support of multiple views of the data
- Sharing of data and multiuser transaction processing

Let us study each one of them separately.

Self-Describing Nature of a Database System

A fundamental characteristic of the database approach is that the database system contains not only the database itself but also a complete definition or description of the database structure and constraints. This definition is stored in the DBMS catalog, which contains information such as the structure of each file, the type and storage format of each data item, and various constraints on the data. The information stored in the catalog is called meta-data, and it describes the structure of the primary database.

The catalog is used by the DBMS software and also by database users who need this information about the database structure. A general-purpose DBMS software package is not written for a specific database application, and hence it must refer to the catalog to know the structure of the files in a specific database, such as the type and format of data it will access. The DBMS software must work equally well with any number of database applications—for example, a university database, a banking database, or a company database—as long as the database definition is stored in the catalog.

In traditional file processing, **data definition** is typically part of the application programs themselves. Hence, these programs are constrained to work with only one specific database, whose structure is declared in the application programs. For example, an application program written in C++ may have struct or class declarations, and a COBOL program has Data Division statements to define its files. Whereas file-processing software can access only specific databases, DBMS software can access diverse databases by extracting the database definitions from the catalog and then using these definitions.

In the example, shown next, the DBMS catalog will store the definitions of all the files shown. These definitions are specified by the database designer prior to creating the actual database and are stored in the catalog. Whenever a request is made to access, say, the Name of a STUDENT record, the DBMS software refers to the catalog to determine the structure of the STUDENT file and the position and size of the Name data item within a STUDENT record.

STUDENT file	Name	Number	Class	Course
	Sachin	10	1	Phd
	Rahul	44	2	Phd

COURSE file	Name	Number	Hours	Dept.
	Visual Basic	VB10	5	Phd
	RDBMS	RD10	4	Phd
	Computer Sc.	CS10	2	CS

SECTION file	Section	Course	Semester	Year	Teacher
	88	VB10	IInd	2005	Tony
	90	RD10	Ist	2005	Greg
	112	CS10	IIIrd	2005	Tom
	120	MIC10	IInd	2005	Bob

GRADE_REPORT	Number	Section	Grade
	10	90	A
	44	88	B

PREREQUISITE	Course	Per. Number
	VB10	CS10
	RD10	MIC10

By contrast, in a typical file-processing application, the file structure and, in the extreme case, the exact location of Name within a STUDENT record are already coded within each program that accesses this data item.

DBMS software can access diverse databases by extracting the database definitions from the catalog and then using these definitions.

Insulation between Programs and Data, and Data Abstraction

In traditional file processing, the structure of data files is embedded in the application programs, so any changes to the structure of a file may require changing all programs that access this file. By contrast, DBMS access programs do not require such changes in most cases. The structure of data files is stored in the DBMS catalog separately from the access programs. We call this property program-data independence.

For example, a file access program may be written in such a way that it can access only STUDENT records of the structure. If we want to add another piece of data to each STUDENT record, say the BirthDate, such a program will no longer work and must be changed. By contrast, in a DBMS environment, we just need to change the description of STUDENT records in the catalog to reflect the inclusion of the new data item BirthDate; no programs are chang... The next time a DBMS program refers to the catalog, the new structure of ST ENT records will be accessed and used.

NOTES

Data Item Name	Starting Position in Record	Length in Characters (bytes)
Name	31	30
StudentNumber	33	8
Class	37	8
Major	39	8

In some types of database systems, such as object-oriented and object-relational systems, users can define operations on data as part of the database definitions. An operation (also called a function or method) is specified in two parts. The interface (or signature) of an operation includes the operation name and the data types of its arguments (or parameters).

The *implementation* (or method) of the operation is specified separately and can be changed without affecting the interface. User application programs can operate on the data by invoking these operations through their names and arguments, regardless of how the operations are implemented. This may be termed program-operation independence.

The characteristic that allows program-data independence and program-operation independence is called data abstraction. A DBMS provides users with a conceptual representation of data that does not include many of the details of how the data is stored or how the operations are implemented. Informally, a data model is a type of data abstraction that is used to provide this conceptual representation.

The data model uses logical concepts, such as objects, their properties, and their interrelationships, that may be easier for most users to understand than computer storage concepts. Hence, the data model hides storage and implementation details that are not of interest to most database users.

The characteristic that allows program-data independence and program-operation independence is called data abstraction.

The internal implementation of a file may be defined by its record length—the number of characters (bytes) in each record—and each data item may be specified by its starting byte within a record and its length in bytes. The STUDENT record would thus be represented. But a typical database user is not concerned with the location of each data item within a record or its length; rather, the concern is that when a reference is made to Name of STUDENT, the correct value is returned. A *conceptual representation* of the STUDENT records is shown earlier. Many other details of file storage organization—such as the access paths specified on a file—can be hidden from database users by the DBMS.

In the database approach, the detailed structure and organization of each file are stored in the catalog. Database users and application programs refer to the conceptual representation of the files, and the DBMS extracts the details of file storage from the catalog when these are needed by the DBMS file access modules. Many data models can be used to provide this data abstraction to database users.

In object-oriented and object-relational databases, the abstraction process includes not only the data structure but also the operations on the data. These operations provide an abstraction of miniworld activities commonly understood by the users.

For example, an operation CALCULATE_GPA can be applied to a STUDENT object to calculate the grade point average. Such operations can be invoked by the user queries or application programs without having to know the details of how the operations are implemented. In that sense, an abstraction of the miniworld activity is made available to the user as an abstract operation.

NOTES

TRANSCRIPT	Name			Transcript		
		Number	Grade	Semester	Year	Section
		VB10	C	Fall	99	119
	Smith					
		VB10	B	Fall	99	112
		RD10	A	Fall	98	85
		MIC10	A	Fall	98	92
	Brown					
		RD10	B	Spring	99	102
		MIC10	A	Fall	99	135
PREREQUISITES	CName	CNumber		Pre.		
				RD10		
		Database	DB10	VB10		
		Computer Sc.	MIC10	MIC20		

Support of Multiple Views of the Data

A database typically has many users, each of whom may require a different perspective or view of the database. A view may be a subset of the database or it may contain virtual data that is derived from the database files but is not explicitly stored. Some users may not need to be aware of whether the data they refer to is stored or derived. A multiuser DBMS whose users have a variety of distinct applications must provide facilities for defining multiple views. For example, one user of the database may be interested only in accessing and printing the transcript of each student; the view for this user is shown in the above first table. A second user, who is interested only in checking that students have taken all the pre-requisites of each course for which they register, may require the view shown in the second table above.

Some users may not need to be aware of whether the data they refer to is stored or derived.

Sharing of Data and Multiuser Transaction Processing

A **multiuser DBMS**, as its name implies, must allow multiple users to access the database at the same time. This is essential if the data for multiple applications is to be integrated and maintained in a single database. The DBMS must include concurrency control software to ensure that several users trying to update the same data can do so in a controlled manner and the result of the updates is correct. For example, when several reservation clerks try to assign a seat on an airline flight, the DBMS should ensure that each seat can be accessed by only one clerk at a time for

STUDENT ACTIVITY 1.1

1. What is a database?

2. What do you understand by Multiple Views of the database?

3. Why do we need database?

4. What are Primary and Foreign Keys?

NOTES

assignment to a passenger. These types of applications are generally called OnLine Transaction Processing (OLTP) applications. A fundamental role of multiuser DBMS software is to ensure that concurrent transactions operate correctly.

The concept of a transaction has become central to many database applications. A transaction is an executing program or process that includes one or more database accesses, such as reading or updating of database records. Each transaction is supposed to execute a logically correct database access if executed in its entirety without interference from other transactions. The DBMS must enforce several transaction properties. The isolation property ensures that each transaction appears to execute in isolation from other transactions, even though hundreds of transactions may be executing concurrently. The atomicity property ensures that either all the database operations in a transaction are executed or none are.

Each transaction is supposed to execute a logically correct database access if executed in its entirety without interference from other transactions.

The preceding characteristics are most important in distinguishing a DBMS from traditional file-processing software.

TYPES OF DATABASES

The DBMS, on which the database system is based can be classified according to the number of users, the database site locations(s) and the expected type and extent of use.

The number of users determines whether the DBMS is classified as single – user or multiuser. A single-user DBMS supports only one user at a time. In other words, if user A is using the database users B and C must wait until user A has completed his/her database work. If a single-user database runs on a personal computer, it is also called a desktop database. In contrast, a multiuser DBMS supports multiple users at the same time. If the multiuser database supports a relatively small number of users (usually fewer than fifty) or a specific department within an organization, it is called a work group database. If the database is used by the entire organization, it is called a workgroup database. If the database is used by the entire organization and supports many users (more than fifty, usually hundreds) across many departments, the database is known as an enterprise database.

The database site location might also be used to classify the DBMS. For example, a DVMS that supports a database located at a single site is called a centralized DBMS. A DBMS that supports a database distributed across several different sites is called a distributed DBMS.

Perhaps the types of use and the extent of such use yield the most relevant and currently favored DBMS classification. For example, transactions such as product or service sales, payments and supply purchases reflect critical day-to-day operations. Such transactions are time-critical and must be recorded accurately and immediately – the sale of a product must be recorded and reflected in the inventory immediately.

A DBMS that powers a database primarily designed to support such "immediate response" transactions is classified as a transactional DBMS or a production DBMS. In contrast a decision support database focuses primarily on the production of

information required to make tactical or strategic decisions at middle and high management levels. Decision support, provided by a decision support system (DSS), typically, requires extensive "Data messaging" (data manipulation) to extract information from historical data to formulate pricing decisions, sales forecasts, market positioning and so on. Because most DSS information is based on historical data, the data retrieval time factor is not likely to be as critical as it is for the transactional database. Additionally, the DSS information tends to be based on complex data derived from many sources.

To make such complex data more easily retrievable, the DSS database structure is quite different from that of a transaction – oriented database. In fact, the term data warehouse is used to describe the database design favored by DSSs.

Quite clearly, properly database design requires the database designer to precisely identify the database's expected use. Designing a transactional database emphasizes data integrity, data consistency and operational speed. The design of a decision support database recognizes the use of historical and aggregated data. Designing a database to be used in centralized, single-user environment requires a different approach from that used in the design of a distributed, multiuser database.

RELATIONAL DATABASE MANAGEMENT SYSTEM

A **DataBase Management System** (DBMS) is a collection of programs that enables users to create and maintain a database. The DBMS is hence a general-purpose software system that facilitates the processes of defining, constructing, manipulating, and sharing databases among various users and applications. Defining a database involves specifying the **data types**, **structures**, and **constraints** for the data to be stored in the database.

Definition: ***A DataBase Management System (DBMS) is a collection of programs that enables users to create and maintain a database.***

Constructing the database is the process of storing the data itself on some storage medium that is controlled by the DBMS. Manipulating a database includes such functions as querying the database to retrieve specific data, updating the database to reflect changes in the miniworld, and generating reports from the data. Sharing a database allows multiple users and programs to access the database concurrently.

Other important functions provided by the DBMS include protecting the database and maintaining it over a long period of time. Protection includes both system protection against hardware or software malfunction (or crashes), and security protection against unauthorized or malicious access.

A typical large database may have a life cycle of many years, so the DBMS must be able to maintain the database system by allowing the system to evolve as requirements change over time.

It is not necessary to use general-purpose DBMS software to implement a computerized database. We could write our own set of programs to create and maintain the database, in effect creating our own specific-purpose DBMS software, in either case—whether we use a general-purpose DBMS or not—we usually have to

deploy a considerable amount of complex software. In fact, most DBMSs are very complex software systems.

RELATIONAL MODEL - KEY CONCEPT

NOTES

The database is used to store information useful to an organization. To represent this information, some means of modeling is used. The components used in modeling are limited to the objects of interest to the organization and the relationships among these objects. One category of objects of concern to any organization is its personnel, and one relationship that exists within this category of objects is that of supervisor to employees. Another area in which the definition, management, and manipulation of a considerable amount of data is required is in **computer-aided design** (CAD) and **computer-aided manufacturing** (CAM). The objecs in these applications consist of the specifications of various components and their interrelationships.

A database model is a collection of logical constructs used to represent the data structure and the data relationships found within the database. Database models can be grouped into two categories: conceptual models and implementation models.

- The conceptual model focuses on the logical nature of the data representation. Therefore, the conceptual model is concerned with what is represented in the database, rather than with how it is represented.
- In contrast to the conceptual model, an implementation model places the emphasis on how the data are represented in the database or how the data structures are implemented to represent what is modeled. Implementation models include the hierarchical database model, the network database model, the relational database model and the object-oriented database model.

File Based or Primitive Models

Entities or objects of interact are represented by records that are stored together in files. Relationships between objects are represented by using directories of various kinds.

Traditional Data Models

Traditional data models are the hierarchical, network and relational models. The hierarchical model evolved from the file-based system and the network model is a superset of the hierarchical model. The concept of data models evolved about the same models as the proposal of the relational model. You were introduced to Hierarchical and Network models in the last chapter. Relational model will be discussed in the next chapter.

The concept of data models evolved about the same models as the proposal of the relational model.

Semantic Data Models

This class of data modes was influenced by the semantic networks developed by artificial intelligence researchers. Semantic networks were developed to organize and represent general knowledge. Semantic data models are able to express greater interdependecies among entities of interest. These interdependecies consist of both

inclusion and exclusion, enabling the models to represent the semantics of the data in the database.

INTEGRITY CONSTRAINTS

NOTES

A database usually contains groups of entities that are similar. For example, a company employing hundreds of employees may want to store similar information concerning each of the employees. These employee entities share the same attributes, but each entity has its own value(s) for each attribute. An entity type defines a collection (or set) of entities that have the same attributes. Each entity type in the database is described by its name and attributes.

The following figure shows two **entity types**, named EMPLOYEE and COMPANY, and a list of attributes for each. A few individual entities of each type are also illustrated, along with the values of their attributes. The collection of all entities of a particular entity type in the database at any point in time is called an entity set; the **entity set** is usually referred to using the same name as the entity type. For example, EMPLOYEE refers to both a type of entity as well as the current set of all employee entities in the database.

EMPLOYEE	COMPANY
Name, Age, Salary	**Name, Headquarters, President**
-	-
(Sachin, 32, 5000)	**(PMG, Mumbai, Sunil)**
-	-
(Rahul, 30, 4000)	**(DNA, Bangalore, Srinath)**
-	-
(Virendra, 26, 4000)	**(TTR, New Delhi, Mohinder)**
-	-
-	-
-	-

An entity type is represented in ER diagrams as a rectangular box enclosing the entity type name. Attribute names are enclosed in ovals and are attached to their entity type by straight lines. Composite attributes are attached to their component attributes by straight lines. Multivalued attributes are displayed in double ovals.

An entity type describes the schema or intension for a set of entities that share the same structure. The collection of entities of a particular entity type are grouped into an entity set, which is also called the extension of the entity type.

Key Attributes of an Entity Type

An important constraint on the entities of an entity type is the key or uniqueness constraint on attributes. An entity type usually has an attribute whose values are distinct for each individual entity in the entity set. Such an attribute is called a **key attribute**, and its values can be used to identify each entity uniquely. For example, the

NOTES

Name attribute is a key of the COMPANY entity type in the last figure, because no two companies are allowed to have the same name.

For the PERSON, entity type, a typical key attribute is SocialSecurityNumber. Sometimes, several attributes together form a key, meaning that the combination of the attribute values must be distinct for each entity. If a set of attributes possesses this property, the proper way to represent this in the ER model, that we describe here, is to define a component attribute and designate it as attribute of the entity type.

Notice that such a composite key must be minimal; that is, all component attributes must be included in the composite attribute to have the uniqueness property. In ER diagrammatic notation, each key attribute has its name underlined inside the oval.

Specifying that an attribute is a key of an entity type means that the preceding uniqueness property must hold for every entity set of the entity type. Hence, it is a constraint that prohibits any two entities from having the same value for the key attribute at the same time. It is not the property of a particular extension; rather, it is a constraint on all extensions of the entity type. This **key constraint**, or for that even the other constraints, is derived from the constraints of the miniworld that the database represents.

It is a constraint that prohibits any two entities from having the same value for the key attribute at the same time.

Some entity types have more than one key attribute. For example, each of the Vehicle ID and Registration attributes of the entity type car (see following figure) is a key in its own right.

The Registration attribute is an example of a composite key formed from two simple component attributes, RegistrationNumber and State, neither of which is a key on its own. An entity type may also have no key, in which case it is called a weak entity type.

CAR

Registration (RegistrationNumber, State), VehcileID, Make, Model, Year, (Colour)

-

(123, DELHI), DL4CJ4759, Hundai, Santro, 2000, (Blue)

-

(123, DELHI), DL7SB7713, Bajaj, Super, 1999, (Yellow)

-

(123, DELHI), DL7SA0009, Escorts, Yamaha, 1997, (Grey)

-

Value Sets (Domains) of Attributes

Each simple attribute of an entity type is associated with a value set (or domain of values), which specifies the set of values that may be assigned to that attribute for each individual entity. In earlier figure, if the range of ages allowed for employees is

between 16 and 70, we can specify the value set of the Age attribute of EMPLOYEE to be the set of integer numbers between 16 and 70.

Similarly, we can specify the value set for the Name attribute as being the set of strings of alphabetic characters separated by blank characters, and so on. Value sets are not displayed in ER diagrams, value sets are typically specified using the basic data types available in most programming languages, such as integer, string, boolean, float, enumerated type, subrange, and so on. Additional data types to represent date, time, and other concepts are also employed.

NOTES

Mathematically, an attribute A of entity type E whose value set is V can be defined as a function from E to the power set P(V) of V

$$\mathbf{A: E \rightarrow P\ (V)}$$

We refer to the value of attribute A for entity *e* as A(*e*). The previous definition covers both single-valued and multivalued attributes, as well as nulls. A null value is represented by the empty set. For single-valued attributes, A(*e*) is restricted to being a singleton set for each entity *e* in E, whereas there is no restriction on multivalued attributes. For a composite attribute A, the value set V is the Cartesian product of $P(V_1)$, $P(V_2)$ ....., $P(V_n)$, where V_1, V_2 ....., V_n are the value sets of the simple component attributes that form A:

$$\mathbf{V = P(V_1) \times P(V_2) \times \ldots \times P(V_n)}$$

Relationship among Entities

The quest for better data management has led to several different ways of solving the file system's critical shortcomings. The resulting theoretical database constructs are represented by various database models. A database model is a collection of logical constructs used to represent the data structure and the data relationships found within the database. Database models can be grouped into two categories: *conceptual models* and *implementation models*.

- The *conceptual model* focuses on the logical nature of the data representation. Therefore, the conceptual model is concerned with what is represented in the database, rather than with how it is represented.

- In contrast to the conceptual model, an *implementation model* places the emphasis on how the data are represented in the database or on how the data structures are implemented to represent what is modeled. Implementation models include the hierarchical database model, the network database model, the relational database model and the object-oriented database model.

Conceptual models use three types of relationship to describe associations among data: one-to-many, many-to-many and one-to-one. Database designers usually use the shorthand notations 1:M, M:N, and 1:1 for them, respectively.

The following examples illustrate the distinctions among the three.

One-to-many relationship

A painter paints many different paintings, but each one of them is painted by only that painter. Thus the painter (the "one") is related to the paintings (the "many").

NOTES

Therefore, database designers label the relationship "PAINTER" paints "PAINTING" as 1:M. Similarly, a customer account (the "one") might contain many invoices, but those invoices (the "many") are related to only a single customer account. The "CUSTOMER" generates "INVOICE" relationship would also be labeled 1:M.

Many-to-many relationship

An employee might learn many job skills and each job skill might be learned by many employees. Database designers label the relationship "EMPLOYEE learns SKILL" as M:N. Similarly, a student can take many courses and each course can be taken by many students, thus yielding the M:N relationship label for the relationship expressed by "STUDENT takes COURSE."

One-to-one relationship

A retail company's management structure may require that each one of its stores be managed by a single employee. In turn, each store manager – who is an employee – only manages a single store. Therefore, the relationship "EMPLOYEE manages STORE" is labeled 1:1.

Database designers use a conceptual database model as the basis for the database blueprint.

Because each database model is evolved from its predecessors, we will examine all the different models briefly in this section. Experience has taught us that you will gain a better understanding of current database design, implementation and management issues once you have introduced to the rudiments of each database model's conceptual framework. In fact, you will discover that many of the "new" database concepts and structures bear a remarkable resembles to some of the "old" database concepts and structures.

Consider the following figure, there are several implicit relationships among the various entity types. In fact, whenever an attribute of one entity type refers to another entity type, some relationship exists. For example, the attribute Manager of DEPARTMENT refers to an employee who manages tile department; the attribute ControllingDepartment of PROJECT refers to the department that controls the project; the attribute Supervisor of EMPLOYEE refers to another employee (the one who supervises this employee); the attribute Department of EMPLOYEE refers to the department for which the employee works; and so on. In the ER model, these references should not be represented as attributes but as relationships, which are discussed here. In the initial design of entity types, relationships are typically captured in the form of attributes. As the design is refined, these attributes get converted into relationships between entity types.

Whenever an attribute of one entity type refers to another entity type, some relationship exists.

DEPARTMENT

Name, Number, {Locations}, Manager, ManagerStartDate

PROJECT

Name, Number, Location, ControllingDepartment

EMPLOYEE

Name (FName, MInit, LName), SSN, Sex, Address, Salary,

BirthDate, Department, Supervisor, {WorksOn (Project, Hours)}

NOTES

DEPENDENT

Employee, DependentName, Sex, BirthDate, Relationship

Degree of a Relationship Type

The degree of a relationship type is the number of participating entity types. Hence, the WORKS_FOR relationship is of degree two. A relationship type of degree two is called binary, and one of degree three is called ternary. An example of a ternary relationship is SUPPLY, where each relationship instance associates three entities—a supplier s, a part p, and a project j— whenever s supplies part p to project j, relationships can generally be of any degree, but the ones most common are binary relationships. Higher-degree relationships are generally more complex than binary relationships.

Relationships as Attributes

It is sometimes convenient to think of a relationship type in terms of attributes. Consider the WORKS_FOR relationship type, one can think of an attribute called Department of the EMPLOYEE entity type whose value for each employee entity is (a reference to) the department entity that the employee works or, Hence, the value set for this Department attribute is the set of all DEPARTMENT entities, which is the DEPARTMENT entity set.

However, when we think of a binary relationship as an attribute, we always have two options. Employees of the entity type DEPARTMENT whose values for each department entity is the set of employee entities who work for that department.

The value set of this Employees attribute is the power set of the EMPLOYEE entity set. Either of these two attributes—Department of EMPLOYEE or Employees of DEPARTMENT—can represent the WORKS_FOR relationship type, if both are represented, they are constrained to be inverses of each other.

Role Names and Recursive Relationships

Each entity type that participates in a relationship type plays a particular role in the relationship. The role name signifies the role that a participating entity from the entity type plays in each relationship instance, and helps to explain what the relationship means. For example, in the WORKS_FOR relationship type, EMPLOYEE plays the role of employee or worker and DEPARTMENT plays the role of department or employer.

Role names are not technically necessary in relationship types where all the participating entity types are distinct, since each participating entity type name can be used as the role name, However, in some cases the same entity type participates more than once in a relationship type in different roles. In such cases the role name becomes essential for distinguishing the meaning of each participation. Such relationship types are called recursive relationships.

FOREIGN KEY

Refer to page 4 of the chapter.

NOTES

SUMMARY

1. Any collection of related information grouped together as a single item is a database.
2. Database uses the table format of rows and columns to store the information.
3. Fields can contain any type of information, as long as each field always contains the same type of information.
4. A database is a logically coherent collection of data with some inherent meaning.
5. A database may be generated and maintained manually or it may be computerized.
6. A DataBase Management System (DBMS) is a collection of programs that enables users to create and maintain a database.
7. Manipulating a database includes such functions as querying the database to retrieve specific data, updating the database to reflect changes in the miniworld, and generating reports from the data.
8. Information from the database, stored in the form of a computerized filling system, can be stored and retrieved quite easily.
9. Manual database systems are usually not fool-proof.
10. A manual database can also be tedious to modify.
11. A computerized database is flexible.
12. In the database approach, a single system of data is maintained that is defined once and then is accessed by various users.
13. A fundamental characteristic of the database approach is that the database system contains not only the database itself but also a complete definition or description of the database structure and constraints.
14. The DBMS software must work equally well with any number of database applications.
15. DBMS access programs do not require such changes in most cases.
16. The characteristic that allows program-data independence and program-operation independence is called data abstraction.
17. In the database approach, the detailed structure and organization of each file are stored in the catalog.
18. In object-oriented and object-relational databases, the abstraction process includes not only the data structure but also the operations on the data.
19. A view may be a subset of the database or it may contain virtual data that is derived from the database files but is not explicitly stored.
20. A multiuser DBMS, as its name implies, must allow multiple users to access the database at the same time.
21. A transaction is an executing program or process that includes one or more database accesses, such as reading or updating of database records.
22. In traditional software development utilizing file processing, every user group maintains its own files for handling its data-pricing applications.
23. DBMS should have the capability to control this redundancy so as to prohibit inconsistencies among the files.
24. When multiple users share a large database, it is likely that most users will not be authorized to access all information in the database.
25. Databases can be used to provide persistent storage for program objects and data structures.

NOTES

26. Traditional database systems often suffered from the so called impedance mismatch problem.
27. In spite of the advantages of using a DBMS, there are a few situations in which such a system may involve unnecessary overhead costs that would not be incurred in traditional file processing.
28. Many persons are involved in the design, use, and maintenance of a large database with hundreds of users.
29. Database designers are responsible for identifying the data to be stored in the database and for choosing appropriate structures to represent and store this data.
30. End users are the people whose jobs require access to the database for querying, updating, and generating reports.
31. Naive end users need to learn very little about file facilities provided by the DBMS.
32. Casual users learn only a few facilities that they may use repeatedly.
33. Sophisticated users try to learn most of the DBMS facilities in order to achieve their complex requirements.
34. System analysts determine the requirements of end users, especially naive and parametric end users, and develop specifications for canned transactions that meet these requirements.
35. DBMS system designers and implementers are persons who design and implement the DBMS modules and interfaces as a software package.
36. In addition to those who design, use, and administer a database, others are associated with the design, development, and operation of the DBMS software and system environment. These persons are typically called "workers behind the scene,"

SELF ASSESSMENT QUESTIONS

1. Which is RDBMS?
2. Write a short note on Database Approach.
3. What do you understand by Multiple Views of the database?
4. What is meant by sharing of Data and Multiuser Transaction Processing?
5. What are the advantages of using RDBMS?
6. Who is Database Administrator?
7. What work is performed by Database Designers?
8. Who are Application Programmers?

Multiple Choice Questions

1. A system where you deal with more than one databases, which are linked to each other is called _____ :
   (a) Relational Database System
   (b) Database System
   (c) Database
2. Data is _____ information :
   (a) raw (b) polished (c) computer output
3. Data administrator grants you the permission to use database :
   (a) True (b) False
4. A database is a collection of _____ data :
   (a) meaningful (b) not required data (c) programs
5. Data in the database can be :
   (a) retrieved (b) updated (c) both
6. The characteristic that allows program-data independence and program-operation independence is called _____ :
   (a) data administration (b) database (c) data abstraction

NOTES

7. In the database approach, the detailed structure and organization of each file are stored in the _____ :
   (a) box (b) catalog (c) magazine
8. Traditional database systems often suffered from the so called impedance _____ problem:
   (a) mismatch (b) match (c) no match
9. _____ are the people whose jobs require access to the database :
   (a) System Analysts (b) End users (c) Programmers
10. _____ determine the requirements of end users :
   (a) Programmers (b) Users (c) System Analysts

True/False Questions

1. Database uses the table format of rows and columns to store the information.
2. A database is a logically coherent collection of data without some inherent meaning.
3. A database may be generated and maintained manually or it may be computerized.
4. Manual database systems are usually fool-proof.
5. A manual database can also be tedious to modify.
6. A computerized database is not flexible.
7. The DBMS software must work equally well with any number of database applications.
8. DBMS access programs do not require such changes in most cases.
9. A multiuser DBMS, as its name implies, must allow multiple users to access the database at the same time.
10. DBMS should have the capability to control the redundancy so as to prohibit inconsistencies among the files.
11. Databases cannot be used to provide persistent storage for program objects and data structures.
12. Traditional database systems often suffered from the so called impedance mismatch problem.
13. Many persons are involved in the design, use, and maintenance of a large database with hundreds of users.
14. Naive end users need to learn very little about file facilities provided by the DBMS.
15. Casual users learn only a few facilities that they may use repeatedly.
16. Sophisticated users try to learn most of the DBMS facilities in order to achieve their complex requirements.
17. DBMS system designers and implementers are persons who design and implement the DBMS modules and interfaces as a software package.

Short Questions with Answers

1. What is a database?

Ans. Any collection of related information grouped together as a single item is a database.

2. What are rows and columns of the database called?

Ans. Rows in a database table are called records, and columns are called fields.

3. What are the advantages of using the database?

Ans. Following are the advantages of using the database: A computerized database provides speed; finding a phone number from among a thousand entries or putting the file in alphabetical order takes just seconds with the database management system; A computerized database is compact; a database with thousands of records can be stored on a single floppy CD and; A computerized database is flexible: it has the ability to examine information from a number of angles, so you can search for a phone number by name, by address, or by pin code and then name.

4. What is data abstraction?

Ans. The characteristic that allows program-data independence and program-operation independence is called data abstraction.

5. What is multiuser database system?

Ans. A multiuser DBMS, as its name implies, must allow multiple users to access the database at the same time. This is essential if the data for multiple applications is to be integrated and maintained in a single database. The DBMS must include concurrency control software to ensure that several users trying to update the same data can do so in a controlled manner and the result of the updates is correct.

NOTES

6. How secure is the database?

Ans. A DBMS should provide a security and authorization subsystem, which the DBA uses to create accounts and to specify account restrictions.

7. What are the main disadvantages of using RDBMS?

Ans. Disadvantages of using RDBMS are: High initial investment in hardware, software, and training; The generality that a DBMS provides for defining and processing data; Overhead for providing security, concurrency control, recovery, and integrity; and Additional problems may arise if the database designers and DBA do not properly design the database or if the database systems applications are not implemented properly.

8. Who are Database designers?

Ans. Database designers are responsible for identifying the data to be stored in the database and for choosing appropriate structures to represent and store this data. Where tasks are mostly undertaken before the database is actually implemented and populated with data, it is the responsibility of database designers to communicate with all prospective database users in order to understand their requirements, and to come up with a design that meets these requirements.

9. Who are end users?

Ans. There are several categories of end users: Casual end users occasionally access the database, but they may need different information each time, They use a sophisticated database query language to specify their requests and are typically middle or high-level managers or other occasional browsers. Naive or parametric end users make up a sizable portion of database end users. Their main job function revolves around constantly querying and updating the database, using standard types of queries and updates—called canned transactions—that have been carefully programmed and tested. The tasks that such users perform are varied: Bank tellers check account balances and post withdrawals and deposits and Reservation clerks for airlines, hotels, and car rental companies check availability for a given request and make reservations.

10. Who are system analysts?

Ans. System analysts determine the requirements of end users, especially naive and parametric end users, and develop specifications for canned transactions that meet these requirements. Application programmers implement these specifications as programs; then they test, debug, document, and maintain these canned transactions. Such analysts and programmers—commonly referred to as software engineers should be familiar with the full range of capabilities provided by the DBMS to accomplish their tasks.

11. Who are the workers behind the scene?

Ans. Workers behind the scene are: DBMS system designers and implementers are persons who design and implement the DBMS modules and interfaces as a software package. A DBMS is a very complex software system that consists of many components, or modules, including modules for implementing the catalog, processing query language, processing the interface,

NOTES

accessing and buffering data, controlling concurrency, and handling data recovery and security. The DBMS must interface with other system software, such as the operating system and compilers for various programming languages. Tool developers include persons who design and implement tools—the software packages that facilitate database system design and use and that help improve performance. Tools are optional packages that are often purchased separately. They include packages for database design, performance monitoring, natural language or graphical interfaces, prototyping, simulation, and test data generation, In many cases, independent software vendors develop and market these tools. Operators and maintenance personnel are the system administration personnel who are responsible for the actual running and maintenance of the hardware and software environment for the database system.

12. Who is database administrator?

Ans. The DataBase Administrator (DBA) is responsible for authorizing access to the database, for coordinating and monitoring its use, and for acquiring software and hardware resources as needed. The DBA is accountable for problems such as breach of security or poor system response time. In large organizations, the DBA is assisted by a staff that helps carry out these functions.

ANSWERS

Multiple Choice Questions

1. a	2. a	3. a	4. a
5. c	6. c	7. b	8. a
9. b	10. c		

True False Questions

1. T	2. F	3. T	4. F
5. T	6. F	7. T	8. T
9. T	10. T	11. F	12. T
13. T	14. T	15. T	16. T
17. T			

CHAPTER 2

DATABASE DEVELOPMENT

LEARNING OBJECTIVES

After going through this chapter, you should appreciate the following:

- Development Process
- Modeling & Database Design
- Planning
- Analysis
- Design and Implementation
- E-R Methods and Diagrams
- Attributes
- Relationship
- Logical Database Design
- Normalization
- First Normal Form
- Second Normal Form
- Third Normal Form
- BCNF
- Translating E-R Diagrams to Relations
- Physical Database Design
- Relational Algebra & SQL Relational Databse Commands
- Data types
- Create Table
- Drop Table
- Alter Table
- Insert Into
- Delete From
- Update
- General Query Syntax (Select)
- Create View
- Drop View
- Set Operators - Union, Intersect, Minus
- Functions
- Group Functions
- Join
- Sub queries.

NOTES

DEVELOPMENT PROCESS

The process of doing database design generally consists of a number of steps which will be carried out by the database designer. Not all of these steps will be necessary in all cases. Usually, the designer must:

- Determine the data to be stored in the database
- Determine the relationships between the different data elements
- Superimpose a logical structure upon the data on the basis of these relationships.

Within the relational model the final step can generally be broken down into two further steps, that of determining the grouping of information within the system, generally determining what are the basic objects about which information is being stored, and then determining the relationships between these groups of information, or objects. This step is not necessary with an Object database.

The tree structure of data may enforce a hierarchical model organization, with a parent-child relationship table. An Object database will simply use a one-to-many relationship between instances of an object class. It also introduces the concept of a hierarchical relationship between object classes, termed inheritance

MODELING

In computer science, data modeling is the process of creating a data model by applying a data model theory to create a data model instance. A data model theory is a formal data model description. See database model for a list of current data model theories.

Managing large quantities of structured and unstructured data is a primary function of information systems. Data models describe structured data for storage in data management systems such as relational databases. They typically do not describe unstructured data, such as word processing documents, email messages, pictures, digital audio, and video.

Types of Data Model

A data model instance may be one of three kinds (according to ANSI in 1975[1]):

1. A conceptual schema (data model) describes the semantics of a domain, being the scope of the model. For example, it may be a model of the interest area of an organization or industry. This consists of entity classes (representing kinds of things of significance in the domain) and relationships (assertions about associations between pairs of entity classes). A conceptual schema specifies the kinds of facts or propositions that can be expressed using the model. In that sense, it defines the allowed expressions in an artificial 'language' with a scope that is limited by the scope of the model.
2. A logical schema (data model) describes the semantics, as represented by a particular data manipulation technology. This consists of descriptions of tables and columns, object oriented classes, and XML tags, among other things.
3. A physical schema (data model) describes the physical means by which data are stored. This is concerned with partitions, CPUs, tablespaces, and the like.

The significance of this approach, according to ANSI, is that it allows the three perspectives to be relatively independent of each other. Storage technology can change without affecting either the logical or the conceptual model. The table/column structure can change without (necessarily) affecting the conceptual model. In each case, of course, the structures must remain consistent with the other model. The table/column structure may be different from a direct translation of the entity classes and attributes, but it must ultimately carry out the objectives of the conceptual entity class structure. Early phases of many software.development projects emphasize the design.of a conceptual data model. Such a design can be detailed into a logical data model. In later stages, this model may be translated into physical data model. However, it is also possible to implement a conceptual model directly.

A contextual data model (list) identifies entity classes (representing things of significance to the organization).

Conceptual Schema

A conceptual data model (semantics) defines the meaning of the things in an organization. This consists relationships (assertions about associations between pairs of entity classes).

Logical Schema

Logical schema|logical data model (schema) describes the logic representation of the properties without regard to a particular data manipulation technology. This consists of descriptions of the attributes (role a data element plays in relation to the thing (entity) it represents.

Physical Schema

Physical schema|physical data model (blueprint) describes the physical means by which data are stored. This is concerned with partitions, CPUs, tablespaces, and the like.

A data definition (configuration) This is the actual language coding of the database schema in the chosen development platform.

A data instantiation holds the values of properties applied to the data in the schema.

The significance of this approach is that it allows the six perspectives to be relatively independent of each other and have different contributors, audiences and purposes. In each case, of course, the structures must remain consistent with the other model instances although the details change. The table/column structure may be different from a direct translation of the entity classes, relationships and attributes, but it must ultimately carry out the objectives of the contextual entity class structure and conceptual relationship structure. Each perspective a separate and distinct vantage point of the data: his view is not a methodology but rather a way of classifying the parts, however development projects and software tools often proceed from Contextual list, to conceptual data model, followed by the Logical schema|logical data model. In later stages when the data platform is known (whether it be database software or filing cabinets), this model may be translated into a Physical schema|physical data model followed by the data definition. When the database actually stores values and is operational data manipulation can take place.

NOTES

Data structure

A data model describes the structure of the data within a given domain and, by implication, the underlying structure of that domain itself. This means that a data model in fact specifies a dedicated grammar for a dedicated artificial language for that domain.

A data model represents classes of entities (kinds of things) about which a company wishes to hold information, the attributes of that information, and relationships among those entities and (often implicit) relationships among those attributes. The model describes the organization of the data to some extent irrespective of how data might be represented in a computer system.

The entities represented by a data model can be the tangible entities, but models that include such concrete entity classes tend to change over time. Robust data models often identify abstractions of such entities. For example, a data model might include an entity class called "Person", representing all the people who interact with an organization. Such an abstract entity class is typically more appropriate than ones called "Vendor" or "Employee", which identify specific roles played by those people.

When designing a data model it is useful[citation needed] to make a distinction between transaction data and reference data, where the transaction data refers to one or more entities of reference data.

A proper conceptual data model describes the semantics of a subject area. It is a collection of assertions about the nature of the information that is used by one or more organizations. Proper entity classes are named with natural language words instead of technical jargon. Likewise, properly named relationships form concrete assertions about the subject area.

There are several versions of this. For example, a relationship called "is composed of" that is defined to operate on entity classes ORDER and LINE ITEM forms the following concrete assertion definition: Each ORDER "is composed of" one or more LINE ITEMS." A more rigorous approach is to force all relationship names to be prepositions, gerunds, or participles, with verbs being simply "must be" or "may be". This way, both cardinality and optionality can be handled semantically. This would mean that the relationship just cited would read in one direction, "Each ORDER may be composed of one or more LINE ITEMS" and in the other "Each LINE ITEM must be part of one and only one ORDER."

Note that this illustrates that often generic terms, such as 'is composed of', are defined to be limited in their use for a relationship between specific kinds of things, such as an order and an order line. This constraint is eliminated in the generic data modeling methodologies.

Generic data model

Generic data models are generalizations of conventional data models. They define standardised general relation types, together with the kinds of things that may be related by such a relation type. This is similar to the definition of a natural language. For example, a generic data model may define relation types such as a 'classification relation', being a binary relation between an individual thing and a kind of thing (a class) and a 'part-whole relation', being a binary relation between two things, one with the role of part, the other with the role of whole, regardless the kind of things that are

NOTES

related. Given an extensible list of classes, this allows the classification of any individual thing and to specify part-whole relations for any individual object. By standardisation of an extensible list of relation types, a generic data model enables the expression of an unlimited number of kinds of facts and will approach the capabilities of natural languages. Conventional data models, on the other hand, have a fixed and limited domain scope, because the instantiation (usage) of such a model only allows expressions of kinds of facts that are predefined in the model.

Generic data models are developed as an approach to solve some shortcomings of conventional data models. For example, different modelers usually produce different conventional data models of the same domain. This can lead to difficulty in bringing the models of different people together and is an obstacle for data exchange and data integration. Invariably, however, this difference is attributable to different levels of abstraction in the models and differences in the kinds of facts that can be instantiated (the semantic expression capabilities of the models). The modelers need to communicate and agree on certain elements which are to be rendered more concretely, in order to make the differences less significant.

There are generic patterns that can be used to advantage for modeling business. These include entity types for PARTY (with included PERSON and ORGANIZATION), PRODUCT TYPE, PRODUCT INSTANCE, ACTIVITY TYPE, ACTIVITY INSTANCE, CONTRACT, GEOGRAPHIC AREA, and SITE. A model which explicitly includes versions of these entity classes will be both reasonably robust and reasonably easy to understand.

More abstract models are suitable for general purpose tools, and consist of variations on THING and THING TYPE, with all actual data being instances of these. Such abstract models are on one hand more difficult to manage, since they are not very expressive of real world things, but on the other hand they have a much wider applicability, especially if they are accompanied by a standardised dictionary. More concrete and specific data models will risk having to change as the scope or environment changes.

One approach to generic data modeling has the following characteristics:

1. A generic data model shall consist of generic entity types, such as 'individual thing', 'class', 'relationship', and possibly a number of their subtypes.
2. Every individual thing is an instance of a generic entity called 'individual thing' or one of its subtypes.
3. Every individual thing is explicitly classified by a kind of thing ('class') using an explicit classification relationship.

The classes used for that classification are separately defined as standard instances of the entity 'class' or one of its subtypes, such as 'class of relationship'. These standard classes are usually called 'reference data'. This means that domain specific knowledge is captured in those standard instances and not as entity types. For example, concepts such as car, wheel, building, ship, and also temperature, length, etc. are standard instances. But also standard types of relationship, such as 'is composed of' and 'is involved in' can be defined as standard instances.

This way of modeling allows the addition of standard classes and standard relation types as data (instances), which makes the data model flexible and prevents data model changes when the scope of the application changes.

NOTES

A generic data model obeys the following rules:

- Candidate attributes are treated as representing relationships to other entity types.
- Entity types are represented, and are named after, the underlying nature of a thing, not the role it plays in a particular context. Entity types are chosen.
- Entities have a local identifier within a database or exchange file. These should be artificial and managed to be unique. Relationships are not used as part of the local identifier.
- Activities, relationships and event-effects are represented by entity types (not attributes).
- Entity types are part of a sub-type/super-type hierarchy of entity types, in order to define a universal context for the model. As types of relationships are also entity types, they are also arranged in a sub-type/super-type hierarchy of types of relationship.
- Types of relationships are defined on a high (generic) level, being the highest level where the type of relationship is still valid. For example, a composition relationship (indicated by the phrase: 'is composed of') is defined as a relationship between an 'individual thing' and another 'individual thing' (and not just between e.g. an order and an order line). This generic level means that the type of relation may in principle be applied between any individual thing and any other individual thing. Additional constraints are defined in the 'reference data', being standard instances of relationships between kinds of things.

Data organization

Another kind of data model describes how to organize data using a database management system or other data management technology. It describes, for example, relational tables and columns or object-oriented classes and attributes. Such a data model is sometimes referred to as the physical data model, but in the original ANSI three schema architecture, it is called "logical". In that architecture, the physical model describes the storage media (cylinders, tracks, and tablespaces). Ideally, this model is derived from the more conceptual data model described above. It may differ, however, to account for constraints like processing capacity and usage patterns.

While data analysis is a common term for data modeling, the activity actually has more in common with the ideas and methods of synthesis (inferring general concepts from particular instances) than it does with analysis (identifying component concepts from more general ones). {Presumably we call ourselves systems analysts because no one can say systems synthesists.} Data modeling strives to bring the data structures of interest together into a cohesive, inseparable, whole by eliminating unnecessary data redundancies and by relating data structures with relationships.

A different approach is through the use of adaptive systems such as artificial neural networks that can autonomously create implicit models of data.

DATABASE DESIGN

Database design is the process of producing a detailed data model of a database. This

logical data model contains all the needed logical and physical design choices and physical storage parameters needed to generate a design in a Data Definition Language, which can then be used to create a database. A fully attributed data model contains detailed attributes for each entity.

NOTES

The term database design can be used to describe many different parts of the design of an overall database system. Principally, and most correctly, it can be thought of as the logical design of the base data structures used to store the data. In the relational model these are the tables and views. In an Object database the entities and relationships map directly to object classes and named relationships. However, the term database design could also be used to apply to the overall process of designing, not just the base data structures, but also the forms and queries used as part of the overall database application within the Database Management System or DBMS.

Design process

This was discussed in the first paragraph of the chapter.

Determining data to be stored

In a majority of cases, the person who is doing the design of a database is a person with expertise in the area of database design, rather than expertise in the domain from which the data to be stored is drawn e.g. financial information, biological information etc. Therefore the data to be stored in the database must be determined in cooperation with a person who does have expertise in that domain, and who is aware of what data must be stored within the system.

This process is one which is generally considered part of requirements analysis, and requires skill on the part of the database designer to elicit the needed information from those with the domain knowledge. This is because those with the necessary domain knowledge frequently cannot express clearly what their system requirements for the database are as they are unaccustomed to thinking in terms of the discrete data elements which must be stored. Data to be stored can be determined by Requirement Specification.

Conceptual schema

Once a database designer is aware of the data which is to be stored within the database, they must then determine how the various pieces of that data relate to one another. When performing this step, the designer is generally looking out for the dependencies in the data, where one piece of information is dependent upon another, i.e., when one piece of information changes, the other will also. For example, in a list of names and addresses, assuming the normal situation where two people can have the same address, but one person cannot have two addresses, the name is dependent upon the address, because if the address is different then the associated name is different too. However, the inverse is not necessarily true, i.e. when the name changes address may be the same.

A common misconception is that the relational model is so called because of the stating of relationships between data elements therein. This is not true. The relational model is so named such because it is based upon the mathematical structures known as relations.

NOTES

Logically structuring data

Once the relationships and dependencies amongst the various pieces of information have been determined, it is possible to arrange the data into a logical structure which can then be mapped into the storage objects supported by the database management system. In the case of relational databases the storage objects are tables which store data in rows and columns.

Each table may represent an implementation of either a logical object or a relationship joining one or more instances of one or more logical objects. Relationships between tables may then be stored as links connecting child tables with parents. Since complex logical relationships are themselves tables they will probably have links to more than one parent.

In an Object database the storage objects correspond directly to the objects used by the Object-oriented programming language used to write the applications that will manage and access the data. The relationships may be defined as attributes of the object classes involved or as methods that operate on the object classes.

Physical database design

The physical design of the database specifies the physical configuration of the database on the storage media. This includes detailed specification of data elements, data types, indexing options, and other parameters residing in the DBMS data dictionary.

