
KARNATAKA STATE



OPEN UNIVERSITY

MBA PROGRAMME
II YEAR – III SEMESTER
SPECIALIZATION: IT



DATABASE MANAGEMENT SYSTEM

COURSE: MBA 116G/SC G3.1

BLOCKS: 1-4

DEPARTMENT OF STUDIES AND RESEARCH
IN MANAGEMENT

DEPARTMENT OF STUDIES AND RESEARCH IN MANAGEMENT

M.B.A III SEMESTER

COURSE - MBSC - 3.1 G

DATABASE MANAGEMENT SYSTEM

BLOCK - 1: INTRODUCTION TO DATABASE SYSTEMS, ENTITY RELATIONSHIP & DATA MODELS

UNIT - 1

INTRODUCTION TO DATABASE SYSTEM 01-17

UNIT - 2

FILE ORGANISATION 18-40

UNIT - 3

DATAMODELS 41-55

UNIT - 4

THE RELATIONAL DATABASE MODEL 56-76

BLOCK-2: ER MODEL AND SQL

UNIT - 5

ER MODEL 77-100

UNIT - 6

NORMALISATION OF DATABASE 101-116

UNIT - 7

INTRODUCTION TO STRUCTURED QUERY LANGUAGE (SQL) 117-140

UNIT - 8

DATA WAREHOUSES AND DATA MINING 141-162

BLOCK-3: DATABASE DESIGN AND CONTROL

UNIT - 9

DATABASE DESIGN 163-192

UNIT - 10

TRANSACTION MANAGEMENT AND CONCURRENENCY CONTROL 193-219

UNIT - 11

DISTRIBUTED DATABASE MANAGEMENT SYSTEM (DDBMS) 220-236

UNIT - 12

BUSINESS INTELLIGENCE AND DATA WAREHOUSES 237-254

BLOCK-4:DATABASE AND INTERNET

UNIT - 13

DATABASE CONNECTIVITY AND TECHNOLOGIES 255-279

UNIT - 14

DATABASE SECURITY AND AUTHORIZATION 280-290

UNIT - 15

DATABASE RECOVERY SYSTEM 291-300

UNIT - 16

DATABASE ADMINISTRATION AND SECURITY 301-332

CREDIT PAGE

Programme Name : MBA Year/Semester : 2nd Year, 3rd Semester Block No: 1 to 4

Course Name : Database Management System Credit : 04 Units No: 1 to 16

Course Design Expert Committee

Prof. Vidya Shankar Vice-Chancellor Karnataka State Open University Mukthagangothri, Mysore-06	Chairman	Prof. Kamble Ashok Dean (Academic) Karnataka State Open University Mukthagangothri, Mysore-06	Member
--	-----------------	---	---------------

Course Designer/Course Co-ordinator

Dr. Rajeshwari H. Assistant Professor, DOS & R in Management KSOU, Mukthagangothri, Mysore-06	BOS Chairman & Member	Dr. P. Savitha Assistant Professor, DOS & R in Management KSOU, Mukthagangothri, Mysore-06	Department Chairman & Member Convener
--	--	---	--

Editorial Committee

Dr. Rajeshwari H. Assistant Professor DOS & R in Management KSOU, Mukthagangothri, Mysore-06	BOS Chairman & Member	Prof. D. S. Guru Dept. of Computer Science UoM, Mysuru	External Subject Expert & Member
Dr. Rajeshwari H. Assistant Professor DOS & R in Management KSOU, Mukthagangothri, Mysore-06	Internal Subject Expert & Member	Dr. P. Savitha Assistant Professor, DOS & R in Management KSOU, Mukthagangothri, Mysore-06	Department Chairman & Member Convener

Course Writers

Dr. Ashok Rao
Former Head,
Network Project,
CEDT,IISc, Bangalore

Course Editor

Nandini H M
Assistant Professor
DoS in Information Technology
KSOU, Mysore.

Dr. Sudhamani M
Assistant Professor
DoS in Computer Science
UoM, Mysuru

**Unit 01-04
Unit 05-08**

**Unit 09-12
Unit 13-16**

Copy Right

Registrar

Karnataka State Open University
Mukthagangothri, Mysuru - 570006

Developed by the Department of Studies and Research in Management, KSOU, under the guidance of Dean (Academic), KSOU, Mysuru

Karnataka State Open University, January-2021

All rights reserved. No part of this work may be reproduced in any form, or any other means, without permission in writing from the Karnataka State Open University.

Further information on the Karnataka State Open University Programmes may obtained from the University's office at Mukthagangothri, Mysuru-570006

Printed and Published on behalf of Karnataka State Open University. Mysuru-570006 by
Registrar (Administration)-2021



Karnataka State Open University
Mukthagangothri, Mysore – 570 006.
Dept. of Studies and Research in Management

MBA IT Specialization
III Semester

Database Management System



Block 1

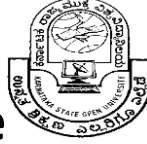
PREFACE

The amount of information available to us is literally exploding, and the value of data as an organizational asset is widely recognized. To get the most out of their large and complex datasets, users require tools that simplify the tasks of managing the data and extracting useful information in a timely fashion. Otherwise, data can become a liability, with the cost of acquiring it and managing it far exceeding the value derived from it.

The goal of this material is to present an introduction to database management systems, with an emphasis on how to design a database and use a DBMS effectively. Not surprisingly, many decisions about how to use a DBMS for a given application depend on what capabilities the DBMS supports efficiently. Therefore, to use a DBMS well, it is necessary to also understand how a DBMS works.

The whole material is organized into four modules each with four units. Each unit lists the objectives of the study along with the relevant questions, illustrations and suggested reading to better understand the concepts.

Wish you happy reading!!!



Karnataka State Open University

Mukthagangothri, Mysore – 570 006.

Dept. of Studies and Research in Management

MBA. IT Specialization

III Semester

DATABASE MANAGEMENT SYSTEM

BLOCK 1

UNIT NO.	TITLE	PAGE NUMBERS
UNIT 1	INTRODUCTION TO DATABASE SYSTEMS	1-16
UNIT 2	FILE ORGANIZATION	17-40
UNIT 3	DATA MODELS	41-55
UNIT 4	THE RELATIONAL DATABASE MODEL	56-76

BLOCK 1 INTRODUCTION

A Database Management System(DBMS) is a software that enables users to create and maintain a database. A database is a collection of related data and data are raw facts. Introductory aspects of DBMS and nuances of file organization is discussed in this module. Unit 1 introduces database systems. Unit 2 has basics of file organization and later two units focus on data models with emphasis on relational database models.