PLANNING

Planning in organizations and public policy is both the organizational process of creating and maintaining a plan; and the psychological process of thinking about the activities required to create a desired future on some scale. As such, it is a fundamental property of intelligent behavior. This thought process is essential to the creation and refinement of a plan, or integration of it with other plans, that is, it combines forecasting of developments with the preparation of scenarios of how to react to them.

The term is also used to describe the formal procedures used in such an endeavor, such as the creation of documents, diagrams, or meetings to discuss the important issues to be addressed, the objectives to be met, and the strategy to be followed. Beyond this, planning has a different meaning depending on the political or economic context in which it is used.

Two attitudes to planning need to be held in tension: on the one hand we need to be prepared for what may lie ahead, which may mean contingencies and flexible processes. On the other hand, our future is shaped by consequences of our own planning and actions.

What should a plan be?

A plan should be a realistic view of the expectations. Depending upon the activities, a plan can be long range, intermediate range or short range. It is the framework within which it must operate. For management seeking external support, the plan is the most important document and key to growth. Preparation of a comprehensive plan will not guarantee success, but lack of a sound plan will almost certainly ensure failure.

Purpose of Plan

Just as no two organizations are alike, so also their plans. It is therefore important to prepare a plan keeping in view the necessities of the enterprise. A plan is an important aspect of business. It serves the following three critical functions: Helps management to clarify, focus, and research their business's or project's development and prospects. Provides a considered and logical framework within which a business can develop and pursue business strategies over the next three to five years. Offers a benchmark against which actual performance can be measured and reviewed.

Importance of the planning Process

A plan can play a vital role in helping to avoid mistakes or recognize hidden opportunities. Preparing a satisfactory plan of the organization is essential. The planning process enables management to understand more clearly what they want to achieve, and how and when they can do it.

A well-prepared business plan demonstrates that the managers know the business and that they have thought through its development in terms of products, management, finances, and most importantly, markets and competition.

Planning helps in forecasting the future, makes the future visible to some extent. It bridges between where we are and where we want to go. Planning is looking ahead.

ANALYSIS

The Analysis phase performs three tasks:

1. It determines the point in the log at which to start the Redo pass.
2. It determines pages in the buffer pool that were dirty at the time of the crash.
3. It identifies transactions that were active at the time of the crash and must be undone.

Analysis begin by examining the most recent begin_checking log record and initializing the dirty page table and transaction table to the copies of those structures in the next end_checkpoint record. Thus, these tables are initialized to the set of dirty pages and active transactions at the time of the checkpoint.

Analysis then scans the log in the forward direction until it reaches the end of the log.

- If an end log record for a transaction T is encountered, T is removed from the transaction table because it is no longer active.
- If a log record other than an end record for a transaction T is encountered, an entry for T is added to the transaction table if it is not already there. Further, the entry for T is modified.
  1. The lastLSN field is set to the LSN of this log record.
  2. If the log record is a commit record, the status is set to C, otherwise it is set to U (indicating that it is to be undone).
- If a redoable log record affecting page P is encountered, and P is not in the dirty page table, an entry is inserted into this table with page id P and recLSN equal to the LSN of this reloadable log record. This LSN identifies the oldest change affecting page P that may not have been written to the disk.

NOTES

At the the end of the Analysis phase, the transaction table contains an accurate list of all transactions that were active at the time of the crash—this is the set of transactions with status U. The dirty page table includes all pages that were dirty at the time of the crash but may also contain some pages that were written to disk. If an end_write log record were written at the completion of each write operation, the dirty page table constructed during Analysis could be made more accurate.

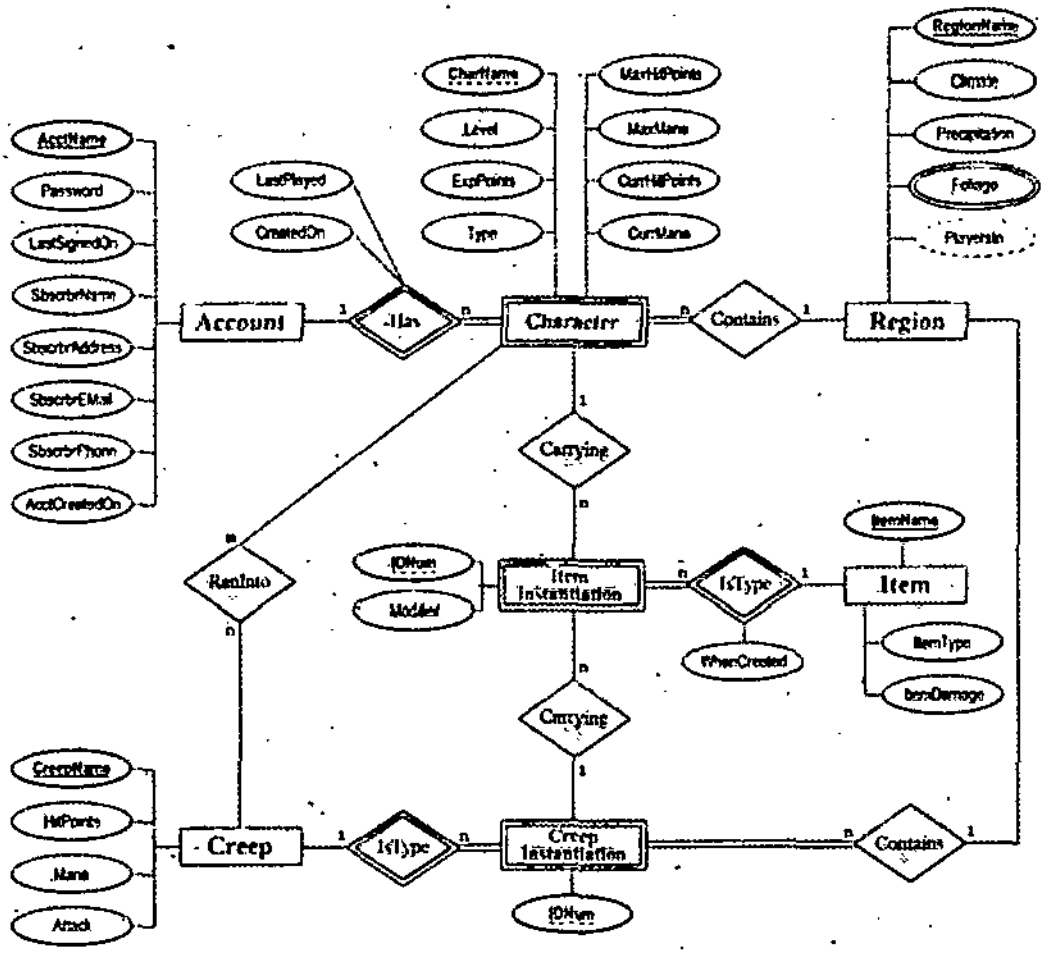
DESIGN AND IMPLEMENTATION

For this refer to Database Design refere to earlier pages of the chapter.

E-R METHODS AND DIAGRAMS

An entity-relationship (ER) diagram is a specialized graphic that illustrates the interrelationships between entities in a database. ER diagrams often use symbols to represent three different types of information. Boxes are commonly used to represent entities. Diamonds are normally used to represent relationships and ovals are used to represent attributes.

An Entity-Relationship Model (ERM) in software engineering is an abstract and conceptual representation of data. Entity-relationship modeling is a relational schema



database modeling method, used to produce a type of conceptual schema or semantic data model of a system, often a relational database, and its requirements in a top-down fashion.

Diagrams created using this process are called entity-relationship diagrams, or ER diagrams or ERDs for short.

NOTES

Overview

The first stage of information system design uses these models during the requirements analysis to describe information needs or the type of information that is to be stored in a database. The data modeling technique can be used to describe any ontology (i.e. an overview and classifications of used terms and their relationships) for a certain universe of discourse (i.e. area of interest). In the case of the design of an information system that is based on a database, the conceptual data model is, at a later stage (usually called logical design), mapped to a logical data model, such as the relational model; this in turn is mapped to a physical model during physical design. Note that sometimes, both of these phases are referred to as "physical design".

There are a number of conventions for entity-relationship diagrams (ERDs). The classical notation is described in the remainder of this article, and mainly relates to conceptual modeling. There are a range of notations more typically employed in logical and physical database design, such as IDEF1X.

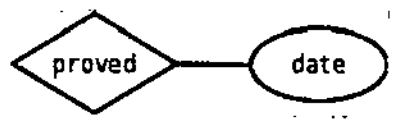
Connection

An entity may be defined as a thing which is recognized as being capable of an independent existence and which can be uniquely identified. An entity is an abstraction from the complexities of some domain. When we speak of an entity we normally speak of some aspect of the real world which can be distinguished from other aspects of the real world.

Two related entities

An entity may be a physical object such as a house or a car, an event such as a house sale or a car service, or a concept such as a customer transaction or order. Although the term entity is the one most commonly used, following Chen we should really distinguish between an entity and an entity-type. An entity-type is a category. An entity, strictly speaking, is an instance of a given entity-type. There are usually many instances of an entity-type. Because the term entity-type is somewhat cumbersome, most people tend to use the term entity as a synonym for this term.

An entity with an attribute



A relationship with an attribute

Entities can be thought of as nouns. Examples: a computer, an employee, a song, a mathematical theorem. Entities are represented as rectangles.

A relationship captures how two or more entities are related to one another. Relationships can be thought of as verbs, linking two or more nouns. Examples: an owns relationship between a

Primary key

NOTES

company and a computer, a supervises relationship between an employee and a department, a performs relationship between an artist and a song, a proved relationship between a mathematician and a theorem. Relationships are represented as diamonds, connected by lines to each of the entities in the relationship.

Entities and relationships can both have attributes. Examples: an employee entity might have a Social Security Number (SSN) attribute; the proved relationship may have a date attribute. Attributes are represented as ellipses connected to their owning entity sets by a line.

Every entity (unless it is a weak entity) must have a minimal set of uniquely identifying attributes, which is called the entity's primary key.

Entity-relationship diagrams don't show single entities or single instances of relations. Rather, they show entity sets and relationship sets. Example: a particular song is an entity. The collection of all songs in a database is an entity set. The eaten relationship between a child and her lunch is a single relationship. The set of all such child-lunch relationships in a database is a relationship set.

Lines are drawn between entity sets and the relationship sets they are involved in. If all entities in an entity set must participate in the relationship set, a thick or double line is drawn. This is called a participation constraint. If each entity of the entity set can participate in at most one relationship in the relationship set, an arrow is drawn from the entity set to the relationship set. This is called a key constraint. To indicate that each entity in the entity set is involved in exactly one relationship, a thick arrow is drawn.

ER diagramming tools

There are many ER diagramming tools. Some of the proprietary ER diagramming tools are Avolution, ConceptDraw, ER/Studio, ERwin, DeZign for Databases, MEGA International, OmniGraffle, Oracle Designer, PowerDesigner, Rational Rose, SmartDraw, Sparx Enterprise Architect, SQLyog, Toad Data Modeler, Microsoft Visio, and Visual Paradigm. A freeware ER tool that can generate database and application layer code (webservices) is the RISE Editor.

Some free software ER diagramming tools that can interpret and generate ER models, SQL and do database analysis are StarUML, MySQL Workbench, and SchemaSpy. Some free software diagram tools which can't create ER diagrams but just draw the shapes without having any knowledge of what they mean or generating SQL are Kivio, Dia. Although DIA diagrams can be translated with tedia2sql.

E-R Diagram Example

Example 1

A publishing company produces scientific books on various subjects. The books are written by authors who specialize in one particular subject. The company employs editors who, not necessarily being specialists in a particular area, each take sole responsibility for editing one or more publications. A publication covers essentially one of the specialist subjects and is normally written by a single author. When writing a particular book, each author works with on editor, but may submit another work for publication to be supervised by other editors. To improve their competitiveness, the company tries to employ a variety of authors, more than one author being a specialist in a particular subject.

NOTES

Example 2

A General Hospital consists of a number of specialized wards (such as Maternity, Paediatry, Oncology, etc). Each ward hosts a number of patients, who were admitted on the recommendation of their own GP and confirmed by a consultant employed by the Hospital.

On admission, the personal details of every patient are recorded. A separate register is to be held to store the information of the tests undertaken and the results of a prescribed treatment. A number of tests may be conducted for each patient. Each patient is assigned to one leading consultant but may be examined by another doctor, if required. Doctors are specialists in some branch of medicine and may be leading consultants for a number of patients, not necessarily from the same ward.

Example 3

A database is to be designed for a Car Rental Co. (CRC). The information required includes a description of cars, subcontractors (i.e. garages), company expenditures, company revenues and customers. Cars are to be described by such data as: make, model, year of production, engine size, fuel type, number of passengers, registration number, purchase price, purchase date, rent price and insurance details. It is the company policy not to keep any car for a period exceeding one year.

All major repairs and maintenance are done by subcontractors (i.e. franchised garages), with whom CRC has long-term agreements. Therefore the data about garages to be kept in the database includes garage names, addressees, range of services and the like. Some garages require payments immediately after a repair has been made; with others CRC has made arrangements for credit facilities. Company expenditures are to be registered for all outgoings connected with purchases, repairs, maintenance, insurance etc. Similarly the cash inflow coming from all sources - car hire, car sales, insurance claims - must be kept of file.

CRC maintains a reasonably stable client base. For this privileged category of customers special credit card facilities are provided. These customers may also book in advance a particular car. These reservations can be made for any period of time up to one month.

Casual customers must pay a deposit for an estimated time of rental, unless they wish to pay by credit card. All major credit cards care accepted. Personal details (such as name, address, telephone number, driving licence, number) about each customer are kept in the database.

Example 4

A database is to be designed for a college to monitor students' progress throughout their course of study. The students are reading for a degree (such as BA, BA(Hons) MSc, etc) within the framework of the modular system. The college provides a number of module, each being characterised by its code , title, credit value, module leader, teaching staff and the department they come from.

A module is co-ordinated by a module leader who shares teaching duties with one or more lecturers. A lecturer may teach (and be a module leader for) more than one module. Students are free to choose any module they wish but the following rules must be observed: some modules require pre-requisites modules and some degree programmes have compulsory modules. The database is also to contain some

NOTES

information about students including their numbers, names, addresses, degrees they read for, and their past performance (i.e. modules taken and examination results).

Example 5

A relational database is to be designed for a medium-sized Company dealing with industrial applications of computers. The Company delivers various products to its customers ranging from a single application program through to complete installation of hardware with customized software. The Company employs various experts, consultants and supporting staff. All personnel are employed on long-term basis, i.e. there are no short-term or temporary staff.

Although the Company is somehow structured for administrative purposes (that is, it is divided into departments headed by department managers) all projects are carried out in an inter-disciplinary way. For each project a project team is selected, grouping employees from different departments, and a Project Manager (also an employee of the Company) is appointed who is entirely and exclusively responsible for the control of the project, quite independently of the Company's hierarchy. The following is a brief statement of some facts and policies adopted by the Company.

There are a variety of methods by which data is merged onto a network, a concept referred to as the media access method. The media access method used depends on the way in which a particular technology such as Ethernet or Token Ring communicates. This section will look at the three most popular methods – contention-based, token passing, and polling.

Contention

Contention-based media access describes a way of getting data on to the network whereby systems 'contend for' or share the media. On a contention-based network, systems can only transmit when the media is free and clear of signals. In this way, devices listen to the media, and if no other system is transmitting, they can go ahead and send data. In cases where more than one system finds the network free and attempts to transmit, a data collision will occur, and systems will need to retransmit. On busy networks, the number of collisions can quickly get very high, adversely affecting performance. Remember that in this scenario, only a single system truly has access to the media at any given time, even though multiple systems may have data to send.

The best example of a contention-based network technology is Ethernet, which uses a scheme called Carrier Sense Multiple Access with Collision Detection (CSMA/CD). The fact that Ethernet is contention-based is a reason why many people thought that the technology would never be a good solution for large networks. As time passed, different techniques were developed to provide a way for contention-based networks to scale to larger sizes. A great example is the use of switches to segment a network, thus significantly reducing (or even eliminating) collisions.

Token Passing

A more orderly scheme for moving data between network systems is found when token passing is used. In token-passing media access environments, a special frame referred to as a token repeatedly circles the network, passed from system to system. If a system has control of the token, it can transmit data. If it doesn't, it must wait for the token to become available again.

While this might sound like a very slow way to go about passing data, it's important to understand that the token moves around the network at incredibly high speeds. Understand also that because this method isn't contention based, there won't be any collisions, further increasing performance

Examples of technologies that use token-passing media access include Token Ring and Fiber Distributed Data Interface (FDDI), both of which will be described in detail later in this chapter.

NOTES

Polling

While contention and token-passing methods are by far the most popular ways in which PCs access LAN media, some technologies rely on a technique called polling. Polling is a deterministic way of allowing systems access to the network while also avoiding collisions. When used, a device referred to as the master polls systems to see if they have data to transmit.

In this way, polling is similar to token passing, except that the central device controls the order in which systems are contacted. The downside of polling is that when the master device fails, the network fails. Most popular in mainframe and minicomputer environments, polling is a technique used in protocols such as Synchronous Data Link Control (SDLC).

The **entity-relationship data model** (E R Model) grew out of exercise of using commercially available DBMSs to model application databases. Earlier commercial systems were based on the hierarchical and network approach. The entity-relationship model is a generalization of these models. It allows the representation of explicit constraints as well as relationships.

Even though the E-R model has some means of describing the physical database model, it is basically useful in the design and communication of the logical database model. In this model, objects of similar structures are collected into an entity set. The relationship between entity sets is represented by a named E-R relationship and is 1:1, 1:M and M:N, mapping from one entity set to another. The database structure employing the E-R model is usually shown using the entity-relationship (E-R) diagrams.

Conceptual modeling is a very important phase in designing a successful database application. Generally, the term database application refers to a particular database and the associated programs that implement the database queries and updates. For example, a BANK database application that keeps track of customer accounts would include programs that implement database updates corresponding to customers making deposits and withdrawals.

These programs provide user-friendly graphical user interfaces (GUIs) utilizing forms and menus for the end users of the application—the bank tellers, in this example. Hence, part of the database application will require the design, implementation, and testing of these application programs. Traditionally, the design and testing of application programs has been considered to be more in the realm of the software engineering domain than in the database domain.

As database design methodologies include more of the concepts for specifying operations on database objects, and as software engineering methodologies specify in more detail the structure of the databases that software programs will use and access, it is clear that these activities are strongly related.

Traditionally, the design and testing of application programs has been considered to be more in the realm of the software engineering domain than in the database domain.

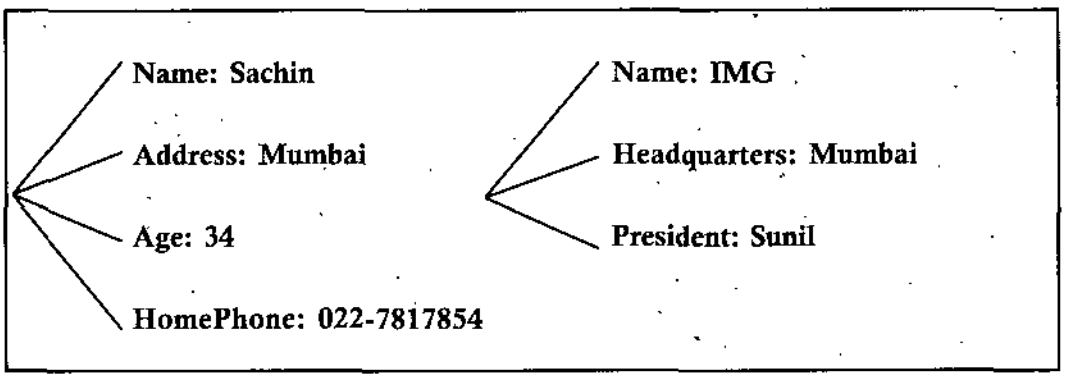
NOTES

ATTRIBUTES AND ENTETIES

The basic object that the ER model represents is an entity, which is a "thing" in the real world with an independent existence. An entity may be an object with a physical existence (for example, a particular person, car, house, or employee) or it may be an object with a conceptual existence (for example, a company, a job, or a university course). Each entity has attributes—the particular properties that describes it. For example, an employee entity may be described by the employee's name, age, address, salary, and job. A particular entity will have a value for each of its attributes. The attribute values that describe each entity become a major part of the data stored in the database.

Each entity has attributes—the particular properties that describes it.

The following figure shows two entities and the values of their attributes. The employee entity e_1 has four attributes: Name, Address, Age, and HomePhone; their values are "Sachin," "Mumbai," "34," and "022-7817854," respectively. The company entity c_1 has three attributes: Name, Headquarters, and President; their values are "IMG," "Mumbai," and "Sunil," respectively.



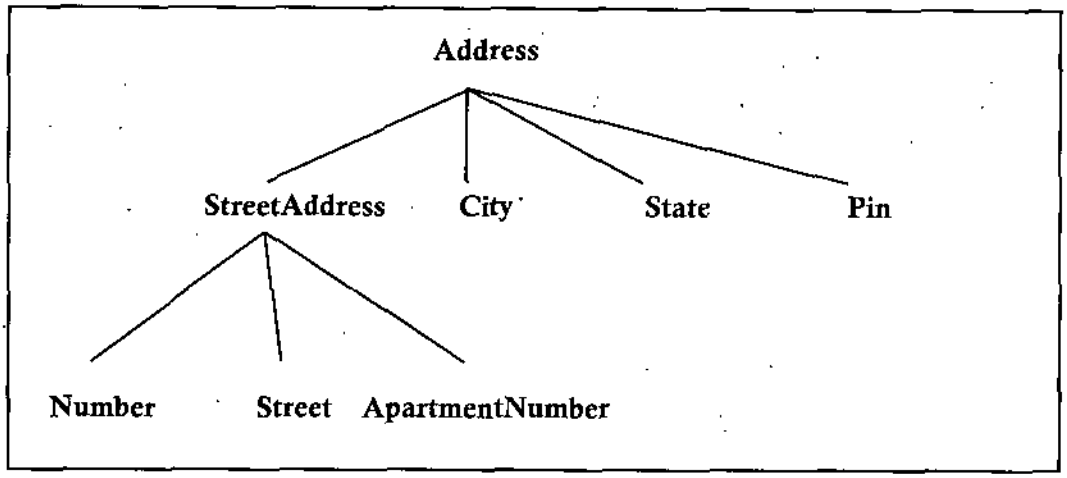
Several types of attributes occur in the ER model: simple versus composite, single-valued versus multivalued, and stored versus derived. Let us first define these attribute types and illustrate their use via examples. We will then introduce the concept of a null value for an attribute.

Composite versus Simple (Atomic) Attributes

Composite attributes can be divided into smaller subparts, which represent more basic attributes with independent meanings. For example, the Address attribute of the employee entity shown in the following figure, can be subdivided into StreetAddress, City, State, and Pin, with the values "2396 Dalal Street," "Mumbai," "Maharashtra," and "400001." Attributes that are not divisible are called simple or atomic attributes.

Composite attributes can form a hierarchy; for example, StreetAddress can be further subdivided into three simple attributes: Number, Street, and ApartmentNumber, in

the figure shown. The value of a composite attribute is the concatenation of tile values of its constituent simple attributes.



Composite attributes are useful to model situations in which a user sometimes refers to the composite attribute as a unit but at other times refers specifically to its components. If the composite attribute is referenced only as a whole, there is no need to subdivide it into component attributes. For example, if there is no need to refer to the individual components of an address (pin code, street, and so on), then the whole address can be designated as a simple attribute.

Single-Valued versus Multivalued Attributes

Most attributes have a single value for a particular entity; such attributes are called single-valued. For example, Age is a **single-valued attribute** of a person. In some cases an attribute can have a set of values for the same entity—for example, a Colours attribute for a car, or a CollegeDegrees attribute for a person. Cars with one colour have a single value, whereas two-tone cars have two values for Colours.

Similarly, one person may not have a college degree, another person may have one, and a third person may have two or more degrees; therefore, different persons can have different numbers of values for the CollegeDegrees attribute. Such attributes are called multivalued. A **multivalued** attribute may have lower and upper bounds to constrain the number of values allowed for each individual entity. For example, the Colours attribute of a car may have between one and three values, if we assume that a car can have at most three colours.

A multivalued attribute may have lower and upper bounds to constrain the number of values allowed for each individual entity.

tored versus Derived Attributes

n some cases, two (or more) attribute values are related—for example, the Age and BirthDate attributes of a person. For a particular person entity, the value of Age can e determined from the current (today's) date and the value of that person's BirthDate. The Age attribute is hence called a derived attribute and is said to be derivable from ne BirthDate attribute, which is called a stored attribute.

ome attribute values can be derived from related entities; for example, an attribute NumberOfEmployees of a department entity can be derived by counting the number f employees related to (working for) that department.

NOTES

Null Values

In some cases a particular entity may not have an applicable value for an attribute. For example, the ApartmentNumber attribute of an address applies only to addresses that are in apartment buildings and not to other types of residences, such as single-family homes. Similarly a CollegeDegrees attribute applies only to persons with college degrees. For such situations, a special value called null is created.

An address of a single-family home would have null for its ApartmentNumber attribute, and a person with no college degree would have null for CollegeDegrees. Null can also be used if we do not know the value of an attribute for a particular entity—for example, if we do not know the home phone of "Sachin". The meaning of the former type of null is not applicable, whereas the meaning of the latter is unknown. The "unknown" category of null can be further classified into two cases.

The first case arises when it is known that the attribute value exists but is missing—for example, if the Height attribute of a person is listed as null. The second case arises when it is not known whether the attribute value exists—for example, if the HomePhone attribute of a person is null.

Null can also be used if we do not know the value of an attribute for a particular entity.

Complex Attributes

Notice that composite and multivalued attributes can be nested in an arbitrary way. We can represent arbitrary nesting by grouping components of a composite attribute between parentheses ( ) and separating the components with commas, and by displaying multivalued attributes between braces { }. Such attributes are called complex attributes.

For example, if a person can have more than one residence and each residence can have multiple phones, an attribute AddressPhone for a person can be specified as shown below.

{AddressPhone({phone(AreaCode,PhoneNumber)},
{Address(StreetAddress(Number,Street,ApartmentNumber),
City,State,Zip)) }

RELATIONSHIP

The quest for better data management has led to several different ways of solving the file system's critical shortcomings. The resulting theoretical database constructs are represented by various database models. A database model is a collection of logical constructs used to represent the data structure and the data relationships found within the database. Database models can be grouped into two categories: *conceptual models* and *implementation models*.

- The *conceptual model* focuses on the logical nature of the data representation. Therefore, the conceptual model is concerned with what is represented in the database, rather than with how it is represented.

- In contrast to the conceptual model, an *implementation model* places the emphasis on how the data are represented in the database or on how the data structures are implemented to represent what is modeled. Implementation models include the hierarchical database model, the network database model, the relational database model and the object-oriented database model.

Conceptual models use three types of relationship to describe associations among data: one-to-many, many-to-many and one-to-one. Database designers usually use the shorthand notations 1:M, M:N, and 1:1 for them, respectively.

The following examples illustrate the distinctions among the three.

One-to-many relationship

A painter paints many different paintings, but each one of them is painted by only that painter. Thus the painter (the "one") is related to the paintings (the "many"). Therefore, database designers label the relationship "PAINTER" paints "PAINTING" as 1:M. Similarly, a customer account (the "one") might contain many invoices, but those invoices (the "many") are related to only a single customer account. The "CUSTOMER" generates "INVOICE" relationship would also be labeled 1:M.

Many-to-many relationship

An employee might learn many job skills and each job skill might be learned by many employees. Database designers label the relationship "EMPLOYEE learns SKILL" as M:N. Similarly, a student can take many courses and each course can be taken by many students, thus yielding the M:N relationship label for the relationship expressed by "STUDENT takes COURSE."

One-to-one relationship

A retail company's management structure may require that each one of its stores be managed by a single employee. In turn, each store manager – who is an employee – only manages a single store. Therefore, the relationship "EMPLOYEE manages STORE" is labeled 1:1.

Database designers use a conceptual database model as the basis for the database blueprint.

Because each database model is evolved from its predecessors, we will examine all the different models briefly in this section. Experience has taught us that you will gain a better understanding of current database design, implementation and management issues once you have introduced to the rudiments of each database model's conceptual framework. In fact, you will discover that many of the "new" database concepts and structures bear a remarkable resembles to some of the "old" database concepts and structures.

Consider the following figure, there are several implicit relationships among the various entity types. In fact, whenever an attribute of one entity type refers to another entity type, some relationship exists. For example, the attribute Manager of DEPARTMENT refers to an employee who manages tile department; the attribute ControllingDepartment of PROJECT refers to the department that controls the project; the attribute Supervisor of EMPLOYEE refers to another employee (the one who supervises this employee); the attribute Department of EMPLOYEE refers

NOTES

to the department for which the employee works; and so on. In the ER model, these references should not be represented as attributes but as relationships, which are discussed here. In the initial design of entity types, relationships are typically captured in the form of attributes. As the design is refined, these attributes get converted into relationships between entity types.

Whenever an attribute of one entity type refers to another entity type, some relationship exists.

DEPARTMENT

Name, Number, {Locations}, Manager, ManagerStartDate

PROJECT

Name, Number, Location, ControllingDepartment

EMPLOYEE

Name (FName, MInit, LName), SSN, Sex, Address, Salary,

BirthDate, Department, Supervisor, {WorksOn (Project, Hours)}

DEPENDENT

Employee, DependentName, Sex, BirthDate, Relationship

Degree of a Relationship Type

The degree of a relationship type is the number of participating entity types. Hence, the WORKS_FOR relationship is of degree two. A relationship type of degree two is called binary, and one of degree three is called ternary. An example of a ternary relationship is SUPPLY, where each relationship instance associates three entities—a supplier s, a part p, and a project j— whenever s supplies part p to project j, relationships can generally be of any degree, but the ones most common are binary relationships. Higher-degree relationships are generally more complex than binary relationships.

Relationships as Attributes

It is sometimes convenient to think of a relationship type in terms of attributes. Consider the WORKS_FOR relationship type, one can think of an attribute called Department of the EMPLOYEE entity type whose value for each employee entity is (a reference to) the department entity that the employee works or, Hence, the value set for this Department attribute is the set of all DEPARTMENT entities, which is the DEPARTMENT entity set.

However, when we think of a binary relationship as an attribute, we always have two options. Employees of the entity type DEPARTMENT whose values for each department entity is the set of employee entities who work for that department.

The value set of this Employees attribute is the power set of the EMPLOYEE entity

set: Either of these two attributes—Department of EMPLOYEE or Employees of DEPARTMENT—can represent the WORKS_FOR relationship type, if both are represented, they are constrained to be inverses of each other.

NOTES

Role Names and Recursive Relationships

Each entity type that participates in a relationship type plays a particular role in the relationship. The role name signifies the role that a participating entity from the entity type plays in each relationship instance, and helps to explain what the relationship means. For example, in the WORKS_FOR relationship type, EMPLOYEE plays the role of employee or worker and DEPARTMENT plays the role of department or employer.

Role names are not technically necessary in relationship types where all the participating entity types are distinct, since each participating entity type name can be used as the role name, However, in some cases the same entity type participates more than once in a relationship type in different roles. In such cases the role name becomes essential for distinguishing the meaning of each participation. Such relationship types are called recursive relationships.

LOGICAL DATABASE DESIGN

Most of the current commercial DBMSs use an implementation data model—such as the relational or te object-relational database model—so the conceptual schema is transformed from the high-level data model into the implementation data model. This step is called Logical Design or Data Model Mapping; its result is a database schema in the implementation data model of the DBMS.

NORMALIZATION

The normalization process, as first proposed by Codd (1972a), takes a relation schema through a series of tests to "certify" whether it satisfies a certain normal form. The process, which proceeds in a top-down fashion by evaluating each relation again the criteria for normal forms and decomposing relations as necessary, can thus be considered as relational design by analysis.

Initially, Codd proposed three normal forms, which he called First, Second, and Third normal form. A stronger definition of 3NF—called Boyce-Codd Normal Form (BCNF)—was proposed later by Boyce and Codd.

All these normal forms are based on the functional dependencies among the attributes of a relation. Later, a **fourth normal form** (4NF) and a fifth normal form (5NF) were proposed, based on the concepts of multivalued dependencies and join dependencies, respectively.

Normalization of data can be looked upon as a process of analyzing the given relation schemas based on their FDs and primary keys to achieve the desirable properties of

(1) minimizing redundancy and

(2) minimizing the insertion, deletion, and update anomalies.

NOTES

Unsatisfactory relation schemas that do not meet certain conditions—the normal form tests—are decomposed into smaller relation schemas that meet the tests and hence possess the desirable properties. Thus, the normalization procedure provides database designers with the following:

- A formal framework for analyzing relation schemas based on their keys and on the functional dependencies among their attributes.
- A series of normal form tests that can be carried out on individual relation schemas so that the relational database can be normalized to any desired degree.

The normal form of a relation refers to the highest normal form condition that it meets, and hence indicates the degree to which it has been normalized. Normal forms, when considered in isolation from other factors, do not guarantee a good database design. It is generally not sufficient to check separately that each relation schema in the database is, say, in BCNF or 3NF. Rather, the process of normalization through decomposition must also confirm the existence of additional properties that the relational schemas, taken together, should possess. These would include two properties:

- The lossless join or nonadditive join property, which guarantees that the spurious tuple generation does not occur with respect to the relation schemas created after decomposition.
- The dependency preservation property, which ensures that each functional dependency is represented in some individual relation resulting after decomposition.

The nonadditive join property is extremely critical and must be achieved at any cost, whereas the dependency preservation property, although desirable, is sometimes sacrificed.

Use of Normal Forms

Most practical design projects acquire existing designs of databases from previous designs, designs in legacy models, or from existing files. Normalization is carried out in practice so that the resulting designs are of high quality and meet the desirable properties stated previously. Although several higher normal forms have been defined, such as the 4NF and 5NF, the practical utility of these normal forms becomes questionable when the constraints on which they are based are hard to understand or to detect by the database designers and users who must discover these constraints. Thus, database design as practiced in industry today pays particular attention to normalization only up to 3NF, BCNF, or 4NF.

Another worth noting point is that the database designers need not normalize to the highest possible normal form. Relations may be left in a lower normalization status, such as 2NF, for performance reasons. The process of storing the join of higher normal form relations as a base relation, which is in a lower normal form—is known as denormalization.

Non-loss decomposition

The problem of database inconsistency and redundancy of data are similar to the problems that exist in the hierarchical and network models. These problems are addressed in the network model by the introduction of virtual fields and in the

hierarchical model by the inroduction of virtual records. In the relational model, the above problems can be remedied by decomposition. Thus, Decomposition can be defined as following:

NOTES

Definition: ***The decomposition of a relation scheme $R = (A_1, A_2, \ldots A_n)$ is its replacement by a set of relation schemes $(R_1, R_2, \ldots R_n)$, such that $R_i \subseteq R$ for $1 \leq i \leq m$ and $R_1 \cup R_2 \cup \ldots \cup R_m = R$.***

A relation scheme R can be decomposed into a collection of relation schemes (R_1, R_2, R_3, . . . R_m) to eliminate some of the anomalies contained in the original relation R. Here the relation schemes R_i ($1 \leq i \leq m$) are subsets of R and the intersection of $R_i \cap R_j$ for $1 \neq j$ need not be empty. Furthermore, the union of R1 ($1 \leq i \leq m$) is equal to R, i.e., $R = R_1 \cup R_2 \cup \ldots \cup R_m$.

The problems in the relation scheme STDINF can be resolved if we replace it with the following relation schemes:

STUDENT_INFO (Name, Phone_No, Major)

TRANSCRIPT(Name, Course, Grade)

TEACHER(Course, Proof)

The *first* relation scheme gives the phone number and the major of each student and such information will be stored only once for each student. Any change in the phone number will thus require a change in only one tuple of this relation.

The *second* relation scheme stores the grade student in each course that the student is or was enrolled in.

The *third* relation scheme records the teacher of each course.

One of the disadvantages of replacing the original relation scheme STDINF with the three relation schemes is that the retrieval of certain information requires a natural join operation to be performed. For instance, to find the majors of student who obtained a grade of A in course 353 requires a join to be performed: (STUDENT_INFO <> TRANSCRIPT). The same information could be derived from the original relation STDINF by selection and projection.

When we replace the original relation scheme STDINF with the relation schemes STUDENT_INFO, TRANSCRIPT and TEACHER, the consistency and referential integrity constraints have to be enforced.

The referential integrity enforcement implies that if a tuple in the relation TRANSCRIPT exists, such as (Jones, 353, in prog), a tuple must exist in STUDENT_INFO with Name = Jones and, further more, a tuple must exist in TEACHER = Course = 353. The attribute Name, which forms part of the key of the relation TRANSCRIPT, is a key of the relation STUDENT_INFO.

Such an attribute (or a group of attributes), which establishes a relationship between specific tuples (of the same or two distinct relations), is called a foreign key. Notice that the attribute Course in relation TRANSCRIPT is also a foreign key, since it is a key of the relation TEACHER.

Note that decomposition of STDINF into the relation schemes STUDENT(Name, Phone_No, Major, Grade) and COURSE(Course, Prof) is a bad decomposition for the following reasons:

NOTES

1. Redundancy and update anomally, because the data for the attributes Phone_No and Majors are repeated.
2. Loss of information, because we lose the fact that a student has a given grade in a particular course.

Name	*Course*	*Phone_No*	*Major*	*Prof*	*Grade*
Jones	353	237-4539	Comp Sci	Smith	A
Ng	329	427-7390	Chemistry	Turner	B
Jones	328	237-4539	Comp Sci	Clark	B
Martin	456	388-5183	Physics	James	A
Dullies	293	371-6259	Decision Sci	Cook	C
Duke	491	823-7293	Mathematics	Lamb	B
Duke	356	823-7293	Mathematics	Bond	in prog.
Jones	492	237-4539	Comp Sci	Cross	in prog.
Baxter	379	839-0827	English	Broes	C

Functional dependencies

Earlier we discussed the concept of uniquely identifying an entity within an entity set by a key, the key being a set of attributes of the entity.

A relation scheme R has a similar concept, which can be explained using functional dependencies.

The first requirement indicates that the dependency of all attributes of R on K is given explicitly in F or can be logically implied from F. The second requirement indicates that no proper subset of K can determine all the attributes of R. Thus, the key used here is minimal with respect to this property and the FD K $\rightarrow$ R is left reduced. A superset of K can then be called a superkey. If there are two or more subsets of R such that the above conditions are satisfied, such subsets are called candidate keys. In such a case one of the candidate keys is designated as the primary key or simply as the key. We do not allow any attribute in the key of a relation to have a null value.

Definition: ***Given a relation scheme R $\{A_1A_2A_3 \ldots A_n\}$ and a set of functional dependencies F, a key of R is a subset of R such that $K \rightarrow A_1A_2A_3 \ldots A_n$ is in F^+ and for any $Y \subset K, Y \rightarrow A_1A_2A_3 \ldots A_n$ is not in F^+.***

Example

If R (ABCDEH) and F = {A $\rightarrow$ BC, CD $\rightarrow$ E, E $\rightarrow$ C, D $\rightarrow$ AEH, ABH $\rightarrow$ BD, DH $\rightarrow$ BC}, then CD is a key of R because CD $\rightarrow$ ABCDEH is in F^+ (since $(CD)^+$ under F is equal to ABCDEH and ABCDH $\subseteq$ ABCDEH). Other candidate keys of R are AD and ED.

Full Functional Dependency

The concept of left reduced FDs and fully functionally dependency is defined below and illustrated in the example given below.

Definition: ***Given a relational scheme R and an FD X → Y, Y is fully functionally dependent on X if there is no Z, where Z is a proper subset of X such that Z → Y. The dependency X → Y is left reduced, there being no extraneous attributes in the left-hand side of the dependency.***

NOTES

Example

In the relation scheme R (ABCDEH) with the FDs F = {A → BC, CD → E, E → C, CD → AH, ABH → BD, DH → BC}. The dependency A → BC is left reduced and BC is fully functionally dependent on A.

However, the functional dependency ABH → D is not left reduced, the attribute B being extraneous in this dependency.

Prime Attribute and Nonprime Attribute

We defined the key of a relation scheme earlier.

We distinguish the attributes that participate in any such key as indicated by the following definition.

Example

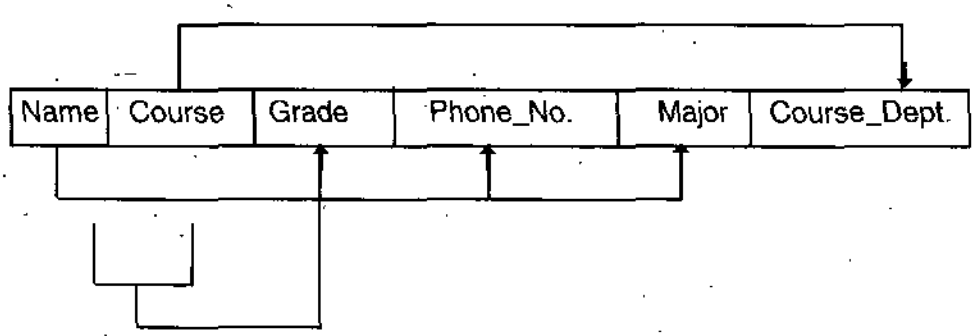
If R (ABCDEH) and F = {A → BC, CD → E, E → C, AH → D}, then AH is the only candidate key of R. The attributes A and H are prime and the attributes B. C. D. and E are nonprime.

Definition: ***An attribute A in a relation scheme R is a prime attribute or simply prime if A is part of any candidate key of the relation. If A is not a part of any candidate key of R, A is called a nonprime attribute or simply nonprime.***

Partial Dependency

Let us introduce the concept of partial dependency below and illustrate the same in the example given next.

Definition: ***Given a relation scheme R with the functional dependencies F defined on the attributes of R and K as a candidate key, if X is a proper subset of K and if F |= X → A. then A is said to be partially dependent on K.***



Example

(a) In the relation scheme STUDENT_COURSE_INFO(Name, Course, Grade,

NOTES

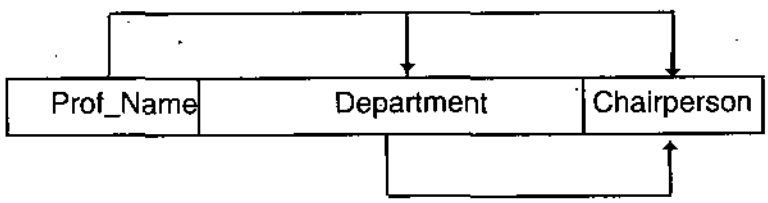
Phone_No, Major, Course_Dept) with the FDs F = {Name → Phone_NoMajor, Course → Course_Dept, NameCourse → Grade}, NameCourse is a candidate key, Name and Course are prime attributes. Grade is fully functionally dependent on the candidate key. Phone_No, Course_Dept and Major are partially dependent on the candidate key.

(b) Given R (A, B, C, D) and F = {AB → C, B → D}, the key of this relation is AB and D is partially dependent on the key.

Transitive Dependency

Another type of dependency which we have to recognize in database design is introduced below and illustrated in the example next.

Definition: ***Given a relation scheme R with the functional dependencies F defined on the attributes of R, let X and Y be subsets of R and let A be attribute of R such that X ⊄ Y, A ⊄ XY. If the set of functional dependencies {X →, Y → A} is implied by F (i.e. F |= X → Y → A and F□¬ |= Y → X) then A is transitively dependent on X.***



Example

(a) In the relation scheme PROF_INFO(Prof_Name, Department, Chairperson) and the function dependencies F = {Prof_Name → Department → Chairperson}, Prof_Name is the key and Chairperson is transitively dependent on the key since Prof_Name → Department → Chairperson.

(b) Given R (A, B, C, D, E) and the function dependencies F = {AB → C, B → D, C → E}, AB is the key and E is transitively dependent on the key since AB → C → E.

FIRST NORMAL FORM

First normal form (1NF) is now considered to be part of the formal definition of a relation in the basic (flat) relational model: historically, it was defined to disallow multivalued attributes, composite attributes, and their combinations. It states that the domain of an attribute must include only atomic (simple, indivisible) values and that the value of any attribute in a tuple must be a single value from the domain of that attribute. Hence, 1NF disallows having a set of values, a tuple of values, or a combination of both as an attribute value for a single tuple. In other words, 1NF disallows "relations within relations" or "relations as attribute values within tuples.' The only attribute values permitted by 1NF are single atomic (or indivisible) values.

Consider the DEPARTMENT relation schema shown here, whose primary key is DNUMBER, and suppose that we extend it by including the DLOCATIONS attribute

as shown in next figure. We assume that each department can have a number of locations. The DEPARTMENT schema and an example relation state are shown in

EMPLOYEE

ENAME	SSN	BDATE	ADDRESS	DNUMBER

DEPARTMENT

DNAME	DNUMBER	DMGRSSN

DEPT_LOCATIONS

DNUMBER	DLOCATION

PROJECT

PNAME	PNUMBER	PLOCATON	DNUM

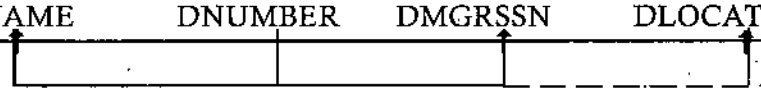
WORKS_ON

SSN	PNUMBER	HOURS

next figure. As we can see, this is not in 1NF because DLOCATIONS is not an atomic attribute, as illustrated by the first tuple in the figure. There are two ways we can look at the DLOCATIONS attribute:

(a) DEPARTMENT

DNAME	DNUMBER	DMGRSSN	DLOCATIONS



(b) DEPARTMENT

DNAME	DNUMBER	DMGRSSN	DLOCATIONS
Research	5	333445555	{Bellaire, Sugarland, Houston}
Administration	4	987654321	{Stafford}
Headquarters	1	888665555	{Houston}

(c) DEPARTMENT

DNAME	DNUMBER	DMGRSSN	DLOCATIONS
Research	5	333445555	Bellaire
Research	5	333445555	Sugarland
Research	5	333445555	Houston
Administration	4	987654321	Stafford
Headquarters	1	888665555	Houston

NOTES

NOTES

- The domain of DLOCATIONS contains atomic values, but some tuples can have a set of these values. In this case, DLOCATIONS is not functionally dependent on the primary key DNUMBER.
- The domain of DLOCATIONS contains sets of values and hence is nonatomic. In this case, DNUMBER → DLOCATIONS, because each set is considered a single member of the attribute domain.