This block consists of 4 units and is organized as follows:

Unit 1-Introduction to Database Systems: Introduction , Basic Definitions and Concepts Traditional File System Versus Database Systems, DBMS Users , Database or DBMS Languages , Schemas, Subschema and Instances, Three Level Architecture of Database Systems (DBMS), Data Models, Types of Database Systems, Comparison between Client/Server and Distributed Database System

Unit 2-File Organization: Introduction, Basic Concepts of Files, File Organization Techniques, Indexing, Comparison of Different File Organizations, Comparison of Different File Organizations, Factors Affecting Choice of File Organization

Unit 3- Data Models : Data Modeling and Data Models, The Importance of Data Models, Data Model Basic Building Blocks, Business Rules, The Evolution of Data Models, Degrees of Data Abstraction

Unit 4- The Relational Database Model: Logical View of Data, Keys, Integrity Rules, Relational Algebra, The Data Dictionary and the System Catalog, Relationships within the Relational Database, Data Redundancy Revisited, Indexes, Codd's Relational Database Rules

UNIT-1: INTRODUCTION TO DATABASE SYSTEMS

STRUCTURE

- 1.0 Objectives
- 1.1 Introduction
- 1.2 Basic Definitions and Concepts
- 1.3 Traditional File System Versus Database Systems
- 1.4 DBMS Users
- 1.5 Database or DBMS Languages
- 1.6 Schemas, Subschema and Instances
- 1.7 Three Level Architecture of Database Systems (DBMS)
- 1.8 Data Independence
- 1.9 Data Models
- 1.10 Types of Database Systems
- 1.11 Comparison between Client/Server and Distributed Database System
- 1.12 Check Your Progress
- 1.13 Summary
- 1.14 Keywords
- 1.15 Questions for self-study
- 1.16 References

1.0 OBJECTIVES

After studying this unit, you will be able to:

- ✓ Describe a DBMS and how a DBMS manages data
- ✓ Compare a traditional file system with a database system
- ✓ Explain Schemas, Subschema and Instances
- ✓ Explain Three Level Architecture of Database Systems
- ✓ Explain a data model and different types of Database Systems,
- ✓ Compare Client/Server and Distributed Database System

1.1 INTRODUCTION

Organizations are increasingly dependent on data for decision making. Managing data is a challenge for organizations as more and more data is available. Managing and extracting information out of it at the right time is the need of the hour. In this digital era data is considered as gold.

A database management system(DBMS) is a software which helps individual/organizations to collect related data in an orderly fashion. A database is a collection of related data and raw facts are known as data. The DBMS software together with the database is called a database system.

Database and DBMS are found useful in managing business, government, schools, universities, banks etc.

1.2 BASIC DEFINITIONS AND CONCEPTS

This section describes some of the important terms used in databases and database systems

1.2.1 Data

Data is defined as raw facts which are processed to create information.

1.2.2 Information

Information is making sense out of the available data.

For example, let us consider the number 37 which is a piece of raw data but once the context is known such as average body temperature in Celsius, the raw data becomes information.

Information is the basis for making decision.

1.2.3 Metadata

Metadata is data on data.

A file name, type, last modified time, last accessed time, file access permission are some of the examples of metadata.

1.2.4 Data dictionary

Data dictionary stores metadata of the database. DBMS refers data dictionary whenever a database has to be modified. A data dictionary is generated for each database which has definitions of the data elements and their relationships which is nothing but the metadata of the database.

1.2.5 Database

A database is a collection of related data.

For example, a university database might contain information about entities such as students, faculty, courses, classrooms and relationship between entities such as students enrolled in courses, faculty teaching courses and the use of rooms for courses.

1.2.6 Components of a Database

A database consists of four components: data, relationship between various data elements, constraints which are nothing but restrictions placed on the data and schema which is the conceptual organization of the entire database as viewed by the database administrator.

1.2.7 Database Management System (DBMS)

It can be defined as a computerized record-keeping system that stores Information and allows the users to add, delete, modify, retrieve and update that information.

Components of DBMS

A DBMS has three main components

1. Data Definition Language (DDL) - It allows the users to define the database, specify the data types, data structures and the constraints on the data to be stored in the database.
2. Data Manipulation Language (DML) and Query Language - DML allows users to insert, update, delete and retrieve data from the database. SQL provides general query facility.
3. Software for Controlled Access of Database.

1.3 TRADITIONAL FILE SYSTEM VERSUS DATABASE SYSTEMS

In this section let us understand the disadvantages of traditional file system and the advantages of the database systems over traditional file system

A traditional file system has the following disadvantages

1. Data redundancy: several files might contain same data. For example in an organization employee name is present in both personal file and payroll file which results in duplicate or redundant data items.
2. Data inconsistency: data redundancy results in data inconsistency. Whenever some piece of data is present in multiple files and when data changes if it is not updated in

all files where it is present it would result in data inconsistency. For example an employee is promoted from junior engineer to assistant engineer and it is not updated in all the files where this information is stored then it would result in data inconsistency.

3. Lack of Data Integration: Many a times data items in different files have to be aggregated to derive required information which makes data integration a difficult task.
4. Program and data dependence: If data format changes, programs manipulating data have to change correspondingly. Similarly programs can access data only in a predefined format.
5. Limited data sharing: Obtaining data from several incompatible files is a difficult task.
6. Poor data control: Data in traditional file system is decentralized which results in poor data control
7. Problem of Security: It is very difficult to enforce security checks and access rights in a traditional file system.
8. Data Manipulation Capability is Inadequate: The data manipulation capability is very limited in traditional file systems since they do not provide strong relationships between data in different files.
9. Needs Excessive Programming: An excessive programming effort is needed to develop a new application program due to very high interdependence between program and data in a file system. Each new application requires that the developers start from the scratch by designing new file formats and descriptions and then write the file access logic for each new file.

The database systems provide the following advantages over the traditional file system:

1. Controlled redundancy: DBMS software helps in controlling duplicate/redundant data.
2. Data consistency: DBMS software takes the responsibility of updating common data item to ensure data consistency.
3. Program data independence: Database system allows to change data organization without affecting the application program that process the data.
4. Sharing of data: As data is stored in a central repository, it can be shared with all authorized users and all programs which needs to access this data.
5. Enforcement of standards: Standard formats can be enforced on data as it is centrally stored.

Storing data in standard format helps easy transfer from one system to another.

6. Improved data integrity and security: Centrally controlled data is both accurate and consistent also data can be secured easily.
7. Data access is efficient
8. Improved backup and recovery facility: Database system can provide facility to take backup and recover from system crash
9. Minimal program maintenance: Independence of program and data results in minimal maintenance of the program.
10. Concurrency control: DBMS ensures data doesn't get corrupted when multiple users access same piece of data.

1.4 DBMS USERS

Database users are the ones who really use and take the benefits of the database. There will be different types of users depending on their needs and way of accessing the database.

1.4.1. Database Administrators

In any organization where many people 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, coordinating and monitoring its use, and acquiring software and hardware resources as needed. The DBA is accountable for problems such as security breaches and poor system response time. In large organizations, the DBA is assisted by a staff that carries out these functions.

1.4.2 Database Designers

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. These 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 create a design that meets these requirements. In many cases, the designers are

on the staff of the DBA and may be assigned other staff responsibilities after the database design is completed. Database designers typically interact with each potential group of users and develop views of the database that meet the data and processing requirements of these groups. Each view is then analyzed and integrated with the views of other user groups. The final database design must be capable of supporting the requirements of all user groups.

1.4.3 End Users

End users are the people whose jobs require access to the database for querying, updating, and generating reports; the database primarily exists for their use.

1.5 DATABASE OR DBMS LANGUAGES

Once the design of a database is completed and a DBMS is chosen to implement the database, the first step is to specify conceptual and internal schemas for the database and any mappings between the two. In many DBMSs where no strict separation of levels is maintained, one language, called the data definition language (DDL), is used by the DBA and by database designers to define both schemas. The DBMS will have a DDL compiler whose function is to process DDL statements in order to identify descriptions of the schema constructs and to store the schema description in the DBMS catalog.

In DBMSs where a clear separation is maintained between the conceptual and internal levels, the DDL is used to specify the conceptual schema only. Another language, the storage definition language (SDL), is used to specify the internal schema. The mappings between the two schemas may be specified in either one of these languages. In most relational DBMSs today, there is no specific language that performs the role of SDL. Instead, the internal schema is specified by a combination of functions, parameters, and specifications related to storage of files. These permit the DBA staff to control indexing choices and mapping of data to storage. For a true three-schema architecture, we would need a third language, the view definition language (VDL), to specify user views and their mappings to the conceptual schema, but in most DBMSs the DDL is used to define both conceptual and external schemas. In relational DBMSs, SQL is used in the role of VDL to define user or application views as results of predefined queries.

Once the database schemas are compiled and the database is populated with data, users must have some means to manipulate the database. Typical manipulations include retrieval, insertion, deletion,

and modification of the data. The DBMS provides a set of operations or a language called the data manipulation language (DML) for these purposes.

1.6 SCHEMAS, SUBSCHEMA AND INSTANCES

A data model—a collection of concepts that can be used to describe the structure of a database. By structure of a database we mean the data types, relationships, and constraints that apply to the data. Most data models also include a set of basic operations for specifying retrievals and updates on the database.

In a data model, it is important to distinguish between the description of the database and the database itself. The description of a database is called the database schema, which is specified during database design and is not expected to change frequently. Most data models have certain conventions for displaying schemas as diagrams. A displayed schema is called a schema diagram.

STUDENT

Name	Student_number	Class	Major
------	----------------	-------	-------

COURSE

Course_name	Course_number	Credit_hours	Department
-------------	---------------	--------------	------------

PREREQUISITE

Course_number	Prerequisite_number
---------------	---------------------

SECTION

Section_identifier	Course_number	Semester	Year	Instructor
--------------------	---------------	----------	------	------------

GRADE_REPORT

Student_number	Section_identifier	Grade
----------------	--------------------	-------

Figure 1.0 Schema diagram for a student's database

The above diagram displays the structure of each record type but not the actual instances of records.

Subschema: A subschema is a subset of the schema having the same properties that a schema has. It identifies a subset of areas, sets, records, and data names defined in the database schema. The subschema allows the user to view only that part of the database that is of interest to him.

Instances: The actual data in a database may change quite frequently. The data in the database at a particular moment in time is called a database state or snapshot. It is also called the current set of occurrences or instances in the database. In a given database state, each schema construct has its own current set of instances.

1.7 THREE LEVEL ARCHITECTURE OF DATABASE SYSTEMS

The three level architecture (Three schema architecture) is illustrated in fig.1.1. Its goal is to separate the user applications from the physical database.

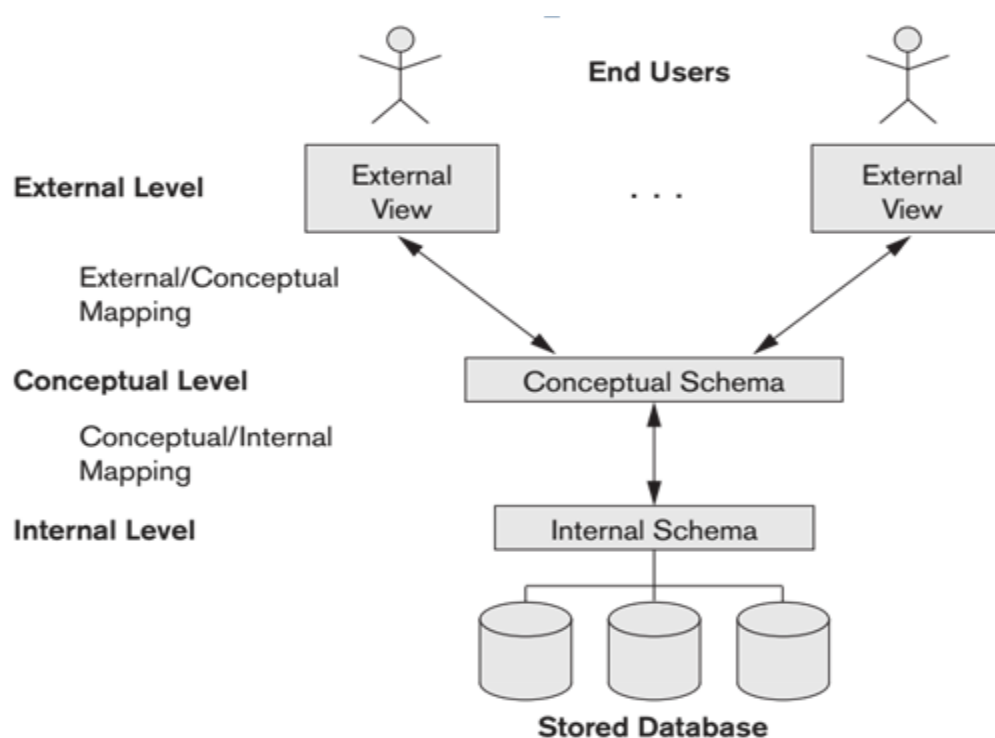


Fig. 1.1 Three Schema Architecture

1. 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.
2. The conceptual level has a conceptual schema, which describes the structure of the whole database for a community of users. The conceptual schema hides the details of physical storage structures and concentrates on describing entities, data types, relationships, user operations, and constraints.