In either case, the DEPARTMENT relation of in figure is not in 1NF; in fact, it does not even qualify as a relation according to our definition of relation. There are three main techniques to achieve first normal form for such a relation:

1. Remove the attribute DLOCATIONS that violates 1NF and place it in a separate relation DEPT_LOCATIONS along with the primary key DNUMBER of DEPARTMENT. The primary key of this relation is the combination {DNUMBER, DLOCATION, as shown in figure. A distinct tuple in DEPT_LOCATIONS exists for each LOCATION of a department. This decomposes the non- 1NF relation into two 1NF relations.
2. Expand the key so that there will be a separate tuple in the original DEPARTMENT relation for each location of a DEPARTMENT, as shown in figure c. In this case, the primary key becomes the combination {DNUMBER, DLOCATION}. This solution has the disadvantage of introducing redundancy in the relation.
3. If a maximum number of values is known for the attribute, for example, if it is known that at most three locations can exist for a department—replace the DLOCATIONS attribute by three atomic attributes: DLOCATION1, DLOCATION2, and DLOCATION3. This solution has the disadvantage of introducing null values if most departments have fewer than three locations. It further introduces a spurious semantics about the ordering among the location values that is not originally intended. Querying on this attribute becomes more difficult; for example, consider how you would write the query: "List the departments that have "Bellaire" as one of their locations" in this design.

Of the three solutions above, the first is generally considered best because it does not suffer from redundancy and it is completely general, having no limit placed on a maximum number of values. In fact. if we choose the second solution, it will be decomposed further during subsequent normalization steps into the first solution.

First normal form also disallows multivalued attributes that are themselves composite. These are called nested relations because each tuple can have a relation within it. The next figure shows how the EMP_PROJ relation could appear if nesting is allowed. Each tuple represents an employee entity, and a relation PROJS(PNUMBER, HOURS) within each tuple represents the employee's projects and the hours per week that employee works on each project. The schema of this EMP_PROJ relation can be represented as follows:

```
EMP_ PROJ(SSN, ENAME {PROJS(PNUNBER, HOURS)})
```

The set braces { } identify the attribute PROJS as multivalued, and we list the component attributes that form PROJS between parentheses ( ). Interestingly, recent trends for supporting complex objects and XML data using the relational model attempt to allow and formalize nested relations within relational database systems, which were disallowed early on by 1NF.

(a) EMP_PROJ

SSN	ENAME	PROJS	
		PNUMBER	HOURS

NOTES

(b) EMP_PROJ

SSN	ENAME	PNUMBER	HOURS
123456789	Smith John B.	1	32.5
		2	7.5
666777888	Naryana Ramesh	3	40.0
453678325	English Joyce A	1	20.0
		2	20.0
765432219	Wong Franklin T	2	10.0
		3	10.0
		10	10.0
		20	10.0
656789654	Zelaya Alicia J	30	30.0
		10	10.0
969798989	Jabbar Ahmed B	10	35.0
		30	5.0
876787678	Wallace Jennifer S	30	20.0
		20	15.0
876545678	Borg James E	20	null

(c) EMP_PROJ1

SSN	ENAME

EMP_PROJ2

SSN	PNUMBER	HOURS

Notice that SSN is the primary key of the EMP_PROJ in relation to above figures a and b, while PNUMBER is the partial key of the nested relation; that is, within each tuple, the nested relation must have unique values of PNUMBER. To normalize this into 1NF, we remove the nested relation attributes into a new relation and propagate the primary key into it; the primary key of the new relation will combine the partial key with the primary key of the original relation. Decomposition and primary key propagation yield the schemas EMP_PROJ1 and EMP_PROJ2 as shown in c part of the figure.

This procedure can be applied recursively to a relation with multiple-level nesting to unnest the relation into a set of 1NF relations. This is useful in converting an unnormalized relation schema with many levels of nesting into 1NF relations. The existence of more than one multivalued attribute in one relation must be handled carefully. As an example, consider the following non-INF relation:

NOTES

```
PERSON (SS#, {CAR_LIC#}, {PHONE#})
```

This relation represents the fact that a person has multiple cars and multiple phones. If a strategy like the second option above is followed, it results in an all-key relation:

```
PERSON_IN_1NF (SS#, CAR_LlC#, PHONE#)
```

To avoid introducing any extraneous relationship between CAR_LIC# and PHONE#, all possible combinations of values are represented for every SS#, giving rise to redundancy. This leads to the problems handled by multivalued dependencies and 4NF. The right way to deal with the two multivalued attributes in PERSON above is to decompose it into two separate relations, using strategy 1 discussed above:

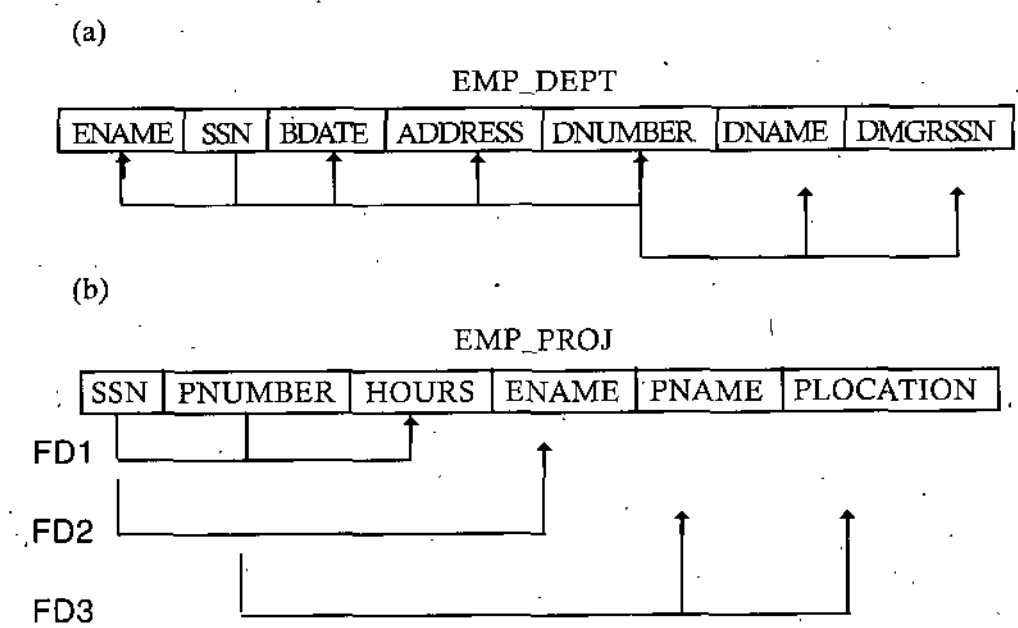
```
P1(SS#, CAR_LlC#) and P2(SS#, PHONE#).
```

SECOND NORMAL FORM

Second normal form (2NF) is based on the concept of full functional dependency. A functional dependency X → Y is a full functional dependency if removal of any attribute A from X means that the dependency does not hold any more; that is, for any attribute A ∈ X, (X – {A}) does not functionally determine Y. A functional dependency X → Y is a partial dependency if some attribute A ∈ X can be removed from X and the dependency still holds; that is, for some A ∈ X, (X – {A}) → Y. In the next figure part b, {SSN, PNUMBER} → HOURS is a full dependency (neither SSN → HOURS nor PNUMBER → HOURS holds). However, the dependency {SSN, PNUMBR} → ENAME is partial because SSN → ENAME holds.

Definition: ***A relation schema R is in 2NF if every nonprime attribute A in R is fully functionally dependent on the primary key of R.***

The test for 2NF involves testing for functional dependencies whose left-hand side attributes are part of the primary key. If the primary key contains a single attribute, the test need not be applied at all. The EMP_PROJ relation in figure part b is in 1NF but is not in 2NF. The nonprime attribute ENAME violates 2NF because of FD2, as



do the nonprime attributes PNAME and PLOCATION because of FD3. The functional dependencies FD2 and FD3 make ENAME, PNAME, and , PLOCATION partially dependent on the primary key {SSN, PNUMBER} of EMP_PROJ, thus violating the 2NF test.

If a relation schema is not in 2NF, it can be "second normalized" or "2NF normalized" into a number of 2NF relations in which nonprime attributes are associated only with the part of the primary key on which they are fully functionally dependent. The functional dependencies FD1, FD2, and FD3 in the figure part b hence lead to the decomposition of EMP=PROJ into the three relation schemas EP1, EP2, and EP3 as shown in figure part a, each of which is in 2NF.

NOTES

THIRD NORMAL FORM

Third normal form (3NF) is based on the concept of transitive dependency. A functional dependency $X \rightarrow Y$ in a relation schema R is a transitive dependency if there is a set of attributes Z that is neither a candidate key nor a subset of any key of R, and both $X \rightarrow Z$ and $Z \rightarrow Y$ hold. The dependency SSN $\rightarrow$ DMGRSSN is transitive through DNUMBER in EMP_DEPT of above figure part a because both the dependencies SSN $\rightarrow$ DNUMBER and DNUMBER $\rightarrow$ DMGRSSN hold and *DNUMBER is neither a key itself nor a subset of the key of* EMP_DEPT. Intuitively, we can see that the dependency of DMGRSSN is undesirable in EMP_DEPT since DNUMBER is not a key of EMP_DEPT.

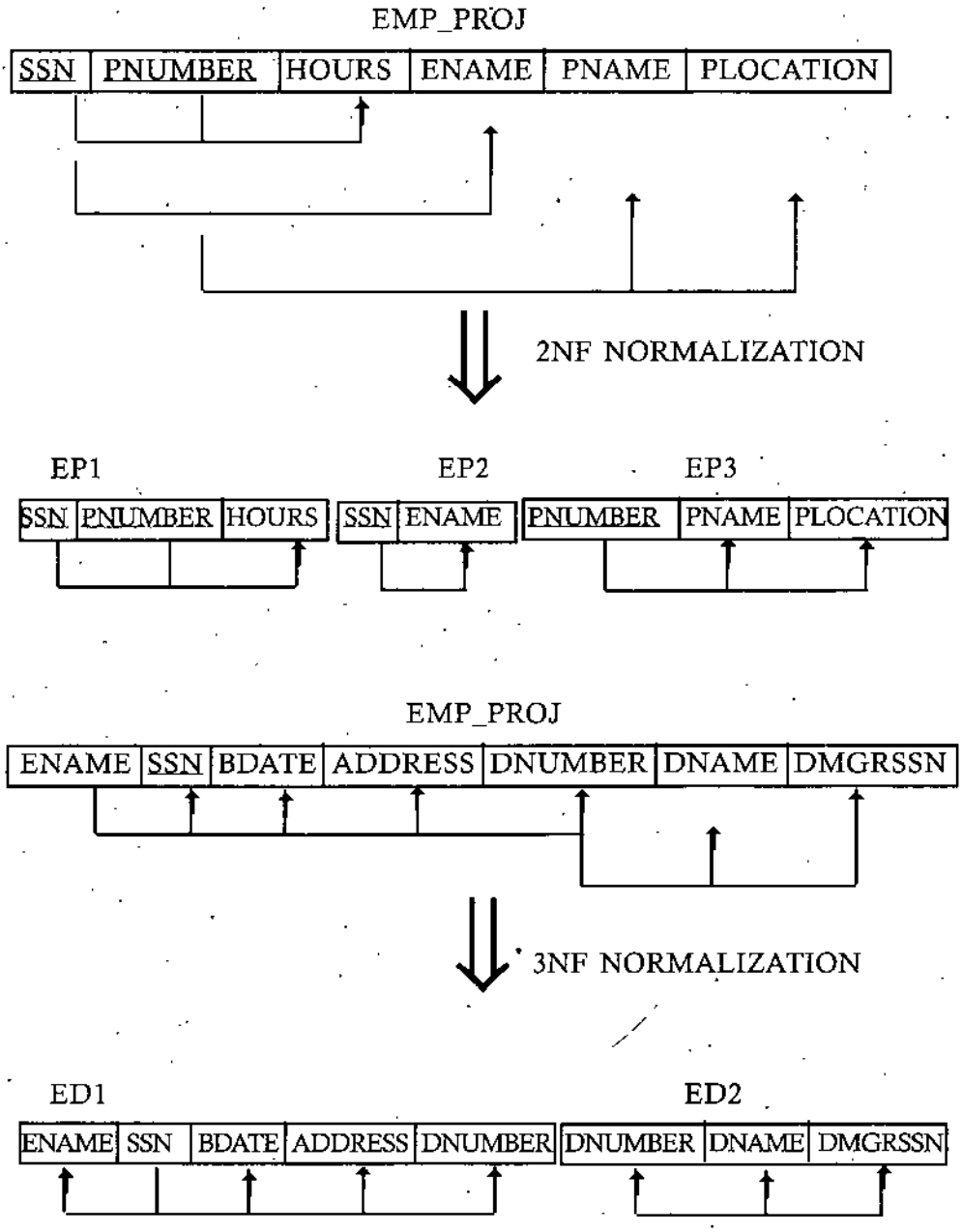
Definition: ***According to Codd's original definition, a relation schema R is in 3NF if it satisfies 2NF and no nonprime attribute of R is transitively dependent on the primary key.***

The relation schema EMP_DEPT is not in 3NF because of the transitive dependency of DMGRSSN (and also DNAME) on SSN via DNUMBER. We can normalize EMP_DEPT by decomposing it into the two 3NF relation schemas ED1 and ED2

NORMAL FORM		TEST	REMEDY (NORMALIZATION)
First (1 NF)	Relation should have no nonatomic attributes or nested relations.	Form new relations for each non-atomic attribute or nested relation.	
Second (2 NF)	For relations where primary key contains multiple attributes, no nonkey attribute should be functionally dependent on a part of the primary key.	Decompose and set up a new relation for each partial key with its dependent attribute(s). Make sure to keep a relation with the original primary key and any attribute that are fully functionally dependent on it.	
Third (3 NF)	Relation should not have a nonkey attribute functionally determined by another nonkey attribute (or by a set of nonkey attributes.) That is, there should be no transitive dependency of a nonkey attribute on the primary key.	Decompose and set up a relation that includes the nonkey attribute(s) that functionally determine(s) other nonkey attribute(s)	

NOTES

(a)



shown in next figure part b. Intuitively, we see that ED1 and ED2 represent independent entity facts about employees and departments. A NATURAL JOIN operation on ED1 and ED2 will recover the original relation EMP_DEPT without generating spurious tuples.

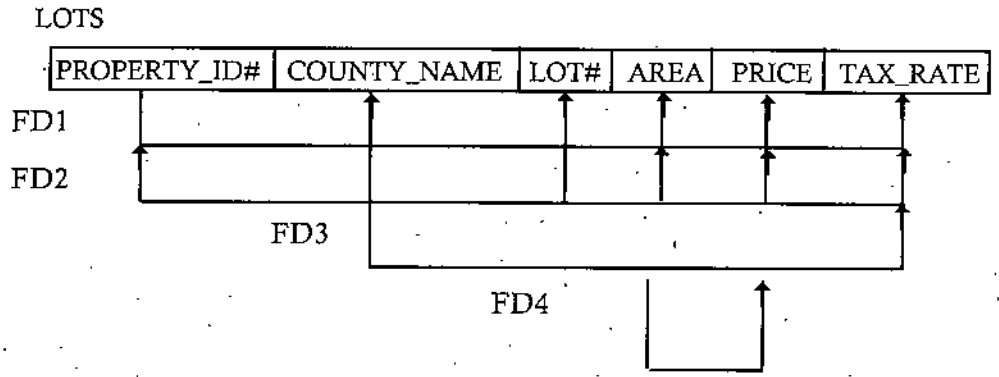
Intuitively, we can see that any functional dependency in which the left-hand side is part (proper subset) of the primary key or any functional dependency in which the left hand side is a nonkey attribute is a "problematic" FD. 2NF and 3NF normalization remove these problem FDs by decomposing the original relation into new relations.

In terms of the normalization process, it is not necessary to remove the partial dependencies before the transitive dependencies, but historically, 3NF has been defined with the assumption that a relation is tested for 2NF first before it is tested for 3NF. The next table informally summarizes the three normal forms based on primary keys, the tests used in each case, and the corresponding "remedy" or normalization performed to achieve the normal form.

BCNF

NOTES

Boyce-Codd Normal Form (BCNF) was proposed as a simpler form of 3NF, but it was found to be stricter than 3NF. That is, every relation in BCNF is also in 3NF; however, a relation in 3NF is not necessarily in BCNF. Intuitively, we can see the need for a stronger normal form than 3NF by going back to the LOTS relation schema of next figure with its four functional dependencies FD1 through FD4.



Suppose that we have thousands of lots in the relation but the lots are from only two counties: Dekalb and Fulton. Suppose also that lot sizes in Dekalb County are only 0.5, 0.6, 0.7, 0.8, 0.9, and 1.0 acres, whereas lot sizes in Fulton County are restricted to 1.1, 1.2, . . ., 1.9, and 2.0 acres. In such a situation we would have the additional functional dependency FD5: AREA → COUNTY_NAME. If we add this to the other dependencies, the relation schema LOTS1A still is in 3NF because COUNTY_NAME is a prime attribute.

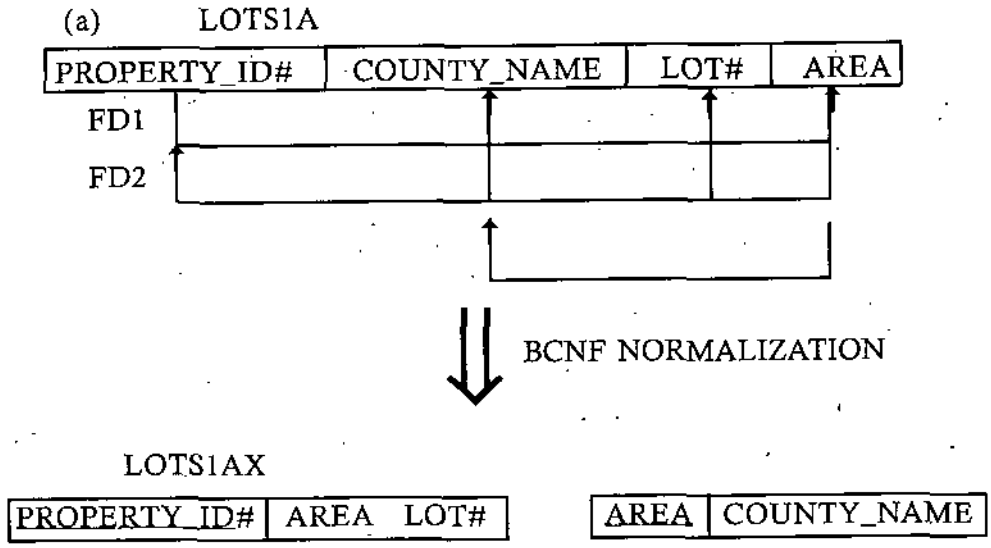
The area of a lot that determines the county, as specified by FD5, can be represented by 16 tuples in a separate relation R(AREA, COUNTY_NAME), since there are only 16 possible AREA values. This representation reduces the redundancy of repeating the same information in the thousands of LOTS1A tuples. BCNF is a stronger normal form that would disallow LOTS1A and suggest the need for decomposing it.

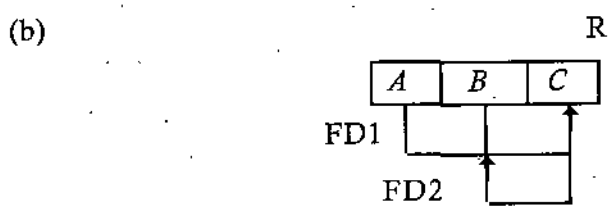
Definition: ***A relation schema R is in BCNF if whenever a nontrivial functional dependency X → A holds in R, then X is a superkey of R.***

The formal definition of BCNF differs slightly from the definition of 3NF. The only difference between the definitions of BCNF and 3NF is that condition (b) of 3NF, which allows A to be prime, is absent from BCNF. In our example, FD5 violates BCNF in LOTS1A because AREA is not a superkey of LOTS1A. Note that FD5 satisfies 3NF in L0TS1A because COUNTY_NAME is a prime attribute (condition b), but this condition does not exist in the definition of BCNF. We can decompose LOTS1A into two BCNF relations LOTS1AX and LOTS1AY, as shown next. This decomposition loses the functional dependency FD2 because its attributes no longer coexist in the same relation after decomposition.

In practice, most relation schemas that are in 3NF are also in BCNF. Only if X → A holds in a relation schema R with X not being a superkey and A being a prime attribute

NOTES





will R be in 3NF but not in BCNF. The relation schema R shown in above figure part b illustrates the general case of such a relation. Ideally, relational database design should strive to achieve BCNF or 3NF for every relation schema. Achieving the normalization status of just 1NF or 2NF is not considered adequate, since they were developed historically as stepping stones to 3NF and BCNF.

As another example, consider the next figure, which shows a relation TEACH with the following dependencies:

FD1:{STUDENT, COURSE}→INSTRUCTOR

FD2:INSTRUCTOR → COURSE

Note that {STUDENT, COURSE} is a candidate key for this relation and that the dependencies shown follow the pattern in part b, with STUDENT as A, COURSE as B, and INSTRUCTOR as C. Hence this relation is in 3NF but not in BCNF. Decomposition of this relation schema into two schemas is not straightforward because it may be decomposed into one of the three following possible pairs:

1. **{STUDENT, INSTRUCTOR} and {STUDENT, COURSE}.**
2. **{COURSE, INSTRUCTOR } and {COURSE, STUDENT}.**
3. **{INSTRUCTOR, COURSE } and {INSTRUCTOR, STUDENT}.**

All three decompositions "lose" the functional dependency FDI. The desireable decomposition of those just shown is 3, because it will not generate spurious tuples after a join.

STUDENT	*COURSE*	*INSTRUCTOR*
Narayan	Database	Mark
Smith	Database	Navathe
Smith	Operating System	Ammar
Smith	Theory	Schulman
Wallace	Database	Mark
Wallace	Operating System	Ahamad
Wong	Database	Omiecinski
Zelaya	Database	Navathe

NOTES

In general, a relation not in BCNF should be decomposed so as to meet this property, while possibly forgoing the preservation of all functional dependencies in the decomposed relations, as is the case in this example. The above algorithm does that and could be used above to give decomposition 3 for TEACH.

TRANSLATING E-R DIAGRAMS TO RELATIONS

Converting any E-R model to a set of tables in a database is followed by a specific set of rules that govern such a conversion. The application of these rules requires understanding the effects of updates and deletions on the tables in the database. Before we discuss these rules in detail, let's briefly review a simpler model, its schema and the SQL commands used to generate the tables.

The model, the Artist database, conforms to the following conditions:

- A painter might paint many paintings. To be considered a painter in the artist database, the painter must have painted at least one painting. This business rule decreases that the cardinality is (1,N) in the relationship between Painter and Painting.
- Each painting is painted by one (and only one) painter.
- A painting might (or might not) be exhibited in a gallery; that is, the Gallery is an optional entity to the Painting entity.

Given this description, we create a simple E-R model and some matching tables for the Artist database shown in figure.

Given these artist database structures, let us now examine the effect of the following actions:

1. **Deleting a painter (row) from the painter table.** If we delete a row (painter) from the painter table, the painting table will contain references to a painter who no longer exists, thereby creating a deletion anomaly. (A painting does not cease to exist just because the painter does.)

   Given this situation, it is wise to restrict the ability to delete a row from a table if there is a foreign key in another table that references it. In short, we

NOTES

should impose a delete restrict requirement on such a table. The restriction means that we can delete a painter from the painter table only if there is no foreign key in another table that requires the painters existence.

The practical effect of this limitation is simple. Suppose that we want to delete painter PTr_num = 123 from the artist database. The Delete restrict clause requires that we must first delete all rows in the painting table that use PTr_num = 123 as the foreign key value. We do this to make the user aware of the consequences of deleting a painter.

The DBMS could also be instructed to delete all painting rows corresponding to the deleted painter (delete cascade), but we chose not to do so for the reasons just stated.

2. **Adding a painter (row) to the painter table.** Adding a painter does not cause any problems, because the painter code does not have any dependencies in other tables.

3. **Making changes in painter table (primary key values).** Changing a painter key causes problems in the database because some paintings in the painting table may make reference to this key. The solution is simple: make sure that a change in the painter's ptr_num automatically triggers the require changes in the ptr_num key found in other tables. Because one change cascades through the system, *the process is called update cascade.*

   In other words, the requirement of update cascade means that all foreign key references to a primary key are updated when the primary key is changed.

4. **Deleting a gallery (row) from the gallery table.** Deleting a Gallery row creates problems in the database if there are rows in the painting table that make reference to that gallery row's primary key.

   Because gallery is optional to painting, we may set all deleted gallery gal_num values to null. Or we may want the database user to be alerted to the problem by specifying a delete restrict clause in the gallery table. The delete restrict clause means that the deletion of a gallery row is permitted only if there is no foreign key (gal_num) in another table that requires the gallery row's existence.

5. **Adding a gallery (row) to the gallery table.** Adding a new row does not affect the database because the gallery does not have dependencies in other tables. (The new row will not be referenced by any foreign key that points to the gallery table.)

6. **Updating the gallery table's primary key.** Changing a primary key value in a gallery row requires that all foreign keys making reference to it be updated, as well. Therefore, we must use an update cascade clause.

PHYSICAL DATABASE DESIGN

See chapter 1.

RELATIONAL ALGEBRA & SQL RELATIONAL DATABSE COMMANDS

NOTES

The name SQL is derived from Structured Query Language. Originally, SQL was called SEQUEL (for Structured English Query Language) and was designed and implemented at IBM Research as the interface for an experimental relational database system called SYSTEM R. SQL is now the standard language for commercial relational DBMSs. A jont effort by ANSI (the American National Standards Institute) and ISO (the International Standards Organization) has led to a standard version of SQL (ANSI 1986), called SQL-86 or SQL1. A revised and much expanded standard called SQL2 (also referred to as SQL-92) was subsequently developed. The next version of the standard was originally called SQL3, but now is called SQL-99.

SQL is a comprehensive database language. It has statements for data definition, query and update. Hence, it is both a DDL and a DML. In addition, it has facilities for defining views on the database, for specifying security and authorization, for defining integrity constraints, and for specifying transaction controls. It also has rules for embedding SQL statements into general-purpose programming language such as Java or COBOL or C/C++.

Data Definition Language

These commands are used to create and maintain a database.

CREATE

Read about it later in the chapter.

ALTER

Read about it later in the chapter.

DROP

Read about it later in the chapter.

RENAME

Column Alias are used to rename a table's columns for the purpose of a particular query. The PRODUCTS_TBL illustrates the use of column aliases.

```
SELECT  COLUMN_NAME  ALIAS_NAME
FROM  TABLE_NAME;
```

The following example displays the product description twice, giving the second column an alias named PRODUCT. Notice the column headers in the output.

```
SELECT  PROD_DESC
        PROD_DESC  PRODUCT
FROM  PRODUCTS_TBL
```

TRUNCATE

SQL offers two options, DELETE and TRUNCATE TABLE for deleting data. These are the two most dangerous commands in SQL. So make sure that you intend to get

STUDENT ACTIVITY 2.1

1. What do you understand by Data Modeling?

2. Describe Entity Relationship Model.

3. Describe the various stages of Information Engineering

4. How would you plan a model?

NOTES

rid of the data you have described in these statements before you execute the statements. There is no Undo button related to these two statements. Results are permanent and final.

Hint: ***Prior to undertaking deletion operation, you are advised to back up your database.***

Assume that you have an entire table that needs to be cleared of data. You have two options for undertaking the deletion. They look like this:

```
DELETE FROM authors
TRUNCATE TABLE authors
```

Both of these statements achieve the same purpose; they delete all the data in the table while leaving intact the column structure associated with the table. As a result, new data can easily be inserted after the deletion takes place.

Difference between TRUNCATE and DELETE

First, DELETE is supported by all SQL databases, while TRUNCATE TABLE might not be. In addition, DELETE is supported in all circumstances, TRUNCATE TABLE might not function in all situations. For example, Microsoft's SQL Server allows you to use TRUNCATE TABLE in most circumstances. However, when you are using the Data Transformation Services to copy a table from one database to another, you cannot use TRUNCATE TABLE to clear the table in the target database of pre-existing data. You must instead use DELETE. In addition, TRUNCATE TABLE undetakes only complete deletions of a table. You cannot use it to delete selected rows and leave others intact.

Data Manipulation Language (DML)

These commands are used for data manipulation.

Select Command

Read about them later in the chapter.

Column Heading Default

You can use * as the indicator for all fields. It is in fact more convenient to use. For example, the above command can be given as

```
SELECT * FROM STUDENT_TBL;
```

This would result in listing all the fileds of the table:

```
SACHIN      1022      MUMBAI
RAHUL       1033      BANGALORE
YUVRAJ      1044      DELHI
DILIP       1055      KOLKATTA
```

Using Arithmetic Operators

You can use the various arithmetical operators like +(addition), – (subtraction), * (multiplication), and /(divison), alongwith the SELECT statement to get the results.

Let me give you examples of each and show how they are used.

NOTES

Addition

It is performed using the (+) symbol.

```
SELECT SALARY + BONUS FROM EMPLOYEE_TBL;
```

The SALARY column is added with the BONUS column for a total for each row of data. You can extend this command to include the following:

```
SELECT FROM EMPLOYEE_TBL WHERE SALARY + BONUS > "25000";
```

This would select all the employees whose SALARY and BONUS added together becomes more than 25,000.

Subtraction

It is performed using the (–) symbol.

```
SELECT SALARY - BONUS FROM EMPLOYEE_TBL;
```

The BONUS column is subtracted from the SALARY column for calculating the difference. You can extend this command to include the following:

```
SELECT FROM EMPLOYEE_TBL WHERE SALARY - BONUS > "25000";
```

This would select all the employees whose SALARY after deducting BONUS becomes more than 25,000.

Multiplication

It is performed using the (*) symbol.

```
SELECT SALARY * 10 FROM EMPLOYEE_TBL;
```

The SALARY column will be multiplied by 10. You can extend this command to include the following:

```
SELECT FROM EMPLOYEE_TBL WHERE SALARY * 10 > "25000";
```

This would return all rows where the product of the SALARY multiplied by 10 is greater than 25,000.

Division

It is performed using the (/) symbol.

```
SELECT SALARY / 10 FROM EMPLOYEE_TBL;
```

The SALARY column is divided by 10. You can extend this command to include the following:

```
SELECT FROM EMPLOYEE_TBL WHERE SALARY / 10 > "25000";
```

This returns all rows where the SALARY divided by 10 is greater than 25,000.

Operator Precedence

The BODMAS rule, as learned in earlier classes applies here too. So in this case the order would be:

Division

Multiplication

Addition and then

Subtraction

NOTES

This can be significantly demonstrated by the following example.

```
2 + 3 * 4 + 5
```

If you do this calulation on the calculator, this would give you the result as 25, which is wrong. The same if done with BODMAS rule would give you the result as 19, which is the right result. Here keeping in mind the BODMAS rule, the multiplication will take place first and then the addition and not the other way around.

Significance of NULL value

NULL is used where you have to specify that there is nothing in it. In SQL it is used very often to check whether the column is blank or not. You can even add a NULL value. You can search for a NULL value using the SELECT statement.

For example,

```
SELECT NAME FROM EMPLOYEE_TBL  WHERE ROLLNO. IS NULL
```

This would search for you the records which do not have a rollno. and gives their names.

NULL values in Arithmetic Expressions

NULL when used in a numeric field signifies that the value is not there. Supposing you are multiplying a field by a variable and by chance that field happens to be NULL, then the result may become abstract. For this you use statement to confirm that the value is not NULL.

For example.

```
SELECT EMP_NAME, SALARY, SALARY*1.5 FROM EMPLOYEE_TBL WHERE
        SALARY IS NOT NULL;
```

This would result in the following:

EMP_NAME	SALARY	SALARY1.5
SACHIN	20000	3000
RAHUL	10000	15000
YUVRAJ	12000	18000
DILIP	8000	12000

This has taken care that the SALARY field is not NULL.

Defining and using Column Alias

These are used to rename a table's column for the purpose of a particular query. For example,

```
SELECT BOOK_NAME, BOOK_NAME ISBN FROM BOOKS_TBL
```

As you know that each book has an ISBN number and a name. This option as mentioned above gives you an option of knowing either one of them. If you know the name of the book, then use BOOK_NAME or in case you know only the ISBN number then use BOOK_NAME ISBN. Both will give you the same result.

Concatenation Operator (||)

It is the process of combining the two separate strings into one string. For example, you can combine the name and last name of an individual using this method.

For example,

```
SELECT "SAC" + "HIN"
```

would result in SACHIN. It is quite useful in cases where you have to save space. There you can combine two fields to club them into one. For example, in most cases of address label printing, city and pin code are printed as one line instead of two lines. This is done using Concatenation Operator.

NOTES

Duplicate rows and their Elimination (DISTINCT keyboard)

The DISTINCT option is used when you have to display only one of the duplicate records. In our example of STUDENT_TBL if there are two records of SACHIN then using the following command only one of them will be displayed.

```
SELECT DISTINCT (NAME) FROM STUDENT_TBL;
```

Only one SACHIN will be selected.

Limiting Rows during selection (using WHERE clause)

WHERE command is used when you have to make selection based on some facts. For example, if you have to select Names and that too when RollNo. is 1033, you will use the WHERE command. For example, the above can be displayed in the form:

```
SELECT NAME FROM STUDENT_TBL WHERE ROLLNO. = "1033"
```

Since there is only one name which has the student roll number as 1033, that name will be selected.

```
RAHUL 1033   BANGALORE
```

The full record will be selected unless you ask it not to do so.

Dates

Date is stored in the computer but the format of representation of the same can be different from system to system. SQL also uses date in its own way. You can recall the system date by using the command called GETDATE() as shown here.

```
SELECT GETDATE()
```

This will give the output as:

```
June 30, 2003
```

You can perform various operations on Date, for example, converting it to character string, adding time, converting it to picture, etc.

Boolean operators

These are also called the Conjunctive Operators. These are used when you have to combine more than one condition for finding out the result. These operators are:

AND, OR and NOT.

Let us see how they are used.

AND

This operator allows the existence of multiple conditions in an SQL statement's WHERE clause. For the action to be taken, all the conditions of the statement must be TRUE.

NOTES

For example,

```
WHERE ROLLNO = "1033" AND NAME = "RAHUL"
```

This will select only the record which has roll number 1033 and name as RAHUL. Both the conditions must be TRUE to get the record selected.

OR

This operator is used to combine multiple conditions in an SQL statement's WHERE clause. Here at least one of the condition should be TRUE for the statement to work.

For example,

```
WHERE ROLLNO = "1033" OR ROLLNO = "1030"
```

Here at least one of the roll number, i.e., 1033 will match and the relative record will be selected.

NOT

This is not just one operator. It has its own set of operators which can be used in conjunction with this. They are: NOT EQUAL, NOT BETWEEN, NOT IN, NOT LIKE, IS NOT NULL, NOT EXISTS, NOT UNIQUE. Did you notice anything?

They are just the negative conditions of the operators used earlier. These are just the other side of those operators but they work in reverse. Instead of matching the conditions, they make sure that the conditions are not matched.

I will not go through the details of each but just give an example to illustrate their working.

Not EQUAL

```
WHERE ROLLNO <> "1050"
```

All records will be selected since there is no records which matches 1050.

Not BETWEEN

```
WHERE ROLLNO NOT BETWEEN "1000" AND "2000"
```

No record will be selected since all of them are between 1000 and 2000.

Not IN

```
WHERE ROLLNO NOT IN ("1000", "1200", "1300")
```

All records will be selected since there is no records which matches the figures given.

Not LIKE

```
WHERE ROLLNO NOT LIKE "2000"
```

All records will be selected since there is no record starting with 2000.

Is Not NULL

```
WHERE ROLLNO IS NOT NULL
```

All records will be selected since there is no records which has NULL value.

Not EXISTS

```
WHERE NOT EXISTS (SELECT ROLLNO FROM STUDENT_TBL WHERE
        ROLLNO = "1033")
```

It searches to see whether the roll number 1033 is not there in the table.

Not UNIQUE

```
WHERE NOT UNIQUE (SELECT ROLLNO FROM STUDENT_TBL)
```

It tests to see whether there are roll numbers in the table that are not UNIQUE.

Logical Operator's Precedence

NOTES

Like arithmetic operators, logical operators too have a precedence. This is different from arithmetic one. The following is the list of precedence:

Arithmetic	*Comparison*	*Logical*
Exponentiation (^)	Equality (=)	Not
Negation (–)	Inequality (<>)	And
Multiplication and division (*,/)	Less than (<)	Or
Integer division (/)	Greater than (>)	Xor
Modulus arithmetic (Mod)	Less than or equal to (<=)	Eqv
Addition and subtraction (+,–)	Greater than or equal to (>=)	Imp
String concatenation (&)	Is	&

Group By Clause

This clause is used in collaboration with the SELECT statement to arrange identical data into groups. The GROUP BY clause follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.

The position of the GROUP BY clause in a query is as follows:

```
SELECT
FROM
WHERE
GROUP BY
ORDER BY
```

The GROUP BY clause must follow the conditions in the WHERE clause and must precede the ORDER BY clause if one is used. The following is the SELECT statement's syntax, including the GROUP BY clause:

```
SELECT COLUMN1, COLUMN2
FROM TABLE1, TABLE2
WHERE CONDITIONS
GROUP BY COLUMN1, COLUMN2
ORDER BY COLUMN1, COLUMN2
```

The following sections give examples and explanations of the GROUP BY clause's use in a variety of situations.

Grouping Selected Data

Grouping data is a simple process. The selected columns are the columns that can be referenced in the GROUP BY clause. If a column is not found in the SELECT statement, it cannot be used in the GROUP BY clause. If the column name has been

NOTES

qualified, the qualified name must go into the GROUP BY clause. When grouping the data, the order of the columns grouped does not have to match the column order in the SELECT clause.

Group Functions

Typical group functions – those that are used with the GROUP BY clause to arrange data in groups – include AVG, MAX, MIN, SUM and COUNT.

Creating Groups and Using Aggregate Functions

There are conditions that the SELECT clause has that must be met when using GROUP BY. Specifically, whatever columns are selected must appear in the GROUP BY clause do not necessarily have to be in the same order as they appear in the SELECT clause. Should the columns in the SELECT clause be qualified, the qualified names of the columns must be used in the GROUP BY clause.

The following are some examples of syntax for the GROUP BY clause:

For example

```
SELECT EMP_ID, CITY
FROM EMPLOYEE_TBL
GROUP BY CITY, EMP_ID;
```

The SQL statement selects the EMP_ID and the CITY from the EMPLOYEE_TBL and groups the data returned by the CITY and then RMP_ID. Note the order of the column selected, versus the order of the columns in the GROUP BY clause.

For example.

```
SELECT EMP_ID, SUM(SALARY)
FROM EMPLOYEE_PAY_TBL
GROUP BY SALARY, EMP_ID;
```

This SQL statement returns the EMP_ID and the total of the salary groups, as well as groups both the salaries and employee IDs.

For example.

```
SELECT SUM(SALARY)
FROM EMPLOYEE_PAY_TBL;
```

This SQL statement returns the total of all the salaries from the EMPLOYEE_PAY_TBL.

For example,

```
SELECT SUM(SALARY)
FROM EMPLOYEE_PAY_TBL
GROUP BY SALARY;
```

This SQL statement returns the totals for the different groups of salaries.

In this first example, you can see that there are three distinct cities in the EMPLOYEE_TBL table.

For example,

```
SELECT CITY
```

```
FROM EMPLOYEE_TBL;
```

This would result in:

```
CITY
......
GANGANAGAR
IMPHAL
NAINITAL
IMPHAL
IMPHAL
IMPHAL
6 rows selected.
```

In the following example, you select the city and a count of all records for each city. You see a count on each of the three distinct cities because you are using a GROUP BY clause.

```
SELECT CITY, COUNT(*)
FROM EMPLOYEE_TBL
GROUP BY CITY;
```

This would give the following result:

```
CITY          COUNT(*)
GANGANAGAR    1
IMPHAL        4
NAINITAL      1
3 rows selected.
```

The following is a query from a temporary table created based on EMPLOYEE_TBL and EMPLOYEE_PAY_TBL.

```
SELECT *
FROM EMP_PAY_TMP;
```

This is what you get as an output:

```
CITY          LAST_NAM    FIRST_NAM    PAY_RATE    SALARY
GANGANAGAR    SHARMA      POONAM                   30000
IMPHAL        PANDIT      LALI           14.75
NAINITAL      GANJU       BABY                     400000
IMPHAL        GANJU       JOHN                     20000
IMPHAL        WALIA       MARY             11
IMPHAL        SANGAR      TARUN            15
6 rows selected.
```

In the following example, you retrieve the average pay rate and salary on each distinct city using the aggregate function AVG. There is no average pay rate for GANGANAGAR or NAINITAL, because no employees living in those cities are paid hourly.

For example,

NOTES

NOTES

```
SELECT CITY, AVG(PAY_RATE), AVG(SALARY) FROM EMP_PAY_TMP
       GROUP BY CITY;
```

This would give the following result:

```
CITY          AVG(PAY_RATE)      AVG(SALARY)
GANGANAGAR                              30000
IMPHAL           13.5833333             20000
NAINITAL                                40000
3 rows selected.
```

In the next example, you combine the use of multiple components in a query to return grouped data. You still want to see the average pay rate and salary, but only for IMPHAL and NAINITAL. You group the data by CITY, of which you have no choice because you are using aggregate functions on the order columns. Lastly, you want to order the report by 2, and then 3, which is the average pay rate and then average salary.

Study the following details and output. For example,

```
SELECT CITY, AVG(PAY_RATE), AVG(SALARY)
FROM EMP_PAY_TMP
WHERE CITY IN ('IMPHAL', 'NAINITAL')
GROUP BY CITY
ORDER BY 2,3;
```

This would result in the following:

```
CITY          ACG(PAY_RATE)      AVG(SALARY)
IMPHAL           13.5833333             20000
NAINITAL                                40000
```

Values are sorted before NULL values; therefore, the record for IMPHAL was displayed first. GANGANAGAR was not selected, but if it were, its record would have been displayed before NAINITAL's record because GANGANAGAR's average salary is Rs.30,000 (the second sort in the ORDER BY clause was on average salary).

The last example in this section shows the use of the MAX and MIN aggregate functions with the GROUP BY clause.

For example,

```
SELECT CITY, MAX(PAY_RATE), MIN(SALARY).
FROM EMP_PAY_TMP
GROUP BY CITY;
```

This would result in the following:

```
CITY          MAX(PAY_RATE)      MIN(SALARY)
GANGANAGAR                              30000
IMPHAL                    15            20000
NAINITAL                                40000
3 rows selected.
```

NOTES

Representing Columns Names with Numbers

Unlike the ORDER BY clause the GROUP BY clause cannot be ordered by using an integer to represent the column name – except when using a UNION and the column names are different.

The following is an example of representing columns names with numbers:

```
SELECT EMP_ID, SUM(SALARY)
FROM EMPLOYEE_PAY_TBL
UNION
SELECT EMP_ID, SUM(PAY_RATE)
FROM EMPLOYEE_PAY_TBL
GROUP BY 2, 1;
```

This SQL statement returns the employee ID and the group totals for the salaries. When using the UNION operator, the results of the two SELECT statements are merged into one result set. The GROUP BY is performed on the entire result set. The order for the groupings is 2 representing salary and 1 representing EMP_ID.

GROUP BY versus ORDER BY

You should understand that the GROUP BY clause works the same as the ORDER BY clause in that both are used to sort data. The ORDER BY clause is specifically used to sort data from a query; the GROUP BY clause also sorts from a query to properly group the data. Therefore, the GROUP BY clause can be used to sort data the same as ORDER BY.

There are some differences and disadvantages of using GROUP BY for sorting operations:

- All non-aggregate columns selected must be listed in the GROUP BY clause.
- Integers cannot be used in the GROUP BY to represent columns after the SELECT keyword, similar to using the ORDER BY clause.
- The GROUP BY clause is generally not necessary unless using aggregate functions.

An example of performing sort operations utilizing the GROUP BY clause in place of the ORDER BY clause is shown next:

```
SELECT LAST_NAME, FIRST_NAME, CITY
FROM EMPLOYEE_TBL
GROUP BY LAST_NAME;
```

This would result in the following:

```
SELECT LAST_NAME, FIRST_NAME, CITY
ERROR at line 1:
ORA - 00979: not a GROUP BY expression
```

In this example, an error was received from the database server stating that FIRST_NAME is not a GROUP BY expression. Remember that all columns and expressions in the SELECT must be listed in the GROUP BY clause, with the exception of aggregate columns (those columns targeted by an aggregate function).

NOTES

In the next example, the previous problem is solved by adding all expression in the SELECT to the GROUP BY clause:

For example,

```
SELECT LAST_NAME, FIRST_NAME, CITY
FROM EMPLOYEE_TBL
GROUP BY LAST_NAME, FIRST_NAME, CITY;
```

This would result in the following:

```
LAST_NAM        FIRST_NAM       CITY
GANJU           BABY            NAINITAL
GANJU           JOHN            IMPHAL
PANDIT          LALI            IMPHAL
SANGAR          TARUN           IMPHAL
SHARMA          POONAM          GANGANAGAR
WALIA           MARY            IMPHAL
6 rows selected.
```

In this example, the same columns were selected from the same table, but all columns in the GROUP BY clause are listed as they appeared after the SELECT keyword. The results were ordered by LAST_NAME first, FIRST_NAME second and CITY third. These results could have been accomplished easier with the ORDER BY clause; however, it may help you better understand how the GROUP BY works if you can visualize how it must first sort data to group data results.

The following example shows a SELECT from EMPLOYEE_TBL and uses the GROUP BY to order by CITY, which leads into the next example.

```
SELECT CITY, LAST_NAME
FROM EMPLOYEE_TBL
GROUP BY CITY, LAST_NAME;
```

This would result in the following:

```
CITY                LAST_NAM
GANGANAGAR          SHARMA
IMPHAL              GANJU
IMPHAL              PANDIT
IMPHAL              SANGAR
IMPHAL              WALIA
NAINITAL            GANJU
6 rows selected.
```

Notice the order of data in the previous results, as well as the LAST_NAME of the individual for each CITY.

All employee records in the EMPLOYEE_TBL table are now counted and the results are grouped by CITY but ordered by the count on each city first.

For example,

```
SELECT CITY, COUNT(*)
```

```
FROM EMPLOYEE_TBL
GROUP BY CITY
ORDER BY 2,1;
```

This would result in the following:

```
CITY          COUNT(*)
GANGANAGAR          1
NAINITAL            1
IMPHAL              4
```

NOTES

Notice the order of the results. The results were first sorted by the count on each city (1-4) and then by city. The count for the first two cities in the output is 1. Because the count is the same, which is the first expression in the ORDER BY clause, the city is then sorted; GANGANAGAR is placed before NAINITAL.

Although GROUP BY and ORDER BY perform a similar function, there is one major difference. The GROUP BY is designed to group identical data, while the ORDER BY is designed merely to put data into a specific order. GROUP BY and ORDER BY can be used in the same SELECT statement, but must follow a specific order. The GROUP BY clause is always placed before the ORDER BY clause in the SELECT statement.

The GROUP BY clause can be used in the CREATIVE VIEW statement to sort data, but the ORDER BY clause is not allowed in the CREATE VIEW statement.

Like operator

The LIKE operator is used to compare a value which is similar to values given by the wildcards. There are 2 wildcards which are used here. They are"

The percent sign (%)

The underscore sign (_)

Here the percent represents zero, one, or multiple characters. The underscore represents a single number of character. Both can be used together. If we use the LIKE command with the following options, we would get the results as shown.

```
WHERE ROLLNO LIKE "100%"
```

It finds the roll number that start with 100.

```
WHERE ROLLNO LIKE "%100%"
```

It finds the roll number that have 100 in any position.

```
WHERE ROLLNO LIKE "_00"
```

It finds the roll number that have 00 in the second and third position.

```
WHERE ROLLNO LIKE "2_%_%"
```

It finds the roll number that starts with 2 and are at least 3 characters in length.

```
WHERE ROLLNO LIKE "%2"
```

It finds the roll number that ends with 2.

```
WHERE ROLLNO LIKE "_2%3"
```

It finds the roll number that have a 2 in the second position and end with a 3.

```
WHERE ROLLNO LIKE "2__3"
```

It finds the roll number in five digit format that start with 2 and end up with 3.

Insert

Read about it later in the chapter.

NOTES

Update

Read about it later in the chapter.

Delete

Read about it later in the chapter.

DATA TYPES

Both PL/SQL and Oracle have their foundations in SQL. Most PL/SQL data types are native to Oracle's data dictionary. Hence, there is a very easy integration of PL/SQL code with the Oracle Engine.

The default data types that can be declared in PL/SQL are number (for storing numeric data), Char (for storing character data), date (for storing date and time data), Boolean (for storing TRUE, FALSE or NULL), number, char and date data types can have NULL values.

The % TYPE attribute provides for further integration. PL/SQL can use the %TYPE attribute to declare variables based on definitions of columns in a table. Hence, if a column's attributes change, the variable's attributes will change as well. This provides for data independence, reduces maintenance costs and allows programs to adapt to changes made to the table.

% TYPE declares a variable or constant to have the same data type as that of a previously defined variable or of a column in a table or in a view. When referencing a table, user may name the table and column or name the owner, the table and column.

NOT NULL causes creation of a variable or a constant that cannot have a null value. If an attempt is made to assign the value NUL to a variable or a constant that has been assigned a NOT NULL constraint, Oracle senses the exception condition automatically and an internal error is returned.

Variables

Variables in PL/SQL blocks are named variables. A variable must begin with a character and can be followed by a maximum of 29 other characters.

Reserved words cannot be used as variable names unless enclosed within double quotes. Variables must be separated from each other by at least one space or by a punctuation mark.

Case is insignificant when declaring variable names. A space cannot be used in a variable name. A variable of any data type either native to the Oracle Engine such as number, char, date and so on or native to PL/SQL such as Boolean (i.e. logical variable content) can be declared.

Assigning Values to variables

The assigning of a value of a variable can be done in two ways

- Using the assignment operator : = (i.e. a colon followed by an equal to sign).
- Selecting or fetching table data values into variables.

NOTES

Constants

Declaring a constant is similar to declaring a variable except that the keyword constant must be added to the variable name and a value immediately assigned. Thereafter, no further assignments to the constant are possible, while the constant is within the scope of the PL/SQL block.

Raw

Raw types are used to store binary data. Character variables automatically converted between character sets by Oracle, if necessary. These are similar to char variables, except that they are not converted between character sets. It is used to store fixed length binary data. The maximum length of a raw variable is 32,767 bytes. However, the maximum length of a database raw column is 255 bytes.

Long raw is similar to long data, except that PL/SQL will not convert between character sets. The maximum length of a long raw variable is 32,760 bytes. The maximum length of a long raw column is 2 GB.

Rowid

This data type is the same as the database Rowid pseudo-column type. It can hold a rowed, which can be considered as a unique key for every row in the database. Rowids are stored internally as a fixed length binary quantity, whose actual fixed length values depending on the operating system.

Various DBMS_ROWID functions are used to extract information about the ROWID pseudo-column. Extended and Restricted are two rowed formats. Restricted is used mostly to be backward compatible with previous versions of Oracle. The extended format takes advantages of new Oracle features.

The DBMS_ROWID package has several procedures and functions to interpret the ROWIDs of records. The following table shows the DBMS_ROWID functions.

Function	*Description*
ROWID_VERIFY	Verifies if the ROWID can be extended; 0 = can be converted to extended format; 1 = cannot be converted to extended format
ROWID_TYPE	0 = ROWID, 1 = extended
ROWID_BLOCK_NUMBER	The block number that contains the record, 1 = extended ROWID
ROWID_OBJECT	The object number of the object that contains the record.
ROWID_RELATIVE_FNO	The relative file number contains the record
ROWID_ROW_NUMBER	The row number of the record.

NOTES

ROWID_TO_ABSOLUTE_FNO	The absolute file number; use need to input rowid_val, schema and object; the absolute file number is returned.
ROWID_TO_EXTENDED	Converts the ROWID from restricted to extended; user need to input restr_rowid, schema, object; the extended number is returned.
ROWID_TO_RESTRICTED	Converts the ROWID from extended to restricted.

ROWID is a pseudo-column that has a unique value associated with each record of the database.

The DBMS_ROWID package is created by the

ORACLE_HOME/RDBMS/ADMIN/DBMSUTIL.SQL script. This script is automatically run when the Oracle instance is created.

LOB Types

A company may decide that some comments about each of its vendors must be stores along with their details. This must be stored along with all the other details that they have on a particular vendor. This can be done in Oracle with the help of LOB types.

The LOB types are used to store large objects. A large object can be either a binary or a character value upto 4 GB in size. Large objects can contain unstructured data, which is accessed more efficiently than long or long raw data, with fewer restrictions. LOB types are manipulated using the DBMS_LOB package. There are four types of LOBs:

- BLOB (Binary LOB) – this stores unstructured binary data upto 4 GB in length. A blob could contain video or picture information.
- CLOB (character LOB) – this stores single byte characters upto 4GB in length. This might be used to store documents.
- BFILE (Binary File) – this stores a pointer to read only binary data stored as an external file outside the database.

Of these LOBs, BFILE is an external to the database. Internal objects store a locator in the Large Object column of a table. Locator is a pointer that specifies the actual location of LOB stored outside the database. The LOB locator for BFILE is a pointer to the location of the binary file stored by the operating system. The BDMS_LOB package is used to manipulate LOBs. Oracle supports data integrity and concurrency for all the LOBs except BFILE as the data is stored outside the database.

Storage for LOB data

The area required to store the LOB data can be specified at the time of creation of the table that includes the LOB column. The create table command has a storage clause that specifies the storage characteristics for the table. The Syntax for this is :

```
CREATE TABLE <tablename> (<columnname> <datatype> <size()>
        <columnname> <datatype> <size()>, <columnname>
        CLOB,....);
```

NOTES

Logical Comparisons

PL/SQL supports the comparison between variables and constants in SQL and PL/SQL statements. These comparisons, often called Boolean expressions, generally consist of simple expressions separated by relational operators (<, >, =, <>, >=, <=) that can be connected by logical operators (AND, OR, NOT). A Boolean expression will always evaluate to TRUE, FALSE or NULL.

Variable Declarations

Communication with the database takes place through variables in the PL/SQL block. Variables are memory locations, which can store data values. As the program runs the contents of variables can and do change. Information from the database can be assigned to a variable or the contents of a variable can be inserted into the database.

Variables can also be modified directly by PL/SQL commands. These variables are declared in the declarative section of the block. Every variable has a specific type as well, which describes what kind of information can be stored in it.

Declaration Syntax

Variables are declared in the declarative section of the block. The general syntax for declaring a variable is

```
Variable_name type [CONSTANT] [NOT NULL] [:= value];
```

Where variable_name is the name of the variable, type is the type and value is the initial value of the variable. For example, the following are legal variable declarations:

```
DECLARE
V_Description VARCHAR2(50);
V_NumberSeats NUMBER := 45;
V_Counter  BINARY_INTEVER := 0;
```

Any legal PL/SQL identifier can be used as a variable name. VARCHAR2, NUMBER and BINARY_INTEGER are valid PL/SQL types. In this example, v_NumberSeats and v_Counter are both initialized to 45 and 0, respectively. If a variable is not initialized, such as v_Description, it is assigned NULL by default. If NOT NULL is present in the declaration the variable must be initialized as it is defined. Furthermore, it is illegal to assign NULL to a variable constrained to be NOT NULL, either when it is declared or in the executable or exception section of the block. The following declaration is illegal because v_TempVar is constrained to be NOT NULL, but is not initialized:

```
DECLARE
V_TempVar NUMBER NO NULL;
```

We can correct this by assigning a default value to v_TempVar, for example:

```
DECLARE
V_TempVar NUMBER NOT NULL := 0;
```

If CONSTANT is present in the variable declaration, the variable must be initialized and its value cannot be changed from this initial value. A constant variable is treated as read-only for the remainder of the block. Constants are often used for values that are known when the block is written, for example:

```
DECLARE
C_MinimumStudentID CONSTANT NUMBER(5) := 10000;
```

If desired, the keyword DEFAULT can be used instead of := as well;

```
DECLARE
V_NumberSeats NUMBER DEFAULT 45;
V_Counter BINARY_INTEGER DEFAULT 0;
V_FirstName VARCHAR2 (20) DEFAULT 'Scott';
```

There can be only one variable declaration per line in the declarative section. The following section is illegal, because two variables are declared on the same line:

```
DECLARE
V_FirstName, v_LastName VARCHAR(20);
```

The correct version of this block would be

```
DECLARE
V_FirstName VARCHAR(20);
V_LastName VARCHAR(20);
```

Variable Initialization

Many languages do not define what uninitialized variables contain. As a result, uninitialized variables can contain random of unknown values at runtime. In these languages, leaving uninitialized variables is not good programming style. In general, it is best to initialize a variable if its value can be determined.

PL/SQL however, does define what an uninitializd variable contains – it is assigned NULL. NULL simply means "missing or unknown value." As a result it is logical that NULL is assigned by default to any uninitialized variable. This is a unique feature of PL/SQL. Many other programming languages (C and Ada included) do not define the value for uninitialized variables. Other languages (such as Java) require that all variables be initialized.

Displaying User Messages On the VDU Screen

Programming tools require a method through which messages can be displayed to the user on the VDU screen.

DBMS_OUTPUT is a package that includes a number of procedures and functions that accumulate information in a buffer so that it can be retrieved later. These functions can also be used to display messages to the user.

PUT_LINE puts a piece of information in the package buffer followed by an end-of-line marker. It can also be used to display a message to the user. Put_line expects a single parameter of character data type. If used to display a message, it is the message string.

To display message to the user, the SERVEROUTPUT should be set to ON. SERVEROUTPUT is a SQL * PLUS environment parameter that displays the information passed as a parameter to the PUT_LINE function. The Syntax for this is:

```
Set ServerOutput [ON/OFF]
```

NOTES

Comments

A comment have two forms, as:

- The comment line begins with a double hyphen (—). The entire line will be treated as comment.
- The comment line begins with a slash followed by an asterisk (/*) till the occurrence of an asterisk followed by a slash (*/). All lines within are treated as comments. This form of specifying comments can be used to span across multiple lines. This technique can also be used to enclose a section of a PL/SQL block that temporarily needs to be isolated and ignored.

NOTES

Control Structure

The flow of control statements can be classified into the following categories:

- Conditional Control
- Iterative Control
- Sequential Control

Conditional Control

PL/SQL allows the use of an If statement to control the execution of a block of code. In PL/SQL the IF – THEN – ELSEIF – ELSE – END IF construct in code blocks allow specifying certain conditions under which a specific block of code should be executed. The Syntax for this is :

```
IF <condition> THEN
   <Action>
ELSEIF <condition> THEN
   <Action>
ELSE
   <Action>
END IF;
```

Example:

Write a PL/SQL code block that will accept a Client_no from the user and adds the amount of Rs. 100 to the Bal_due column if the Bal_due column has a minimum balance of Rs. 5000. The process is fired on the Client_Master table.

```
DECLARE
/* Declaration of memory variables and constants to be
        used in the Execution section.*/
   Bal_due number(11,2);
   mClient_no varchar2(7);
   add_amt number(4) := 100;
   MIN_BAL constant number(7,2) := 5000.00;

BEGIN
/* Accept Client_no from the user*/
```

NOTES

```
mClient_no := &mClient_no;

/* Retrieving the balance from the Client_Master table
     where the Client_no in the table is equal to the
     Client_no entered by the user.*/
   SELECT Bal_due
   FROM Client_Master
   WHERE Client_no = mClient_no;

/* Checking if the resultant balance is greater than or
     equal to the minimum balance of Rs. 5000. If the
     condition is satisfied an amount of Rs. 100 is
     added in the balance due of the corresponding
     Client_no.*/

   IF Bal_due >= MIN_BAL THEN
     UPDATE Client_Master
     SET Bal_due = Bal_due + add_amt
     WHERE Client_no = mClient_no;
   END IF;
END;
```

Iterative Control

Iterative control indicates the ability to repeat or skip sections of a code block. A loop marks a sequence of statements that has to be repeated. The keyword loop has to be placed before the first statement in the sequence of statements to be repeated, while the keyword end loop is placed immediately after he last statement in the sequence. Once a loop begins to execute, it will go on forever. Hence a conditional statement that controls the number of times a loop is executed always accompanies loops.

PL/SQL supports the following structures for iterative control:

Simple Loop

In simple loop, the key word loop should be placed before the first statement in the sequence and the keyword end loop should be written at the end of the sequence to end the loop. The Syntax for this is:

```
Loop
   <Sequence of statements>
End loop:
```

Example:

```
DECLARE
   i number := 10
BEGIN
   LOOP
```

```
      i  := 1 + 2
      EXIT WHEN I = 10
   END LOP
   dbms_output.put_line(to_char(i));
END;
```

NOTES

The WHILE loop

Syntax:

```
WHILE <condition>
Loop
   <Action>
End loop:
```

Example:

Write a PL/SQL code block to calculate the area of a circle for a value of radius varying from 3 to 7. Store the radius and the corresponding values of calculated area in an empty table named Areas, consisting of two columns Radius and Area.

Table Name : Areas

Radius Area

```
DECLARE
/* Declaration of memory variables and constants to be
      used in the Execution section. */
   pi constant number(4,2) := 3.14;
   radius number(5);
   area number(14,2);

BEGIN
/* Initialize the radius to 3, since calculations are
      required for radius to 3 to 7 */
   radius :=3;

/* Set a loop so that it fires till the radius value
      reaches 7 */
   WHILE radius <= 7
   LOOP
/* Area calculation for a circle */
   area := pi * power(radius,2);

/* Insert the value for the radius and its corresponding
      area calculated in the table */
INSERT INTO areas VALUES (radius, area);

/* Increment the value of the variable radius by 1 */
```

NOTES

```
    radius := radius + 1;
END LOOP;
END;
```

The above PL/SQL code block initializes a variable radius to hold the value of 3. The area calculations are required for the radius between 3 and 7. The value for area is calculated first with radius 3 and the radius and area are inserted into the table Areas. Now, the variable holding the value of radius is incremented by 1, i.e. it now holds the value 4. Since the code is held within a loop structure, the code continues to fire till the radius value reaches 7. Each time the value of radius and area is inserted into the areas table.

After the loop is completed the table will now hold the following:

Table name : Areas

Radius	Area
3	28.26
4	50.24
5	78.5
6	113.04
7	153:86

The FOR Loop

The Syntax for this is:

```
FOR variable IN [RESERVE] start..end
Loop
   <Action>
END LOOP;
```

Example:

Write a PL/SQL block of code for inverting a number 5639 to 9365.

```
DECLARE
   /* Declaration of memory variables and constants to be
        used in the Execution section. */
   given_number varchar(5) := '5639';
   str_length number(2);
   inverted_number varchar(5);

BEGIN
/* Store the length of the given number */
   str_length := length(given_number);
/* Initialize the loop such that it repeats for the number
        of times equal to the length of the given number.
        Also, since the number is required to be inverted,
        the loop should consider the last number first
        and store it i.e. in reverse order  */
```

```
   FOR cntr IN REVERSE 1..str_length
/* Variables used as counter in the for loop need not be
      declared i.e. cntr declaration is not required */

   LOOP
/* The last digit of the number is obtained using the
      substr function and stored in a variable, while
      retaining the previous digit stored in the variable
      */
   inverted_number    :=    inverted_number    ||
      substr(given_number, cntr, 1);
END LOOP;
/* Display the initial number, as well as the inverted
      number which is stored in the variable on screen
      */
   dbms_output.put_line ('The Given number is ' ||
      given_number );
   dbms_output.put_line ('The Inverted number is ' ||
      inverted_number );
END;
```

NOTES

The above PL/SQL code block stores the given number as well its length in two variables. Since the FOR loop is set to repeat till the length of the number is reached and in reverse order, the loop will fire 4 times beginning form the last digit, i.e., 9. This digit is obtained using the function SUBSTR and stored in a variable. The loop now fires again to fetch and store the second last digit of the given number. This is appended to the last digit stored previously. This repeats till each digit of the number is obtained and stored. The resultant display after execution of the PL/SQL code will be

Output:

The Given number is 5639

The Inverted number is 9365

Sequential Control

The GOTO statement

The GOTO statement changes the flow of control within a PL/SQL block. This statement allows execution of a section of code, which is not in the normal flow of control. The entry point into such a block of code is marked using the tags <<userdefined name>>. The GOTO statement can then make use of this user-defined name to jump into that block of code for execution.

The Syntax for this is:

```
GOTO <codeblock name>;
```

Example:

Write a PL/SQL block of code to achieve the following: if the price of product

NOTES

P00001' is less than 500, then change the price to 500. The price change is to be recorded in the old_price_table along with Product_no and the date on which the price was last changed.

Table Name : **Product_master**

Product_no	Sell_price
P00001	350
P00002	400
P00003	850
P00004	900
P00005	250

Table Name : **old_price_table**

Product_No	Date_change	Old_Price

```
DECLARE
/* Declaration of memory variables and constants to be
      used in the Execution section. */
   Selling_price number(10,2);

BEGIN
/* Fetch the sell_price of product_no 'P00001' into a
      variable */
   SELECT Sell_price into selling_price
      FROM Product_Master
      WHERE Product_no = 'P00001';

/* If the sell_price is less than 500, pass the execution
      control to a user labeled section of code, labeled
      as add_old_price in this example. If the price is
      equal to or greater than 500, display a message,
      giving the current sell_price of the product */
IF Selling_price < 500 THEN
   GOTO add_old_price;
   ELSE
      Dbms_output.put_line('Current Price of P00001 is'
         || selling_price);
   END IF;

/* A labeled section of code which updates the sell_price
      of product 'P00001' to 500. The product_no, current
      date and the old_price are inserted in to the
      table old_price_table and a message displaying
      the new price is displayed. */
```

```
<<add_old_price>>
   UPDATE Product_Master
      SET Sell_price = 500
      WHERE Product_no = 'P00001';
   INSERT INTO old_price_table
      (Product_no, Date_change, Old_price)
         VALUES ('P00001', sysdate, selling_price);
   Dbms_output.put_line('The new Price of P00001 is
      500');
END;
```

NOTES

The PL/SQL code first fetches the first fetches the Sell_price of the Product_no 'P00001' into a variable selling_price. It then checks whether the value that is held in the variable Selling_price it is less than 500. If so, the control is passed to a different section of code, labeled as add_old_price. In this block of code, the value of sell_price for product_no 'P00001' in the Product_Master table is updated to 500. Also, the Product_no, the current date and the Old_price are inserted into the old_price_table that keeps an audit trail of the change made to the product_master table.

In case the Sell_price for Product_no 'P00001' in the Product_Master table is already equal or greater than 500, a message stating the current price of the product 'P00001' is displayed.

PL/SQL Control Structures

PL/SQL, like other third-generation languages, has a variety of control structures that allow you to control the behavior of the block as it runs. These structures include conditional statements and loops. It is these structures, combined with variables, that give PL/SQL its power and flexibility.

IF-THEN-ELSE

The syntax for an IF-THEN-ELSE statement is

```
IF Boolean_expression1 THEN
   Sequence_of_statements;
   [ELSEIF Boolean_expression2 THEN
   sequence_of_statements;]
   ...
   [ELSE
   sequence_of_statements;]
   END IF]
```

where Boolean_expression is any expression that evaluates to a Boolean value, defined in the previous section, "Boolean Expressions," The ELSEIF clauses are optional and there can be many ELSEIF clauses as desired. For example, the following block shows an IF-THEN-ELSE statement with one ELSEIF clause and one ELSE clause:

```
DECLARE
   V_NumberSeats rooms.number_seats%TYPE;
   V_Comment VARCHAR(35);
```

NOTES

```
BEGIN
   /* Retrieve the number of seats in the room identified
        by ID 20008.
      Store the result in v_NumberSeats. */
   SELECT number_seats
      INTO v_NumberSeats
      FROM rooms
      WHERE room_id = 20008;
   IF v_NumberSeats < 50 THEN
      V-Comment := 'Fairly small';
   ELSEIF v_NumberSeats < 100 THEN
      V_Comment := 'A little bigger';
   ELSE
      V_Comment := 'Lots of room';
   END IF;
END;
```

The behavior of the preceding block is the same as the keywords imply. If the first condition evaluates to TRUE, the first sequence of statements is executed. In this case, the first condition is

```
V_NumberSeats < 50
```

And the first sequence of statement is

```
V_Comment := 'Fairly small';
```

If the number of seats is not less than 50, the second condition

```
V_NumberSeats < 100
```

Is evaluated. If this evaluates to TRUE, the second sequence of statements

```
V_Comment := 'A little bigger';
```

Is executed. Finally, if the number of seats is not less than 100, the final sequence of statements

```
V_comment := "Lots of room';
```

Is executed. Each sequence of statements is executed only if its associated Boolean condition evaluates to TRUE.

In the example, each sequence of statements has only one procedural statement. However, in general, you can have as many statements (procedural or SQL) as desired. The following block illustrates this:

```
   V_NumberSeats  rooms.number_seats%TYPE;
   V_Comment  VARCHAR(35);
BEGIN
   /* Retrieve the number of seats in the room identified
        by ID 20008.
      Store the result in v_NumberSeats. */
   SELECT number_seats
```

```
      INTO v_NumberSeats
      FROM rooms
      WHERE room_id = 20008;
   IF v_NumberSeats < 50 THEN
      V_Comment := 'Fairly small';
   INSERT INTO temp_table (char_col)
      VALUES ('Nice and cozy');
   ELSEIF v_NumberSeats < 100 THEN
      V_Comment := 'A little bigger';
      INSERT INTO temp_table (char_col)
         VALUES ('Some breathing room');
   ELSE
      V_Comment := 'Lots of room';
   END IF;
END;
```

NOTES

Loops

PL/SQL provides a facility for executing statements repeatedly, via loops. Loops are divided into four categories. Simple loops, WHILE loops and numeric FOR loops are loops that are discussed in the following sections.

Simple Loops

The most basic kind of loops, simple loops, have the syntax

```
LOOP
   Sequence_of_statements;
END LOOP;
```

The sequence_of_statements will be executed infinitely, because this loop has no stopping condition. However, we can add one with the EXIT statement, which has the following syntax:

```
EXIT [WHEN condition];
```

For example, the following block inserts 50 rows into temp_table.

```
DECLARE
   V_counter BINARY_INTEGER := 1;
BEGIN
   LOOP
      -- Insert a row into temp_table with the current
         value of the loop counter.
      INSERT INTO temp_table
         VALUES (v_counter, 'Loop index');
      V_Counter := v_Counter + 1;
      -- Exit condition - when the loop counter > 50 as will
      -- break out of the loop.
```

```
      IF v_Counter > 50 THEN
         EXIT;
      END IF;
   END LOOP;
END;
```

NOTES

The statement

```
EXIT WHEN condition;
```

Is equivalent to

```
IF condition THEN
   EXIT;
END IF;
```

So, we can rewrite the example with the following block, which behaves exactly the same way:

```
DECLARE
   V_Counter BINARY_INTEGER := 1;
BEGIN
   LOOP
      - Insert a row into temp_table with the current
        value of the
      - loop counter.
      INSERT INTO temp_table
         VALUES (v_Counter, 'Loop index');
      V_Counter := v_Counter + 1;
      - Exit condition - when the loop counter > 50 we will
      - break out of the loop.
      EXIT WHEN v_Counter > 50;
   END LOOP;
END;
```

WHILE Loops

The syntax for a WHILE loop is

```
WHILE condition LOOP
   Sequence_of_statements;
END LOOP;
```

The condition is evaluated before each iteration of the loop. If it evaluates to TRUE, sequence_of_statements is executed. If condition evaluates to FALSE or NULL, the loop is finished and control resumes after the END LOOP statement. Now we can rewrite the example using a WHILE loop as follows:

```
DECLARE
   V_Counter BINARY_INTEGER := 1;
BEGIN
```

```
    - Test the loop counter before each loop iteration to
    - insure that it is still less than 50
    WHILE V_Counter <= 50 LOOP
       INSERT INTO temp_table
          VALUES (v_Counter, 'Loop index');
       V_Counter := v_Counter + 1;
    END LOOP;
END;
```

NOTES

The EXIT or EXIT WHEN statement can still be used inside a WHILE loop to exit the loop prematurely, if desired.

Keep in mind that if the loop condition does not evaluates to TRUE the first time it is checked, the loop is not executed at all. If we remove the initialization of v_Counter in our example, the condition v_Counter < 50 will evaluate to NULL, and no rows will be inserted into temp_ table:

```
DECLARE
   V_Counter BINARY_INTEGER;
BEGIN
   - this condition will evaluate to NULL, since V_Counter
   - is initialized to NULL by default.
   WHILE v_Counter <= 50 LOOP
      INSERT INTO temp_table
         VALUES (v_Counter, 'Loop index');
      V_Counter := v_Counter + 1;
   END LOOP;
END ;
```

Numeric FOR Loops

The number of iterations for simple loops and WHILE loops is not known in advance – it depends on the loop condition. Numeric FOR loops, on the other hand, have defined number of iterations. The syntax is

```
FOR loop_counter IN [REVERSE] low_bound..high_boud LOOP
   Sequence_of_statements;
END LOOP;
```

Where loop_counter is an implicitly declared index variable, low_bound and high_bound specify the number of iterations and sequence_of_statements is the content of the loop.

The bounds of the loop are evaluated once. The determines the total number of iterations that loop_counter will take on the values ranging from low_bound to high_bound, incrementing by 1 each time until the loop is complete. We can rewrite our looping example using a FOR loop as follows:

```
BEGIN
   FOR v_Counter IN 1..50 LOOP
```

```
        INSERT INTO temp_table
          VALUES (v_Counter, 'Loop Index');
    END LOOP;
END;
```

NOTES

Scooping Rules

The loop index for a FOR loop is implicitly declared as a BINARY_INTEGER. It is not necessary to declare it prior to the loop. If it is declared, the loop index will hide the outer declaration in the same way that a variable declaration in an inner block can hide a declaration in an outer block. See the following example:

```
DECLARE
   V_Counter NUMBER := 7;
BEGIN
   - Inserts the value 7 into temp_table.
   INSERT INTO temp_table (num_col)
      VALUES (v_counter);
   - This loop redeclares v_Counter as a BINARY_INTEGER,
      which hides
   - the NUMBER declaration of v_Counter.
   FOR v_Counter IN 20..30 loop
   - inside the loop, v_Counter ranges from 20 to 30.
   INSERT INTO temp_table (num_col)
      VALUES (v_Counter);
   END LOOP;
   - Inserts another 7 into temp_table.
   INSERT INTO temp_table (num_col)
      VALUES (v_Counter);
END;
```

Using REVERSE

If the REVERSE keyword is present in the FOR loop, the loop index will iterate from the high value to the low value. Note in the following example that the syntax is the same – the low value is still referenced first:

```
BEGIN
   FOR v_Counter in REVERSE 10..50 LOOP
      - v_Counter will start with 50 and will be decremented
        by
      - 1 each time through the loop.
      NULL;
   END LOOP;
END;
```

Loop Ranges

The high how value don't have to be numeric literals. They can be any expression that can be converted to a numeric value. Here is an example:

```
DECLARE
   V_LowValue NUMBER := 10;
   V_HighValue  NUMBER := 40;
BEGIN
   FOR v_Counter IN REVERSE v_LowValue .. v_HighValue
        LOOP
      INSERT INTO temp_table
         VALUES (v_Counter, 'Dynamically specified loop
         ranges');
      END LOOP;
END;
```

NOTES

CREATE TABLE

This command allows you to create a schema under which all the controls will be mentioned.

This is done using the syntax.

```
CREATE TABLE NewTable (NewValue) INT)
```

Executing this statement creates a new table named New Table with a column named New Value that takes integer data. If you want to add additional column, simply add the additional column names followed by their data type so that each pairing of name and data type between the parentheses is separated by a comma, as follows:

```
CREATE TABLE NewTable (NewValue INT, NextValue VARCHAR(6))
```

Since creating a table entails creating columns and creating columns requires the use of data types, it's useful to review the data types available. The following table provides a way of reacquainting yourself with the data types which are available with SQL.

Name	*Storage*
binary	Up to 8,000 bytes of binary data
bit	Integers 0 or 1
char	A fixed-length string of characters not encoded as Unicode
datetime	Date and time values from 1/1/175 to 12/31/9999
decimal	Numbers containing decimal fractions from -10^ 38-1 to 10^38-1
float	Floating-point decimals from -1.79E+308 to 1.79E+308
image	A variable-length string of bits (binary data) with a maximum size of 2^31
int	Integers from -2^31-1 to 2^31-1
money	Numbers representing monetary values from -2^63 to 2^63-1
nchar	A fixed-length string of characters encoded as Unicode

NOTES

ntext	A variable-length string of characters not encoded as Unicode with a maximum size of 2^31-1
nvachar	A variable-length string of characters encoded as Unicode
numeric	Same as decimal
real	Floating-point decimals from -3.40E+38 to 3.40E+38
smalldatetime	Date and time values from 1/1/1900 to 6/6/2079
smallint	integers from -2^15-1 to 2^15-1
smallmoney	Numbers representing monetary values from -214,748.3648 to 214,748.3647
text	A variable-length string of characters not encoded as Unicode with a maximum size of 2^31-1
timestamp	A unique number in the database
tinyint	Integers from 0 to 255-failed copy
verbinary	A variable-length string of bits (binary data) with a maximum size of 8,000 bytes
varchar	A variable-length string of characters not encoded as Uicode
uniqueidentifier	A globally unique identifier (GUID), that is, a number in the world.

One thing that you have to consider is the ultimate size of your database and to certain extent, its speed. In general, you want to plan your tables by choosing table names and column names that are descriptive and self-documenting. People have to remember how to use your tables and columns and table names and column do not take up large amounts of space. This statement should not suggest, however, that 256-character column names are a good idea. The data type for a blank occurrence of that column. Many SQL databases require on database creation that you set the maximum file size for the database. Under these circumstances, you want to choose your data types wisely so that you do not bump up against that maximum size too quickly.

In choosing data types, therefore, apply the following suggestions:

- Choose varchar over char whenever possible. Char (256) sets aside 1 filled byte and 255 blank bytes for the valye "A." Varchar (256), while it does incur slight overhead to allow variable-length strings, only uses the number of bytes it requires for the string.
- If you absolutely know that you will never ever need to store a Unicode string, use the standard data types (like char) instead of the data types that begin with "n" (line nchar). Storing Unicode incurs some additional storage, because you are not using a character set that can be represented by a single byte. However, you need to be absolutely certain that you will need to store Unicode, because you won't be able to if you try.
- Use a tiny data type or a small data type whenever it makes sense to do so. For example, you can use tinyint, smallint or int. The int data type takes the most storage, because it has to be prepared to store very large integers. The

smallint data type restricts the possible range of integers and takes less storage. The tinyint type takes the least storage but has a maximum value of 255.

- Avoid the binary and next data types whenever possible. If you need to store a collection of binary large objects (BLOBs), such as a collection of pictures, consider storing the path to the graphics file in the database and the BLOB in a folder set aside for holding the BLOB files. Your database will be smaller, storage for the BLOBs will actually be less, because you won't have the overhead the database wraps around the BLOB to keep track of it and response time for queries will be faster. The same is true for large chunks of text.

NOTES

In addition to planning data types, you need to consider default value and nullability when planning column layouts in tables. The default value is what is placed in the column when a row is created if no value for the column is provided. To provide a default value, you use the default keyword as follows:

```
CREATE TABLE NewTable (NewValue INT DEFAULT 0)
```

The column NewValue now takes a value of 0 if no other value is provided for it by an INSERT statement. A default value of 0 is somewhat redundant for any numeric data type, because the data type itself usually defaults the column to 0.

However, there are many instances where 0 is not the appropriate default, such as when you want to specify a guaranteed interest rate for a life insurance contract. In addition, you need to specify whether the column can be null, that is, whether it can contain no data at all. You do so using the following syntax:

```
CREATE TABLE NewTable (NewText CHAR(6) NULL)
```

This table definition states that the column NewText can be null, which will be its value if an INSERT does not supply another value. You can also specify a default value for a nullable column as follows:

```
CREATE TABLE NewTable (NewText CHAR(6) NULL DEFAULT
        'ABCDEF')
```

Here, an INSERT that does not provide a value for this column creates a row that contains 'ABCDEF' in the column NewText. However, another INSERT statement could later set the value of this column to NULL if we so desired. If your database supports filenames or filegroups that represent the files that containing the data on the disk, you can usually specify which files to use in creating a table using the ON keyword. You should check you database's documentation for the exact conventions to use, because they can differ. Most tables have more than one or two columns, so let's take a brief look at a table definition:

```
CREATE TABLE Friends
(
   friend_id char(4) NOT NULL,
   friend_name  varchar(40)   NULL,
   city       varchar(20)     NULL,
   state      char(2)         NULL,
   country    varchar(30)     NULL,
              DEFAULT("INDIA")
)
```

NOTES

Things to note here are:

- The use of identification to make the statement more readable.
- Each column has its own line.
- Any additional lines necessary for a given column are indented well underneath that column name.

In addition, note that you can use NOT NULL to indicate that a column absolutely cannot contain NULL as a value.

DROP TABLE

You can also delete tables with the DROP statement. The syntax is straightforward:

```
DROP TABLE [dbo].[friends]
```

Keep in mind that DROP needs a keyword to identify what you want to delete, in this case a table. Following this keyword is the table name. DROP is final and does not prompt you to have your sanity checked before you carry through the deletion. Make sure you have the names right. If there is data in the table, it will be deleted along with the table.

ALTER TABLE

As you might guess, ALTER TABLE is a statement that changes an existing table. Its basic syntax is as follows:

```
ALTER TABLE MyTable ADD MyColumn VARCHAR (20) NULL
```

This form of the statement adds a column. As you can see, the name of the table follows the ALTER TABLE keywords and then the action to be taken follows. This example adds two constraints to a table; it was taken from a script that SQL Server generated for building the table:

```
ALTER TABLE [dbo].[sales] ADD
    FOREIGN KEY
    (
        [F_id]
    ) REFERENCES (dbo).[names] (
        [F_id]
    ),
    FOREIGN KEY
    (
        [name_id]
    ) REFERENCES (dbo).[address] (
        [name_id]
    )
```

You can also drop a column or a constraint using syntax like the following:

```
ALTER TABLE MyTable DROP COLUMN MYColumn
```

You use one ADD keyword and each item to add follows in a comma-separated list. Use separate ALTER TABLE statements for each intended action. ALTER TABLE statements can become confusing when they are complex and complex ALTER TABLEs can lead to accidental and unintended consequences. Keep them as simple and straightforward as possible.

NOTES

INSERT INTO

INSERT can be used in serveral ways. The most important one is to add data in the database. Various options of INSERT are shown next.

Remember: ***Do not forget that SQL statements can be in upper- or lowercase. The data, depending on how it is stored in the database, is not case-sensitive. These examples use both lower- and uppercase just to show that is does not affect the outcome.***

Inserting Data into a Table

Use the INSERT statement to insert new data into a table. There are a few options with the INSERT statement: look at the following basic syntax to beigin:

```
INSERT INTO SCHEMA.TABLE_NAME
VALUES ('value1', 'Value2', [ NULL ] );
```

Using this INSERT statement syntax, you must include every column in the specified table in the VALUES list. Notice that each value in this list is separated by a comma. The values inserted into the table must be enclosed by quotation marks for character and date data types. Quotation marks are not required for numeric data types of NULL values using the NULL keyword. A value should be present for each column in the table.

In the following example, you insert a new record into the PRODUCTS_TBL table.

Table structure

```
products.tbl
```

Column Name	Null?	Data Type
PROD_IDNOT	NULL	VARCHAR2(10)
PROD_DESC	NOT NULL	VARCHAR2(25)
COST	NOT NULL	NUMBER(6,2)

Sample INSERT statement.

```
INSERT INTO PRODUCTS_TBL
VALUES ('7725', 'LEATHER GLOVES', 24.99);
```

You will get the output as:

```
1 row created.
```

In this example, you insert three values into a table with three columns. The inserted values are in the same order as the columns listed in the table.

STUDENT ACTIVITY 2.2

1. What are Entity Types and Entity Sets?

2. Write a short note on Relationship among Entities.

3. Describe the working of First normal form with example.

4. What do you understand by Second normal form?

The first two values are inserted using quotation marks, because the data types of the corresponding columns are of character type.

The third value's associated column, COST, is a numeric data type and does not require quotation marks, although they can be used.

NOTES

Tip: ***The schema name, or table owner, has not been specified as part of the table name, as it was shown in the syntax. The schema name is not required if you are connected to the database as the user who owns the table.***

Inserting Data into Limited columns of a Table

There is a way you can insert data into a table's limited columns. For instance, suppose you want to insert all values for an employee except a pager number. You must, in this case, specify a column list as well as a VALUES list in your INSERT statement.

```
INSERT INTO EMPLOYEE_TBL
(EMP_ID, LAST_NAME, FIRST_NAME, MIDDLE_NAME, ADDRESS, CITY,
        STATE, PIN, PHONE)
VALUES
(`123456789', `TENDULKAR', `SACHIN', `RAMESH', `123 JUHU
        BEACH ROAD', `MUMBAI', `MAHARASHTRA', `400001',
        `9810223293');
```

In this case you will get the output as:

```
1 row created
```

The syntax for inserting values into a limited number of columns in a table is as follows:

```
INSERT INTO SCHEMA TABLE_NAME (`COLUMN1', `COLUMN2')
VALUES (`VALUE1', `VALUE2');
```

You use ORDERS_TBL and insert values into only specified columns in the following example:

Table structure

```
ORDERS_TBL
```

Column Name	Null?	Data Type
ORD_NUMNOT	NULL	VARCHAR2(10)
CUST_ID	NOT NULL	VARCHAR2(10)
PROD_IDNOT	NULL	VARCHAR2(10)
QTY	NOT NULL	NUMBER(4)
ORD_DATE		DATE

Sample INSERT statement

```
INSERT INTO ORDERS_TBL (ORD_NUM, CUST_ID, PROD_ID, QTY)
VALUES (`23A16', `109', `7725', 2)
```

You will get the output as:

```
1 row created
```

NOTES

You have specified a column list enclosed by parentheses after the table name in the INSERT statement. You have listed all columns into which you want to insert data. ORD_DATE is the only excluded column. You can see, if you look at the table definition, that ORD_DATE does not require data for every record in the table. You know that ORD_DATE does not require data because NOT NULL is not specified in the table definition. NOT NULL tells us that NULL values are not allowed in the column. Furthermore, the list of values must appear in the order in which you want to insert them according to the column list.

Remember: ***The column list in the INSERT statement does not have to reflect the same order of columns as in the definition of the associated table, but the list of values must be in the order of the associated columns in the column list.***

Inserting Data from Another Table

You can insert data into a table based on the results of a query from another table using a combination of the INSERT statement and the SELECT statement. A query is a question that the user asks the database, and the data returned is the answer. In the case of combining the INSERT statement with the SELECT statement, you are able to insert the data retrieved from a query into a table.

The syntax for inserting data from another table is:

```
INSERT NTO SCHEMA TABLE_NAME [(`COLUMN1', `COLUMN2')]
SELECT (*.|(`COLUMN1', `COLUMN2')]
FROM TABLE_NAME
[WHERE CONDITION(S)]
```

You see three new keywords in this syntax, which are covered here briefly. These keywords are SELECT, FROM, and WHERE. SELECT is the main command used to initiate a query in SQL. FROM is the clause in the query that specifies the names of tables in which the target data should be found. The WHERE clause, also part of the query is used to place conditions on the query itself. An example condition may state: WHERE NAME = 'SACHIN'.

A condition is a way of placing criteria on data affected by a SQL statement.

The following example uses a simple query to view all data in the PRODUCTS_TBL table. SELECT * tells the database server that you want information on all columns of the table. Because there is no WHERE clause, you want to see all records in the table as well. For example,

```
SELECT * FREE PRODUCTS_TBL;
```

The output of this would be as follows:

```
PROD-ID          PROD_DESC                        COST
11235            WITCHES COSTUME                 29.99
222              PLASTIC PUMPKIN 18 INCH          7.75
90               LIGHTED LANTERNS                 14.5
15               ASSORTED COSTUMES                 0.0
9                CANDY CORN                       1.35
```

NOTES

```
6                PUMPKIN CANDY                    1.45
87               PLASTIC SPIDERS                  1.05
119              ASSORTED MASKS                   4.95
1234             KEY CHAIN                        5.95
2345             OAK BOOKSHELF                   59.99
11 rows selected.
```

Now, insert values into the PRODUCTS_TMP table based on the preceding query. You can see that 11 rows are created in the temporary table.

Your input here would be:

```
INSERT INTO PRODUCTS_TMP
SELECT * FROM PRODUCTS_TBL;
```

The result of this would be:

```
11 rows selected
```

The following query shows all data in the PRODUCTS_TMP table that you just inserted.

The input here would be:

```
SELECT * FROM PRODUCTS_TMP;
```

And the result would be:

```
PROD-ID          PROD_DESC                        COST
11235            WITCHES COSTUME                 29.99
222              PLASTIC PUMPKIN 18 INCH          7.75
90               LIGHTED LANTERNS                 14.5
15               ASSORTED COSTUMES                10.0
9                CANDY CORN                       1.35
6                PUMPKIN CANDY                    1.45
87               PLASTIC SPIDERS                  1.05
119              ASSORTED MASKS                   4.95
1234             KEY CHAIN                        5.95
2345             OAK BOOKSHELF                   59.99
11 rows selected.
```

Inserting NULL Values

Inserting a NULL value into a column of a table is a simple matter. You might want to insert a NULL value into a column if the value of the column in question is unknown. For instance, not every person carries a pager, so it would be inaccurate to enter an erroneous pager number—not to mention, you would not be budgeting space. A NULL value can be inserted into a column of a table using the keyword NULL.

The syntax for inserting a NULL value follows:

```
INSERT INTO SCHEMA.TABLE_NAME VALUES
('COLUMN1', NULL, 'COLUMN3');
```

The NULL keyword should be used in the associated column that exists in the table.

That column will not have data in it for that row if you enter NULL. In the syntax, a NULL value is being entered in the place of COLUMN2.

NOTES

Study the two following examples:

```
INSERT INTO ORDERS_TBL (ORD_NUM, COST_ID, PROD_ID, QTY,
       ORD_DATE)
VALUES (`23A16', `109', `7725', 2, NULL);
```

The output of this would be:

```
1 row created.
```

In the first example, all columns in which to insert values are listed, which also happen to be every column in the ORDERS_TBL table. You insert a NULL value for the ORD_DATE column, meaning that you either do not know the order date, or there is no order date at this time.

```
INSERT INTO ORDERS_TBL
VALUES (`23A16', `109', `7725', 2, `');
```

The result of this would be:

```
1 row created.
```

There are two differences from the first statement in the second example, but the results are the same. First, there is not a column list. Remember that a column list is not required if you are inserting data into all columns of a table. Second, instead of inserting the value NULL into the ORD_DATE column, you insert '' (two single quotation marks together), which also symbolizes a NULL value (because there is nothing between them.)

DELETE FROM

The DELETE command is used to remove entire rows of data from a table. The DELETE command is not used to remove values from specific columns; a full record, including all columns, is removed. The DELETE statement must be used with caution—it works all too well.

To delete a single record or selected records from a table, the DELETE statement must be used with the following syntax:

```
DELETE FROM SCHEMA.TABLE_NAME
(WHERE CONDITION);
```

For example,

```
DELETE FROM ORDERS_TBL
WHERE ORD-NUM = `23A16';
```

The output of this would be:

```
1 row deleted.
```

Notice the use of the WHERE clause. The WHERE clause is an essential part of the DELETE statement if you are attempting to remove selected rows of data from a table. You rarely issue a DELETE statement without the use of the WHERE clause. If you do, your results are similar to the following example.

```
DELETE FROM ORDERS_TBL;
11 rows deleted.
```

NOTES

Tip: ***If the WHERE clause is omitted from the DELETE statement, all rows of data are deleted from the table. As a general rule, always use a WHERE clause with the DELETE statement.***

Tip: ***The temporary table that was populated from the original table earlier can be very useful for testing the DELETE and UPDATE commands before issuing them again the original table.***

UPDATE

This statement is used to update the existing data in the table. Pre-existing data in a table can be modified using the UPDATE command. The UPDATE command does not add new records to a table, nor does it remove records—it simply updates existing data. The update is generally used to update one table at a time in a database, but can be used to update multiple columns of a table at the same time. An individual row of data in a table can be updated, or numerous rows of data can be updated in a single statement, depending on what's needed.

Updating the Value of a Single Column

The most simple form of the UPDATE statement is its use to update a single column in a table. Either a single row of data or numerous records can be updated when updating a single column in a table.

The syntax for updating a single column follows:

```
UPDATE TABLE_NAME
SET COLUMN_NAME = `VALUE'
(WHERE CONDITION)
```

The following example updates the QTY column in the ORDERS table to the new value 1 for the ORD_NUM 23A16, which you have specified using the WHERE clause.

```
UPDATE ORDER_TBL
SET QTY = 1
WHERE ORD_NUM = `23A16';
```

The output of this would be:

```
1 row updated.
```

The following example is identical to the previous example, except for the absence of the WHERE clause:

```
WHERE ORDERS_TBL
SET QTY = 1;
```

The output of this would be:

```
11 rows updated.
```

Notice that in this example, 11 rows of data were updated. You set the QTY to 1, which updated the quantity column in the ORDERS_TBL table for all rows of data. Is this really what you wanted to do? Perhaps in some cases, but rarely will you issue an UPDATE statement without a WHERE clause.

NOTES

Caution: ***Extreme caution must be used when using the UPDATE statement without a WHERE clause. The target column is updated for all rows of data in the table if conditions are not designated using the WHERE clause.***

Updating Multiple Columns in One or More Records

Next, you see how to update multiple columns with a single UPDATE statment. Study the following syntax;

```
UPDATE TABLE_NAME
SET COLUMN1 = `VALUE',
   (COLUMN2 = `VALUE',]
   (COLUMN3 = `VALUE']
[WHERE CONDITION];
```

The output of this would be:

```
1 row updated.
```

A comma is used to separate the two columns being updated. Again, the WHERE clause is optional, but usually necessary.