3. 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.

Notice that the three schemas are only descriptions of data; the actual data is stored at the physical level only. In the three-schema architecture, each user group refers 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 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.

1.8 DATA INDEPENDENCE

The three-schema architecture can be used to further explain the concept of data independence, which can be defined as the capacity to change the schema at one level of a database system without having to change the schema at the next higher level. We can define two types of data independence.

1. **Logical data independence** is the capacity to change the conceptual schema without having to change external schemas or application programs. We may change the conceptual schema to expand the database (by adding a record type or data item), to change constraints, or to reduce the database (by removing a record type or data item).
2. **Physical data independence** is the capacity to change the internal schema without having to change the conceptual schema. Hence, the external schemas need not be changed as well. Changes to the internal schema may be needed because some physical files were reorganized—for example, by creating additional access structures—to improve the performance of retrieval or update.

1.9 DATA MODELS

A data model is a collection of concepts that can be used to describe the structure of a database which provides the necessary means to achieve abstraction. By structure of a database we mean the data types, relationships, and constraints that apply to the data.

1.9.1 Types of Data Models

Many data models have been proposed, which we can categorize according to the types of concepts they use to describe the database structure. High-level or conceptual data models provide concepts that are close to the way many users perceive data, whereas low-level or physical data models provide concepts that describe the details of how data is stored on the computer storage media, typically disks. Concepts provided by physical data models are generally meant for computer specialists, not for end users. Between these two extremes is a class of representational data models, which provide concepts that may be easily understood by end users but that are not too far removed from the way data is organized in computer storage. Representational data models hide many details of data storage on disk but can be implemented on a computer system directly.

Conceptual data models use concepts such as entities, attributes, and relationships. An entity represents a real-world object or concept, such as an employee or a project from the miniworld that is described in the database. An attribute represents some property of interest that further describes an entity, such as the employee's name or salary. A relationship among two or more entities represents an association among the entities, for example, a works-on relationship between an employee and a project. Entity-relationship model is a popular high-level conceptual data model.

Representational data models are the models used most frequently in traditional commercial DBMSs. These include the widely used relational data model, as well as the so-called legacy data models—the network and hierarchical models—that have been widely used in the past. Representational data models represent data by using record structures and hence are sometimes called record-based data models.

We can regard the object data model as an example of a new family of higher-level implementation data models that are closer to conceptual data models. A standard for object databases called the ODMG object model has been proposed by the Object Data Management Group (ODMG). Object data models are also frequently utilized as high-level conceptual models, particularly in the software engineering domain.

Physical data models describe how data is stored as files in the computer by representing information such as record formats, record orderings, and access paths. An access path is a

search structure that makes the search for particular database records efficient, such as indexing or hashing. An index is an example of an access path that allows direct access to data using an index term or a keyword.

Another class of data models is known as self-describing data models. The data storage in systems based on these models combines the description of the data with the data values themselves. In traditional DBMSs, the description (schema) is separated from the data. These models include XML as well as many of the key-value stores and NOSQL systems that were recently created for managing big data.

1.10 TYPES OF DATABASE SYSTEMS

Several criteria can be used to classify DBMSs. The first is the data model on which the DBMS is based. The main data model used in many current commercial DBMSs is the relational data model, and the systems based on this model are known as SQL systems. The object data model has been implemented in some commercial systems but has not had widespread use. Recently, big data systems, also known as key-value storage systems and NOSQL systems, use various data models: document-based, graph-based, column-based, and key-value data models. Many legacy applications still run on database systems based on the hierarchical and network data models.

The relational DBMSs are evolving continuously, and, in particular, have been incorporating many of the concepts that were developed in object databases. This has led to a new class of DBMSs called object-relational DBMSs. We can categorize DBMSs based on the data model: relational, object, object-relational, NOSQL, key-value, hierarchical, network, and other.

Some experimental DBMSs are based on the XML (eXtended Markup Language) model, which is a tree-structured data model. These have been called native XML DBMSs. Several commercial relational DBMSs have added XML interfaces and storage to their products.

The second criterion used to classify DBMSs is the number of users supported by the system. Single-user systems support only one user at a time and are mostly used with PCs. Multiuser systems, which include the majority of DBMSs, support concurrent multiple users.

The third criterion is the number of sites over which the database is distributed. A DBMS is centralized if the data is stored at a single computer site. A centralized DBMS can support

multiple users, but the DBMS and the database reside totally at a single computer site. A distributed DBMS (DDBMS) can have the actual database and DBMS software distributed over many sites connected by a computer network. Big data systems are often massively distributed, with hundreds of sites. The data is often replicated on multiple sites so that failure of a site will not make some data unavailable.

Homogeneous DDBMSs use the same DBMS software at all the sites, whereas heterogeneous DDBMSs can use different DBMS software at each site.

The fourth criterion is cost. Today we have open source (free) DBMS products like MySQL and PostgreSQL that are supported by third-party vendors with additional services. Many DBMS are sold in the form of licenses—site licenses allow unlimited use of the database system with any number of copies running at the customer site. Another type of license limits the number of concurrent users or the number of user seats at a location. Standalone single-user versions of some systems like Microsoft Access are sold per copy or included in the overall configuration of a desktop or laptop.