Remember: ***The SET keyword is used only once for each UPDATE statement. If more than one column is to be update, a comma is used to separate the columns to be updated.***

GENERAL QUERY SYNTAX (SELECT)

SELECT commands with where clause using conditional expressions

This command accompanied by many options and clauses, is used to compose queries against a relational database. SELECT is the main command used in SQL to initiate a query. Let us see how it can be used. The basic syntax for the command is:

```
SELECT [ * | ALL | DISTINCT COLUMN1, COLUMN2 ] FROM TABLE1
       [, TABLE2 ];
```

Here the the various terms used are:

SELECT	It is the main query
FROM	It is the keyword followed by a list of one or more tables from which you want to select data
ALL	This option is used to display all vaues for a column, including duplicates.
DISTINCT	This option is used to eliminate duplicate rows. The default between DISTINCT and ALL is ALL, which may not be specified.

NOTES

It can be used in various ways. Some of them are discussed here.

Supposing you have a table of various students with Names, Roll Nos., Address, etc. You can use the following Select statement to list out only names from it.

```
SELECT NAME FROM STUDENT_TBL;
```

This would result in the following:

```
SACHIN
RAHUL
YUVRAJ
DILIP
```

Tip: ***Commas are used to separate arguments in a list in SQL statements. Some common lists of columns in a query lists of tables to be selected from in a query, values to be inserted into a table, and values grouped at a condition in a query's WHERE clause.***

Now we see the other options of SELECT statement.

Selecting All the Columns

Using the All option, you can use the statement in the following way.

```
SELECT ALL ROLLNO. FROM STUDENT_TBL;
```

This would result in the following:

```
1022
1033
1044
1055
```

These are the Roll numbers which are there in the table. Since ALL is a default option, you could have given the above command in the following way too.

```
SELECT ROLLNO. FROM STUDENT_TBL;
```

This would again have given the same result.

Selecting Specific Column

In the above case we had to select one field only. We can in fact select both Name and RollNo. fields together. Let us see how it is done.

```
SELECT ALL NAME, ROLLNO. FROM STUDENT_TBL;
```

This would result in the following:

```
SACHIN    1022
RAHUL     1033
YUVRAJ    1044
DILIP     1055
```

Similary if you want you can select the various or all fields.

CREATE VIEW

In SQL, the command to specify a view is Create View. The view is givan a table name (or view name), a list of attribute names, and a query to specify the contents of the view. If none of the view attributes results from applying functions, or arithmetic operations, we do not have to specify attribute names for the view, since they would be same as the names of the attributes of the defining ables in the default case. The views V1 and V2 are created in the following examples.

NOTES

V1

```
CREATE VIEW     WORKS_ON1
As Select       Fname, Lname, Pname, Hours
From            EMPLOYEE, PROJECT, WORKS_ON
Where           Ssn=Essn AND Pno=Pnumber
```

V2

```
CREATE VIEW     DEPT_INFO(Dept_name, No_of_emps,
                Total_sal)
As Select       Dname, COUNT (*), SUM (Salary)
From            DEPARTMENT, EMPLOYEE
Where           Dnumber=Dno
Group by        Dname;
```

In V1 we did not specify any new attribute names of the view WORKS_ON1; in this case, WORKS_ON1 inherits the names of the view attributes from the defining tables EMPLOYEE, PROJECT, and WORKS_ON. View V2 explicity specifies new attribute names for the view DEPT_INFO using a one-to-one correspondence between the attributes specified in the CREATE VIEW clause and those specified in the SELECT clause of the query that defines the view.

DROP VIEW

If we do not need a view any more, we can use DROP VIEW comand to dispose of it. For example, to get rid of the view V1, we can use the SQL statement in V1A.

```
V1A   DROP VIEW       WORKS_ON1;
```

SET OPERATORS - UNION, INTERSECT AND MINUS

You can combine multiple queries by using UNION clause. But the condition to apply union clause is that both the tables should have same structure and specify the column name of both the tables in the same sequence. The syntax for this is:

```
SELECT [STATEMENT] UNION SELECT [STATEMENT]
```

For example,

```
SELECT NAME, ROLL_NO FROM STUDENT_TBL UNION NAME, ROLL_NO
       FROM MARKS_TBL;
```

NOTES

The above statement would display name and roll number from two tables, mainly, STUDENT_TBL and MARKS_TBL. The duplicate rows would be automatically deleted by the UNION clause.

Several set theoretic operations are used to merge the elements of two sets in various ways, including UNION, INTERSECTION, and SET DIFFERENCE (also called MINUS). These are binary operations; that is, each is applied to two sets (of tuples). When these operations are adapted to relational databases, the two relations on which any of these three operations are applied must have the same type of tuples; this condition has been union compatibility. Two relations $R(A_1, A_2, \ldots A_n)$ and $S(B_1, B_2, \ldots B_n)$ are said to be union compatible if they have the same degree n and if dom(A_i) = dom (Bi) fir $1 \leq i \leq n$. This means that the two relations have the same number of attributes and each corresponding pair of attributes has the same domain.

We can define the three opearions UNION, INTERSECTION, and SET DIFFERENCE on two union-compatible relations R and S as follows:

- UNION: The result of this operation, denoted by $R \cup S$, is a relation that includes all tuples that are either in R or in S or in both R and S. Deuplicate tuples are eliminated.
- INTERSECTION: The result of this operation, denoted by $R \cap S$, is a relation that includes all tuples that are in both R and S.
- SET DIFFERENCE: (or MINUS): The result of this operation, denoted by R – S, is a relation that includes all tuples athat are in R but not in S.

We will adopt the convention that the resulting relation has the same attribute names as the *first* relation R. It is always possible to rename the attributes in the result using the rename operator.

Notice that both UNION and INTERSECTION are *commutative operations*; that is,

$$R \cup S = S \cup R \text{ and } R \cap S = S \cap R$$

Both UNION and INTERSECTION can be treated as n-ary operations applicable to any number of relations because both are *associative operations*; that is,

$$R \cup (S \cup T) = (R \cup S) \cup T \text{ and } (R \cap S) \cap T = R \cap (S \cap T)$$

The MINUS operation is *not commutative*; that is, in general,

$$R - S \neq S - R$$

FUNCTIONS

Functions are the keywords in SQL and are used to manipulate values within columns for output purposes. A function is a command always used in conjunction with a column name for expression. There are several types of functions in SQL.

Types of SQL Functions

Following are the various types of functions:

Aggregate functions

Date and Time functions

Character functions

Conversion functions

Miscellaneous functions.

Single Row Functions

NOTES

Single Row functions operate on the single row and return one result per row. They can accept one or more arguments and return one value for each row. They can be used with SELECT, WHERE, and ORDER BY clause and also they can be nested. An argument can be of the following types:

- Users supplied constant
- Variable name
- Variable value
- Expression

Single row functions can fall in the following categories:

- Character
- Number
- Date
- Conversion
- General

Character Functions

There are various types of functions in SQL. Character functions are used to modify the appearance of character values. Various character functions are: CHR, CONCAT, INITCAP, INSTR, LEFT, LENGTH, LOCATE, LOWER, LPAD, LTRIM, REPLACE, RIGHT, RPAD, RTRIM, SUBSTR/SUBSTRING, TRANSLATE, TRIM and UPPER. Some of them have been discussed earlier. Let us read about them one by one.

CHR

This function returns the character equivalent of the number it uses as an argument.

For example,

```
SELECT ROLLNO, CHR(ROLLNO) FROM STUDENT_TBL;
```

This would give the following result.

```
ROLLNO  CH
22      -
33      !
44      ,
55      7
```

CONCAT

This function was talked earlier as Concotation Operators (||).

NOTES

INITCAP

This function makes the first character to the uppercase and all other characters to lowercase.

For example,

```
SELECT NAME BEFORE, INITCAP(FIRSTNAME) AFTER FROM
       STUDENT_TBL;
```

This would give the following result.

```
BEFORE  AFTER
SACHIN  Sachin
RAHUL   Rahul
YUVRAJ  Yuvraj
DILIP   Dilip
```

INSTR

This function is used to find out where in a string a particular pattern occurs. For example,

```
SELECT NAME, INSTR(NAME, 'A', 2, 1) FROM STUDENT_TBL;
```

This would give the following result.

```
NAME    INSTR(LASTNAME, 'A', 2, 1)
SACHIN  2
RAHUL   2
YUVRAJ  5
DILIP   0
```

In Sachin, A is at 2nd position, so in Rahul. In the case of Yuvraj it is there on the 5th position, while in Dilip it is not there, hence 0 is returned.

LEFT

This function returns the leftmost character from the string.

For example,

```
SELECT LEFT(NAME, 3) FROM STUDENT_TBL;
```

This would give the following result.

```
NAME    LEFT(NAME, 3)
SACHIN  SAC
RAHUL   RAH
YUVRAJ  YUV
DILIP   DIL
```

The first 3 characters from the left are selected.

LENGTH()

This function is used to find the length of a string, number, date, or expression in bytes. The syntax is:

```
LENGTH (CHARACTER STRING)
```

For example,

```
SELECT LENGTH(NAME) FROM STUDENT_TBL
```

This would return the number of characters in the Name field.

LOCATE

NOTES

This function is used to the first occurence of the substring in the string. It returns a 0 if the string is not there.

For example,

```
SELECT LOCATE('AC', 'SACHIN');
```

This would give the following result.

```
2
```

In Sachin, AC appears at the 2nd position, that is why 2 has been returned.

LOWER

It has been discussed earlier.

LPAD

See RPAD.

LTRIM

It is used to clip a part of a string. It is used to trim characters from the left of a string. The syntax is:

```
LTRIM(CHARACTER STRING [ , 'set'])
```

For example,

```
SELECT LTRIM(NAME, 'SA') FROM STUDENT_TBL
```

This would remove SA from the name, if it is there. The net result would be:

```
SACHIN      1022     MUMBAI
RAHUL       1033     BANGALORE
YUVRAJ      1044     DELHI
DILIP       1055     KOLKATA
```

In the first record, it was found and thus SA has been trimmed.

REPLACE

It has been discussed earlier.

RIGHT

This function returns the rightmost character from the string.

For example,

```
SELECT RIGHT(NAME, 3) FROM STUDENT_TBL;
```

This would give the following result.

```
NAME     RIGHT(NAME, 3)
SACHIN   HIN
RAHUL    HUL
```

```
YUVRAJ   RAJ
DILIP    LIP
```

The first 3 characters from the right are selected.

RPAD()

NOTES

PAD and TRIM are used twice over with Left and Right. So you have functions as LPAD and RPAD. I am going to give them separately. In fact, what this function does is that it fills up the extra space in the field with the padding either from the left or from the right. The syntax is:

```
LPAD(CHARACTER  STRING)
RPAD(CHARACTER  STRING)
```

For example,

```
SELECT  LPAD(NAME)  FROM  STUDENT_TBL
```

Supposing we have the field length of 10 in the case of NAME and the data we have, as in the case of SACHIN, only 6 characters, the rest 4 characters would be filled up with padding records. The net result would be:

```
....SACHIN  1022      MUMBAI
.....RAHUL  1033      BANGALORE
....YUVRAJ  1044      DELHI
.....DILIP  1055      KOLKATA
```

Similarly if the command had been given for right padding, the result would have been.

```
SACHIN....  1022      MUMBAI
RAHUL.....  1033      BANGALORE
YUVRAJ....  1044      DELHI
DILIP.....  1055      KOLKATA
```

RTRIM

It is similar to LTRIM but the trimming of characters starts from the right. The syntax is:

```
RTRIM(CHARACTER STRING [ ,' 'set'])
```

For example,

```
SELECT  LTRIM(NAME,  'UL')  FROM  STUDENT_TBL
```

This would remove UL from the name, if it is there as the last 2 characters. The net result would be:

```
SACHIN      1022      MUMBAI
RAH         1033      BANGALORE
YUVRAJ      1044      DELHI
DILIP       1055      KOLKATA
```

In the second record, it was found and thus UL has been trimmed.

SUBSTR

It is used to take out a set of characters from a string. The syntax is:

```
SUBSTR(COLUMN  NAME,  STARTING  POSITION,  LENGTH)
```

Here Column Name is the name of the field from the characters have to be obtained, STARTING POSITION is the position from where the displaying would start and LENGTH is the number of characters it would display. For example,

```
SELECT SUBSTR(NAME, 2, 2) FROM STUDENT_TBL
```

This would start from the second character and display the next 2 characters. The net result would be:

```
CH          1022      MUMBAI
AH          1033      BANGALORE
VR          1044      DELHI
LI          1055      KOLKATA
```

Notice that all other fields remain the same.

TRANSLATE

This was discussed earlier.

TRIM

See LTRIM and RTRIM.

UPPER

This was discussed earlier.

Case Conversion Functions

Various functions under this category are: LOWER, INITCAP and UPPER. All of them have been discussed earlier.

Character Manipulation Function

Various functions under this category are: CONCAT and INSTR. They have been discussed earlier.

Number Functions

Following is the list of number functions arranged in the alphabetical order and what they do.

ABS

This function returns the absolute value. The syntax for this function is:

```
ABS (value)
```

ACOS

This function returns arc cosine of value. The syntax for this function is:

```
ACOS(value)
```

ASIN

This function returns arc sin of value. The syntax for this function is:

```
ASIN(value)
```

NOTES

NOTES

ATAN

This function returns arc tangent of value. The syntax for this function is:

```
ATAN(value)
```

CEIL

This function returns smallest integer larger than or equal to value. The syntax for this function is:

```
CEIL(value)
```

COS

This function returns cosine of the value. The syntax for this function is:

```
COS(value)
```

COSH

This function returns hyperbolic cosine of the value. The syntax for this function is:

```
COSH(value)
```

EXP

This function returns e raised to the valueth power.

The syntax for this function is:

```
EXP(n)
```

FLOOR

This function returns largest integer smaller than or equal to value. The syntax for this function is:

```
FLOOR(value)
```

LN

This function returns natural (base e) logarithm of value. The syntax for this function is:

```
LN(number)
```

LOG

This function returns base logarithm of value. The syntax for this function is:

```
LOG(base, number)
```

MOD

This function returns modulus of value divided by divisor. The syntax for this function is:

```
MOD(value, divisor)
```

NVL

This function returns substitute for value if value is NULL. The syntax for this function is:

```
NVL(value, substitute)
```

POWER

This function returns value raised to an exponent. The syntax for this function is:

```
POWER(value, exponent)
```

ROUND

This function returns rounding of value to precision. The syntax for this function is:

```
ROUND (date, 'format')
```

SIGN

This function returns 1 if value is positive, –1 if negative, 0 if zero. The syntax for this function is:

```
SIGN(value)
```

SIN

This function returns sine of value. The syntax for this function is:

```
sin(value)
```

SINH

This function returns hyperbolic sine of value. The syntax for this function is:

```
SINH(value)
```

SQRT

This function returns square root of value. The syntax for this function is:

```
SQRT(value)
```

TAN

This function returns tangent of value. The syntax for this function is:

```
TAN(value)
```

TANH

This function returns hyperbolic tangent of value. The syntax for this function is:

```
TANH(value)
```

TRUNC

This function returns value truncated to precision. The syntax for this function is:

```
TRUNC(value,precision)
```

VSIZE

This function returns storage size of value in Oracle. The syntax for this function is:

```
VSIZE(value)
```

Working with Dates

Date is already stored in computer. You can recall it and then manipulate using the following functions as per your need.

LAST_DAY

This function is used to return the last of a specified month. It is there supposing you forget how many days are there in that particular month.

For example,

```
SELECT ENDDATE, LAST_DAY(ENDDATE) FROM EMP_TBL;
```

NOTES

NOTES

This would give the following, depending upon the data you have in EMP_TBL.

```
ENDDATE        LASTDAY(ENDDATE)
01-01-2003     31-01-2003
01-02-2003     28-02-2003
01-03-2003     31-03-2003
```

MONTHS_BETWEEN()

This function is used to return the number of months elapsed between two months. For example,

```
SELECT STARTDATE, ENDDATE, MONTHS_BETWEEN(STARTDATE,
       ENDDATE) DURATION FROM PROJECT;
```

This would gvie the following, depending upon the data you have in EMP_TBL.

```
ENDDATE        LASTDAY(ENDDATE)  DURATION
01-01-2003     30-01-2003        .93548387
01-02-2003     01-02-2003        0
01-03-2003     15-03-2003        .48387097
```

NEXT_DAY()

This function is used to return the first day of the week that is equal to or later than another specified date.

For example,

```
SELECT STARTDATE, NEXT_DAY(STARTDATE, 'MONDAY') FROM
       EMP_TBL;
```

This would give the following, depending upon the data you have in EMP_TBL.

```
STARTDATE      NEXTDATE
01-01-2003     06-01-2003
01-02-2003     03-02-2003
01-03-2003     03-03-2003
```

ADD_MONTHS()

This function is used to add a number of months to a specified date.

For example,

```
SELECT STARTDATE, ENDDATE ORIGINAL, ADD_MONTHS(ENDDATE,3)
       FROM EMP_TBL;
```

This would give the following, depending upon the data you have in EMP_TBL.

```
STARTDATE      ORIGINAL      ADD_MONTH
01-01-2003     31-01-2003    30-04-2003
01-02-2003     28-02-2003    31-05-2003
01-03-2003     31-03-2003    30-06-2003
```

ROUND()

It rounds off date according to format. Various formats available for rounding are:

Format	Meaning
cc,scc	century (rounds up to January 1st of next century, as of midnight exactly on the morning of January 1st 1950, 2050 and so on)
syear,syyy,y,yy,yyy,yyyy and year	year(rounds up to January 1st of the next year as of midnight exactly on the morning of July 1st)
q	quarter (rounds up in the 2nd month of the quarter as of midnight exactly on the morning of the 16th, regardless of the number of days in the month)
month,mon,mm	month (rounds up as of midnight exactly on the morning of the 16th regardless of the number of days in the month)
ww	rounds to closest Monday
w	rounds to closest day which is the same day as the first day of the month
ddd,dd,,j	rounds up to the next day as of noon exactly. This is the same as ROUND with no format
day,dy,d	rounds up to next Sunday (first day of the week) as of noon exactly on Wednesday
hh,hh12,hh24	rounds up to the next whole hour as of 30 minutes and 30 seconds after the hour
mi	rounds up to the next whole minute as of 30 seconds of this minute.

NOTES

TRUNC

It truncates number to precision.

Arithmetic Operation on Dates

Date functions similar to character string functions, are used to manipulate the representation of data and time. Among the various functions which can be performed on the Date are; format the date and time in the required format, compare date values with one another, compare intervals between dates, etc.

Date Functions and their Usuage

Date is used in calculations too. It is possible that you have to convert its characters first for using it.

Converting Dates to Character Strings

You can convert date to month, date, year. For this we have the function called TO_CHAR.

For example,

```
SELECT ENDDATE TO_CHAR(ENDDATE, "Month dd, yyyt')
       'DATE_CHAR' FROM PROJECT_TBL;
```

This would return the following, depending upon the data available in PROJECT_TBL.

```
ENDDATE      DATE_CHAR
01-01-2003   January 1, 2003
```

```
01-02-2003    February 1, 2003
01-03-2003    March 1, 2003
```

Converting Character Strings to Dates

NOTES

This is the reverse of the above. Well, it is possible that you may need to do this too. Here the function to be used is TO_DATE.

For example,

```
SELECT TO_DATE('JANUARY 01 2003', 'MONTH DD YYYY') FROM
       EMPLOYEE_TBL;
```

This would give you the following result.

```
TO_DATE('
01-JAN-2003
01-JAN-2003
01-JAN-2003
01-JAN-2003
```

This will depend on the type of data you have in your EMPLOYEE_TBL.

Data type Conversion Functions

As we have seen in the case of dates above, the conversion sometimes becomes necessary. But the conversion is not limited to dates only. You need to convert data also sometimes. The functions are TO_CHAR and TO_NUMBER.

Implicit Conversions

When the Oracle Server automatically converts data to the expected datatype, the conversion is called implicit convesion. For an assignment the Oracle Server can automatically convert the following:

From	*To*
VARCHAR2ORCHAR	NUMBER
VARCHAR2ORCHAR	DATE
NUMBER	VARCHAR2
DATE	VARCHAR2

For expression evaluation, the Oracle Server can automatically convert the following:

From	*To*
VARCHAR2ORCHAR	NUMBER
VARCHAR2ORCHAR	DATE

Explicit Conversion

Explicit datatype conversion are done by using the conversion functions. Conversion functions convert one datatype to another. SQL provides the three explicit conversion functions listed below:

Function	*Purpose*
`TO_CHAR(number | date, [fmt], [nlparams])`	Converts a number or date value to a VARCHAR2 character

string with format model fmt. nlparams parameter specifies the following characters, which are returned by number format elements:

- Decimal character
- Group separator
- Local currency symbol
- International currency symbol

```
TO_NUMBER(char, [fmt], [nlsparams])
```

Converts a character string containing digits to a number in the format specified by the optional format model fmt. The nlsparams parameter has the same purpose as in the TO_CHAR function for number conversion.

```
TO_DATE(char, [fmt], [nlsparams])
```

Converts a character string representing a date to a date value according to the fmt specified. If fmt is omitted, the format is DD-MON-YY. The nlsparams parameter has the same purpose in this function as in the TO_CHAR function for date conversion.

TO_CHAR Function with Dates

This has been discussed earlier.

TO_CHAR Function For Numbers

This function can be used for converting a number into a character.

TO_NUMBER

This function is used to convert a character string to its numerical equivalent. In the following example, we have converted the NAME to number and then multiplied with another number to get the result.

```
SELECT NAME, TESTNUM, TESTNUM*TO_NUMBER(NAME) FROM
    STUDENT_TBL;
```

This would give you the following result.

```
NAME        TESTNUM          TESTNUM*TO_NUMBER(NAME)
13          23               299
40          95               3800
74          68               5032
```

TO_DATE Functions

This is used to convert text into a Date format. The syntax is similar to TO_CHAR.

```
TO_DATE(expression, 'date_picutre')
```

For example,

```
SELECT TO_DATE('20030101', 'yyyymmdd') "NEW DATE" FROM
    STUDENT_TBL;
```

This would give the following result.

```
NEW DATE
01-JAN-2003
```

NOTES

Or you can have the following format.

```
SELECT TO_DATE('20030101', 'yyyy/mm/dd') "NEW DATE" FROM
       STUDENT_TBL;
```

This would give the following result.

```
NEW DATE
01/JAN/2003
```

Valid Date, Time and Other Formats

Following is the list of valid date, time and other formats:

Element	*Description*
DATE FORMATS	
SCC or CC	Century; S prefixes BC date with -
Years in dates YYYY or SYYYY	Year; S prefixes BC date with -
YYY or YY or Y	Last three, two or one digits of year
Y,YYY	Year with comma in this position
[YYY,IYY,IY,I	Four, three, two or one digit year based on the ISO standard
SYEAR or YEAR	Year spelled out; S prefixes BC date with -
BC or AD	BC/AD indicator
B.C. or A.D.	
Q	Quarter of year
MM	Month, two-digit value
MONTH	Name of month, padded with blanks to length of nine characters
MON	Name of month, three-letter abbreviation
RM	Roman numerical month
WW or W	Week of year or month
DDD or DD or D	Day of year; month or week
DAY	Name of day padded with blanks to length of 9 characters
DY	Name of day; three-letter abbreviation
J	Julian day; the number of days since 31 December 4713 BC
TIME FORMATS	
AM or PM	Meridian indicator
A.M. or P.M.	Meridian indicator with periods

HH or HH12 or HH24	Hour of day or hour (1-12) or hour (0-23)
MI	Minute (0-59)
SS	Second(0-59)
SSSSS	Seconds past midnight (0-86399)

OTHER FORMATS

/.,	Punctuation is reproduced in the result
"of the"	Quoted string is reproduced in the result
TH	Original number (for example, DDTH for 4th)
SP	Spelled out number (for example, DDSP for FOUR)
SPTH or THSP	Spelled out ordinal numbers (for example, DDSPTH for FOURTH)

NOTES

NVL Function and its Usage

This function is used to return data from one expression if another expression is NULL. NVL can be used with most data types, however, the value and the substitue must be the same data types. The syntax for this is:

```
NVL('VLAUE', 'SUBSTITUTION')
```

For example,

```
SELECT NVL(SALARY, '00000') FROM EMPLY_TBL;
```

This statement finds NULL values and substitutes 00000 for any NULL values.

DECODE Function and its Usage

This function is used to search a string for a value or string, and if the string is found, an alternate string is displayed as part of the query results. The syntax is:

```
DECODE(COLUMN NAME, 'SEARCH1', 'RETURN1', ['SEARCH2',
        'RETURN2', 'DEFAULT VALUE'])
```

For example,

```
SELECT DECODE(LAST_NAME, 'BANGIA', 'RAMESH', 'OTHER') FROM
       STUDENT_TBL;
```

This query searches the value of all last names in EMPLOYEE_TBL; if the value BANGIA is found, RAMESH is displayed in its place. All other names are displayed as OTHER, which is called the default value.

Remember: ***When embedding functions within functions in an SQL statement, remember that the innermost function is resolved first, and then each function is subsequently resolved from the inside out.***

GROUP FUNCTIONS

Typical group functions—those that are used with the GROUP BY clause to arrange

NOTES

data in groups—include AVG, MAX, MIN, SUM, and COUNT. You will learn about them in the next para.

Types of group functions

The various type of group functions are: MAX, MIN, SUM, AVG and COUNT. They are discussed and their usage shown below.

Using MAX Function

Returns the maximum value associated with an expression. The syntax is:

```
MAX(sql_expression)
```

The sql_expression is typically a column name, although arithmetic expressions can be used. ALL and DISTINCT may be used to qualify sql_expression, although DISTINCT has no real meaning in the context of this function. There is only one maximum value for any given expression. ALL forces the function to apply to all values of the expression; it is the default mode of operation for the function. Te return type matches that of sql_expression. For example,

```
SELECT MAX (ytd_sales) FROM titles
```

Using MIN Function

Returns the minimum value associated with an expression. The syntax is:

```
MIN (sql_expression)
```

The sql_expression is typically a column name, although arithmetic expressions can be used. ALL and DISTINCT may be used to qualify sql_expression, although DISTINCT has no real meaning in the context of this function. There is only one minimum value for any given expression. ALL forces the function to apply to all values of the expression; it is the default mode of operation for the function. Te return type matches that of sql_expression. For example,

```
SELECT min(ytd_sales) FROM titles
```

Using AVG Function

Returns the average of the values in a group defined either by AVG or GROUP BY. The syntax is

```
AVG (sql_expression)
```

The sql_expression is typically a column name, but it can be a more complex expression. ALL or DISNTINCT can be used to modify the expression. The return value matches the data type of the sql_expression. For example,

```
SELECT AVG(DISTINCT price) FROM titles
```

Using SUM Function

Returns the sum of the items designated in the expression. The syntax is

```
SUM(sql_expression)
```

The sql_expression is typically a column name. However, other types of expressions can be used. Sql_expression, must represent numeric data and NULL are discarded. ALL and DISTINCT may be used to modify sql_expression and ALL is the default. DISTNCT forces the sum of only the unique values' duplicate values are discarded. The sum data type matches that of sql_expression. For example,

```
SELECT SUM(advance) FROM titles
```

Using COUNT Function

Returns the count of the number of items in a group defined by COUNT or GROUP By. The syntax is:

```
COUNT(sql_expression)
```

NOTES

The sql_expression is typically a column name, although more complex expressions are possible. ALL and DISTINCT may be used to qualify sql_expression. The return type is an integer representing the count. For example,

```
SELECT COUNT(city) FROM authors
```

Using COUNT(*)

This function is used to count all rows in a table. So when you give the command COUNT(*), it will give you the number the rows the table has.

For example,

```
SELECT COUNT(*) FROM STUDENT_TBL;
```

This will give you,

```
4
```

Since there are only 4 rows in the table, the result value 4 has been given.

DISTINCT clause with COUNT(*)

DISTINCT as we know is the command to eliminate the duplicates. So when you use this with the COUNT function, the counting is done on the number of rows which are unique and not duplicate in nature. For example,

```
SELECT DISTINCT COUNT(*) FROM STUDENT_TBL;
```

This will result in following:

```
4
```

Since there are no duplicates, the value return has been the same.

Using NVL Function with Group Functions

All the group functions expect COUNT ignore the NULL values in the column. For example,

```
SELECT AVG_SALARY FROM EMPLOYEE_TBL;
```

When the above command is executed, the AVG function will find the average of all the records excluding the ones which have the NULL values.

JOIN

An SQL JOIN clause combines records from two tables in a database. It creates a set that can be saved as a table or used as is. A JOIN is a means for combining fields from two tables by using values common to each. ANSI standard SQL specifies four types of JOINs: INNER, OUTER, LEFT, and RIGHT. In special cases, a table (base table, view, or joined table) can JOIN to itself in a self-join.

A programmer writes a JOIN predicate to identify the records for joining. If the

NOTES

evaluated predicate is true the combined record is then produced in the expected format, for example a record set or a temporary table.

Sample tables

All subsequent explanations on join types in this article make use of the following two tables. The rows in these tables serve to illustrate the effect of different types of joins and join-predicates. In the following tables, Department. DepartmentID is the primary key, while Employee.DepartmentID is a foreign key.

Employee Table

LastName	DepartmentID
Rafferty	31
Jones	33
Steinberg	33
Robinson	34
Smith	34
Jasper	NULL

Department Table

DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing

Note: The "Marketing" Department currently has no listed employees. Employee "Jasper" has not been assigned to any Department yet.

Inner join

An inner join requires each record in the two joined tables to have a matching record. An inner join essentially combines the records from two tables (A and B) based on a given join-predicate. The result of the join can be defined as the outcome of first taking the Cartesian product (or cross-join) of all records in the tables (combining every record in table A with every record in table B) - then return all records which satisfy the join predicate. Actual SQL implementations will normally use other approaches where possible, since computing the Cartesian product is not very efficient. This type of join occurs most commonly in applications, and represents the default join-type.

SQL specifies two different syntactical ways to express joins. The first, called "explicit join notation", uses the keyword JOIN, whereas the second uses the "implicit join notation". The implicit join notation lists the tables for joining in the FROM clause of a SELECT statement, using commas to separate them. Thus, it specifies a cross-join, and the WHERE clause may apply additional filter-predicates. Those filter-predicates function comparably to join-predicates in the explicit notation.

NOTES

One can further classify inner joins as equi-joins, as natural joins, or as cross-joins (see below).

Programmers should take special care when joining tables on columns that can contain NULL values, since NULL will never match any other value (or even NULL itself), unless the join condition explicitly uses the IS NULL or IS NOT NULL predicates.

As an example, the following query takes all the records from the Employee table and finds the matching record(s) in the Department table, based on the join predicate. The join predicate compares the values in the DepartmentID column in both tables. If it finds no match (i.e., the department-id of an employee does not match the current department-id from the Department table), then the joined record remains outside the joined table, i.e., outside the (intermediate) result of the join.

Example of an explicit inner join:

```
SELECT *
FROM   employee
       INNER JOIN department
       ON employee.DepartmentID = department.DepartmentID
```

is equivalent to:

```
SELECT *
FROM   employee, department
WHERE  employee.DepartmentID = department.DepartmentID
```

Explicit Inner join result:

Employee.LastName	*Employee.DepartmentID*	*Department.DepartmentName*	*Department.DepartmentID*
Smith	34	Clerical	34
Jones	33	Engineering	33
Robinson	34	Clerical	34
Steinberg	33	Engineering	33
Rafferty	31	Sales	31

Notice that the employee "Jasper" and the department "Marketing" does not appear. Neither of these has any matching records in the respective other table: "Jasper" has no associated department and no employee has the department ID 35. Thus, no information on Jasper or on Marketing appears in the joined table. Depending on the desired results, this behavior may be a subtle bug. Outer joins may be used to avoid it.

Equi-join

An equi-join, also known as an equijoin, is a specific type of comparator-based join, or theta join, that uses only equality comparisons in the join-predicate. Using other comparison operators (such as <) disqualifies a join as an equi-join. The query shown above has already provided an example of an equi-join:

```
SELECT Employee.lastName, Employee.DepartmentID,
       Department.DepartmentName
FROM Employee INNER JOIN Department
```

NOTES

```
ON Employee.DepartmentID = Department.DepartmentID;
SQL provides optional syntactic sugar for expressing equi-
      joins, by way of the USING construct (Feature ID
      F402):
SELECT    Employee.lastName,    DepartmentID,
      Department.DepartmentName
FROM Employee INNER JOIN Department
USING(DepartmentID);
```

The USING clause is supported by MySQL, Oracle and PostgreSQL.

Natural join

A natural join offers a further specialization of equi-joins. The join predicate arises implicitly by comparing all columns in both tables that have the same column-name in the joined tables. The resulting joined table contains only one column for each pair of equally-named columns.

The above sample-query for inner joins can be expressed as a natural join in the following way:

```
SELECT *
FROM   employee NATURAL JOIN department
```

The result appears slightly different, however, because only one DepartmentID column occurs in the joined table.

DepartmentID	*Employee.LastName*	*Department.DepartmentName.*
34	Smith	Clerical
33	Jones	Engineering
34	Robinson	Clerical
33	Steinberg	Engineering
31	Rafferty	Sales

The Oracle database implementation of SQL selects the appropriate column in the naturally-joined table from which to gather data. An error-message such as "ORA-25155: column used in NATURAL join cannot have qualifier" is an error to help prevent or reduce the problems that could occur may encourage checking and precise specification of the columns named in the query, and can also help in providing compile time checking (instead of errors in query).

Cross join

A cross join, cartesian join or product provides the foundation upon which all types of inner joins operate. A cross join returns the cartesian product of the sets of records from the two joined tables. Thus, it equates to an inner join where the join-condition always evaluates to True or join-condition is absent in statement.

If A and B are two sets, then the cross join is written as A × B.

The SQL code for a cross join lists the tables for joining (FROM), but does not include any filtering join-predicate.

Example of an explicit cross join:

```
SELECT *
FROM   employee CROSS JOIN department
```

Example of an implicit cross join:

```
SELECT *
FROM   employee, department;
```

NOTES

Employee.LastName	*Employee.DepartmentID*	*Department.DepartmentName*	*Department.DepartmentID*
Rafferty	31	Sales	31
Jones	33	Sales	31
Steinberg	33	Sales	31
Smith	34	Sales	31
Robinson	34	Sales	31
Jasper	NULL	Sales	31
Rafferty	31	Engineering	33
Jones	33	Engineering	33
Steinberg	33	Engineering	33
Smith	34	Engineering	33
Robinson	34	Engineering	33
Jasper	NULL	Engineering	33
Rafferty	31	Clerical	34
Jones	33	Clerical	34
Steinberg	33	Clerical	34
Smith	34	Clerical	34
Robinson	34	Clerical	34
Jasper	NULL	Clerical	34
Rafferty	31	Marketing	35
Jones	33	Marketing	35
Steinberg	33	Marketing	35
Smith	34	Marketing	35
Robinson	34	Marketing	35
Jasper	NULL	Marketing	35

The cross join does not apply any predicate to filter records from the joined table. Programmers can further filter the results of a cross join by using a WHERE clause.

Outer joins

An outer join does not require each record in the two joined tables to have a matching

NOTES

record. The joined table retains each record—even if no other matching record exists. Outer joins subdivide further into left outer joins, right outer joins, and full outer joins, depending on which table(s) one retains the rows from (left, right, or both).

(For a table to qualify as left or right its name has to appear after the FROM or JOIN keyword, respectively.)

No implicit join-notation for outer joins exists in standard SQL.

Left outer join

The result of a left outer join (or simply left join) for table A and B always contains all records of the "left" table (A), even if the join-condition does not find any matching record in the "right" table (B). This means that if the ON clause matches 0 (zero) records in B, the join will still return a row in the result—but with NULL in each column from B. This means that a left outer join returns all the values from the left table, plus matched values from the right table (or NULL in case of no matching join predicate).

For example, this allows us to find an employee's department, but still shows the employee(s) even when their department does not exist (contrary to the inner-join example above, where employees in non-existent departments are excluded from the result).

Example of a left outer join, with the additional result row italicized:

```
SELECT *
FROM   employee  LEFT OUTER JOIN department
                        ON  employee.DepartmentID  =
       department.DepartmentID
```

Employee.LastName	*Employee.DepartmentID*	*Department.DepartmentName*	*Department.DepartmentID*
Jones	33	Engineering	33
Rafferty	31	Sales	31
Robinson	34	Clerical	34
Smith	34	Clerical	34
Jasper	NULL	NULL	NULL
Steinberg	33	Engineering	33

Right outer joins

A right outer join (or right join) closely resembles a left outer join, except with the treatment of the tables reversed. Every row from the "right" table (B) will appear in the joined table at least once. If no matching row from the "left" table (A) exists, NULL will appear in columns from A for those records that have no match in A.

A right outer join returns all the values from the right table and matched values from the left table (NULL in case of no matching join predicate).

For example, this allows us to find each employee and his or her department, but still show departments that have no employees.

Example right outer join, with the additional result row italicized:

```
SELECT *
FROM   employee RIGHT OUTER JOIN department
                    ON  employee.DepartmentID  =
       department.DepartmentID
```

NOTES

Employee.LastName	*Employee.DepartmentID*	*Department.DepartmentName*	*Department.DepartmentID*
Smith	34	Clerical	34
Jones	33	Engineering	33
Robinson	34	Clerical	34
Steinberg	33	Engineering	33
Rafferty	31	Sales	31
NULL	NULL	Marketing	35

In practice, explicit right outer joins are rarely used, since they can always be replaced with left outer joins (with the table order switched) and provide no additional functionality. The result above is produced also with a left outer join:

```
SELECT *
FROM   department LEFT OUTER JOIN employee
                    ON  employee.DepartmentID  =
       department.DepartmentID
```

Full outer join

A full outer join combines the results of both left and right outer joins. The joined table will contain all records from both tables, and fill in NULLs for missing matches on either side.

For example, this allows us to see each employee who is in a department and each department that has an employee, but also see each employee who is not part of a department and each department which doesn't have an employee.

Example full outer join:

```
SELECT *
FROM   employee
       FULL OUTER JOIN department
                    ON  employee.DepartmentID  =
       department.DepartmentID
```

Employee.LastName	*Employee.DepartmentID*	*Department.DepartmentName*	*Department.DepartmentID*
Smith	34	Clerical	34
Jones	33	Engineering	33
Robinson	34	Clerical	34
Jasper	NULL	NULL	NULL

Steinberg	33	Engineering	33
Rafferty	31	Sales	31
NULL	NULL	Marketing	35

NOTES

Some database systems (like MySQL) do not support this functionality directly, but they can emulate it through the use of left and right outer joins and unions. The same example can appear as follows:

```
SELECT *
FROM   employee
       LEFT JOIN department
                    ON  employee.DepartmentID  =
       department.DepartmentID
UNION
SELECT *
FROM   employee
       RIGHT JOIN department
                    ON  employee.DepartmentID  =
       department.DepartmentID
WHERE  employee.DepartmentID IS NULL
```

Self-join

A self-join is joining a table to itself. This is best illustrated by the following example.

Example

A query to find all pairings of two employees in the same country is desired. If you had two separate tables for employees and a query which requested employees in the first table having the same country as employees in the second table, you could use a normal join operation to find the answer table. However, all the employee information is contained within a single large table.

Considering a modified Employee table such as the following:

Employee Table

EmployeeID	LastName	Country	DepartmentID
123	Rafferty	Australia	31
124	Jones	Australia	33
145	Steinberg	Australia	33
201	Robinson	United States	34
305	Smith	United Kingdom	34
306	Jasper	United Kingdom	NULL

An example solution query could be as follows:

```
SELECT F.EmployeeID, F.LastName, S.EmployeeID, S.LastName,
       F.Country
FROM Employee F, Employee S
```

```
WHERE F.Country = S.Country
AND F.EmployeeID < S.EmployeeID
ORDER BY F.EmployeeID, S.EmployeeID;
```

Which results in the following table being generated.

Employee Table after Self-join by Country

EmployeeID	*LastName*	*EmployeeID*	*LastName*	*Country*
123	Rafferty	124	Jones	Australia
123	Rafferty	145	Steinberg	Australia
124	Jones	145	Steinberg	Australia
305	Smith	306	Jasper	United Kingdom

For this example, note that:

F and S are aliases for the first and second copies of the employee table.

The condition F.Country = S.Country excludes pairings between employees in different countries. The example question only wanted pairs of employees in the same country.

The condition F.EmployeeID < S.EmployeeID excludes pairings where the EmployeeIDs are the same.

F.EmployeeID < S.EmployeeID also excludes duplicate pairings. Without it only the following less useful part of the table would be generated (for the United Kingdom only shown):

EmployeeID	*LastName*	*EmployeeID*	*LastName*	*Country*
305	Smith	305	Smith	United Kingdom
305	Smith	306	Jasper	United Kingdom
306	Jasper	305	Smith	United Kingdom
306	Jasper	306	Jasper	United Kingdom

Only one of the two middle pairings is needed to satisfy the original question, and the topmost and bottommost are of no interest at all in this example.

Alternatives

The effect of outer joins can also be obtained using correlated subqueries. For example

```
SELECT employee.LastName, employee.DepartmentID,
       department.DepartmentName
FROM   employee LEFT OUTER JOIN department
                    ON employee.DepartmentID =
       department.DepartmentID
```

can also be written as

```
SELECT employee.LastName, employee.DepartmentID,
   (SELECT department.DepartmentName
    FROM department
```

NOTES

```
        WHERE employee.DepartmentID = department.DepartmentID
    )
FROM    employee
```

Implementation

NOTES

Much work in database-systems has aimed at efficient implementation of joins, because relational systems commonly call for joins, yet face difficulties in optimising their efficient execution. The problem arises because (inner) joins operate both commutatively and associatively. In practice, this means that the user merely supplies the list of tables for joining and the join conditions to use, and the database system has the task of determining the most efficient way to perform the operation. A query optimizer determines how to execute a query containing joins. A query optimizer has two basic freedoms:

Join order: Because joins function commutatively and associatively, the order in which the system joins tables does not change the final result-set of the query. However, join-order does have an enormous impact on the cost of the join operation, so choosing the best join order becomes very important.

Join method: Given two tables and a join condition, multiple algorithms can produce the result-set of the join. Which algorithm runs most efficiently depends on the sizes of the input tables, the number of rows from each table that match the join condition, and the operations required by the rest of the query.

Many join-algorithms treat their inputs differently. One can refer to the inputs to a join as the "outer" and "inner" join operands, or "left" and "right", respectively. In the case of nested loops, for example, the database system will scan the entire inner relation for each row of the outer relation.

One can classify query-plans involving joins as follows:

left-deep: using a base table (rather than another join) as the inner operand of each join in the plan

right-deep: using a base table as the outer operand of each join in the plan

bushy: neither left-deep nor right-deep; both inputs to a join may themselves result from joins

These names derive from the appearance of the query plan if drawn as a tree, with the outer join relation on the left and the inner relation on the right (as convention dictates).

Join algorithms: Three fundamental algorithms exist for performing a join operation.

Nested loops

Please refer to main articles: Nested loop join and block nested loop

Use of nested loops produces the simplest join-algorithm. For each tuple in the outer join relation, the system scans the entire inner-join relation and appends any tuples that match the join-condition to the result set. Naturally, this algorithm performs poorly with large join-relations: inner or outer or both. An index on columns in the inner relation in the join-predicate can enhance performance.

The block nested loops (BNL) approach offers a refinement to this technique: for

every block in the outer relation, the system scans the entire inner relation. For each match between the current inner tuple and one of the tuples in the current block of the outer relation, the system adds a tuple to the join result-set. This variant means doing more computation for each tuple of the inner relation, but far fewer scans of the inner relation.

NOTES

Merge join

If both join relations come in order, sorted by the join attribute(s), the system can perform the join trivially, thus:

Consider the current "group" of tuples from the inner relation; a group consists of a set of contiguous tuples in the inner relation with the same value in the join attribute.

For each matching tuple in the current inner group, add a tuple to the join result. Once the inner group has been exhausted, advance both the inner and outer scans to the next group.

Merge joins offer one reason why many optimizers keep track of the sort order produced by query plan operators—if one or both input relations to a merge join arrives already sorted on the join attribute, the system need not perform an additional sort. Otherwise, the DBMS will need to perform the sort, usually using an external sort to avoid consuming too much memory.

Hash join

A hash join algorithm can only produce equi-joins. The database system pre-forms access to the tables concerned by building hash tables on the join-attributes. The lookup in hash tables operates much faster than through index trees. However, one can compare hashed values only for equality, not for other relationships.

SUB QUERIES

The most common operation in SQL databases is the query, which is performed with the declarative SELECT keyword. SELECT retrieves data from a specified table, multiple related tables in a database or the result of an expression. While often grouped with Data Manipulation Language (DML) statements, the standard SELECT query is considered separate from SQL DML, as it has no persistent effects on the data stored in a database. Note that there are some platform-specific variations of SELECT that can persist their effects in a database, such as the SELECT INTO syntax that exists in some databases.

SQL queries allow the user to specify a description of the desired result set, but it is left to the devices of the database management system (DBMS) to plan, optimize, and perform the physical operations necessary to produce that result set in as efficient a manner as possible. An SQL query includes a list of columns to be included in the final result immediately following the SELECT keyword. An asterisk ("*") can also be used as a "wildcard" indicator to specify that all available columns of a table (or multiple tables) are to be returned. SELECT is the most complex statement in SQL, with several optional keywords and clauses, including:

The FROM clause which indicates the source table or tables from which the data is to be retrieved. The FROM clause can include optional JOIN clauses to join related tables to one another based on user-specified criteria.

NOTES

The WHERE clause includes a comparison predicate, which is used to restrict the number of rows returned by the query. The WHERE clause is applied before the GROUP BY clause. The WHERE clause eliminates all rows from the result set where the comparison predicate does not evaluate to True.

The GROUP BY clause is used to combine, or group, rows with related values into elements of a smaller set of rows. GROUP BY is often used in conjunction with SQL aggregate functions or to eliminate duplicate rows from a result set.

The HAVING clause includes a comparison predicate used to eliminate rows after the GROUP BY clause is applied to the result set. Because it acts on the results of the GROUP BY clause, aggregate functions can be used in the HAVING clause predicate.

The ORDER BY clause is used to identify which columns are used to sort the resulting data, and in which order they should be sorted (options are ascending or descending). The order of rows returned by an SQL query is never guaranteed unless an ORDER BY clause is specified.