1.11 COMPARISON BETWEEN CLIENT/SERVER AND DISTRIBUTED DATABASE SYSTEM

Client/Server Database System	Distributed Database System
In this, different platforms are often difficult to manage.	In this, different platforms can be managed easily
Here, application is usually distributed across clients	Here, application is distributed across sites.
In this database system, whole system comes to a halt if server crashes.	Here, failure of one site doesn't bring the entire system down as system may be able to reroute the one site's request to another site.
Maintenance cost is low.	Maintenance cost is much higher.
In this system, access to data can be easily controlled.	In DDS not only does the access to replicate the data has to be controlled at multiple locations but also the network has to be made secure.
In this, new sites cannot be added easily	In this, new sites can be added with little or no problem.
Speed of database access is good.	Speed of database access is much better.

1.12 CHECK YOUR PROGRESS

1. What is the difference between data and information?
2. A data model is a collection of concepts that can be used to describe the _____ of a database.
3. Data stored in database at a particular moment in time is a database _____.
4. A physical view represents how the users view the data. true/false
5. Manager's salary details are to be hidden from Employees Table. This technique is called as
(a) Conceptual level data hiding (b) Physical level data hiding
(c) External level data hiding (d) Logical level data hiding.
6. Which statement is false regarding data independence?
(a) Hierarchical data model suffers from data independence.
(b) Network model suffers from data independence.
(c) Relational model suffers from logical data independence.
(d) Relational model suffers from physical data independence.
7. What are Metadata?
8. Define data model.
9. What is schema?
10. Give some applications of DBMS.

Answers to check your progress:

1. raw facts – data, meaning of data - information
2. Structure
3. state
4. false
5. c
6. d
7. data about data
8. A data model is a relatively simple representation, usually graphical, of more complex real-world data structures.
9. conceptual organization of the entire database as viewed by the database administrator.
10. Banking, Airlines, Universities, Finance, Telecom

1.13 SUMMARY

In this unit we defined a database as a collection of related data, where data means recorded facts. A typical database represents some aspect of the real world and is used for specific purposes by one or more groups of users. A DBMS is a generalized software package for implementing and maintaining a computerized database. The database and software together form a database system. We identified several characteristics that distinguish the database approach from traditional file-processing applications, and we discussed the main categories of database users, or the actors on the scene.

In this unit we introduced the main concepts used in database systems. We defined a data model and we distinguished three main categories

- High-level or conceptual data models (based on entities and relationships)
- Low-level or physical data models
- Representational data models (record-based, object oriented)

We distinguished the schema - description of a database from the database itself. The schema does not change very often, whereas the database state changes every time data is inserted, deleted, or modified. Then we described the three-schema DBMS architecture, which allows three schema levels:

- An internal schema describes the physical storage structure of the database.
- A conceptual schema is a high-level description of the whole database.
- External schemas describe the views of different user groups.

A DBMS that cleanly separates the three levels must have mappings among the schemas to transform requests and query results from one level to the next. We used the three-schema architecture to define the concepts of logical and physical data independence.

Then we discussed the main types of languages that DBMSs support. A data definition language (DDL) is used to define the database conceptual schema. In most DBMSs, the DDL also defines user views and, sometimes, storage structures; in other DBMSs, separate languages or functions exist for specifying storage structures. This distinction is fading away in today's relational implementations, with SQL serving as a catch all language to perform multiple roles, including view definition. The storage definition part (SDL) was included in SQL's early versions, but is

now typically implemented as special commands for the DBA in relational DBMSs. The DBMS compiles all schema definitions and stores their descriptions in the DBMS catalog.

A data manipulation language (DML) is used for specifying database retrievals and updates. DMLs can be high level or low level. A high-level DML can be embedded in a host programming language or it can be used as a standalone language; in the latter case it is often called a query language. We continued with an overview of the three-tier architectures for database applications. Finally, we classified DBMSs according to several criteria: data model, number of users, number of sites and cost.

1.14 KEYWORDS

- **Data**- raw facts
- **Database** – Databases are specialized structures that allow computer-based systems to store, manage, and retrieve data very quickly.
- **DBMS** - A database management system (DBMS) is a computerized system that enables users to create and maintain a database.
- **Data model** - a collection of concepts that can be used to describe the structure of a database
- **Schema** - is the description of the database.

1.15 QUESTIONS FOR SELF STUDY

1. Describe the three-schema architecture. Why do we need mappings among schema levels?
2. What is the difference between logical data independence and physical data independence? Which one is harder to achieve? Why?
3. What are the disadvantages of database system?
4. What are the responsibilities of DBA?
5. What is meant by Data Independence? State its importance in database technology
6. Explain the client server architecture. Also write advantages and disadvantages of it.
7. Explain data dictionary by giving suitable example.
8. Discuss the importance of data modeling.

1.16 REFERENCES

1. Coronel, C., & Morris, S. (2016). Database systems: design, implementation, & management. Cengage Learning.
2. Gupta, S. B., & Mittal, A. (2009). Introduction to database management system. Laxmi Publications, Ltd..
3. Ramakrishnan, R., & Gehrke, J. (2000). Database management systems. McGraw-Hill.

UNIT-2: FILE ORGANIZATION

STRUCTURE

- 2.0 Objectives
- 2.1 Introduction
- 2.2 Basic Concepts of Files
- 2.3 File Organization Techniques
- 2.4 Indexing
- 2.5 Comparison of Different File Organizations
- 2.6 Factors Affecting Choice of File Organization
- 2.7 Check your progress
- 2.8 Summary
- 2.9 Keywords
- 2.10 Questions for self-study
- 2.11 References

2.0 OBJECTIVES

After studying this unit, you will be able to:

- ✓ Explain basic concepts of files
- ✓ Discuss various file organization techniques
- ✓ Explain indexing
- ✓ Compare different file organization techniques
- ✓ Describe factors affecting choice of file organization

2.1 INTRODUCTION

Databases are stored physically as files of records, which are typically stored on magnetic disks. This unit deal with the organization of databases in storage and the techniques for accessing them efficiently using various algorithms, some of which require auxiliary data structures called indexes. The collection of data that makes up a computerized database must be stored physically on some computer storage medium. The DBMS software can then retrieve,

update, and process this data as needed. Computer storage media form a storage hierarchy that includes two main categories - Primary storage and Secondary storage. This unit discusses various file organizations in detail.