The following is an example of a SELECT query that returns a list of expensive books. The query retrieves all rows from the Book table in which the price column contains a value greater than 100.00. The result is sorted in ascending order by title. The asterisk (*) in the select list indicates that all columns of the Book table should be included in the result set.

```
SELECT *
    FROM Book
    WHERE price > 100.00
    ORDER BY title
```

The example below demonstrates the use of multiple tables in a join, grouping, and aggregation in an SQL query, by returning a list of books and the number of authors associated with each book.

```
SELECT Book.title,
       count(*) AS Authors
    FROM Book
         JOIN Book_author ON Book.isbn = Book_author.isbn
    GROUP BY Book.title
```

Example output might resemble the following:

Title	Authors
SQL Examples and Guide	3
The Joy of SQL	1
How to use Wikipedia	2
Pitfalls of SQL	1

Under the precondition that isbn is the only common column name of the two tables and that a column named title only exists in the Books table, the above query could be rewritten in the following form:

```
SELECT title,
```

```
        count(*) AS Authors
    FROM Book
        NATURAL JOIN Book_author
    GROUP BY title
```

However, many vendors either do not support this approach, or it requires certain column naming conventions. Thus, it is less common in practice.

NOTES

Data retrieval is very often combined with data projection when the user is looking for calculated values and not just the verbatim data stored in primitive data types, or when the data needs to be expressed in a form that is different from how it's stored. SQL allows the use of expressions in the select list to project data, as in the following example which returns a list of books that cost more than 100.00 with an additional sales_tax column containing a sales tax figure calculated at 6% of the price.

```
SELECT isbn,
        title,
        price,
        price * 0.06 AS sales_tax
    FROM Book
    WHERE price > 100.00
    ORDER BY title
```

Universal quantification is not explicitly supported by sql, and must be worked out as a negated existential quantification.

SUMMARY

1. The database is used to store information useful to an organization.
2. The entity-relationship data model (E R Model) grew out of exercise of using commercially available DBMSs to model application databases.
3. The relationship between entity sets is represented by a model named E-R relationship.
4. Conceptual modeling is a very important phase in designing a successful database application.
5. A database model is a collection of logical constructs used to represent the data structure and the data relationships found within the database.
6. Tranditional data models are the hierarchical, network and relational models.
7. Semantic data modes were influenced by the semantic networks developed by artificial intelligence researchers.
8. Composite attributes can be divided into smaller subparts, which represent more basic attributes with independent meanings.
9. Most attributes have a single value for a particular entity.
10. There can be multivalued attributes too.
11. Some attribute values can be derived from related entities.
12. In some cases a particular entity may not have an applicable value for an attribute. It is called the NULL value.
13. A database usually contains groups of entities that are similar.
14. An entity type describes the schema or intension for a set of entities that share the same structure.

NOTES

15. An important constraint on the entities of an entity type is the key or uniqueness constraint on attributes.
16. It is a constraint that prohibits any two entities from having the same value for the key attribute at the same time.
17. Each simple attribute of an entity type is associated with a value set (or domain of values), which specifies the set of values that may be assigned to that attribute for each individual entity.
18. Database models can be grouped into two categories: conceptual models and implementation models.
19. The conceptual model focuses on the logical nature of the data representation.
20. An implementation model places the emphasis on how the data are represented in the database or on how the data structures are implemented to represent what is modeled.
21. Each database model is evolved from its predecessors.
22. The degree of a relationship type is the number of participating entity types.
23. It is sometimes convenient to think of a relationship type in terms of attributes.
24. Each entity type that participates in a relationship type plays a particular role in the relationship.
25. The process of database design is an iterative rather than a linear or sequential process.
26. During the design process, the database designer does simply depend on interviews to help define entities, attributes and relationships.
27. Converting any E-R model to a set of tables in a database is followed by a specific set of rules that govern such a conversion.
28. The normalization process, as first proposed by Codd (1972a), takes a relation schema through a series of tests to "certify" whether it satisfies a certain normal form.
29. Codd proposed three normal forms, which he called First, Second, and Third normal form.
30. A stronger definition of 3NF—called Boyce-Codd Normal From (BCNF)—was proposed later by Boyce and Codd.
31. Normalization of data can be looked upon as a process of analyzing the given relation schemas based on their FDs and primary keys to achieve the desirable properties of : minimizing redundancy and minimizing the insertion, deletion, and update anomalies.
32. Normal forms, when considered in isolation from other factors, do not guarantee a good database design.
33. Normalization is carried out in practice so that the resulting designs are of high quality and meet the desirable properties.
34. The process of storing the join of higher normal form relations as a base relation which is in a lower normal form—is known as denormalization.
35. The problem of database inconsistency and redundancy of data are similar to the problems that exist in the hierarchical and network models.
36. The first normal form states that the domain of an attribute must include only atomic (simple, indivisible) values and that the value of any attribute in a tuple must be a single value from the domain of that attribute.
37. The only attribute values permitted by 1NF are single atomic (or indivisible) values.
38. First normal form also disallows multivalued attributes that are themselves composite.
39. Second normal form (2NF) is based on the concept of full functional dependency.
40. Third normal form (3NF) is based on the concept of transitive dependency.
41. In terms of the normalization process, it is not necessary to remove the partial dependencies

before the transitive dependencies, but historically, 3NF has been defined with the assumption that a relation is tested for 2NF first before it is tested for 3NF.

42. Boyce-Codd normal form (BCNF) was proposed as a simpler form of 3NF, but it was found to be stricter than 3NF.
42. In practice, most relation schemas that are in 3NF are also in BCNF.
43. SQL statements have 3 Data Definition Language (DDL) statements, CREATE, ALTER and DROP.
44. SQL also has Data Manipulation Language (DML)statements such as SELECT, UPDATE, and DELETE.
45. Data Control Commands (DCL) allow us to control access to data within the database.
46. Some data control commands are: ALTER PASSWORD, GRANT, REVOKE and CREATE SYNONYM.
47. TCL (Transactional Control Commands are the commands which allow the user to manage database transactions. For example commands like: COMMIT, ROLLBACK, SAVEPOINT and SET TRANSACTION.
48. SELECT command accompanied by many options and clauses, is used to compose queries against a relational database.
49. INSERT command is used to insert new data into a table.
50. UPDATE command does not add new records to a table, nor does it remove records—it simply updates existing data.
51. DELETE command is used to remove entire rows of data from a table.
52. CREATE command allows you to create a schema under which all the controls will be mentioned.
53. DROP command can be used, for example, to drop a user from using the database.
54. ALTER command is used to change the attributes like user, password, privileges, etc.
55. RENAME allows you to rename a table from the database.
56. TRUNCATE allows you to truncate the running job.
57. GRANT command is used to grant both system-level and object-level privileges of an exising database user account.
58. REVOKE command removes privileges that have granted to database users.
59. COMMIT is the transactional command used to save changes invoked by a transaction to the database.
60. ROLLBACK is the transactional control command used to undo transactions that have not already been saved to the database.
61. ADDITION is performed using (+) symbol.
62. SUBTRACTION is performed using (–) symbol.
63. MULTIPLICATION is performed using (*) symbol.
64. DIVISION is performed using (/) symbol.
65. Arithmetic operators are performed in the sequence of Division, Multiplication, Addition and Subtraction.
66. NULL is used where you have to specify that there is nothing in it.
67. CONCATENATION is the process of combining the two separate strings into one string.
68. The DISTINCT option is used when you have to display only one of the duplicate records.
69. WHERE command is used when you have to make selection based on some facts.
70. The character means the various alphabets which are used in the language.

NOTES

NOTES

71. REPLACE function is used to replace every occurence of a character(s) with a specified character(s).
72. The LIKE operator is used to compare a value which is similar to values given by the wildcards.
73. The EXISTS operator is used to search for the presence of a row in a specified table that meets certain criteria.
74. UNIQUE operator searches every row of a specified table for uniqueness, i.e., no duplicates.
75. The ALL operator is used to compare a value to all values in another values set.
76. The ANY operator is used to compare a value to any applicable value in the list according to the condition.
77. AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause.
78. OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.
79. ORDER BY clause is used in sorting.
80. You can choose multiple columns too for sorting.
81. CHR function returns the character equivalent of the number it uses as an argument.
82. INITCAP function makes the first character to the uppercase and all other characters to lowercase.
83. INSTR function is used to find out where in a string a particular pattern occurs.
84. LEFT function returns the leftmost character from the string.
85. LENGTH function is used to find the length of a string, number, date, or expression in bytes.
86. LOCATE function is used to the first occurance of the substring in the string.
87. LTRIM is used to clip a part of a string.
88. RIGHT function returns the rightmost character from the string.
89. NUMBER functions are: ABS, ACOS, ASIN, ATAN, CEIL, COS, COSH, EXP, FLOOR, LN, LOG, MOD, NVL, POWER, ROUND, SIGN, SIN, SINH, SQRT, TAN, TANH, TRUNC and VSIZE.
90. LAST_DAY function is used to return the last day of a specified month.
91. MONTHS_BETWEEN function is used to return the number of months elapsed between two months.
92. NEXT_DAY function is used to return the first day of the week that is equal to or later than another specified date.
93. ADD_MONTH function is used to add a number of months to a specified date.
94. ROUND rounds off date according to format.

SELF ASSESSMENT QUESTIONS

1. What do you understand by Data Modeling?
2. Describe Entity Relationship Model.
3. Describe the various Data Models.
4. Write a short note on Entities and Attributes.
5. What are Entity Types and Entity Sets?
6. Write a short note on Key Attributes of an Entity Type.
7. Write a short note on Relationship among Entities.
8. What is E-R Diagram?

9. What do you understand by Functional Dependencies?
10. What are Trivial and Non-Trivial dependencies?
11. Describe the Closure of a set of dependencies and attributes.
12. What do you understand by Irreducible set of dependencies?
13. Describe the Non-loss decomposition.
14. Describe the working of First normal form with example.
15. What do you understand by Second normal form?
16. What is Third normal form?
17. Describe in details the Boyce/Codd normal form.
18. Write short notes on :

Functional dependencies	Full Functional dependencies
Prime Attribute and Nonprime Attribues	Transitive Dependency
General Definition of Second normal form	General Definition of Third normal form.
File Based Primitive Models	Traditional Data Models
Semantic Data Models	Composite versus Simple Attributes
Single Value versus Multivalues Attributes	Null Values
Complex Attributes	Value Sets of Attributes
One-to-Many relationship	Many-to-Many relationship
One-to-one relationship	Degree of a Relationship Type
Relationships as Attributes	
Converting an E-R Model into a database structure.	

19. Which are the Data Definition Language statements?
20. Which are Data Manipulation Language statements?
21. What are the Data Control Commands?
22. Which are Transactional Control Commands?
23. Describe with example the use of the following functions:

INSERT	UPDATE
DELETE	CREATE
DROP	ALTER
RENAME	TRUNCATE
GRANT	REVOKE
COMMIT	ROLLBACK
ADDITION	SUBTRACTION
MULTIPLICATION	DIVISION
NULL	CONCATENATION
DISTINCT	WHERE
GREATER THAN	LESS THAN
LIKE	EXITS
UNIQUE	ANY
AND	OR
ORDER	CHR
INITCAP	INSTR
LEFT	LENGTH
LOCATE	LTRIM
RIGHT	NUMBER
LAST_DAY	NEXT_DAY
MONTHS_BETWEEN	ADD_MONTH
ROUND	

NOTES

NOTES

Multiple Choice Questions

1. E-R represents the relationship between :
   (a) attributes (b) Data (c) entitites
2. Tranditional data models are the hierarchical, network and _____ :
   (a) physical (b) relational (c) manual
3. Entities not having an applicable value for an attribute are called :
   (a) NULL (b) NIL (c) NO
4. Database models can be grouped into two categories: _____ models and implementation models :
   (a) concurrent (b) clumsy (c) conceptual
5. _____ data modes were influenced by the semantic networks developed by artificial intelligence researchers :
   (a) Semantic (b) Traditional (c) Primitive
6. Codd proposed three normal forms, which he called First, Second, and _____ normal form :
   (a) Fourth (b) Many (c) Third
7. A stronger definition of 3NF—called Boyce-Codd Normal From (BCNF)—was proposed by Boyce and _____ :
   (a) Paul Allen (b) Codd (c) Bill Gates
8. The process of storing the join of higher normal form relations as a base relation which is in a lower normal form—is known as _____:
   (a) normalization (b) denormalization
9. Third normal form (3NF) is based on the concept of _____ dependency:
   (a) transitive (b) functional (c) none of above
10. In practice, most relation schemas that are in 3NF are also in _____ .
    (a) 1NF (b) 2NF (c) 3CNF
11. Data Definition Language (DDL) statements are: CREATE, ALTER and :
    (a) DROP (b) UPDATE (c) REVOKE
12. Out of the following which are Data Manipulation Language (DML) statements:
    (a) SELECT (b) UPDATE (c) EVOKE
13. Out of the following which are Data Control Commands :
    (a) ALTER (b) PASSWORD (c) GRANT
14. ROLLBACK is a :
    (a) TCL (Transactional Control Command)
    (b) DML (Data Manipulation Language)
    (c) Data Definition Language)
15. For entering new data we use :
    (a) UPDATE (b) CREATE (c) INSERT
16. For creating a new database/table we use:
    (a) CREATE (b) UPDATE (c) INSERT
17. For changing the user, password, etc, we use:
    (a) UPDATE (b) INSERT (c) ALTER
18. A privilege is given to a user by command:
    (a) GRANT (b) REVOKE (c) COMMIT
19. ADDITION is performed using :
    (a) (–) symbol (b) (+) symbol (c) (*) symbol
20. MULTIPLICATION is performed using :
    (a) (*) symbol (b) (+) symbol (c) (–) symbol

21. INITCAP function makes which character to the uppercase :
    (a) Last (b) Second (c) First
22. To find out the length of a string, we use the function called:
    (a) STRING (b) LENGTH (c) FUNCTION

True/False Questions

NOTES

1. The entity-relationship data model (E R Model) grew out of exercise of using commercially available DBMSs to model application databases.
2. The relationship between entity sets is represented by a model named E-R relationship.
3. A database model is a collection of logical constructs used to represent the data structure and the data relationships found within the database.
4. Semantic data models were not influenced by the semantic networks developed by artificial intelligence researchers.
5. Most attributes have a single value for a particular entity.
6. There cannot be multivalued attributes.
7. In some cases a particular entity may not have an applicable value for an attribute. It is called the NULL value.
8. An entity type does not describe the schema or intension for a set of entities that share the same structure.
9. It is a constraint that prohibits any two entities from having the same value for the key attribute at the same time.
10. Database models cannot be grouped into two categories: conceptual models and implementation models.
11. An implementation model places the emphasis on how the data are represented in the database or on how the data structures are implemented to represent what is modeled.
12. It is sometimes convenient to think of a relationship type in terms of attributes.
13. The process of database design is an iterative rather than a linear or sequential process.
14. Converting any E-R model to a set of tables in a database is followed by a specific set of rules that govern such a conversion.
15. The normalization process, as first proposed by Codd (1972a), takes a relation schema through a series of tests to "certify" whether it satisfies a certain normal form.
16. Codd proposed three normal forms, which he called First, Second, and Third normal form.
17. Normalization of data can be looked upon as a process of analyzing the given relation schemas based on their FDs and primary keys to achieve the desirable properties of : minimizing redundancy and minimizing the insertion, deletion, and update anomalies.
18. Normal forms, when considered in isolation from other factors, do guarantee a good database design.
19. The problem of database inconsistency and redundancy of data are similar to the problems that exist in the hierarchical and network models.
20. The only attribute values not permitted by 1NF are single atomic (or indivisible) values.
21. First normal form also disallows multivalued attributes that are themselves composite.
22. Third normal form (3NF) is based on the concept of transitive dependency.
23. In practice, most relation schemas that are in 3NF are also in BCNF.
24. SQL statements have 3 Data Definition Language (DDL) statements, CREATE, ALTER and DROP.
25. Data Control Commands (DCL) does not allow you to control access to data within the database.
26. TCL (Transactional Control Commands are the commands which allow the user to manage database transactions. For example commands like: COMMIT, ROLLBACK, SAVEPOINT and SET TRANSACTION.
27. INSERT command is used to insert new data into a table.

NOTES

28. DELETE command is not used to remove entire rows of data from a table.
29. DROP command cannot be used, for example, to drop a user from using the database.
30. RENAME allows you to rename a table from the database.
31. GRANT command is not used to grant both system-level and object-level privileges to an exising database user account.
32. COMMIT is the transactional command used to save changes invoked by a transaction to the database.
33. ADDITION is performed using (–) symbol.
34. MULTIPLICATION is performed using (/) symbol.
35. Arithmetic operators are performed in the sequence of Division, Multiplicaion, Addition and Subtraction.
36. CONCATENATION is the process of combining the two separate strings into one string.
37. WHERE command is used when you have to make selection based on some facts.
38. The ALL operator is used to compare a value to all values in another values set.
39. AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause.
40. ORDER BY clause is used in sorting.
41. CHR function returns the character equivalent of the number it uses as an argument.
42. INSTR function is used to find out where in a string a particular pattern occurs.
43. LENGTH function is used to find the length of a string, number, date, or expression in bytes.
44. LTRIM is not used to clip a part of a string.
45. NUMBER functions are: ABS, ACOS, ASIN, ATAN, CEIL, COS, COSH, EXP, FLOOR, LN, LOG, MOD, NVL, POWER, ROUND, SIGN, SIN, SINH, SQRT, TAN, TANH, TRUNC and VSIZE.
46. MONTHS_BETWEEN function is used to return the number of months elapsed between two months.

Short Questions with Answers

1. What is an entity-relationship model?

Ans. The entity-relationship model is a generalization of these models. It allows the representation of explicit constraints as well as relationships. Even though the E-R model has some means of describing the physical database model, it is basically useful in the design and communication of the lofical database model.

2. What is conceptual modeling?

Ans. Conceptual modeling is a very important phase in designing a successful database application. Generally, the term database application refers to a particular database and the associated programs that implement the database queries and updates. For example, a BANK database application that keeps track of customer accounts would include programs that implement database updates corresponding to customers making deposits and withdrawals.

3. What is an entity?

Ans. The basic object that the ER model represents is an entity, which is a "thing" in the real world with an independent existence. An entity may be an object with a physical existence (for example, a particular person, car, house, or employee) or it may be an object with a conceptual existence (for example, a company, a job, or a university course). Each entity has attributes—the particular properties that describes it.

4. What is entity type?

Ans. An entity type describes the schema or intension for a set of entities that share the same structure. The collection of entities of a particular entity type are grouped into an entity set, which is also called the extension of the entity type.

5. What type of relationships exist in terms of databases?

Ans. There are three main type of relationships: One to one relationship; On to many relationship and Many to Many relationship.

6. What is a functional dependency?

Ans. A functional dependency is basically a constraint between two sets of attributes from the database. It is due to the consequence of the interrelationship among attributes of an entity represented by a relation or due to the relationship between entities that is also represented by a relation.

NOTES

7. What is normalization?

Ans. The normalization process, as first proposed by Codd (1972), takes a relation schema through a series of tests to "certify" whether it satisfies a certain normal form. The process, which proceeds in a top-down fashion by evaluating each relation again the criteria for normal forms and decomposing relations as necessary, can thus be considered as relational design by analysis.

8. How many normal forms are there?

Ans. Initially, Codd proposed three normal forms, which he called First, Second, and Third normal form. A stronger definition of 3NF—called Boyce-Codd Normal Form (BCNF)—was proposed later by Boyce and Codd. All these normal forms are based on the functional dependencies among the attributes of a relation. Later, a fourth normal form (4NF) and a fifth normal form (5NF) were proposed, based on the concepts of multivalued dependencies and join dependencies, respectively.

9. What are the properties of first normal form?

Ans. First normal form disallows having a set of values, a tuple of values, or a combination of both as an attribute value for a single tuple. In other words, 1NF disallows "relations within relations" or "relations as attribute values within tuples." The only attribute values permitted by 1NF are single atomic (or indivisible) values.

10. What is second normal form based on?

Ans. Second normal form (2NF) is based on the concept of full functional dependency.

11. What is third normal form based on?

Ans. Third normal form (3NF) is based on the concept of transitive dependency.

12. Write the syntax of the following: CREATE, ALTER, DROP, TRUNCATE, SELECT, DATES, AND, OR, Not EQUAL, Not BETWEEN, Not IN, Not LIKE, Is Not NULL, Not EXISTS, Not UNIQUE, GROUP BY, INSERT, UPDATE SQL commands.

Ans. **CREATE**

CREATE TABLE NewTable (NewValue) INT)

ALTER

ALTER TABLE MyTable ADD MyColumn VARCHAR (20) NULL

DROP

DROP TABLE [dbo].[friends]

TRUNCATE

DELETE FROM authors

TRUNCATE TABLE authors

SELECT

SELECT [* | ALL | DISTINCT COLUMN1, COLUMN2] FROM TABLE1 [, TABLE2];

DATES

SELECT GETDATE()

AND

WHERE ROLLNO = "1033" AND NAME = "RAHUL"

OR

WHERE ROLLNO = "1033" OR ROLLNO = "1030"

NOTES

Not EQUAL

WHERE ROLLNO <> "1050"

Not BETWEEN

WHERE ROLLNO NOT BETWEEN "1000" AND "2000"

Not IN

WHERE ROLLNO NOT IN ("1000", "1200", "1300")

Not LIKE

WHERE ROLLNO NOT LIKE "2000"

Is Not NULL

WHERE ROLLNO IS NOT NULL

Not EXISTS

WHERE NOT EXISTS (SELECT ROLLNO FROM STUDENT_TBL WHERE ROLLNO = "1033")

Not UNIQUE

WHERE NOT UNIQUE (SELECT ROLLNO FROM STUDENT_TBL)

GROUP BY

```
SELECT LAST_NAME, FIRST_NAME, CITY
FROM EMPLOYEE_TBL
GROUP BY LAST_NAME;
```

INSERT

```
INSERT INTO SCHEMA.TABLE_NAME
VALUES ('value1', 'Value2', [ NULL ] );
```

UPDATE

```
UPDATE TABLE_NAME
SET COLUMN_NAME = 'VALUE'
(WHERE CONDITION)
```

ANSWERS

Multiple Choice Questions

1. c	2. b	3. a	4. c
5. a	6. c	7. b	8. b
9. a	10. c	11. a	12. a
13. b	14. a	15. c	16. a
17. c	18. a	19. a	20. b
21. a	22. c		

True False Questions

1. T	2. T	3. T	4. F
5. T	6. F	7. T	8. F
9. T	10. F	11. T	12. T
13. T	14. T	15. T	16. T
17. T	18. F	19. T	20. F
21. T	22. T	23. T	24. T
25. F	26. T	27. T	28. F
29. T	30. T	31. F	32. T
33. F	34. F	35. T	36. T
37. T	38. T	39. T	40. T
41. T	42. T	43. T	44. F
45. T	46. T		

CHAPTER 3

DATA ADMINISTRATION

LEARNING OBJECTIVES

After going through this chapter, you should appreciate the following:

- Data Administration
- Client/Server and Distributed Databases Data administration functions
- Data administration tools - Repositories
- CASE Tools
- Concurrency Control
- Database Security
- Database Recovery
- Client/Server Architecture
- Functions of Client/Server
- Advantages
- Issues
- Distributed Databases
- Objectives
- Distributed DBMS
- Location transparency
- Replication transparency
- Failure transparency
- Commit protocol
- Concurrency transparency.

NOTES

DATA ADMINISTRATION

In any organization where many persons use the same resources, there is a need for a chief administrator to oversee and manage these resources. In a database environment, the primary resource is the database itself, and the secondary resource is the DBMS and related software. Administering these resources is the responsibility of the DataBase Administrator (DBA). The DBA is responsible for authorizing access to the database, for coordinating and monitoring its use, and for acquiring software and hardware resources as needed. The DBA is accountable for problems such as breach of security or poor system response time. In large organizations, the DBA is assisted by a staff that helps carry out these functions.

CLIENT/SERVER AND DISTRIBUTED DATABASES DATA ADMINISTRATION FUNCTIONS

Architectures for DBMSs have followed trends similar to those for general computer system architectures. Earlier architectures used mainframe computers to provide the main processing for all system functions, including user application programs and user interface programs, as well as all the DBMS functionality. The reason was that most users accessed such systems via computer terminals that did not have processing power and only provided display capabilities. Therefore, all processing was performed remotely on the computer system, and only display information and controls were sent from the computer to the display terminals, which were connected to the central computer via various types of communications networks.

As prices of hardware declined, most users replaced their terminals with PCs and workstations. At first, database systems used these computers similarly to how they had used display terminals, so that the DBMS itself was still a centralized DBMS in which all the DBMS in which all the DBMS functionality, application program execution, and user interface processing were carried out on one machine. Gradually, DBMS systems started to exploit the available processing power at the user side, which led to Client/Server DBMS architectures.

Basic Client/Server Architectures

First, we discuss client/server architecture in general, then we see how it is applied to DBMSs. The client/server architecture was developed to deal with computing environments in which a large number of PCs, workstations, file servers, printers, database servers, Web servers, and other equipment are connected via a network. The idea is to define specialized servers with specific functionalities. For example, it is possible to connect a number of PCs or small workstations as clients to a file-server that maintains the files of the client machines. Another machine can be designated as a printer server by being connected to various printers; thereafter, all print requests by the clients are forwarded to this machine. Web servers or e-mail servers also fall into the specialized server category.

In this way, the resources provided by specialized servers can be accessed by many client machines. The client machines provide the user with the appropriate interfaces to utilize these servers, as wel as with local processing power to run local applications. This concept can be carried over to software, with specialized programs—such as a

DBMS or a CAD (computer aided design) package—being stored on specific server machines and being made accessible to multiple clients.

The concept of client/server architecture assumes an underlying framework that consists of many PCs and workstations as well as a smaller number of mainframe machines, connected via LANs and other types of computer networks. A client in this framework is typically a user machine that provides user interface capabilities and local processing. When a client requires access to additional functionality—such as database access—that does not exist at the machine, it connects to a server that provides the needed functionality. A server is a system containing both hardware and software that can provide services to the client machines, such as file access, printing, archiving, or database access. In the general case, some machines install only client software, others only server software, and still others may include both client and server software. However, it is more common that client and server software usually run on separate machines. Two main types of basic DBMS architecture were created on this underlying client/server framework: two-tier and three-tier.

NOTES

Two-Tier Client/Server Architectures for DBMSs

The client/server architecture is increasingly being incorporated into commercial DBMS packages. In relational database management systems (RDBMSs), many of which started as centralized systems, the system components that were first moved to client sider were the user interface and application programs. Because SQL provided a standard language for RDBMSs, this created a logical dividing point between client and server. Hence, the query and transaction functionality related to SQL processing remained on the server side. In such architecture, the server is often called a query server or transaction server because it provides these two functionalities. In an RDBMS the server is also often called SQL server.

In such a client/server architecture, the user interface programs and application programs can run on the client side. When DBMS acess is required, the program establishes a connection to the DBMS, which is on the server side; once the connection is created, the client program can communicate with the DBMS. A standard called Open Database Connectivity (ODBC) provides an application programming interface (API), which allows client-side programs to call the DBMS, as long as both client and server machines have the necessary software installed. Most DBMS vendors provide ODBC drivers for their systems. A client program can actually connect to serveral RDBMSs and send query and transaction requests using the ODBC API, which are then processed at the server sites. Any query results are send back to the client program, which can process or display the results as needed. A related standard for the Java programming lnaguage, called JDBC, has also been defined. This allows Java client programs to access the DBMS through a standard interface.

Three-Tier and n-Tier Architecture for Web Application

Many Web applications use an architecture called the three-tier architecture, which adds an intermediate layer between the client and the database server. This immediate layer or middle tier is sometimes called the application server and sometimes the Web server, depending on the application. This server plays an intermediary role for storing business rules, procedures or constraints, that are used to access data from the database server. It can also improve database security by checking a client's credentials before forwarding a request to the database server. Clients contain GUI interfaces and some additional application-specific business rules. The intermediate server accepts

NOTES

requests from the client, processes the request and sends database commands to the database server, and then acts as a conduit for passing (partially) processed data from the databaser server to the clients, where it may be processed further and filtered to be presented to users in GUI format. Thus, the userr interface, application rules, and data access act as the three tiers.

Advances in encryption and decryption technology make it safer to transfer sensitive data from server to client in encrypted form, where it will be decrypted. The latter can be done by the hardware or by advanced software. This technology gives higher levels of data security, but the network security issues remain a major concern. Various technologies for data compression also help to transfer large amounts of data from servers to clients over wired and wireless networks.

DATA ADMINISTRATION TOOLS - REPOSITORIES

Data administration is, in computing science, the administration of the organisation of data, usually as stored in databases under some Database Management System or alternative systems such as electronic spreadsheets.

In many smaller organisations, Data Administration is not performed at all, or is but a small parcel of the Database Administrator's work.

Data Administration ideally begins at software conception, ensuring there is a data dictionary to help keeping consistency and avoid redundancy and modelling the database so as to make it logical and usable, by means of the normalisation technique.

Quite often such modelling is mistaken as diagramming, because of the prevalence of entity-relationship diagrams.

Data Resource Management refers to the development and maintenance of data models to facilitate data sharing between different systems, particularly in a corporate context. DRM is concerned with both data quality and compatibility between data models.

A large amount of multimedia data as well as metadata is stored for retrieval purposes. A central repository containing multimedia data may be maintained by a DBMS and may be organized into a hierarchy of storage level—local disks, tertiary disks and tapes, optical disks, and so on. Examples include repositories of satellite images, engineering drawings and designs, space photographs, and radiologhy scanned pictures.

CASE TOOLS

CASE (Computer-aided Software/System Engineering) refers to the methods dedicated to an engineering discipline for the development of information systems together with automated tools that can be used in this process.

CASE can be described as harboring two key ideas :

- Computer Assistance in software development and/or maintenance
- An engineering approach to the software development and/or maintenance.

Some typical CASE tools are:

- Code generation tools
- Data modeling tools
- UML
- Refactoring tools
- QVT or Model transformation Tools

Classification of CASE Tools

Existing CASE Environments can be classified along 4 different dimensions :

- Life-Cycle Support
- Integration Dimension
- Construction Dimension
- Knowledge Based CASE dimension

Let us take the meaning of these dimensions along with their examples one by one :

Life-Cycle Based CASE Tools

This dimension classifies CASE Tools on the basis of the activities they support in the information systems life cycle. They can be classified as Upper or Lower CASE tools.

Upper Case Tools support strategic, planning and construction of conceptual level product and ignore the design aspect. They support traditional diagrammatic languages such as ER Diagrams, DFD, Structure charts etc.

Lower Case Tools concentrate on the back end activities of the software life cycle and hence support activities like physical design, debugging, construction, testing, integration of software components, maintenance, reengineering and reverse engineering activities.

Integration Dimension

Three main CASE Integration dimension have been proposed :

- CASE Framework
- ICASE Tools
- Integrated Project Support Environment(IPSE)

History of CASE

All aspects of the software development lifecycle can be supported by software tools, and so the use of tools from across the spectrum can, arguably, be described as CASE; from project management software through tools for business and functional analysis, system design, code storage, compilers, translation tools, test software, and so on.

However, it is the tools that are concerned with analysis and design, and with using design information to create parts (or all) of the software product, that are most frequently thought of as CASE tools. CASE applied, for instance, to a database software product, might normally involve:

- Modelling business / real world processes and data flow

NOTES

- Development of data models in the form of entity-relationship diagrams
- Development of process and function descriptions
- Production of database creation SQL and stored procedures

The term CASE was originally coined by software company, Nastec Corporation of Southfield, Mich. in 1982 with their original integrated graphics and text editor GraphiText, which also was the first microcomputer-based system to use hyperlinks to cross-reference text strings in documents — an early forerunner of today's web page link. GraphiText's successor product, DesignAid was the first microprocessor-based tool to logically and semantically evaluate software and system design diagrams and build a data dictionary. Under the direction of Albert F. Case, Jr. vice president for product management and consulting (the rumor that he changed his last name is untrue), and Vaughn Frick, director of product management, the DesignAid product suite was expanded to support analysis of a wide range of structured analysis and design methodologies, notably Yourdori/Demarco, Gane & Sarson, Ward-Mellor (real-time) SA/SD and Warnier-Orr (data driven). The next entrant into the market was Excelerator from Index Technology in Cambridge, Mass. While DesignAid ran on Convergent Technologies and later Burroughs Ngen networked microcomputers, Index launched Excelerator on the IBM PC/AT platform. While, at the time of launch, and for several years, the IBM platform did not support networking or a centralized database as did the Convergent Technologies or Burroughs machines, the allure of IBM was strong, and Excelerator came to prominence.

CASE tools were at their peak in the early 1990s. At the time IBM had proposed AD/Cycle which was an alliance of software vendors centered around IBM's mainframe:

The application development tools can be from several sources: from IBM, from vendors, and from the customers themselves. IBM has entered into relationships with Bachman Information Systems, Index Technology Corporation, and Knowledgeware, Inc. wherein selected products from these vendors will be marketed through an IBM complementary marketing program to provide offerings that will help to achieve complete life-cycle coverage.

With the decline of the mainframe, AD/Cycle and the Big CASE tools died off, opening the market for the mainstream CASE tools of today. Interestingly, nearly all of the leaders of the CASE market of the early 1990s ended up being purchased by Computer Associates, including IEW, IEF, ADW, Cayenne, and Learmonth & Burchett Management Systems (LBMS).

Many CASE tools not only output code but also generate other output typical of various systems analysis and design methodologies such as SSADM. e.g.:

- database schema
- data flow diagrams
- entity relationship diagrams
- program specifications
- user documentation

CONCURRENCY CONTROL

NOTES

In computer science, especially in the fields of computer programming (see also concurrent programming, parallel programming), operating systems, multiprocessors, and databases, concurrency control ensures that correct results for concurrent operations are generated, while getting those results as quickly as possible.

Concurrency control in databases

Concurrency control in database management systems (DBMS) ensures that database transactions are performed concurrently without the concurrency violating the data integrity of a database. Executed transactions should follow the ACID rules, as described below. The DBMS must guarantee that only serializable (unless Serializability is intentionally relaxed), recoverable schedules are generated. It also guarantees that no effect of committed transactions is lost, and no effect of aborted (rolled back) transactions remains in the related database.

Transaction ACID rules

- **Atomicity** - Either the effects of all or none of its operations remain when a transaction is completed - in other words, to the outside world the transaction appears to be indivisible, atomic.
- **Consistency** - Every transaction must leave the database in a consistent state.
- **Isolation** - Transactions cannot interfere with each other. Providing isolation is the main goal of concurrency control.
- **Durability** - Successful transactions must persist through crashes.

Concurrency control mechanism

The main categories of concurrency control mechanisms are:

- **Optimistic** - Delay the synchronization for a transaction until its end without blocking (read, write) operations, and then abort transactions that violate desired synchronization rules.
- **Pessimistic** - Block operations of transaction that would cause violation of synchronization rules.

There are several methods for concurrency control. Among them:

- Two-phase locking
- Strict two-phase locking
- Conservative two-phase locking
- Index locking
- Multiple granularity locking

A Lock is a database system object associated with a database object (typically a data item) that prevents undesired (typically synchronization rule violating) operations of other transactions by blocking them. Database system operations check for lock existence, and halt when noticing a lock type that is intended to block them.

There are also non-lock concurrency control methods, among them:

NOTES

- Conflict (serializability, precedence) graph checking
- Timestamp ordering
- commitment ordering

Also Optimistic concurrency control methods typically do not use locks.

Almost all currently implemented lock-based and non-lock-based concurrency control mechanisms guarantee schedules that are conflict serializable (unless relaxed forms of serializability are needed). However, there are many research texts encouraging view serializable schedules for possible gains in performance, especially when not too many conflicts exist (and not too many aborts of completely executed transactions occur), due to reducing the considerable overhead of blocking mechanisms.

Concurrency control in operating systems

Operating systems, especially real-time operating systems, need to maintain the illusion that many tasks are all running at the same time. Such multitasking is fairly simple when all tasks are independent from each other. However, when several tasks try to use the same resource, or when tasks try to share information, it can lead to confusion and inconsistency. The task of concurrent computing is to solve that problem. Some solutions involve "locks" similar to the locks used in databases, but they risk causing problems of their own such as deadlock. Other solutions are lock-free and wait-free algorithms.

DATABASE SECURITY

Database security is the system, processes, and procedures that protect a database from unintended activity. Unintended activity can be categorized as authenticated misuse, malicious attacks or inadvertent mistakes made by authorized individuals or processes. Database security is also a specialty within the broader discipline of computer security.

Definition: ***Database security is the system, processes, and procedures that protect a database from unintended activity.***

Traditionally databases have been protected from external connections by firewalls or routers on the network perimeter with the database environment existing on the internal network opposed to being located within a demilitarized zone.

Additional network security devices that detect and alert on malicious database protocol traffic include network intrusion detection systems along with host-based intrusion detection systems.

Process Controls

Database security is more critical as networks have become more open. Databases provide many layers and types of information security, typically specified in the data dictionary, including the following controls:

- Access control
- Auditing

- Authentication
- Encryption
- Integrity controls

NOTES

Database security can begin with the process of creation and publishing of appropriate security standards for the database environment. The standards may include specific controls for the various relevant database platforms; a set of best practices that cross over the platforms; and linkages of the standards to higher level polices and governmental regulations. An important procedure when evaluating database security is performing vulnerability assessments against the database. A vulnerability assessment attempts to find vulnerability holes that could be used to break into the database.

Database administrators or information security administrators run vulnerability scans on databases to discover misconfiguration of controls within the layers mentioned above along with known vulnerabilities within the database software.

The results of the scans should be used to harden the database in order to mitigate the threat of compromise by intruders.

A program of continual monitoring for compliance with database security standards is another important task for mission critical database environments. Two crucial aspects of database security compliance include patch management and the review and management of permissions (especially public) granted to objects within the database. Database objects may include table or other objects listed in the Table link. The permissions granted for SQL language commands on objects are considered in this process. One should note that compliance monitoring is similar to vulnerability assessment with the key difference that the results of vulnerability assessments generally drive the security standards that lead to the continuous monitoring program. Essentially, vulnerability assessment is a preliminary procedure to determine risk where a compliance program is the process of on-going risk assessment.

The compliance program should take into consideration any dependencies at the application software level as changes at the database level may have effects on the application software or the application server. In direct relation to this topic is that of application security.

Application level authentication and authorization mechanisms should be considered as an effective means of providing abstraction from the database layer.

The primary benefit of abstraction is that of a single sign-on capability across multiple databases and database platforms. A Single sign-on system should store the database user's credentials (login id and password), and authenticate to the database on behalf of the user. Another security layer of a more sophisticated nature includes the real-time monitoring of database protocol traffic (SQL) over the network. Analysis can be performed on the traffic for known exploits or network traffic baselines can be captured overtime to build a normal pattern used for detection of anomalous activity that could be indicative of intrusion. These systems can provide a comprehensive Database audit trail in addition to the intrusion detection (and potentially protection) mechanisms.

NOTES

When a network level audit system is not feasible a native database audit program should be instituted. The native audit trails should be extracted on a regular basis and transferred to a designated security system where the database administators do not have access. This ensures a certain level of segragation of duties that may provide evidence the native audit trails were not modified by authenticed administrators. Generally, the native audit trails of databases do not provide sufficient controls to enforce separation of duties; therefore, the network and/or kernel module level host based monitoring capabilities provides a higher degree of confidence for forsenices and preservation of evidence.

After an incident occurs, the usage of Database Forensics can be employed to determine the scope.

A database security program should include the regular review of permissions granted to individually owned accounts and accounts used by automated processes. The accounts used by automated processes should have appropriate controls around password storage such as sufficient encryption and access controls to reduce the risk of compromise. For individual accounts, a two-factor authentication system should be considered in a database environment where the risk is commensurate with the expenditure for such an authentication system.

In conjunction with a sound database security program, an appropriate disaster recovery program should exist to ensure that service is not interrupted during a security incident or any other incident that results in an outage of the primary database environment. An example is that of replication for the primary databases to sites located in different geographical regions.

Access Control

Access control is the ability to permit or deny the use of a particular resource by a particular entity. Access control mechanisms can be used in managing physical resources (such as a movie theater, to which only ticketholders should be admitted), logical resources (a bank account, with a limited number of people authorized to make a withdrawal), or digital resources (for example, a private text document on a computer, which only certain users should be able to read).

Item Control or Electronic Key Management is an area within (and possibly integrated with)an access control system which concerns the managing of possession and location of small assets or physical (mechanical) keys.

Access Control System Operation

When a credential is presented to a reader, the reader sends the credential's information, usually a number, to a control panel, a highly reliable processor. The control panel compares the credential's number to an access control list, grants or denies the presented request, and sends a transaction log to a database.

When access is denied based on the access control list, the door remains locked. If there is a match between the credential and the access control list, the control panel operates a relay that in turn unlocks the door.

The control panel also ignores a door open signal to prevent an alarm. Often the reader provides feedback, such as a flashing red LED for an access denied and a flashing green LED for an access granted.

NOTES

The above description illustrates a single factor transaction. Credentials can be passed around, thus subverting the access control list. For example, Alice has access rights to the server room but Bob does not. Alice either gives Bob her credential or Bob takes it; he now has access to the server room. To prevent this, two-factor authentication can be used. In a two factor transaction, the presented credential and a second factor are needed for access to be granted. The second factor can be a PIN, a second credential, operator intervention, or a biometric input. Often the factors are characterized as

- something you have, such as a credential,
- something you know, e.g. a PIN, or
- something you are, typically a biometric input.

Access Control System Components

An access control point, which can be a door, turnstile, parking gate, elevator, or other physical barrier where granting access can be electrically controlled. Typically the access point is a door. An electronic access control door can contain several elements. At its most basic there is an electric lock.

The lock is unlocked by an operator with a switch. To automate this, operator intervention is replaced by a reader. The reader could be a keypad where a code is entered, it could be a card reader, or it could be a biometric reader. Readers do not usually make an access decision but send a card number to an access control panel that verifies the number against an access list. To monitor the door position a magnetic door switch is used. In concept the door switch is not unlike those on refrigerators or car doors.

Generally only entry is controlled and exit is uncontrolled. In cases where exit is also controlled a second reader is used on the opposite side of the door. In cases where exit is not controlled, free exit, a device called a request-to-exit (REX) is used.

Request-to-exit devices can be a pushbutton or a motion detector. When the button is pushed or the motion detector detects motion at the door, the door alarm is temporarily ignored while the door is opened. Exiting a door without having to electrically unlock the door is called mechanical free egress. This is an important safety feature. In cases where the lock must be electrically unlocked on exit, the request-to-exit device also unlocks the door.

Credential

A credential is something you know, such as number or PIN, something you have, such as an access badge, something you are, such as a biometric feature, or some combination of these. The typical credential is an access card, key fob, or other key. There are many card technologies including magnetic stripe, bar code, Wiegand, 125 kHz proximity, contact smart cards, and contactless smart cards. Typical biometric technologies include fingerprint, facial recognition, iris recognition, retinal scan, voice, and hand geometry.

Bar Code Technology

A bar code is a series of alternating dark and light stripes that are read by an optical scanner. The organization and width of the lines is determined by the bar code protocol

NOTES

selected. There are many different protocols but code 39 is the most popular in the security industry. Sometimes the digits represented by the dark and light bars are also printed to allow people to read the number without an optical reader.

The advantage of using bar code technology is that it is cheap and easy to generate the credential and it can easily be applied to cards or other items.

The disadvantage of this technology is that it is cheap and easy to generate a credential making the technology susceptible to fraud and the optical reader can have reliability problems with dirty or smudged credentials. One attempt to reduce fraud is to print the bar code using carbon-based ink and then cover the bar code with a dark red overlay. The bar code can then be read with an optical reader tuned to the infrared spectrum, but can not easily be copied by a copy machine. This does not address the ease with which bar code numbers can be generated from a computer using almost any printer.

Magnetic Stripe Technology

Magnetic stripe technology, usually called mag-stripe, is so named because of the stripe of magnetic oxide tape that is laminated on a card. There are three tracks of data on the magnetic stripe. Typically the data on each of the tracks follows a specific encoding standard, but it is possible to encode any format on any track.

A mag-stripe card is cheap compared to other card technologies and is easy to program.

The magnetic stripe holds more data than a bar code can in the same space. While a mag-stripe is more difficult to generate than a bar code, the technology for reading and encoding data on a mag-stripe is widespread and easy to acquire. Magnetic stripe technology is also susceptible to misreads, card wear, and data corruption.

Wiegand Card Technology

Wiegand card technology is a patented technology using embedded ferromagnetic wires strategically positioned to create a unique pattern that generates the identification number. Like magnetic stripe or bar code, this card must be swiped through a reader to be read. Unlike those other technologies the identification media is embedded in the card and not susceptible to wear. This technology once gained popularity because of the difficulty in duplicating the technology creating a high perception of security. This technology is being replaced by proximity cards because of the limited source of supply, the relatively better tamper resistance of proximity readers, and the convenience of the touch-less functionality in proximity readers.

Proximity Card Technology

The Wiegand effect was used in early access cards. This method was abandoned in favor of other technologies. The new technologies retained the Wiegand upstream data so that the new readers were compatible with old systems. Readers are still called Wiegand but no longer use the Wiegand effect. A Wiegand reader radiates a 1" to 5" electrical field around itself. Cards use a simple LC circuit.

When a card is presented to the reader, the reader's electrical field excites a coil in the card. The coil charges a capacitor and in turn powers a integrated circuit.

The integrated circuit outputs the card number to the coil which transmits it to the reader.

A common proximity format is 26 bit Wiegand. This format uses a facility code, sometimes also called a site code. The facility code is a unique number common to all of the cards in a particular set. The idea is that an organization will have their own facility code and a set of numbered cards incrementing from 1. Another organization has a different facility code and their card set also increments from 1. Thus different organizations can have card sets with the same card numbers but since the facility codes differ, the cards only work at one organization. This idea worked fine for a while but there is no governing body controlling card numbers, and different manufacturers can supply cards with identical facility codes and identical card numbers to different organizations.

NOTES

Thus there is a problem of duplicate cards. To counteract this problem some manufacturers have created formats beyond 26 bit Wiegand that they control and issue to organizations.

In the 26 bit Wiegand format, bit 1 is an even parity bit. Bits 2-9 are a facility code. Bits 10-25 are the card number. Bit 26 is an odd parity bit. Other formats have a similar structure of a leading facility code followed by the card number and including parity bits for error checking.

Smart Card

There are two types of smart cards: contact and contactless Both have an embedded microprocessor and memory. The smart card differs from the card typically called a proximity card in that the microchip in the proximity card has only one function: to provide the reader with the card's identification number. The processor on the smart card has an operating system and can handle multiple applications such as a cash card, a pre-paid membership card, and even an access control card.

The difference between the two types of smart cards is found in the manner with which the microprocessor on the card communicates with the outside world.

A contact smart card has eight contacts, which must physically touch contacts on the reader to convey information between them. A contactless smart card uses the same radio-based technology as the proximity card with the exception of the frequency band used. Smart cards allow the access control system to save user information on a credential carried by the user rather than requiring more memory on each controller.

PIN

A personal identification number (PIN) falls in the category of what you know rather than what you have. The PIN is usually a number consisting of four to eight digits. Less and the number is too easy to guess. More and the number is too difficult to remember.

The advantage to using a PIN as an access credential is that once the number is memorized, the credential cannot be lost or left somewhere.

The disadvantage is the difficulty some people have in remembering numbers that are not frequently used and the ease with which a PIN can be observed and therefore

NOTES

used by unauthorized people. The PIN is even less secure than a bar code or magnetic stripe card.

Computer Security

In computer security, access control includes authentication, authorization and audit. It also includes measures such as physical devices, including biometric scans and metal locks, hidden paths, digital signatures, encryption, social barriers, and monitoring by humans and automated systems.

In any access control model, the entities that can perform actions in the system are called subjects, and the entities representing resources to which access may need to be controlled are called objects. Subjects and objects should both be considered as software entities, rather than as human users: any human user can only have an effect on the system via the software entities that they control. Although some systems equate subjects with user IDs, so that all processes started by a user by default have the same authority, this level of control is not fine-grained enough to satisfy the Principle of least privilege, and arguably is responsible for the prevalence of malware in such systems.

In some models, for example the object-capability model, any software entity can potentially act as both a subject and object.

Access control models used by current systems tend to fall into one of two classes: those based on capabilities and those based on access control lists (ACLs). In a capability-based model, holding an unforgeable reference or capability to an object provides access to the object (roughly analogous to how possession of your house key grants you access to your house); access is conveyed to another party by transmitting such a capability over a secure channel. In an ACL-based model, a subject's access to an object depends on whether its identity is on a list associated with the object (roughly analogous to how a bouncer at a private party would check your ID to see if your name is on the guest list); access is conveyed by editing the list. Different ACL systems have a variety of different conventions regarding who or what is responsible for editing the list and how it is edited.

Both capability-based and ACL-based models have mechanisms to allow access rights to be granted to all members of a group of subjects (often the group is itself modeled as a subject).

Access control systems provide the essential services of identification and authentication (I&A), authorization, and accountability where:

- identification and authentication determine who can log on to a system, and the association of users with the software subjects that they are able to control as a result of logging in;
- authorization determines what a subject can do;
- accountability identifies what a subject (or all subjects associated with a user) did.

Identification and Authentication (I&A)

Identification and authentication (I&A) is the process of verifying that an identity is bound to the entity that asserts it. The I&A process assumes that there was an initial vetting of the identity, during which an authenticator was established. Subsequently, the entity asserts an identity together with an authenticator as a means for validation.

The only requirements for the identifier is that it must be unique within its security domain.

Authenticators are commonly based on at least one of these four factors:

- Something you know, such as a password or a personal identification number (PIN). This assumes that only the owner of the account knows the password or PIN needed to access the account.
- Something you have, such as a smart card or token. This assumes that only the owner of the account has the necessary smart card or token needed to unlock the account.
- Something you are, such as fingerprint, voice, retina, or iris characteristics.
- Where you are, for example inside or outside a company firewall, or proximity of login location to a personal GPS device.

NOTES

Authorization

Authorization applies to subjects rather than to users (the association between a user and the subjects initially controlled by that user having been determined by I&A). Authorization determines what a subject can do on the system.

Most modern operating systems define sets of permissions that are variations or extensions of three basic types of access:

- Read (R): The subject can
  - Read file contents
  - List directory contents
- Write (W): The subject can change the contents of a file or directory with the following tasks:
  - Add
  - Create
  - Delete
  - Rename
- Execute (X): If the file is a program, the subject can cause the program to be run. (In Unix systems, the 'execute' permission doubles as a 'traverse directory' permission when granted for a directory.)

These rights and permissions are implemented differently in systems based on Discretionary Access Control (DAC) and Mandatory Access Control (MAC).

Accountability

Accountability uses such system components as audit trails (records) and logs to associate a subject with its actions. The information recorded should be sufficient to map the subject to a controlling user. Audit trails and logs are important for

- Detecting security violations
- Re-creating security incidents

If no one is regularly reviewing your logs and they are not maintained in a secure and consistent manner, they may not be admissible as evidence.

STUDENT ACTIVITY 3.1

1. What do you understand by Database Security?

2. How a Database is Protected?

3. What is Data administration?

4. What are CASE (Computer-aided Software/System Engineering) tools?

NOTES

Many systems can generate automated reports based on certain predefined criteria or thresholds, known as clipping levels. For example, a clipping level may be set to generate a report for the following:

- More than three failed logon attempts in a given period
- Any attempt to use a disabled user account

These reports help a system administrator or security administrator to more easily identify possible break-in attempts.

Access Control Techniques

Access control techniques are sometimes categorized as either discretionary or non-discretionary. The three most widely recognized models are Discretionary Access Control (DAC), Mandatory Access Control (MAC), and Role Based Access Control (RBAC). MAC and RBAC are both non-discretionary.

Discretionary Access Control

Discretionary access control (DAC) is an access policy determined by the owner of an object. The owner decides who is allowed to access the object and what privileges they have.

Two important concepts in DAC are

- File and data ownership: Every object in the system has an owner. In most DAC systems, each object's initial owner is the subject that caused it to be created. The access policy for an object is determined by its owner.
- Access rights and permissions: These are the controls that an owner can assign to other subjects for specific resources.

Access controls may be discretionary in ACL-based or capability-based access control systems. (In capability-based systems, there is usually no explicit concept of 'owner', but the creator of an object has a similar degree of control over its access policy.)

Mandatory Access Control

Mandatory access control (MAC) is an access policy determined by the system, not the owner. MAC is used in multilevel systems that process highly sensitive data, such as classified government and military information. A multilevel system is a single computer system that handles multiple classification levels between subjects and objects.

- Sensitivity labels: In a MAC-based system, all subjects and objects must have labels assigned to them. A subject's sensitivity label specifies its level of trust. An object's sensitivity label specifies the level of trust required for access. In order to access a given object, the subject must have a sensitivity level equal to or higher than the requested object.
- Data import and export: Controlling the import of information from other systems and export to other systems (including printers) is a critical function of MAC-based systems, which must ensure that sensitivity labels are properly maintained and implemented so that sensitive information is appropriately protected at all times.

Two methods are commonly used for applying mandatory access control:

- Rule-based access controls: This type of control further defines specific conditions for access to a requested object. All MAC-based systems implement a simple form of rule-based access control to determine whether access should be granted or denied by matching:
- An object's sensitivity label
- A subject's sensitivity label
- Lattice-based access controls: These can be used for complex access control decisions involving multiple objects and/or subjects. A lattice model is a mathematical structure that defines greatest lower-bound and least upper-bound values for a pair of elements, such as a subject and an object.

Few systems implement MAC. XTS-400 is an example of one that does.

Role Based Access Control

Role-based access control (RBAC) is an access policy determined by the system, not the owner. RBAC is used in commercial applications and also in military systems, where multi-level security requirements may also exist. RBAC differs from DAC in that DAC allows users to control access to their resources, while in RBAC, access is controlled at the system level, outside of the user's control. Although RBAC is non-discretionary, it can be distinguished from MAC primarily in the way permissions are handled. MAC controls read and write permissions based on a user's clearance level and additional labels. RBAC controls collections of permissions that may include complex operations such as an e-commerce transaction, or may be as simple as read or write. A role in RBAC can be viewed as a set of permissions.

Three primary rules are defined for RBAC:

1. Role assignment: A subject can execute a transaction only if the subject has selected or been assigned a role.
2. Role authorization: A subject's active role must be authorized for the subject. With rule 1 above, this rule ensures that users can take on only roles for which they are authorized.
3. Transaction authorization: A subject can execute a transaction only if the transaction is authorized for the subject's active role. With rules 1 and 2, this rule ensures that users can execute only transactions for which they are authorized.

Additional constraints may be applied as well, and roles can be combined in a hierarchy where higher-level roles subsume permissions owned by sub-roles.

Most IT vendors offer RBAC in one or more products.

Public Policy

In public policy, access control to restrict access to systems ("authorization") or to track or monitor behavior within systems ("accountability") is an implementation feature of using trusted systems for security or social control.

Auditing

The most general definition of an audit is an evaluation of a person, organization, system, process, project or product. Audits are performed to ascertain the validity

NOTES

and reliability of information, and also provide an assessment of a system's internal control. The goal of an audit is to the person/organization/system etc. under evaluation based on work done on a test basis.

Due to practical constraints, an audit seeks to provide only reasonable assurance that the statements are free from material error.

Hence, statistical sampling is often adopted in audits. In the case of financial audits, a set of financial statements are said to be true and fair when they are free of material misstatements - a concept influenced by both quantitative and qualitative factors.

Traditionally audits were mainly associated with gaining information about financial systems and the financial records of a company or a business. However recently auditing has begun to include other information about the system, such as information about environmental performance. As a result there are now professions that conduct environmental audits.

In financial accounting, an audit is an independent assessment of the fairness by which a company's financial statements are presented by its management. It is performed by competent, independent and objective person or persons, known as auditors or accountants, who then issue an auditor's report on the results of the audit.

Such systems must adhere to generally accepted standards set by governing bodies that regulate businesses. It simply provides assurance for third parties or external users that such statements present 'fairly' a company's financial condition and results of operations.

Authentication

In art, antiques, and anthropology, a common problem is verifying that a given artifact was produced by a certain famous person, or was produced in a certain place or period of history.

There are two types of techniques for doing this.

The first is comparing the attributes of the object itself to what is known about objects of that origin. For example, an art expert might look for similarities in the style of painting, check the location and form of a signature, or compare the object to an old photograph. An archaeologist might use carbon dating to verify the age of an artifact, do a chemical analysis of the materials used, or compare the style of construction or decoration to other artifacts of similar origin. The physics of sound and light, and comparison with a known physical environment, can be used to examine the authenticity of audio recordings, photographs, or videos.

Attribute comparison may be vulnerable to forgery. In general, it relies on the fact that creating a forgery indistinguishable from a genuine artifact requires expert knowledge, that mistakes are easily made, or that the amount of effort required to do so is considerably greater than the amount of money that can be gained by selling the forgery.

Criminal and civil penalties for fraud, forgery, and counterfeiting can reduce the incentive for falsification, depending on the risk of getting caught.

The second type relies on documentation or other external affirmations. For example, the rules of evidence in criminal courts often require establishing the chain of custody

of evidence presented. This can be accomplished through a written evidence log, or by testimony from the police detectives and forensics staff that handled it. Some antiques are accompanied by certificates attesting to their authenticity. External records have their own problems of forgery and perjury, and are also vulnerable to being separated from the artifact and lost.

NOTES

Currency and other financial instruments commonly use the first type of authentication method. Bills, coins, and cheques incorporate hard-to-duplicate physical features, such as fine printing or engraving, distinctive feel, watermarks, and holographic imagery, which are easy for receivers to verify.

Information content

The authentication of information can pose special problems, and is often wrapped up with authenticating identity.

Literary forgery can involve imitating the style of a famous author. If an original manuscript, typewritten text, or recording is available, then the medium itself (or its packaging - anything from a box to e-mail headers) can help prove or disprove the authenticity of the document.

However, text, audio, and video can be copied into new media, possibly leaving only the informational content itself to use in authentication.

Various systems have been invented to allow authors to provide a means for readers to reliably authenticate that a given message originated from or was relayed by them. These involve authentication factors like:

- A difficult-to-reproduce physical artifact, such as a seal, signature, watermark, special stationery, or fingerprint.
- A shared secret, such as a passphrase, in the content of the message.
- An electronic signature; public key infrastructure is often used to cryptographically guarantee that a message has been signed by the holder of a particular private key.

The opposite problem is detection of plagiarism, where information from a different author is passed of as a person's own work. A common technique for proving plagiarism is the discovery of another copy of the same or very similar text, which has different attribution. In some cases excessively high quality or a style mismatch may raise suspicion of plagiarism.

Factual verification

Determining the truth or factual accuracy of information in a message is generally considered a separate problem from authentication. A wide range of techniques, from detective work to fact checking in journalism, to scientific experiment might be employed.

DATABASE RECOVERY

Database protection can begin with the process of creation and publishing of appropriate protection standards for the database environment. The standards may include specific controls for the various relevant database platforms; a set of best

practices that cross over the platforms; and linkages of the standards to higher level polices and governmental regulations. An important procedure when evaluating database security is performing vulnerability assessments against the database. A vulnerability assessment attempts to find vulnerability holes that could be used to break into the database.

NOTES

A database protection program should include the regular review of permissions granted to individually owned accounts and accounts used by automated processes. The accounts used by automated processes should have appropriate controls around password storage such as sufficient encryption and access controls to reduce the risk of compromise. For individual accounts, a two-factor authentication system should be considered in a database environment where the risk is commensurate with the expenditure for such an authentication system.

CLIENT/SERVER ARCHITECTURE

Three of the four important characteristics of the database approach are:

1. insulation of programs and data (program-data and program-operation independence),
2. support of multiple user views, and
3. use of a catalog to store the data-base description (schema).

Here, we specify an architecture for database systems, called the Three-Schema Architecture, that was proposed to help achieve and visualize these characteristics.

The Three-Schema Architecture

The goal of the three-schema architecture is to separate the user applications and the physical database. In this architecture, schemas can be defined at the following three levels:

The External level

The external or view level includes a number of external schemas or user views. Each external schema describes the part of the database that a particular user group is interested in and hides the rest of the database from that user group. Here, each external schema is typically implemented using a representational data model, possibly based on an external schema design in a high-level data model.

The Conceptual level

The conceptual schema hides the details of physical storage structures and concentrates on describing entities, data types, relationships, user operations, and constraints. Usually, a representational data model is used to describe the conceptual schema when a database system is implemented. This implementation conceptual schema is often based on a conceptual schema design in a high-level data model.

The Internal level

The internal level has an internal schema, which describes the physical storage structure of the database. The internal schema uses a physical data model and describes the complete details of data storage and access paths for the database. The conceptual

level has a conceptual schema, which describes the structure of the whole database for a community of users.

The three-schema architecture is a convenient tool with which the user can visualize the schema levels in a database system.

NOTES

Most DBMSs do not separate the three levels completely, but support the three-schema architecture to some extent. Some DBMSs may include physical-level details in the conceptual schema.

In most DBMSs that support user views, external schemas are specified in the same data model that describes the conceptual-level information. Some DBMSs allow different data models to be used at the conceptual and external levels.

Mappings

Notice that the three schemas are only descriptions of data; the only data that actually exists is at the physical level. In a DBMS based on the three-schema architecture, each user group refers only to its own external schema. Hence, the DBMS must transform a request specified on an external schema into a request against the conceptual schema, and then into a request on the internal schema for processing over the stored database.

If the request is a database retrieval, the data extracted from the stored database must be reformatted to match the user's external view. The processes of transforming requests and results between levels are called mappings. These mappings may be time-consuming, so some DBMSs—especially those that are meant to support small databases—do not support external views. Even in such systems, however, a certain amount of mapping is necessary to transform requests between the conceptual and internal levels.

The processes of transforming requests and results between levels are called mappings.

FUNCTIONS OF CLIENT/SERVER—ADVANTAGES AND ISSUES

A Client Server system has one or more client processes and one or more server processes, and a client process can send a query to any one server process. Clients are responsible for user-interface issues, and servers manage data and execute transactions. Thus, a client process could run on a personal computer and send queries to a server running on a mainframe.

This architecture has become very popular for several reasons. First, it is relatively simple to implement due to its clean separation of functionality and because the server s centralized. Second, expensive server machines are not underutilized by dealing with mundane user-interactions, which are now relagated to inexpensive client machines. Third, users can run a graphical user interface that they are familiar with, rather than the (possibly unfamiliar and unfriendly) user interface on the server.

While writing Client Server applications, it is important to remember the boundary between the client and the server and keep the communication between them as set-

oriented as possible. In particular, opening a cursor and fetching tuples one at a time generates many messages and should be avoided. Even if we fetch serveral tuples and cache them at the client, messages must be exchanged when the cursor is advanced to ensure that the current row is locked.

NOTES

DISTRIBUTED DATABASES OBJECTIVES

A distributed database is a database that is under the control of a central database management system (DBMS) in which storage devices are not all attached to a common CPU. It may be stored in multiple computers located in the same physical location, or may be dispersed over a network of interconnected computers.

Collections of data (eg. in a database) can be distributed across multiple physical locations. A distributed database is distributed into separate partitions/fragments. Each partition/fragment of a distributed database may be replicated (ie. redundant fail-overs, RAID like).

Besides distributed database replication and fragmentation, there are many other distributed database design technologies. For example, local autonomy, synchronous and asynchronous distributed database technologies. These technologies' implementation can and does depend on the needs of the business and the sensitivity/confidentiality of the data to be stored in the database, and hence the price the business is willing to spend on ensuring data security, consistency and integrity.

Basic architecture

A database Users access the distributed database through:

Local applications: applications which do not require data from other sites.

Global applications: applications which do require data from other sites.

Important considerations

Care with a distributed database must be taken to ensure the following:

- **The distribution is transparent** — users must be able to interact with the system as if it were one logical system. This applies to the system's performance, and methods of access amongst other things.
- **Transactions are transparent** — each transaction must maintain database integrity across multiple databases. Transactions must also be divided into subtransactions, each subtransaction affecting one database system...

Advantages of distributed databases

- **Reflects organizational structure** — database fragments are located in the departments they relate to.
- **Local autonomy** — a department can control the data about them (as they are the ones familiar with it.)
- **Improved availability** — a fault in one database system will only affect one fragment, instead of the entire database.
- **Improved performance** — data is located near the site of greatest demand, and the database systems themselves are parallelized, allowing load on the

databases to be balanced among servers. (A high load on one module of the database won't affect other modules of the database in a distributed database.)

- **Economics** — it costs less to create a network of smaller computers with the power of a single large computer.
- **Modularity** — systems can be modified, added and removed from the distributed database without affecting other modules (systems).

NOTES

Disadvantages of distributed databases

- **Complexity** — extra work must be done by the DBAs to ensure that the distributed nature of the system is transparent. Extra work must also be done to maintain multiple disparate systems, instead of one big one. Extra database design work must also be done to account for the disconnected nature of the database — for example, joins become prohibitively expensive when performed across multiple systems.
- **Economics** — increased complexity and a more extensive infrastructure means extra labour costs.
- **Security** — remote database fragments must be secured, and they are not centralized so the remote sites must be secured as well. The infrastructure must also be secured (e.g., by encrypting the network links between remote sites).
- **Difficult to maintain integrity** — in a distributed database, enforcing integrity over a network may require too much of the network's resources to be feasible.
- **Inexperience** — distributed databases are difficult to work with, and as a young field there is not much readily available experience on proper practice.
- **Lack of standards** – there are no tools or methodologies yet to help users convert a centralized DBMS into a distributed DBMS.
- **Database design more complex** – besides of the normal difficulties, the design of a distributed database has to consider fragmentation of data, allocation of fragments to specific sites and data replication.

DISTRIBUTED DBMS

Distributed databases bring the advantages of distributed computing to the database management domain. A distributed computing system consists of a number of processing elements, not necessary homogenous, that are interconnected by a computer network, and that cooperate in performing certain assigned tasks. As a general goal, distributed computing systems partition a big, unmanagable problem into smaller pieces and solve it efficiently in a coordinated manner. The economic viability of this approach stems from two reasons: more computer power is harnessed to solve a complex task, and each autonomous processing element can be managed independently and develop its own applications.

We can define a distributed database (DDB) as a collection of multiple logically interrelated database distributed over a computer network, and a distributed database management system (DDBMS) as a software system that manages a distributed database while making the distribution transparent to the user.

NOTES

A collection of files stored at different nodes of a network and the maintaining of interrelationships among them via hyperlinks has become a common organization on the Internet, with files of Web pages. The common functions of database management, including uniform query processing and transaction processing, do not apply to this scenario yet. The technology is, however, moving in a direction such that distributed World Wide Web (WWW) databases will become a reality in the near future.

LOCATION TRANSPARENCIES—LOCATION, REPLICATION, FAILURE, COMMIT PROTOCOL AND CONCURRENCY, ETC.

Following types of transparencies are possible:

- Distribution or network transparency. This refers to freedom for the user from the operating details of the network. It may be divided into location transparency and naming transparency. Locations refers to the fact that command used to perform a task is independent of the location of data and the location of data and the location of the system where the command was issued. Naming transparency implies that once a name is specified, the named objects can be accessed unambiguously without additional specification.
- Replication transparency. Copies of data may be stored at multiple sites for better availability, performance, and reliability. Replication transparency makes the user unaware of the existence of copies.
- Fragmentation transparency. Two type of fragmentation are possible. Horizontal fragmentation distributes a relation into sets of tuples (rows). Vertical fragmentation distributes a relation into subrelations where each subrelation is defined by a subset of the columns of the original relation. A global query by the user must be transformed into several fragment queries. Fragmentation transparency makes the user unaware of the existence of fragments.
- Other transparencies include design transparency and execution transparency—referring to freedom from knowing how the distributed database is designed and where a transaction executes.

NOTES

SUMMARY

1. Database security is the system, processes, and procedures that protect a database from unintended activity.
2. Databases provide many layers and types of information security, typically specified in the data dictionary.!
3. Two crucial aspects of database security compliance include patch management and the review and management of permissions (especially public) granted to objects within the database.
4. A Single sign-on system should store the database user's credentials (login id and password), and authenticate to the database on behalf of the user.
5. Access control is the ability to permit or deny the use of a particular resource by a particular entity.
6. An electronic access control door can contain several elements. At its most basic there is an electric lock.
7. The bar code can then be read with an optical reader tuned to the infrared spectrum, but can not easily be copied by a copy machine.
8. Wiegand card technology is a patented technology using embedded ferromagnetic wires strategically positioned to create a unique pattern that generates the identification number.
9. There are two types of smart cards: contact and contactless. Both have an embedded microprocessor and memory.
10. The PIN is usually a number consisting of four to eight digits.
11. Identification and authentication (I&A) is the process of verifying that an identity is bound to the entity that asserts it.
12. Authorization determines what a subject can do on the system.
13. Accountability uses such system components as audit trails (records) and logs to associate a subject with its actions. The information recorded should be sufficient to map the subject to a controlling user.
14. Discretionary access control (DAC) is an access policy determined by the owner of an object. The owner decides who is allowed to access the object and what privileges they have.
15. Mandatory access control (MAC) is an access policy determined by the system, not the owner.
16. Role-based access control (RBAC) is an access policy determined by the system, not the owner.
17. The most general definition of an audit is an evaluation of a person, organization, system, process, project or product.
18. The authentication of information can pose special problems, and is often wrapped up with authenticating identity.
19. In cryptography, encryption is the process of transforming information (referred to as plaintext) using an algorithm (called cipher) to make it unreadable to anyone except those possessing special knowledge, usually referred to as a key.
20. Data integrity is a term used in computer science and telecommunications that can mean ensuring data is "whole" or complete, the condition in which data are identically maintained during any operation.
21. In any organization where many persons use the same resources, there is a need for a chief administrator to oversee and manage these resources.
22. The client/server architecture was developed to deal with computing environments in

NOTES

which a large number of PCs, workstations, file servers, printers, database servers, Web servers, and other equipment are connected via a network.

23. Many Web applications use an architecture called the three-tier architecture, which adds an intermediate layer between the client and the database server.
24. Data administration is, in computing science, the administration of the organisation of data, usually as stored in databases under some Database Management System or alternative systems such as electronic spreadsheets.
25. CASE (Computer-aided Software/System Engineering) refers to the methods dedicated to an engineering discipline for the development of information systems together with automated tools that can be used in this process.
26. A Lock is a database system object associated with a database object (typically a data item) that prevents undesired (typically synchronization rule violating) operations of other transactions by blocking them.
27. Database protection can begin with the process of creation and publishing of appropriate protection standards for the database environment.
28. A Client Server system has one or more client processes and one or more server processes, and a client process can send a query to any one server process.
29. A distributed database is a database that is under the control of a central database management system (DBMS) in which storage devices are not all attached to a common CPU.
30. Distributed databases bring the advantages of distributed computing to the database management domain.

SELF ASSESSMENT QUESTIONS

1. What do you understand by Database Security?
2. What are Process Controls?
3. How a Database is Protected?
4. What are 2-phase Command and Working Protocols?
5. What is Data administration?
6. What are CASE (Computer-aided Software/System Engineering) tools?
7. How would you lock a database?
8. How database can be protected?
9. What is a Client Server system?
10. What is a distributed database?
11. Describe the followings :

Access Control System Operation	Access Control System Components
Credential	Bar Code Technology
Magnetic Stripe Technology	Wiegand Card Technology
Proximity Card Technology	Smart Card
PIN	Computer Security
Identification and Authentication (I&A)	Authorization
Accountability	Access Control Techniques
Public Policy	

Multiple Choice Questions

NOTES

1. REX is :
   (a) Request to exit (b) Request to exist (c) Ready to exit
2. PIN is:
   (a) Private Identification Number
   (b) Personal Identification Number
   (c) Personal Information Number
3. ACL is :
   (a) Alter Control List (b) Access Clear List (c) Access Control List
4. I&A is :
   (a) Identification and Authentication
   (b) Information and Authentication
   (c) Identification and Access
5. DAC is :
   (a) Discretionary Add Control
   (b) Discretionary Access Control
   (c) Direct Access Control
6. MAC is :
   (a) Man Access Control
   (b) Mandatory Access Clear
   (c) Mandatory Access Control
7. RBAC is :
   (a) Role Based Access Control
   (b) Role Basic Access Control
   (c) Real Based Access Control
8. DBA is :
   (a) Database Administrator (b) Database Advisor (c) Database Accurator
9. RDBMS is :
   (a) Relative Database Management System
   (b) Relational Database Management System
   (c) Rotational Database Management System
10. CASE is :
    (a) Computer Aided Software/System Engineer
    (b) Computer Added Software/System Engineering
    (c) Computer Aided Software/System Engineering
11. DDB is :
    (a) Distributed Database
    (b) Deleted Database
    (c) Diluted Database
12. DDBMS is :
    (a) Deleted Database Management System
    (b) Distributed Database Management System
    (c) Diluted Database Management System

NOTES

True/False Questions

1. Databases provide many layers and types of information security, typically specified in the data dictionary.
2. Access control is the ability to permit or deny the use of a particular resource by a particular entity.
3. An electronic access control door cannot contain several elements. At its most basic there is an electric lock.
4. Wiegand card technology is a patented technology using embedded ferromagnetic wires strategically positioned to create a unique pattern that generates the identification number.
5. Identification and authentication (I&A) is the process of verifying that an identity is bound to the entity that asserts it.
6. Authorization determines what a subject cannot do on the system.
7. Discretionary access control (DAC) is an access policy determined by the owner of an object. The owner decides who is allowed to access the object and what privileges they have.
8. Mandatory access control (MAC) is an access policy determined by the owner.
9. The most general definition of an audit is an evaluation of a person, organization, system, process, project or product.
10. The authentication of information can pose special problems, and is often wrapped up with authenticating identity.
11. Database protection can begin with the process of creation and publishing of appropriate protection standards for the database environment.

Short Questions with Answers

1. What is Access Control?

Ans. Access control is the ability to permit or deny the use of a particular resource by a particular entity. Access control mechanisms can be used in managing physical resources (such as a movie theater, to which only ticketholders should be admitted), logical resources (a bank account, with a limited number of people authorized to make a withdrawal), or digital resources (for example, a private text document on a computer, which only certain users should be able to read).

2. What is Wiegand Card Technology?

Ans. Wiegand card technology is a patented technology using embedded ferromagnetic wires strategically positioned to create a unique pattern that generates the identification number. Like magnetic stripe or bar code, this card must be swiped through a reader to be read. Unlike those other technologies the identification media is embedded in the card and not susceptible to wear.

3. What is Bar Code Technology?

Ans. A bar code is a series of alternating dark and light stripes that are read by an optical scanner. The organization and width of the lines is determined by the bar code protocol selected. There are many different protocols but code 39 is the most popular in the security industry. Sometimes the digits represented by the dark and light bars are also printed to allow people to read the number without an optical reader.

4. What is Bar Code Technology?

Ans. There are two types of smart cards: contact and contactless. Both have an embedded microprocessor and memory. The smart card differs from the card typically called a proximity card in that the microchip in the proximity card has only one function: to provide the reader with the card's identification number. The processor on the smart card

has an operating system and can handle multiple applications such as a cash card, a pre-paid membership card, and even an access control card.

5. What is PIN?

Ans. A personal identification number (PIN) falls in the category of what you know rather than what you have. The PIN is usually a number consisting of four to eight digits. Less and the number is too easy to guess. More and the number is too difficult to remember.

NOTES

6. What are Discretionary Access Control (DAC), Mandatory Access Control (MAC), and Role Based Access Control (RBAC)?

Ans. **Discretionary Access Control**

Discretionary access control (DAC) is an access policy determined by the owner of an object. The owner decides who is allowed to access the object and what privileges they have.

Mandatory Access Control

Mandatory access control (MAC) is an access policy determined by the system, not the owner. MAC is used in multilevel systems that process highly sensitive data, such as classified government and military information. A multilevel system is a single computer system that handles multiple classification levels between subjects and objects.

Role Based Access Control

Role-based access control (RBAC) is an access policy determined by the system, not the owner. RBAC is used in commercial applications and also in military systems, where multi-level security requirements may also exist. RBAC differs from DAC in that DAC allows users to control access to their resources, while in RBAC, access is controlled at the system level, outside of the user's control.

7. What is Factual Verification?

Ans. Determining the truth or factual accuracy of information in a message is generally considered a separate problem from authentication. A wide range of techniques, from detective work to fact checking in journalism, to scientific experiment might be employed.

8. What is a Client Server?

Ans. The client/server architecture was developed to deal with computing environments in which a large number of PCs, workstations, file servers, printers, database servers, Web servers, and other equipment are connected via a network. The idea is to define specialized servers with specific functionalities. For example, it is possible to connect a number of PCs or small workstations as clients to a file-server that maintains the files of the client machines. Another machine can be designated as a printer server by being connected to various printers; thereafter, all print requests by the clients are forwarded to this machine. Web servers or e-mail servers also fall into the specialized server category.

9. What does Data Administrator do?

Ans. Data Administration ideally begins at software conception, ensuring there is a data dictionary to help keeping consistency and avoid redundancy and modelling the database so as to make it logical and usable, by means of the normalisation technique.

10. What are CASE Tools?

Ans. CASE (Computer-aided Software/System Engineering) refers to the methods dedicated to an engineering discipline for the development of information systems together with automated tools that can be used in this process.

11. What is Concurrency Control?

Ans. Concurrency control in database management systems (DBMS) ensures that database transactions are performed concurrently without the concurrency violating the data integrity of a database. Executed transactions should follow the ACID rules. The DBMS must guarantee that only serializable (unless Serializability is intentionally relaxed), recoverable

schedules are generated. It also guarantees that no effect of committed transactions is lost, and no effect of aborted (rolled back) transactions remains in the related database.

12. What are Distributed databases?

NOTES

Ans. A distributed database is a database that is under the control of a central database management system (DBMS) in which storage devices are not all attached to a common CPU. It may be stored in multiple computers located in the same physical location, or may be dispersed over a network of interconnected computers.

Distributed databases bring the advantages of distributed computing to the database management domain. A distributed computing system consists of a number of processing elements, not necessary homogenous, that are interconnected by a computer network, and that cooperate in performing certain assigned tasks.

ANSWERS

Multiple Choice Questions

1. a	2. b	3. c	4. a
5. b	6. c	7. a	8. a
9. b	10. c	11. a	12. b

True False Questions

1. T	2. T	3. F	4. T
5. T	6. F	7. T	8. F
9. T	10. T	11. T	

CHAPTER 4

DATABASE APPLICATIONS

Learning Objectives

After going through this chapter, you should appreciate the following:

- Financial Systems
- Marketing System
- Foreign Trade
- Inventory Information Systems.

NOTES

FINANCIAL SYSTEMS

An important area of computerization is the Accounting System. Usually this is the second application to be computerized in any organization, the first being the Payroll System. A well maintained accounting setup with capabilities of the quick production of reports and summaries is an asset to any business activity in that the management is aware of the latest up to date financial standing of the business thereby facilitating right decision making. In a non computerized accounting system the books of accounts are normally completed only by two to thee months after closing of the period and knowledge of what happened before two or three months is only of historical importance.

An accounting system is ideally suited for computerization, since the system involves mass processing of data which otherwise take many man hours to complete. Accuracy is of paramount important as regards accounting data is concerned which is often lost in a manual system. Also the type of accounting data are of simple numerical in nature involving large volumes and laborious calculations.

In olden days, in a manual system, closing of accounts of production of periodical reports were done rather infrequently after days of poring over the books of accounts and painstakingly arriving at the required totals during which time all routine accounting activities were to be suspended. Today, with the advent of computers, it has become possible to prepare profit and loss figures monthly or if necessary even on a daily basis.

Unfortunately to the technical uninitiated, the accounting system will remains a mystery and except for students with a commerce background, computer professionals are generally unaware of the nuances of the system. This is primarily because the terminologies used and the concept of accounts are rather strange to students of other faculties.

To understand a computerized accounting system it is therefore imperative that one should have a clear background to this field of study as otherwise when talking to the accounting personnel, he would be faced with terminologies like debits and credits which would be rather incoherent to a software personnel.

Let us now try to understand the accounting system using our household as a typical example. You must have heard many a times "to put your house in order" usage when things go wrong. Is it not?

While a household is strictly not a business organization, we need to make "both ends meet" in a household. What does this means? You should be able to live off well without much difficulties with your income from all sources. In other words, you should control your expenditure so that it never exceeds your income and even if at some stage it exceeds due to unavoidable reasons, you should be able to mobilize enough funds to offset such expenses. This is exactly what is required in a business organization too, except that in a business, success is measured in terms of profits you make. Even in a household, if you can make a saving, well and good. Is it not so.

How do we achieve this objective? First and foremost is that we should account very pisa that is being spent and earned under different heads of accounts. We will now see what are the different heads of accounts possible in a household to maintain a typical accounting system.

1. Land and Buildings.
2. Machineries – for e.g. Mixer Grinder, Refrigerator etc.
3. Furniture – e.g. Cots, Chairs, table etc.
4. Tools – e.g. Stove, Utensils, Hammer, Screw driver etc.
5. Stocks – Many items Like Rice, Sugar, Bulbs, etc kept in stock.
6. Cash – Amount of cash kept in a Cash box for day to day minor expenditure.
7. Bank – Amount kept in a Bank account for expenditure of a larger nature.
8. Debtors – There may be persons who owe money to you for various reasons. For e.g. you have given a loan to Mr. X.
9. Creditors – You may perhaps owe money to M/s ABC traders for purchases made.
10. Loans taken – From may be the Bank for construction of the house.
11. Salaries and allowances – Paid to domestic systems.
12. Telephone charges
13. Electricity charges
14. Cost of stationery
15. Travelling expenses
16. Cost of Provisions
17. Cost of Milk
18. Cost of Vegetables
19. School Fees
20. Entertainment Expenses
21. Salary received from employer
22. Proceeds of Coconut Sales
23. Interest received on deposits from the bank
24. Proceeds of sale – other agricultural products
25. M/s. ABC Coconut traders – They buy Coconut from you regularly
26. M/s. XYZ Agricultural traders – They buy other agricultural products from you
27. Varkey's supermarket – From whom you regularly buy your domestic requirements.
28. Pramod Kumar – The milk man who supplies milk regularly
29. Moosa Koya – Supplier of Vegetables
30. Bank of India – A loan has been taken from this bank for an emergency

The above accounts heads are by no means all that there is to it. There could be many more account heads. But for understanding an accounts system, the above heads would suffice.

NOTES

On analysis, you can find heads 1 though 8 are of a peculiar nature. They are all some from of an ASSET as far as a household is concerned. You could sell them and get money according to the worth of the assets. Cash is hard cash, you can use them, money in the bank can be encashed, money from debtors can be realized and so on. If you decide to migrate to America, you can convert these heads into solid cash. Do you get the concept of an ASSET?

ASSETS are of two kinds, Fixed Assets and Current Assets. Items 1 through 4 are fixed assets, while items 5 through 8 are examples of current assets. Fixed assets are those which cannot be very easily sold while it is easy liquidity as far as current assets are concerned, i.e. easy it convert to into cash. Items 9 and 10 are LIABILITIES to the household. If you are going away for good, Creditors should be paid off, loan taken should be repaid. Thus these items are a burden to you. Isn't it?

In an actual business environment, there will be appearing a major account head under the caption "LIABILITIES", titled Capital Account. This represents the initial investment or money received from shareholders to start the organization or for further expansion of business. Until the organization is wound up or liquidated, these funds received remain as a liability in the books of accounts.

Items 11 to 24 represents heads coming under the Profit and Loss accounts (generally known as P & L account heads). These are further divided into INCOME heads (items 21 to 24) and EXPENDITURE heads (item 11 to 20). Expenditure could also be subdivided into direct and indirect expenditure heads. We have considered here only direct expenditures and indirect expenditure will be discussed later.

Finally items 25 and 26 are the debtors to the household while items 27 to 30 are the creditors. Debtors are those who owe you money for whatever reasons it may be like in this case, they owe you money for purchases made from you. Creditors are those to whom you owe money in the course of business transactions with them. Here we owe them money for purchases made form them. In the case of item 30, you owe Bank of India money to be repaid against a loan taken from them.

With the above background, let us now study the various procedures of book keeping involved in an Accounting System.

Double Entry System of Book Keeping

You would have heard in your Physics classes that "matter can neither be created nor be destroyed". It only changes form. Similarly in an accounting system, we have a parallel.

Income and Expenditure, direct or indirect are what constitutes an accounting environment. When an expenditure is incurred, you always get something in return. You can say that the expenses under any head of account gets converted into an income under some other head of account. Remember that income or expenditure need not necessarily be in the form of hard currency, it could be in any other form which has an equivalent value in terms of money. So it follows that any transaction has two sides to it one affecting an expenditure head and the other an income head. We have here used the terms income and expenditure in the literal sense. In accounting parlance, the corresponding items are Debits and Credits. Translating our concept of income and expenditure to accounting terminology, we say that for every Debit transaction, there is a corresponding Credit transaction. We will illustrate this by an example.

Suppose we spend Rs. 20 from our cash box in the household and purchase vegetables for use at dinner. We debit Rs. 20 to the head of account "Cost of vegetables" and Credit Rs. 20 to the head of account titled "Cash". What does this mean? Rs. 20 has gone out of the cash box (expended) and 'Cost of vegetables' account has gained by a similar amount. You can visualize the various heads of accounts as boxes where you are going to keep money pertaining to that account. Thus 'Cash' is box as is the 'Cost of vegetables'. What really has happened here, money has changed boxes. Rs. 20 from Cash has gone to the box 'Cost of vegetables. You will now wonder than where do the vegetables come from. Actually you have paid the money to the vegetable vendor and got in return vegetables worth Rs. 20. so when we say the 'Cost of vegetables' head has gained by Rs. 20, it only means that vegetables worth Rs. 20 has gone into this box. For accounting purposes, the actual material purchased is of little consequence, but accounts are only interested in the money equivalent to the item purchased.

Thus if a cheque payment is made to your servant for Rs. 500 towards his salary for the month, we credit 'Bank' account with Rs. 500 and debit "Salaries and allowances" account with an equivalent amount. (In fact, the payment is made to your servant and this is of no consequence to your Accounting Syste,).

You will find the above concept is rather strange, but you will get used to it as we move along. This is primarily because, we confuse accounting terms with their meanings in the literal sense and it will take time to comprehend these terms in the spirit of a real accounting system.

We have now seen that the real difficult task in accounting to decide which accounts is to be debited and which corresponding account is to be credited when there is a transaction in a business or household. Fortunately, this is the job of an Accountant and we as computer students are not very much interested in this aspect. However, we should try to understand these concepts generally to have some superficial knowledge of the system.

Account heads are basically divided into Personal Accounts and Impersonal accounts. Impersonal accounts are further classified into Property or Real and Nominal or Fictious Accounts.

Personal A/C

Where an accounting transaction affects a person such as individual or any body of individuals such as an association, club or a company and the like in a Credit transaction, is known as a Personal a/c e.g. Debtors such as M/s. XYZ agricultural traders, Creditors such as Varkey's Super Market, Pramod Kumar etc.

Suppose we sell coconuts worth Rs. 5000 to M/s ABC Coconut traders. We debit ABC Coconut trades with Rs. 5000 and credit a similar amount to the head "Proceed of Coconut Sales"

Similarly if we buy vegetables worth Rs. 30 from Moosa Koya, we debit the account "Cost of vegetables" with Rs. 300 while crediting Moosa Koya with a similar amount.

The general principle in respect of a Personal a/c is to debit the Receiver and Credit and Giver.

In the above example, Moosa Koya is the giver of vegetables worth Rs 300 and who receives it, the head titled "Cost of vegetables". Now verify the above principle with what we have done with the transaction.

NOTES

Property or Real A/c

All commodities having commercial value and which can be touched and seen are known as properties in real existence normally involved in either exchange or transfer transactions are called as such.

For e.g. Land and Buildings, Machineries, Furniture etc.

Proceeds of Coconut sales and other agricultural sales.

Suppose you buy 3 chairs against payment of Cash from a furniture shop. You debit the cost of say Rs 1200 of the chairs to "furniture" Account while crediting it to "Cash"account. Here Rs 1200 has goneout of 'Cash' a/c and 'Furniture a/c' where the Rs. 1200 worth of chairs has come is in debited.

Nominal or Fictious A/c

Expenses and Income of various types incurred or earned on availing or rendering services of any nature which do not cause any property go away are called Nominal or Fictious Accounts. Expenses result in permanent losses which cannot be recovered at all while income received cannot be claimed by any person.

For example, Electricity charges, Salaries and allowances etc.

Interest received, Commission received for services rendered etc.

The general principle in respect of this type of account is to debit expenses or losses while crediting incomes or gains.

The above principles enumerated are known as the Golden rules of Accounting.

Journal

Journal is the prime book of entry. It is also known as the first book of an accounting entry. Every accounting transaction is first required to be recorded into this journal debit transactions are entered into the debit column and credit transactions into the 'Credit' column. The account number (Head of accounts are coded) and the particulars of the transactions are written into the appropriate columns. Given below is a typical format of a Journal.

JOURNAL

Date	*Reference No.*	*Particulars*	*A/c. Head*	*Debit*
Credit				
3-11-02	CV/327	Cash Paid for 3 chairs	BL006	1200.00
3-11-02	CM/135/02	Furniture A/c.	BL0003	1200.00

The reference number shows the document number of the source document from which entries are made into the Journal. BL006 is the code no. for Cash account and BL003 the code number for the furniture account. When cash is paid out, a cash payment voucher is prepared and authorized by the Accounts Officer and this voucher number is entered in the 'reference' column. The supplier of the chairs ABC traders would give you a Cash bill when payments are received and this bill number is entered as 'Reference' against the Furniture A/c.

We will now study the various source documents from which an accounting transaction is normally generated.

Invoice (Bill)

When items or services are sold on credit for e.g. sale of coconuts to M/s. ABC traders, a bill is prepared and issued to the customer describing the goods or services sold, their quantity and value. Form this bill, an entry of sale of goods or services on credit is recorded into the journal. In case of a sales invoice the bills will be serially numbered and this bill number is entered in the 'Reference' column of a journal.

Similarly, when goods or services are purchased on credit, from a supplier of such goods or services, a bill is received from the supplier as detailed above from which the entry for purchases on credit is reordered into the journal. Purchase bill will not be serially numbered since they are received from various suppliers.

The format of invoices has already been elaborated in the chapters on sales and purchases.

Cash Memo

When purchase are made from a supplier against payment of cash, the supplier issues a cash memo which is subsequently journalized. This cash memo is very similar to an invoice in nature except that this memo also serves the purpose of a receipt of payment for goods supplied. Purchase Cash memos also contain a number, which may not be serial in nature since purchases are made form various suppliers.

Likewise, when items or services are sold for cash payment, a cash bill is issued to the customers in the same format of a sales invoice with numbers given serially. This also serves as an acknowledgement for payment received for goods or services sold.

Receipts

Whenever payments are made either in cash or by cheque for any purpose, a receipt is issued by the receiver and likewise when money is received, receipts are issued bin acknowledgement. When we issue receipts, they will be serially numbered. Receipts from others will contain a receipt number, but not in any serial order.

Cash/Cheque payment Voucher

Whenever payment is made in cash or cheque for expenses of any nature for which an official receipt cannot be obtained form the receiver of the money as in the case of say travelling expenses paid to an employee, a payment voucher is prepared by the payer and the signature is obtained thereon from the receiver, wherein the details of such expenses are recorded together with the account number to which it should be debited. The entries from these vouchers are also entered into the journal. Vouchers are serially numbered for reference.

Debit Note

When purchases are made on credit and later on, if any part of it or whole of it are returned for any genuine reasons, to the supplier, a memo is issued to the supplier indicating therein, the reason for the return of the items and the value thereof. At the same time, the supplier is informed that his account which was previously credited with the value of purchases as giver of items or services has now been debited with the value of items returned to him. As a receiver of value of the returned items his account is debited. This sort of a memo is called a Debit Note.

NOTES

Credit Note

When items come in as returned or services rendered are disapproved by the customers to whom they were supplied or offered on credit, a memo is issued to the customers acknowledging the receipt of items returned or accepting the disapproval of services rendered and the value of returns. Thus the customer is informed that this account in the books of the supplier, which was previously debited as receiver is now credited with the value of returns as giver of that value. This sort of a memo is called a Credit note.

We will consider an indirect expenditure. For example, Depreciation of an asset. When you buy a chair, though it is an expense is not considered as such for calculating profit or loss of an organization. This is because a chair can be used for many years and taking the full cost of the chair as an expense in the year of purchase is not quite right. So what do we do. Let us assume that the cost of the chair is Rs 400 and that this chair can be used for 10 years. Then every year the chair gets depreciated by 10%. It is not so. This amount Rs. 40 is treated as an expense in the year of purchase. So we create the following transaction.

1. To depreciation of a chair – Furniture a/c – Credit Rs 40
2. "Depreciation a/c – Debit Rs. 40

Thus the "furniture account" gets reduced by Rs 40 and the "Depreciation Account" gains by Rs 40. This depreciation account is reckoned for profit and loss calculations.

Note that the net asset value of the chair has now become Rs 360 and next year, it will be depreciated only by Rs 36 i.e. 10% of Rs 360. You will observe that the net asset value of the chair never becomes zero.

How a certain asset is depreciated depends on tax laws of the land and directions contained in the company's Act for the relevant financial year.

Such sort of transactions which are of indirect nature are written up in a Journal Voucher and this document also becomes a Source document of an accounting system. Given below is the format of a Journal Voucher.

Every journal voucher is authorized by an officer of the accounts department to confirm that the debits and credits are charged to the correct account heads.

Now to summarize, we have the following source documents in an accounting system.

1. Bill (Invoices)
2. Cash Memos
3. Receipts
4. Cash/Cheque Payment Vouchers
5. Debit notes
6. Credit notes
7. Journal vouchers

Using the above documents as input, an accounting system is computerized. We will now study how this is done in detail. The various statements and reports produced in an accounting system will be explained at the appropriate placed.