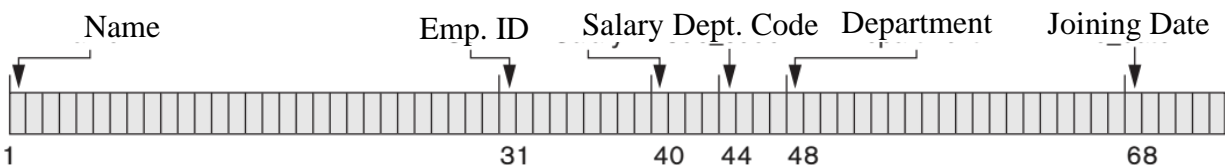
2.2 BASIC CONCEPTS OF FILES

A file is a collection of related sequence of records. A collection of field names and their corresponding data types constitutes a record. A data type, associated with each field, specifies the types of values a field can take.

2.2.1 Records and Record Types

Data is usually stored in the form of records. Each record consists of a collection of related data values or items, where each value is formed of one or more bytes and corresponds to a particular field of the record. Collection of field names and their corresponding data types constitutes a record type or record format definition. A data type, associated with each field, specifies the types of values a field can take. The data type of a field is usually one of the standard data types used in programming. These include numeric (integer, long integer, or floating point), string of characters (fixed-length or varying), Boolean (having 0 and 1 or TRUE and FALSE values only), and sometimes specially coded date and time data types. In many cases, all records in a file are of the same record type. If every record in the file has exactly the same size (in bytes), the file is said to be made up of fixed-length records. If different records in the file have different sizes, the file is said to be made up of variable-length records.

2.2.1.1 Fixed Length Records



An example of fixed-length record with six fields and size of 71 bytes.

In the above example **Name** field has 30 bytes. Whatever be the number of characters a name has in each record, exactly 30 bytes would be reserved for **Name** data in every record. Same holds good for rest of the fields such as Emp.ID, Salary etc. An advantage of fixed length record files is Insertion and deletion of records is easy to implement.

It has two main disadvantages:

1. It causes wastage of space as size of the data stored in many fields may be less than the length of the fields. In the above example length of the Name field is 30 bytes; if a Name has 20 characters then remaining 10 bytes are unused and go to waste.
2. Changing the length of each field is difficult as it requires changing the structure of the program and the database.

2.2.1.2 Variable Length Records

A file may have variable-length records for several reasons

1. The file records are of the same record type, but one or more of the fields are of varying size (variable-length fields).
2. The file records are of the same record type, but one or more of the fields may have multiple values for individual records; such a field is called a repeating field and a group of values for the field is often called a repeating group.
3. The file records are of the same record type, but one or more of the fields are optional; that is, they may have values for some but not all of the file records (optional fields).
4. The file contains records of different record types and hence of varying size (mixed file). This would occur if related records of different types were clustered (placed together) on disk blocks;



An Example for a variable-length record.

In the above example all the three fields can have data of variable length. Hence the size of each record can vary.

Advantage of Variable Length Records:

1. Database automatically adjusts the size of the record
2. It saves memory space
3. Future enhancements can be easily implemented as it is a flexible approach

Disadvantage of Variable Length Records:

DBMS overhead increases as it has to keep track of size of each record.

2.3 FILE ORGANIZATION TECHNIQUES

A file organization is a way of arranging the records in a file when the file is stored on secondary storage/primary storage (RAM, disk, tape etc.). There are different types of file organizations that are used by applications.

The different types of file organizations are as follows:

1. Heap file organization
2. Sequential file organization
3. Indexed—Sequential file organization
4. Hashing or Direct file organization.

2.3.1 Heap File Organization

In this simplest and most basic type of organization, records are placed in the file in the order in which they are inserted, so new records are inserted at the end of the file. Such an organization is called a heap or pile file.

Inserting a new record is very efficient. The last disk block of the file is copied into a buffer, the new record is added, and the block is then written back to disk. The address of the last file block is kept in the file header. However, searching for a record using any search condition involves a linear search through the file block by block—an expensive procedure. If only one record satisfies the search condition, then, on the average, a program will read into memory and search half the file blocks before it finds the record. For a file of b blocks, this requires searching $(b/2)$ blocks, on average. If no records or several records satisfy the search condition, the program must read and search all b blocks in the file.

To delete a record, a program must first find its block, copy the block into a buffer, delete the record from the buffer, and finally write the emptied buffer block back to the disk. This leaves unused space in the disk block. Deleting a large number of records in this way results in wasted storage space. Another technique used for record deletion is to have an extra byte or bit, called a deletion marker, stored with each record. A record is deleted by setting the

deletion marker to a certain value. A different value for the marker indicates a valid (not deleted) record. Search programs consider only valid records in a block when conducting their search. Both of these deletion techniques require periodic reorganization of the file to reclaim the unused space of deleted records. During reorganization, the file blocks are accessed consecutively, and records are packed by removing deleted records. After such reorganization, the blocks are filled to capacity once more. Another possibility is to use the space of deleted records when inserting new records, although this requires extra bookkeeping to keep track of empty locations.

To read all records in order of the values of some field, we create a sorted copy of the file. We need to keep in mind sorting is an expensive operation for a large disk file and special techniques for external sorting are used.

2.3.2 Sequential File Organization

We can physically order the records of a file on disk based on the values of one of their fields called the ordering field. This leads to an ordered or sequential file. If the ordering field is also a key field of the file, a field guaranteed to have a unique value in each record, then the field is called the ordering key for the file.

Ordered records have some advantages over unordered files. First, reading the records in order of the ordering key values becomes extremely efficient because no sorting is required. The search condition may be of the type $\langle \text{key} = \text{value} \rangle$, or a range condition such as $\text{value1} < \text{key} < \text{value2}$. Second, finding the next record from the current one in order of the ordering key usually requires no additional block accesses because the next record is in the same block as the current one (unless the current record is the last one in the block). Third, using a search condition based on the value of an ordering key field results in faster access when the binary search technique is used, which constitutes an improvement over linear searches, although it is not often used for disk files. Ordered files are blocked and stored on contiguous cylinders to minimize the seek time.

A binary search for disk files can be done on the blocks rather than on the records. Suppose that the file has b blocks numbered $1, 2, \dots, b$; the records are ordered by ascending value of their ordering key field; and we are searching for a record whose ordering key field value is K .

A binary search usually accesses $\log_2(b)$ blocks, whether the record is found or not—an improvement over linear searches, where, on the average, $(b/2)$ blocks are accessed when the record is found and b blocks are accessed when the record is not found.

A search criterion involving the conditions $>$, $<$, \geq , and \leq on the ordering field is efficient, since the physical ordering of records means that all records satisfying the condition are contiguous in the file.