Computerization

The first step in computerizing an accounting system is to identify the heads of accounts required and code them as appropriate. We have already done this at the beginning of this chapter. In an actual business environment there would be many more heads of accounts and these should be studied in detail and classified into major and minor groups, such as Fixed Assets and within fixed assets, Land and Buildings, Plant & machinery etc. A suitable coding structure should then be designed. A typical coding structure is given below.

NOTES

The account heads in a business can generally be classified as follows:

Balance Sheet Heads

Within this major grouping, we can have various subgroups as under.

1. ASSETS
2. LIABILITIES

Within ASSETS, minor grouping can be done as

1. FIXED ASSETS: Within which sub grouping can be done as
    1. Land
    2. Building
    3. Plant & Machinery
    4. Furniture etc.

    etc.
2. CURRENT ASSETS: sub groups within as
    1. Stocks
    2. Cash
    3. Bank
4. Debtors etc.

Similarly within liabilities, there can be Capital a/c and Current liabilities with in which Creditors, Loans etc. can be classified.

Profit & Loss Account Heads:

These are classified into

1. Income – Account Heads representing income within which there could be
    1. Direct income Heads like
        1. Sales A/c – Coconuts
        2. Sales A/c – Other agricultural products.
    2. Indirect Income Heads like
        1. Interest gained from deposits
        2. Refunds received
        3. Sale of Assets etc.

NOTES

2. Expenditure – Account Heads representing expenditures within which there could be

    1. Direct Expenditure like
        1. Salaries & Allowances
        2. Travelling expenses
        3. Cost of Raw materials consumed

        etc.
    2. Indirect Expenditure like
        1. Depreciation of an Asset
        2. Bad debts written off

        etc.

3. DEBTORS : All debtors are given a Head of account like
    1. ABC traders
    2. Metro Trading Co.
    3. Global Enterprises

    etc.

    Sometime debtors can be classified as
    1. Customers who are regular buyers of your product.
    2. Individual debtors with whom you have occasional transactions
    3. Staff members of the organization etc.

4. CREDITORS: All creditors are also given a Head of account like
    1. Varkey's Super Market
    2. Delhi Hardware Stores
    3. S.K. & F
    4. Fertilisers & Chemicals of Travancore
    5. Mr. Pramod Kumar

    etc.

Creditors can also be classified as

1. Suppliers of Raw materials
2. Suppliers of Machinery and Space parts
3. Machinery Maintenance companies
4. Sundry creditors

etc. depending on the type of creditors in any organization.

Classification of account heads as above is not a very difficult job as you can seek

the help of the Accountant who are quite familiar with the specific requirements of an organization. After such classifications, codes can be allotted using any of the coding technique. For example

NOTES

BL/01/01/01 - could represent Land under fixed assets under ASSETS head under Balance Sheet a/c heads.

BL – Balance Sheet – Major Group

BL01 - " – Asset – Inter-Group

BL0101 - " " _ Fixed Assets Minor Group

BL010101 – " " " – Land etc.

BL010102 - can represent Buildings

BL020104 - Creditors etc.

BL – Balance Sheet

02 – Liabilities

01 – Current Liabilities

04 – Creditors

Similarly PL/01/02/03 – Can represent Sale of assets under indirect income under Income head of P & L Account.

A good coding system, needless to say, goes a long way in efficient computerisation of an accounting function.

After the codes are thus allotted, the next step is to design the Master Files. In a typical accounting system, thee are two Master Files.

1. Account Master File
2. Budget Master File – used in Budgetory control system which will be discussed later.

Accounts Master File

This master file is designed depending on whether you are going to do batch processing or on line processing. In a batch processing the master files are updated only periodically, normally once in a month. In an on line processing environment, the master file are updated immediately on a transaction taking place and will always reflect the latest financial position of the organization.

Master File Layout

Sl No.	*Description*	*Width*	*Dec*	*Type*	*Remarks*
1.	Account Number	8		Character	Code allotted
2.	Description of the account	30		Character	Name of account

NOTES

3.	Financial Year ending	8		Date	Year ending date e.g. 31/03/02
4.	Current Balance	10	2	Numeric	
5.	Todate debits	10	2	Numeric	
6	Todate credits	10	2	Numeric	
7	Opening Balance – April	10	2	Numeric	
8	" – May	10	2	Numeric	
9	" – June	10	2	Numeric	
10	" – July	10	2	Numeric	
11	" – August	10	2	Numeric	
12	" – September	10	2	Numeric	
13	" – October	10	2	Numeric	
14	" – November	10	2	Numeric	
15	" – December	10	2	Numeric	
16	" – January	10	2	Numeric	
17	Opening Balance – February	10	2	Numeric	
18	" – March	10	2	Numeric	
*19	Closing Balance for the year	10	2	Numeric	

We have assumed here that the financial year of the organization is April to March of the succeeding year, which also happens to be the Tax year. Some organization do have a separate financial year e.g. July to June of next year or the Calender year itself.

Current Balance

This is the latest balance up to the last transaction entered into the computer.

This can be a positive or negative figure. Negative figures are preceded by a minus sign e.g. -6325.00. In other words, the current balance represent the sum total of all debits and credits of all transactions that has occurred in the current financial year plus any opening balance for the respective financial year. In computers, credits are considered as negative figures and debits as positive ones, normally. There is no harm done even if you enter a debit as a negative figure as when balancing is done, usual mathematical rules of addition or subtraction are followed.

To date Debits/Credits

These fields are used to store the total of all debit transaction and credit transactions that has occurred in a financial year in respect of the given account number contained in the file layout.

Opening Balance:

This is the balance in respect of the account number as at the beginning of every

calendar month. The usage of this is primarily for selective printing of a ledger for a given month, which will be explained later in this chapter.

Closing Balance for the year

This is the balance as the end of the financial year which can again be positive or negative in respect of the account number.

NOTES

The above master file is usually maintained in the account number order. At the beginning of the financial year, To date debits/credits fields as well as opening balances from May through March are initialized to Zero. The closing balance for the year also is set to zero. The fields current balance and opening balance for April would be the closing balance of the account for the previous year in respect of all balance sheet, Debtors & Creditors account heads. In Profit & Loss accounts, these are also initialized to zero.

At any point of time, the current balance would be equal to (Opening balance for April + To date Debits – To date Credits). Similarly the Closing balance for the year would be the same figure as that of the current balance at the end of the year. So this field is strictly superfluous and can be removed. If you have understood this, you are doing well with the comprehension of the accounting system.

The first program to be written in an accounting system thus would be to create, maintain and query the Accounts Master File. Normally, the data contained in this file should never be modified except at the beginning of a financial year when the Numeric fields of the file would be initialized to zero.

The Program should check that when modifications are done, the current balance and opening balance for April should be the only fields where data other than zero can be entered in respect of non P & L accounts. For all other fields if the data is 1st April of any year, you can zeroise the fields. For any other month, no changes to data should be permitted. In other words, data in the above file should never be manually changed except at the beginning of every financial year to initialize the file. Querying this file is always allowed.

When the master file creation is complete we should next prepare the transaction files. Any for of expenditure or income is entered into this transaction file. In a manual system, transactions which are authenticated through any one of the source documents described earlier are first posted into a journal. In the computerized system, this is not necessary. Records are created and appended to the transaction file direct from the source documents. Often transaction records are generated automatically when the source document is itself prepared on a computer. For example, when an invoice is printed, the corresponding transactions are appended to the accounting transaction file by the sales system. Thus we see that all other systems computerized in an organization is ultimately linked to the accounting system. We will discuss this aspect a little more in detail later.

We will now look at a transaction file layout.

Transaction File Layout

Sl No.	*Description*	*Width*	*Dec*	*Type*	*Remarks*
1.	Date of transaction	8		Date	

NOTES

2.	Source document Number	8		Character	e.g. I/94/032
3.	Document Date	8		Date	
4.	Document Code	3		Character	
5.	Narration	30		Character	
6.	Account Number	8		Character	
7.	Amount	10	2	Numeric	(Minus for Credit amount)
8.	Cost Centre code	2		Numeric	
9.	Bank Code	2		Numeric	(For cheque payments & receipts)
10.	Cheque Number	8		Character	

All transactions emanate from a source document and it is this source document number and date of document which are entered into Fields 2 & 3. For e.g. I/94/032 may be an invoice number representing sale of items to a specific customer.

All document are coded as under

INV	-	Invoices
CM	-	Cash Memos
RPT	-	Receipts – Cash
CSH	-	Cash payment Voucher
RPQ	-	Receipts – Cheque
CQP	-	Cheque Payment Voucher
DN	-	Debit Note
CN	-	Credit Note
JV	-	Journal Voucher etc.

Details in brief regarding the transaction are entered into the 'narrations' Field.

While a debit mount is entered as such, a credit amount is prefixed by a minus sign.

Expenses and income are sometimes analyzed department wise or cost centre wise. A cost centre is a department or section which directly contributes to the cost of production of an item. Accounts department, Personnel department etc. are generally non profit earning departments and expenses incurred by them do not directly affect the cost of production. These cost centres are coded and keyed in into the last field of the transaction record.

Data are entered into the transaction file either through a data entry program or directly from linked computerized systems like Payroll, Inventory, Sales or Purchase systems. We will first concentrate on a data entry procedure done in line.

We had earlier mentioned that for every transaction there will a debit and credit entry. That is for every transaction two heads of accounts are affected. The 'Reverse Account Number; shown in the screen layout is to generate the second transaction.

When after data entry, you respond with a "Y" to the prompt "Confirm Recording", two records will be generated, the first one charged to the account number and the second one charged to the reverse account. The amount in the first record if a debit, in the second record, it will be created as a credit (with a minus sign prefixed) or vice versa. These two records are appended to the transaction file which is generally maintained in Account Number, date of transaction order. Apart from this the accounts Master File would be updated to reflect the latest current balances and todate debits and credits in respect of the two account numbers affected in the transaction. This result in some amount of duplication of data which can be eliminated by using certain short cuts which will be described later. However, with storage space becoming cheaper with technological advances in the field of computers, this draw back can easily be ignored. Needles to mention, data entered should be well validated and visually verified before recording and updating.

Direct Data Entry through linked systems

From Sales System:

When an invoice is printed, it represents a sales activity. The customer to whom the invoice is being sent in a debtor to the organization, if the sale is done on credit basis. Therefore the grow value of the invoice is debited to the 'customer's personal account' and credited to 'Sales account'. These transactions automatically generated and appended to the accounts transaction file and the master updated.

Similarly when a cash Sale is made to a customer, a cash bill is prepared. The amount of the bill has been received in cash and therefore debited to' 'Cash account' and credited to 'Sales Account' and suitable transactions are generated and appended to the transaction file and the corresponding master record updated.

From Payroll System:

Salaries and allowances are calculated in this system and when department wise earnings summary is being prepared, the total salaries and allowances of the employees in the departments are computed and transaction records created by debiting such amount to the head 'Salaries and allowances' and credited to 'Cash account' when salary payments are made in cash. If payments are being made by cheque, the amount will be credited to the 'Bank account'. Accounts master file is then updated as appropriate.

From inventory control System:

Whenever an issue is made form the stores, the value of items issued is credited to stock account and debited to the corresponding account pertaining to the usage of the items. For examples if raw materials are issued from stores to the production department, the value of issues is debited to 'Cost of raw materials' Master records are also updated automatically.

From Purchase System:

When purchases are made from suppliers on credit, the supplier sends an invoice showing value of items supplied. The value of goods thus received is debited to Stock account and credited to the suppliers account (Creditor). If purchases are made against cash, the value of items purchased is credited to 'Cash account' and debited to 'Stock account' and master records are automatically updated.

NOTES

Similarly when debit notes or credit notes are prepared consequent to purchase returns or issue / sales returns, the corresponding transactions are generated by the system and appended to the transaction file while updating the accounts master file.

Subsidiary Book of Accounts

In the manual accounting system, the main journal of accounts is subdivided into many subsidiary journals like

1. Cash Book
2. Bank Book
3. Sales Journal/Register
4. Purchase Journal/register
5. Debit note Journal or Goods Returns Outward Book.
6. Credit note Journal or Goods Returns Inward Book.

Etc.

These subsidiary books of accounts can easily be prepared in a Computerized accounting system periodically. Cash book is prepared on a day to day basis for reconciliation of the cash balances, to find out whether there has been under/over payments or whether a transaction has been omitted to be entered into the transaction file.

Let us now see how cash reconciliation is done. The input to this program is the account transaction file. The documents codes CSH and RPT representing Cash payments and receipts are only selected for this purpose. The opening balance of cash at the beginning of the day is keyed in as a parameter or retrieved from a file, specially kept to store daily cash opening and closing balances.

We thus see that all cash transactions made during a day are listed in this Cash book together with the opening balance of cash at the beginning of the day. When the last transaction is printed, the total cash receipts for the day and total cash payments are printed and the closing cash balance is computed and given as the last entry of the cash book.

If this cash closing balance does not tally with the cash in the cash box, an error has occurred in either the receipts or payments or in data entry which must be investigated and the errors corrected. The cash closing balance will be the opening cash balance for tomorrow.

Earlier we talked about duplication of data in the account transaction files when on line data entry of transactions are made. Cash Receipts are debited to Cash account while cash payments are credited. To do this for every transaction would be unnecessary duplication of data in the Cash accounts. To avoid this, while on line data entry is made, the cash account related transactions are omitted and when the cash book is printed and reconciled at the end of the day, the total receipts of cash can be debited to the Cash account while crediting cash payments to Cash account. This practice is generally not recommended.

Similar to Cash book preparation, Bank books and other subsidiary books of accounts can also be prepared on the computer periodically usually at the end of the month. Observe that the opening balances of each account head is provided in the Accounts

Master for each month of the financial year. This figure together with the related transactions in respect of Sales a/c, Purchase a/c etc. are used to prepare the relevant journals, at the end giving the total debits and credits of all transactions and the closing balance of the month computed in respect of the account.

These books can also be prepared either quarterly, half yearly or annually as desired.

In an online accounting system, the transactions are entered as and when they take place and this provide an upto date record of the financial position of the organization. Let us now see what other reports are prepared form the data thus stored to make meaningful analysis and consequent decision making or strategical finance planning.

Ledger

This the main book of accounts and is the final book of accounting entries. All transactions are posted into the ledger from either the source documents or from the journal book in such a manner that every account head and transactions pertaining to the account appears in the ledger separately and independent of each other, together with their debit and credit effects to provide summarized information of all heads of accounts in respect of all associated transactions.

In a computerized accounting system, the ledger is prepared monthly when all the transactions pertaining to the month are data entered into the accounts transaction file. Together with the Accounts master file, the ledger is printed.

A ledger is printed account headwise with all the transactions listed datewise with the document reference numbers, particulars of transaction and a debit or credit amount as appropriate. The ledger account details for the 'month are preceded by the opening balance figure for the respective account and at the end of all transactions listed, the closing balance figure is computed by adding up all debit amounts and credit amounts separately and the difference calculated by subtracting credit total from the debit total and if this total is positive the net balance is printed under the debit column and if other wise the balance printed under the credit column. The closing balance for the month is entered into the Accounts Master record as the opening balance for the succeeding month.

At the end of the ledger, total of all debit and credit transactions excluding the closing balance figures are printed in respect of all account heads put together.

Obviously these two figures would be the same since in a double entry book keeping system, for every debit there is a corresponding credit. If it is not, there is some mistake somewhere and it has to be localised and corrected and the ledger reprinted.

This ledger can be printed selectively, i.e. either monthly, quarterly or annually and such selections are provided to the program through input parameters at the time of running the program. For e.g. you can give the 'from date' and 'to date' and transactions will be picked out to fall between the dates supplied and the appropriate opening balance extracted from the Accounts Master. You can also print the ledger for selected account heads too.

In a manual accounting system, you will hear about the balancing of the Ledger accounts and this is achieved automatically in a computerized accounting system, the method of which is explained above. The user is transparent to such intricacies in a computerized system.

NOTES

Understanding Ledger Accounts

Personal Accounts

Every personal account showing debit balance (i.e. excess of debit side over credit side) will reveal the amount by which the debit side is more than the credit side. Debit side. Debit balance is recoverable from the person whose account shows a debit balance. A debit balance to a personal account is an asset and therefore the more debit balance to a personal account is an asset and therefore the more debit balances to personal accounts, more the assets are in the form of outstanding recoverables.

Similarly a personal account showing credit balance means such balances are payable to the person whose account shows a credit balance. So credit balances to personal accounts are a liability. More the credit balances to personal accounts, more the amounts payable to others.

Property or Real Accounts

All real accounts show a debit balance in the ledger except Sales and Returns outward accounts which show credit balances as they represent goods. A debit balance to every other real account shows the value of the properties in possession of an organisation on any day. It represents the wealth and financial position of the company in the form of properties owned and therefore more the debit balances in such accounts, the more wealthy the organization is.

Nominal or Fictitious Accounts

Nominal accounts which represent non-recoverable expenses will have debit balances and is a loss of the company. More such debit balances, more are the losses of the organization.

Nominal accounts with credit balances represent gains to the organization and it is quite welcome to the organization. More the nominal accounts with credit balances, more incomes and gains to the organization.

The duty of analysing a ledger is the function of the accounts department and computer users need not normally worry about such things.

Trial Balance

A ledger for a month when printed will run into many pages and it would be very difficult for anyone to go through them in detail. You can imagine what would be the volume when a ledger is printed which could be in the order of 5000 to 10,000 transactions in a medium sized organized in a month. To make this ledger more readable and comprehensible often the summary of all transactions together with the opening balances is computed and an account headwise summary report of the closing balances in respect of all account heads are printed. This is known as Trial Balance in a computerized system.

Every account head preparing in a ledger is listed in this trial balance. At the end the total of debits and credits in respect of all accounts are also printed. As mentioned before these totals should be the same if we have meticulously entered all transactions into the computer. If they do not tally, the reasons should be investigated in depth.

In a manual accounting system, the trial balance has a different definition. It is a statement of balances of all the ledger accounts, extracted from the ledger at the end

of a specific period to determine whether the grand total of debits and credits tally and thus (find out that all transactions have been properly journalised and) posted into the ledger. In contrast, the computer ledger tallies or not. Moreover, in the manual system, Cash and Bank expenses are only posted as debits and the corresponding credits are posted as a total extracted from the cash book or the Bank Book at a later date, may be at the end of the month when the subsidiary books are individually reconciled

A computerised accounting system is thus less prone to errors in posting and accuracy of accounts can be ascertained, if necessary even on a day to day basis. And the ledger at any point of time is exhaustively maintained and no short cuts are ever to be employed to save posting efforts in contrast to the manual system. When the trial balance tallies, it only means arithmetic accuracy in respect of the double entry book keeping system. It is not a proof of accounting accuracy. Many mistakes could still be there through trial balance be tailed arithmetically. Let us look at them.

Errors of Principles

Where accounts are wrongly debited or credited due to improper application of the golden rules of accounting, we call them an error of principles. For example a Personal account rule is applied to a transaction where it whould have been considered as a Property account. Suppose Rs 500 were paid to Ms. ABC Enterprises for purchase of an electric iron. We had in the transaction debited Ms. ABC enterprises with Rs 500 and credited a similar amount to Cash account. Actually it should have been debited to a Property account 'Plant & Machinery'. The trial balance would tally with this error, but the accounts is still incorrect. Is it not so? How do we correct such a mistake.

Rectification of the entry referred above can be done by creating a Journal Voucher debiting the said amount to 'Machinery account' and crediting it to 'Ms. ABC Enterprises' account which was wrongly debited earlier. This journal voucher should be duly authorised by an officer of the accounts department. Given below is the appropriated journal voucher specimen.

This journal voucher is now keyed in into the accounts transaction file and the master records would be updated accordingly to rectify this error of principle.

Error of Commission

This is an error where principles of accounts are not violated, but there has been a transcription error. For example instead of debiting and crediting the affected account heads as above with Rs. 500 only Rs. 50 were debited and credited. In an online accounting system, this type of error is quite common, that we key in the amount only once and the other corresponding account is automatically created by the computer and updated as appropriate by referring to the Reverse Account (Refer the screen layout of transaction data entry).

Now both debit and credit side has a shortfall of Rs 450 and the trail balance will tally.

To rectify this error, create a journal voucher debiting the 'Machinery account' by Rs 450 and crediting 'Cash account' by a similar amount with a suitable narration in the Particular column, duly authorised by the Accounts officer. This voucher is then data entered into the transaction file and master updated accordingly.

NOTES

Errors of Compensation

When an error on one side of an account gets compensated by an error on the other side, they are called 'Compensatory errors' As in the previous example, the shortfall of Rs 450 can be considered as this type of error since the short debit of Rs. 450 compensates short credit of Rs. 450.

Another example of this type of error is when goods worth Rs. 5000 was bough from M/s. Varkey's supermarket and by mistake, this was treated as a sales transaction and the following wrong entries were made.

1. Debited Rs. 5000 to M/s. Varkey's supermarket account.
2. Credited Rs. 5000 to Sales Account

Though the trial balance would tally, the entries are not correct, one mistake covers the other mistake. Isn't it? To rectify this error, we have to completely reverse the entries as follows.

1. Credit M/s. Varkey's super market A/c. With Rs. 5000

   Debit Sales a/c with Rs 5000
2. Debit Stock a/c with Rs 5000

   Credit M/s. Varkey's supermarket with Rs. 5000

The journal voucher should be prepared with the above entries, duly authorised and data entered and master updated.

This sort of an error could also happen during data entry by keying in the wrong account code. A transaction a/c number DB004 was wrongly entered as CR004. To correct this type of errors too you should reverse the entries as explained above.

Suitable narrations should be given in the particulars column to explain the transactions as incident to error corrections.

Errors of Omission

When both of the debit and credit aspects in respect of a transaction is omitted while data entry or when a whole document is omitted to be data entered, an error of omission occurs. In such cases, do what should have been done. Enter the data omitted prepare the trial balance once over again.

Validation procedures of Hash total checking or Control total checking could eliminate such errors of omission.

Suspense Account

If the trial balance does not tally due to a small difference and finding out the error is going to delay the preparation of the final accounts, in such cases, in order to avoid such delays, the amount of the difference between the debit and credit sides is temporarily placed in a Suspense a/c and the trial balance tallied. For example, if the debit side total of the trial balance is Rs 20,000 and that of the credit side is Rs 19,500 the difference of Rs 500 is placed into the credit side of the Suspense a/c and journalised which is data entered and the trial balance would then tally.

After the final accounts are prepared, the error will be localised and suitable reverse entries are passed to eliminate any balances in the suspense a/c.

There is another occasion when you can make use of the Suspense a/c. Say for example, you have received from a customer a cheque for Rs. 5000 without any explanations as to why this amount is being paid. While you would debit this amount to Bank a/c you do not know where this amount is to be credited since you do not know the customer number or his name (Possibly he had forgotten to attach a covering letter along with the payment). In this case, you credit the amount to suspense account. Then at a later date, the position become clear, the suspense a/c is debited with the amount and credited to the right account and appropriately date entered.

Final Accounts

Final Accounts consist of 3 parts.

1. Trading Account
2. Profit & Loss Account
3. Balance Sheet

A tallied trial balance is the base for preparing the final accounts.

Trading Account

This is prepared to find out the result of direct trading or manufacturing and trading activity in the form of gross profit or gross loss, that is the difference between the value of sales and cost of purchases or cost of manufacture without taking into consideration the indirect costs or expenses. It is prepared in the form of a ledger account, by debiting it with such of the expenses or debit balances from the trial balance, which makes up the total cost of purchases and cost of manufacture. The account is then credited with the sales balances after deducting there from purchase returns if any.

In a trading account, the first entry on the debit side is normally the opening stock if any. The next entry will be the purchases after deducting purchase returns if any. Thereafter, expenses such as clearing and forwarding charges, packing charges, salaries, handling charges and such other expenses incurred on purchase and which are directly attributable to purchase or direct cost of manufacture are entered on the debit side of the trading account to arrive at the direct cost of goods sold.

In manufacturing organizations, the same account is called Trading and Manufacturing account and in addition to the accounts already shown above, other expenses directly incurred on account of manufacturing and specifically attributable to cost of manufacture, such as factory expenses, lighting and electricity charges of the factory, power and fuel charges, salary of factory staff and labour and similar other expenses on account of production or manufacturing activity are debited to this account. Credit side of the trading account will usually have the Sales account balances as the first entry after deducting therefrom sales returns if any. The last entry on the credit side would be the closing stock value determined on physical verification of stock and valuation at the end of the financial year. The excel of credit side total of this account over the debit side totals reveal the Gross profit and if the debit totals are in excel of the credit totals, it reveals the Gross loss, which are carried forward or brought down to the Profit and Loss account. Following is the format of a trading Account.

To prepare this report in a computerized system, it is just a matter of extraction of the relevant accounts details from the transaction and accounts master files. Or

NOTES

alternatively, this can be taken from a temporary file created on preparing the trial balance consisting of the relevant accounts heads and their balances. This file should have the account number, particulars of the account, debit or credit balances in respect of all account pertaining in the trading activity.

The trick in organizing this reports is by allotting such account code number from which you can easily pick out the heads participating in the trading function. For example, we have an account head titled "Stock a/c" in the Accounting system. If we were to subclassify this head into say

1. Raw material Stocks
2. Finished Goods Stocks
3. Machinery Stock – Factory
4. Machinery Spaces – Non Factory related
5. General Stock

etc. it would be fairly easy to find out Purchase of stock required to appear in the trading account. Similarly notional accounts should also be classified as the ones that are affecting the trading account and others.

In most of the organizations where the accounting system is computerized the practice is the to create the containing all the account heads affecting the trading account. Using this file and the accounts master file, appropriate records are selected and the trading account printed. Value of closing stock is input through the keyboard after physical verification and valuation.

Profit and Loss Account

The purpose of preparing the account is to find out the net profit made by the organization, during a particular period. The gross profit or gross loss is not the actual indicator of the net results of the business operation, because while arriving at the gross profit or loss, the indirect expenses are not taken into consideration.

While preparing the P & L account, the first entry on the credit side is the gross profit brought down from the trading a/c. If the trading a/c shows a loss, this would appear on the debit side of the P & L account. Thereafter all expenses of a nominal or fictitious nature are entered into the debit side of the P & L account.

Incomes of a nominal nature are entered into the credit side of the account. This account is then balanced and if the credit side is more than the debit side, the excess of credit side over the debit side will be the net profit whereas the excess of debit side over credit side indicate the net loss.

In a computerized accounting system, P & L accounts are coded separately to easily identify them. In our example, of heads of accounts, P & L heads are coded starting with 'PL' for example, 'PL01' shows "salaries & Allowances" etc. These account heads are picked up from the Accounts master and together with the results of the trading accounts, P & L account is printed. Before the certain notional expenditures are computed like depreciation of fixed assets etc. and included in the input data of P & L programs.

Data to prepare this report is extracted from the Accounts Master using the P & L account heads file and the heads are groups into Income and Expenditure and printed

as above without any rounding off and finally net profit / loss is computed and printed in the end.

Balance Sheet

The entries in the balance sheet are classified as follows

NOTES

Assets

They are the value of properties in possession of the company including all the receivables and recoverablles from debtors and other sources. They are further classified:

1. Fixed Assets.
2. Investment in Securities
3. Current Assets
4. Loans and Advances
5. Stocks, Cash and Bank Balances etc.

Liabilities

They are the source or causes of all liabilities which the organization owes to or is indebted to pay to others including the liabilities to the owners of the organization in the form of capital, net profit, reserves, accumulated profits in any form, provisions for various purposes and funds created out of profits.

While preparing a balance sheet, assets can be listed on one side and the liabilities on the other side or liabilities can be listed first followed by the assets.

At the end, both the assets and the liabilities are totalled both side totals will be the same if everything is done in order. Those account balances taken in the P & L or trading a/c must not be included while preparing the balance sheet. Only the closing stock shown on the credit side of the trading a/c should be taken into the balance sheet, being assets still in possession of the organization. The net profit or loss from the P & L a/c is taken into the liabilities side for giving effect to the Capital a/c. The Capital of an organization is the asset in excel of all liabilities towards the owners as well as others.

When a computerized Accounting System is designed, the staff in the accounts department should be consulted to get the full idea about their requirements and the nature of the heads of accounts so that preparation of the final accounts is made easier from the Accounts Master maintained on an up to date basis. Using a properly designed system, it would be possible to take the P & L account and Balance sheet of an organization at any point of time. Usually these are prepared only at the end of a financial year.

Apart from the above reports, many other useful statements are also prepared as byproduct of a good computerized accounting system. Let us look at them briefly.

Bank Reconciliation Statements

Any organization who buys or sells items on credit usually make use of banking facilities and for this purpose accounts are opened with one or more banks. Opening an accounts and depositing money with the bank is like transferring one of the cash

NOTES

boxes into the safe custody of a bank referred to as a bank account, which money is rightfully owned by the depositor. So, the organization normally keeps only a small amount of money in the cash a/c, while the lion's share is kept in a bank a/c. They use money from both these accounts to meet their expenses.

When money is received by means of a cheque, they are deposited in the bank and is debited to the Bank a/c of the organization's books of accounts. Similarly when cheque payments are made by the company, they are credited to the bank a/c. Often money is transferred from Cash a/c to Bank a/c when cash balances accumulate and vice versa when cash balances deplete. Such transactions which result in neither an income nor an expense are called 'Contra Entries' in accounting parlance. However the transactions are journalised and data entered in a computerized accounting system.

When an account is opened with a bank, a debtor – creditor relation comes into existence between the organisation and the bank. Amounts are deposited into the bank a/c as well as withdrawn from time to time. Payments are made by a cheque from the bank account and often receipts are got by cheques which are deposited in the bank.

Records of these transactions are entered into the Bank a/c of the company. Simultaneously the bank also maintains a record of such transactions. But often the records maintained by the bank may not tally with Bank a/c maintained by the organization due to various reasons. Therefore it is necessary to keep a check on the bank transactions maintained by the company with the statements of transactions provided by the bank normally once in a month. This is done by a Bank Reconciliation statement.

We will now look into the possible reasons why the bank statement and the bank a/c maintained by the company differ.

1. The bank might have credited to your account by interest accrued on your balances or debited interest on overdraft amount drawn.
2. A cheque issued by the organization might not have been presented to the bank.
3. A cheque received from a party, though deposited in the bank might not have been cleared as yet and credited to your account.

At the end of the month, when the statement of account is received from the bank which would include details such as cheque number, amount deposited or withdrawn, date of transaction etc., these details are keyed into a file and is compared with the records in the transaction file (only RPQ and CPQ document codes are considered for this purpose) and varying records can be identified and printed out.

The net balances of such accounts when added to your bank balances at the end of the month as per your records would give the balances to tally with the balance shown in the statement of account provided to you by the Bank. This process is known as Bank Reconciliation. The fields compared are cheque number and amount which are available in the accounts transaction file as well as the statement of account received from the bank. If the company has accounts with more than one bank, the bank reconciliation statement is prepared in respect of each bank.

Input parameters are opening bank balance, Closing bank balances as well as the dates between which the reconciliation is to be prepared. The simple rule observed in

preparing a bank reconciliations is "Do what the bank has done". If the bank has not deducted an amount which we have deducted, then we add back the amount to be in line with the Bank records. We correct ourselves by reversing our act, to be in line with the act of bank.

Ratio Analysis

The statements described above are prepared purely based on accounting principles and fall short of management requirements to take proper decisions. Ratios help to express performances, results and financial information either in terms of percentages or in terms of relations between different sets of values to enable management to understand how well the financial resources are being utilized and to determine what remedial measures are to be taken for improved performances.

Let us look at some of such ratios.

Current Ratio

This is also known as working capital ratio or solvency ratio. This reveals the relation between current assets and liabilities. Current assets are convertible into cash within the current period (normally for the period of one financial year) to meet the current liabilities arising during the same period.

Current Ratio = Current Assets / Current Liabilities

The more the current assets are in relation to the current liabilities, the better the financial ability of the organization to meet its financial liabilities. A current ratio of a figure less than 1 is quite dangerous.

Current assets include cash in hand and in the bank; stock of raw materials, finished goods, debtors from whom money is recoverable, short term investments etc. while current liabilities include bank overdrafts, creditors to whom money is payable, bills and other short term liabilities. If these heads are so coded to identify them easily, the computer using the Accounts master file as Input can quickly work out this ratio and displayed whenever needed.

Acid Text Ratio

This is also known as quick ratio or liquid ratio, which is an improvement over the current ratio. This involves the testing of the liquidity of the current assets into cash in the shortest possible time without difficulties to meet the immediate current liabilities.

Acid Test Ratio = Quick Assets / Quick Liabilities

Stock which is strictly speaking saleable into cash generally not considered as quick assets. Similarly a bank overdraft is not a quick liability since it is a long term facility offered to you by a bank.

A 1:1 quick ratio between quick assets and liabilities can be considered as a safe ratio. With proper coding to identify quick assets and liabilities, this ratio can be worked out by the computer using accounts master file as input to the program.

Stock turnover Ratio:

This is used to find out how fast the stocks are utilized or disposed. i.e. the rate at

NOTES

which stocks are sold and thus converted to money. The faster such conversion, the better will be the business performance.

Stock Turnover ratio = Cost of goods sold / Average inventory at cost

Average inventory at cost = Opening stock + Closing Stock / 2

When the Sales system and the accounting system are integrated this ratio can easily be calculated on a computer. Cost of goods sold is not the sale proceeds, since it includes a profit element in it. It is the actual cost of the goods, in other words the cost of production of the goods.

Debtor turnover ratio:

This ratio is prepared to find out as to how much of the total sale is held by debtors without having paid for them. It indicates the number of days credit facility extended to or availed by the customers. Ratio reveals the number of days the sales remain unpaid. It enables management to have control on the debtors and to make efforts in recovering outstanding from debtors in time.

Debtor Turnover Ratio = Debtors balances / Sales per day

Sales per day = Net Sales / Number of working days (360 approximately)

Gross Profit Percentage

Gross Profit Percentage = Gross Profit * 100 / Net Sales

This reveals margin of profit on sales. Increase in gross profit do not necessarily indicate good performance since increase in selling price with out a corresponding decrease in manufacturing cost is unfoverable. Decrease in gross profit can also be due to uneconomic purchase of raw materials, improper valuation of stock balances etc.

Net Profit Ratio

This is a more realistic indicator of the success in business since this includes all direct and indirect cost of the trading or manufacturing activity and indicates the actual returns on investment in a business.

Net Profit Ratio = Net Profit * 100 / Net Sales

Return on Capital

Return on Capital employed = Net Profit * 100 / Gross Capital Employed

Gross capital is the share capital received form the shareholders as well as borrowed capital by way of long term loans, debentures and reserves of a capital and revenue nature. Capital is represented by the assets in existence with the organization and employed for the purpose of generating profits. That portion of the assets of a fictitious nature like goodwill, investments outside the organization etc. are excluded while computing the gross capital employed.

Net Worth

This indicates the relation of capital (Share Capital, Reserves and Surplus, accumulated profits etc.) with the fixed assets of the organization. Value of fixed assets must be reasonably low in comparison to the capital as otherwise it shows non productive

investments. Higher the value of current assets in comparison to fixed assets, better the financial strength of the organization. More the owners capital in relation to outside capital by way of loans, debentures etc. better the financial structure and represent sound policies and profitable activities of a business with least dependence on creditors for finances.

NOTES

The above ratios can be excellent by product of a good computerized accounting System. The trick in preparing these ratios lie in a good coding system from which appropriate figures can easily be extracted for computations. A very good interaction and involvement of the accounting personnel while designing the system can pay rich dividends in the form of good management information emanating from the Accounting system.

Cash Flow Analysis

This shows the inflow and outflow of cash during the individual months of this year in each major and minor heads of accounts. The relevant figures can be extracted from the Accounts Master file and transaction files. Let us look at the usual format of a Cash flow statement. The above report can be prepared monthly, quarterly, half yearly or annually as desired.

All transactions relating to the bank and Cash accounts are only considered for this purpose. A cash flow statement for a given period is usually a good indicator to the possible cash flow of the next ensuing period and therefore similar statements are also prepared by managers to forecast their cash requirements for a period yet to come. This is normally done using an Electronic spread sheet and doing 'What-if' analysing with varying parameters of cash inflow and outflow.

Apart from what we have discussed so far, a computerized accounting system should have the following Sub systems.

1. Accounts Receivable
2. Accounts Payable
3. Depreciation of Fixed assets calculations
4. Preparation of various schedules in support of the Balance sheet.
5. Budgetory Control System etc.

MARKETING SYSTEM

A market system is any systematic process enabling many market players to bid and ask: helping bidders and sellers interact and make deals. It is not just the price mechanism but the entire system of regulation, qualification, credentials, reputations and clearing that surrounds that mechanism and makes it operate in a social context.

Because a market system relies on the assumption that players are constantly involved and unequally enabled, a market system is distinguished specifically from a voting system where candidates seek the support of voters on a less regular basis. However, the interactions between market and voting systems are an important aspect of political economy, and some argue they are hard to differentiate, e.g. systems like cumulative voting and runoff voting involve a degree of market-like bargaining and tradeoff, rather than simple statements of choice.

NOTES

Types

In economics, market forms are studied. These look at the impacts of a particular form on larger markets, rather than technical characteristics of how bidders and sellers interact.

Heavy reliance on many interacting market systems and forms is a feature of capitalism, and advocates of socialism often criticize market features. This article does not discuss the political impact of any particular system nor applications of a particular mechanism to any particular problem in real life.

For more on specific types of real-life markets, see commodity markets, insurance markets, bond markets, energy markets, flea markets, debt markets, stock markets, online auctions, real estate market, each of which is explained in its own article with features of its application, referring to market systems as such if needed.

Protocols

The market itself provides a medium of exchange for the contracts and coupons and cash to seek prices relative to each other, and for those to be publicized. This publication of current prices is a key feature of market systems, and is often relevant far beyond the current groups of buyers and sellers, affecting others' supply and demand decisions, e.g. whether to produce more of a commodity whose price is now falling. Market systems are more abstract than their application to any one use, and typically a 'system' describes a protocol of offering or requesting things for sale. Well-known market systems that are used in many applications include:

- auctions - the most common, including:

  Dutch auctions

  reverse auctions

  silent auctions
- rationing (including the command economy of some states)
- regulated market (including most real-life examples as above)
- black market (the term 'black' indicating lack of regulation)

The term 'laissez-faire' ("let alone") is sometimes used to describe some specific compromise between regulation and black market, resulting in the political struggle to define or exploit "free markets". This is not an easy matter to separate from other debates about the nature of capitalism.

There is no such thing as a "free" market other than in the sense of a black market, and most free-market advocates favor at least some form of regulated market, e.g. to prevent outright fraud, theft, and retain some degree of credibility with the larger public. This political debate is out of the scope of this article, other than to note that the "free" market is usually a "less regulated" market, but not qualitatively different from other regulated markets, in any society with laws, and that what opponents of "free markets" usually seek is some kind of moral purchasing rather than pure rationing.

As this debate suggests, key debates over market systems relate to their accessibility, safety, fairness, and ability to guarantee clearance and closure of all transactions in a reasonable period of time.

Importance of trust

The degree of trust in a political or economic authority (such as a bank or central bank) is often critical in determining the success of a market. A market system depends inherently on a stable money system to ensure that units of account and standards of deferred payment are uniform across all players - and to ensure that the balance of contracts due within that market system are accepted as a store of value, i.e. as "collateral" of the holder of the contract, which justifies "credit" from a lender of cash.

Banks, themselves, are often described in terms of markets; as "transducers of trust" between lenders (who deposit money) and borrowers (who take it out again). Trust in the bank to manage this process makes more economic activity possible. However, critics say, this trust is also quite easy to abuse, and has many times proven difficult to limit or control (see business cycle), resulting in 'runs on banks' and other such 'crises of trust' in 'the system'.

However, market systems are usually flexible enough to be refined and have its detailed rules adjusted so as to regain the trust of participants relatively quickly - most market systems tend to degrade gracefully, with a few exceptions, e.g. hyperinflation, South Sea bubble, tulip boom, dotcom boom, depression, that are very damaging, but nonetheless relatively infrequent.

FOREIGN TRADE

Computer is very much used in the various calculations of foreign trade. It is more or less on the lines of the Inventory System, discussed next.

INVENTORY INFORMATION SYSTEMS

An inventory information approval system, or IIAS, is a point-of-sale technology used by retailers that accept FSA debit cards, which are issued for use with medical flexible spending accounts (FSAs), health reimbursement accounts (HRAs), and some health savings accounts (HSAs) in the United States.

By the end of 2007, all grocery stores, discount stores, and online pharmacies that accept FSA debit cards must have an IIAS; by the end of 2008, most chain pharmacies must have an IIAS as well.

The first IIAS was developed by the online retailer drugstore.com for its "FSA store" in 2005; it was first introduced to brick-and-mortar retailing by Walgreens in 2006. Wal-Mart became the first discounter with an IIAS in late 2006.

How IIAS works

IIAS is similar to the system used by grocery stores ever since they introduced the first barcode scanners in the 1970s to separate items eligible for purchase under the Food Stamp Program from those that are not eligible. Every item in the grocery store's database is flagged "yes" or "no" for food-stamp eligibility; the scanner automatically keeps a separate total for food-stamp items. In the beginning, the cashier pressed a special "food-stamp total" key, and the customer presented paper food

NOTES

stamps; today, the customer swipes an Electronic Benefit Transfer (EBT) card and selects the "food stamp" account, and the register charges only the food-stamp total to the EBT card. The remaining balance must be paid for by other means.

IIAS works in much the same way, but with medical FSAs, HRAs, or HSAs instead of food stamps: (Usually, the term "FSA" is used to cover all of them; HRAs, HSAs, and non-medical FSAs are relatively rare, and HSAs can also have regular debit cards though many of them have FSA debit cards instead.)

Every item in the store's scanner database is flagged "yes" or "no" for FSA eligibility. (This flag is obviously separate from the one for food stamps, if there is one.)

Prescription drugs are usually not in the main scanner database (though they may be made scannable by tying the pharmacy system into the scanners), but they are almost always FSA-eligible; therefore, the pharmacy department is often categorically flagged as FSA-eligible, the only department to be so treated. (In contrast, multiple departments of most grocery stores are categorically flagged as food-stamp eligible, including the meat, produce, and dry-grocery departments.)

At checkout, the scanner (for brick-and-mortar retailers) or shopping cart (for online retailers) keeps a separate total for those items that are "FSA-eligible".

If an FSA debit card is presented for payment, the scanner or shopping cart will charge the card, but for no more than the "FSA-eligible" total.

If there are other items in the order (or if the FSA debit card didn't pay for all eligible items), the scanner or shopping cart then demands another form of payment, such as cash, check, credit card or debit card, to pay for the remaining items.

IIAS does have one additional requirement that is not normally found with food stamps, though the U.S. Department of Agriculture can audit retailers directly for similar purposes: Beginning January 1, 2007, the merchant must make a record of each transaction available to the employer, or more commonly, to the employer's FSA or HRA provider. This can be done contemporaneously with the transaction, or it may be provided later if the Internal Revenue Service ever audits the employer.

Please note that the terminology used by the IRS in its descriptions of IIAS may seem obtuse; this is not only because it's the IRS, but also because IIAS was first developed by an online retailer (drugstore.com) and only later adapted to brick-and-mortar retailing. For example, IIAS is described by the IRS as an "inventory control" system tied to SKUs; but it's generally easier to understand as it was implemented by Walgreens and Wal-Mart, i.e., as a point-of-sale system tied to UPC codes.

IRS requirements to use IIAS

Though IIAS was first used in 2005, it was not officially approved by the Internal Revenue Service until July 2006, in IRS Notice 2006-69. At the same time, the IRS decided to crack down on FSA/HRA providers that were not following prior IRS guidance on FSA debit cards. As part of this, the IRS decided that grocery and discount stores would not be allowed to accept FSA debit cards unless they installed an IIAS; they decided it would be too easy to misuse the cards if they could be used at grocers and discounters for anything they sold, even if the grocer or discounter also had a pharmacy. However, they permitted stand-alone chain or independent pharmacies (known as "true pharmacies") to accept the card without an IIAS.

Grocers and discounters immediately challenged the IRS ruling, claiming that their pharmacies were being discriminated against, and that since most "true pharmacies" sold ineligible goods as well, the risk from them was just as great. Therefore, two changes were made by IRS Ruling 2007-02 in December 2006:

Grocers and discounters are allowed to keep accepting the cards until December 31, 2007; this was to give them sufficient time to install an IIAS.

"True pharmacies" are required to install an IIAS after December 31, 2008, unless at least 90% of the individual pharmacy's sales are of "FSA-eligible" items, i.e., prescription drugs or over-the-counter (OTC) items.

Most major pharmacy chains report that 60-65% of their sales come from the pharmacy; therefore, OTC would have to account for 25-30% of their total sales for them to qualify, which is unlikely--especially since each individual pharmacy must qualify separately. Therefore, only independent pharmacies are likely to qualify for the exemption.

Because of this ruling, by 2009 most grocers, discounters, and chain or Internet pharmacies in the U.S. must have an IIAS in place in order to accept FSA debit cards.

Importance of IIAS

In addition to the above IRS requirements, IIAS is important in promoting the use of tax-favored health accounts, especially FSAs (which are usually set up by employees), for these reasons:

While other IRS-approved "auto-adjudication" systems for electronic substantiation of FSA debit card charges are geared towards health plan expenses, such as copay matching or electronic transmittal of explanations of benefits, IIAS is the only one that is designed for use with over-the-counter drugs and similar items (OTC) as well as prescription drugs.

IIAS is the first system with 100% "auto-adjudication" of an entire class of FSA debit card charges that has been widely adopted by the FSA industry. A few FSA vendors had previously used proprietary systems which provided 100% auto-adjudication of prescription charges thru a pharmacy benefits manager, but they ran into numerous technical and educational issues and were not adaptable to OTC.

Some of the IRS rules on what OTC items are and aren't eligible for FSAs have proven rather arcane in practice; for example, condoms are OK since they prevent pregnancy, but K-Y Jelly isn't if it's used to lubricate them. IIAS effectively manages this problem by verifying eligibility of each OTC item at point-of-sale.

Both paper claims and manual substantiation of FSA debit card charges often required the submission of receipts with "full names" of OTC items; but many stores abbreviate item names in such a way that it is almost impossible to tell if the item is eligible or not. Also, most providers did not reimburse sales tax on paper claims with "mixed" FSA/non-FSA receipts because they could not "split" the tax line item without being versed in the sales tax laws of every state and locality in the U.S., a near impossibility. IIAS avoids this by having the retailer itself verify item eligibility and "split" the sales tax.

The process of demanding receipts or reimbursement for FSA debit card charges that are not "auto-adjudicated", known as "pay and chase" in the industry (a term recognized by the IRS in Notice 2007-02), proved particularly cumbersome for OTC

items due to the lack of "auto-adjudication" systems and the high potential for fraudulent or erroneous charges; IIAS eliminates this by providing an "auto-adjudication" system for OTC while preventing many fraudulent or erroneous charges at retailers.

NOTES

Since IIAS eliminates many of the roadblocks that previously existed for use of medical FSAs at retailers (especially for OTC items), it is hoped that it will lead to increased enrollment in medical FSAs.