Ordering does not provide any advantages for random or ordered access of the records based on values of the other non-ordering fields of the file. In these cases, we do a linear search for random access. To access the records in order based on a non-ordering field, it is necessary to create another sorted copy in a different order of the file.

Inserting and deleting records are expensive operations for an ordered file because the records must remain physically ordered. To insert a record, we must find its correct position in the file, based on its ordering field value, and then make space in the file to insert the record in that position. For a large file this can be very time consuming because, on the average, half the records of the file must be moved to make space for the new record. This means that half the file blocks must be read and rewritten after records are moved among them. For record deletion, the problem is less severe if deletion markers and periodic reorganization are used.

One option for making insertion more efficient is to keep some unused space in each block for new records. However, once this space is used up, the original problem resurfaces. Another frequently used method is to create a temporary unordered file called an overflow or transaction file. With this technique, the actual ordered file is called the main or master file. New records are inserted at the end of the overflow file rather than in their correct position in the main file. Periodically, the overflow file is sorted and merged with the master file during file reorganization. Insertion becomes very efficient, but at the cost of increased complexity in the search algorithm. One option is to keep the highest value of the key in each block in a separate field after taking into account the keys that have overflowed from that block. Otherwise, the overflow file must be searched using a linear search if, after the binary search, the record is not found in the main file. For applications that do not require the most up-to-date information, overflow records can be ignored during a search.

Modifying a field value of a record depends on two factors: the search condition to locate the

record and the field to be modified. If the search condition involves the ordering key field, we can locate the record using a binary search; otherwise we must do a linear search. A non-ordering field can be modified by changing the record and rewriting it in the same physical location on disk—assuming fixed length records. Modifying the ordering field means that the record can change its position in the file. This requires deletion of the old record followed by insertion of the modified record.

Reading the file records in order of the ordering field is efficient if we ignore the records in overflow, since the blocks can be read consecutively using double buffering. To include the records in overflow, we must merge them in their correct positions; in this case, first we can reorganize the file, and then read its blocks sequentially. To reorganize the file, first we sort the records in the overflow file, and then merge them with the master file. The records marked for deletion are removed during the reorganization.

2.3.3 Index Sequential File Organization

Index sequential file organization is used to overcome the disadvantages of sequential file organization. It also preserves the advantages of sequential access. This organization enables fast searching of records with the use of index. Some basic terms associated with index sequential file organization are as follows:

Block: Block is a unit of storage in which records are saved.

Index: Index is a table with a search key by which block of a record can be found.

Pointer: Pointer is a variable which points from index entry to starting address of block.

To manipulate any record, search key of index is entered to find the starting address of block and then required record is searched sequentially within the block.

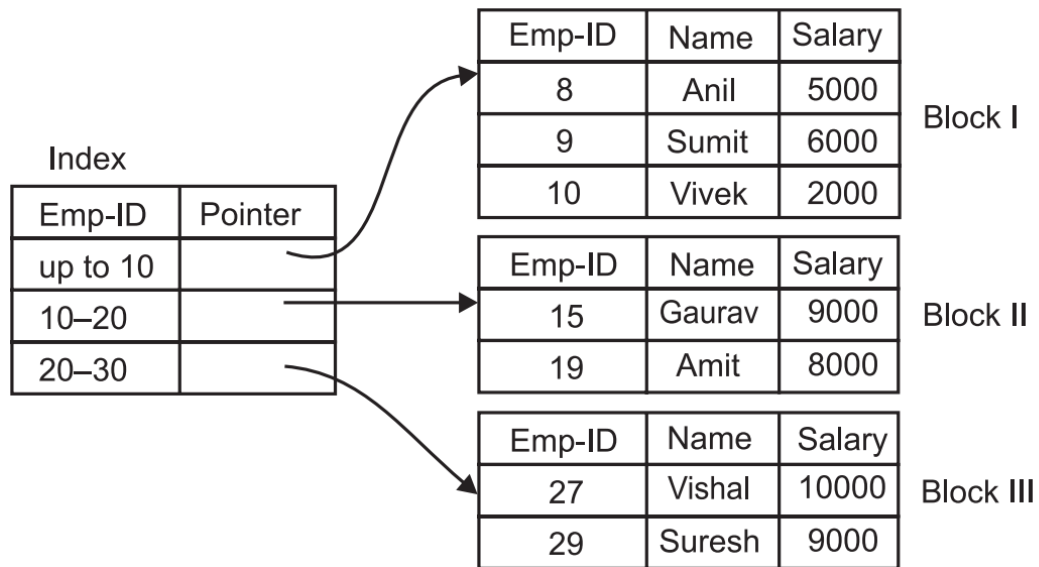


Fig.2.0 Index sequential file organization.

1. Components of an Indexed Sequential File:

- a) Prime area: During the creation of index sequential file, records are written in prime area. Records are maintained according to any key. Hence, prime area must be a sequential file.
- b) Overflow area: Overflow area is used during addition of new records.

There are two types of overflow area:

 - Cylinder overflow area: This area consists of free area on each cylinder which is reserved for overflow records for that particular cylinder.
 - Independent overflow area: This area is used to store overflow records from anywhere.
- c) Record characteristics: Usually fixed length records are used.
- d) Indexes: Index is a collection of entries and each entry corresponds to block of storage.
 - First level index: The lowest level index is known as first level index.
 - Higher level index: Higher level indexing is used when first level index becomes too large.
 - Cylinder index: The indexes can be made according to the hardware boundaries. Cylinder index entries consists one entry for each cylinder.
 - Master index: The highest level of index is known as master index.

2. Operations on Index Sequential Files:

- a) Creating an index sequential file: After allocating free space and making necessary entries in file system directory all records are written into prime area in sequential manner according to key value.
- b) Opening and closing an existing file: It is same as sequential file operations.
- c) Reading records from an index sequential file (or searching any record) : To search any record enter the key value of record then first search the block of record in index then search record sequentially within the block.
- d) Modification of records: To modify a record, first search that record and then modify it. Updated record is stored at same position if it has same key value and size equal to the original record. Otherwise original record is deleted and new record is inserted.
- e) Deletion of records: To delete a record, first search that record. Specific codes are used to indicate deleted records. Records consist of flags. Specific code is inserted to the flag of that record that needs to be deleted.
- f) Insertion of records: To insert a record, first find the desired block according to key value. Then confirm that record does not exist already. The record is first placed in overflow area and then copied to the desired block.

2.3.4 Hashing

Another type of file organization is based on hashing, which provides very fast access to records under certain search conditions. This organization is usually called a hash file. The search condition must be an equality condition on a single field, called the hash field. In most cases, the hash field is also a key field of the file, in which case it is called the hash key. The idea behind hashing is to provide a function h , called a hash function or randomizing function, which is applied to the hash field value of a record and yields the address of the disk block in which the record is stored. A search for the record within the block can be carried out in a main memory buffer. For most records, we need only a single-block access to retrieve that record. Hashing is also used as an internal search structure within a program whenever a group of records is accessed exclusively by using the value of one field.

2.3.4.1 Internal Hashing

For internal files (files in main memory), hashing is typically implemented as a hash table through the use of an array of records.

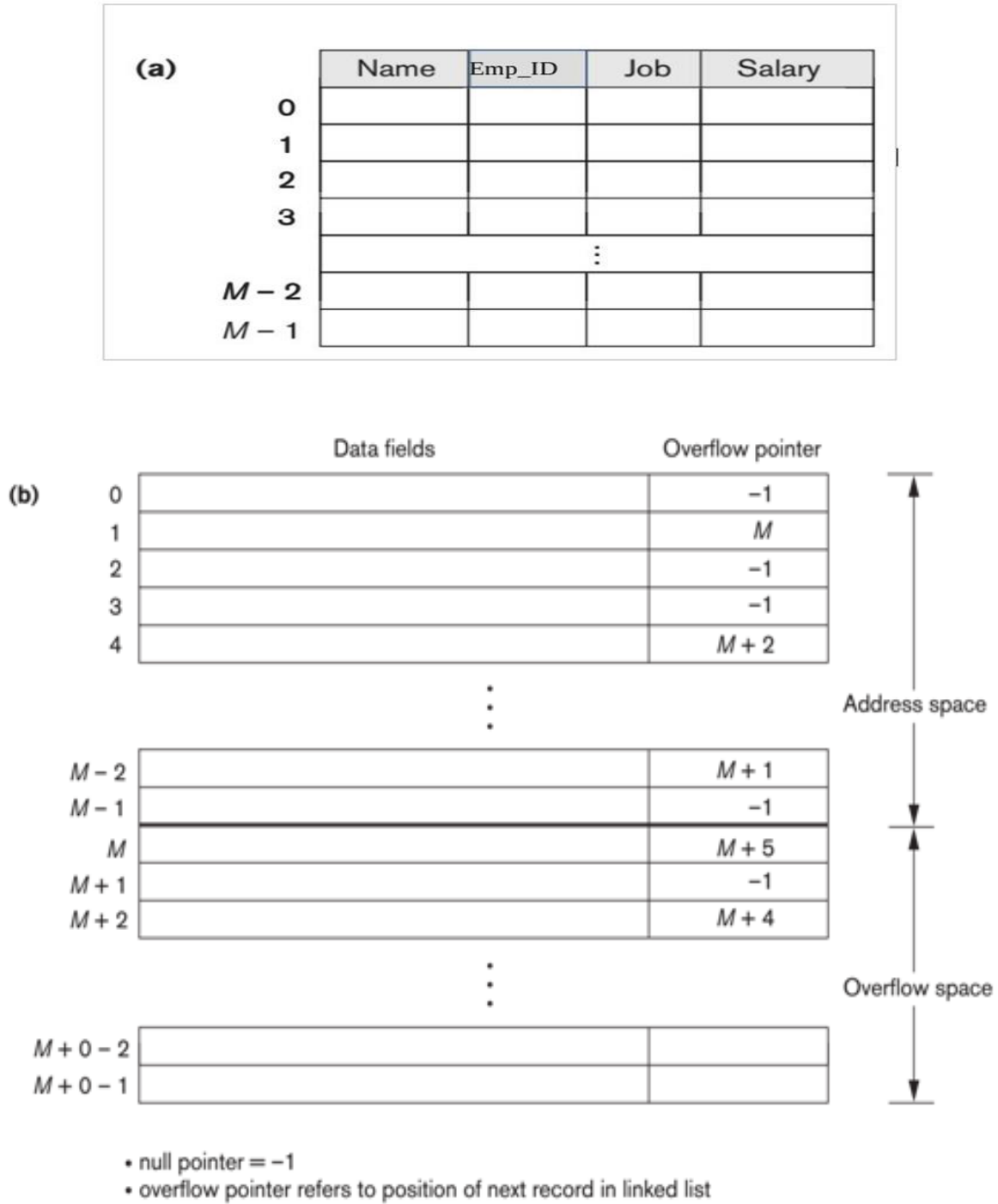


Fig.2.1 (a) and (b) Examples for Internal Hashing

Fig.2.1 (a) and (b) Internal hashing data structures. (a) Array of M positions for use in internal hashing. (b) Collision resolution by chaining records.

Suppose that the array index range is from 0 to $M - 1$, as shown in Figure 2.1(a) then we have M slots whose addresses correspond to the array indexes. We choose a hash function that transforms the hash field value into an integer between 0 and $M - 1$. One common hash function is the $h(K) = K \bmod M$ function, which returns the remainder of an integer hash field value K after division by M ; this value is then used for the record address.

Non integer hash field values can be transformed into integers before the mod function is applied. For character strings, the numeric (ASCII) codes associated with characters can be used in the transformation—for example, by multiplying those code values. For a hash field whose data type is a string of 20 characters, Algorithm 2.1(a) can be used to calculate the hash address.

Two simple hashing algorithms: (a) Applying the mod hash function to a character string K . (b) Collision resolution by open addressing.

```
(a) temp ← 1;
for i ← 1 to 20 do temp ← temp * code(K[i ]) mod M ;
hash_address ← temp mod M;

(b) i ← hash_address(K); a ← i;
if location i is occupied
    then begin i ← (i + 1) mod M;
        while (i ≠ a) and location i is occupied
            doi ← (i + 1) mod M;
        if (i = a) then all positions are full
        elsenew_hash_address ← i;
    end;
```

Other hashing functions can be used. One technique, called folding, involves applying an arithmetic function such as addition or a logical function such as exclusive or to different portions of the hash field value to calculate the hash address (for example, with an address space from 0 to 999 to store 1,000 keys, a 6-digit key 235469 may be folded and stored at the address: $(235+964) \bmod 1000 = 199$). Another technique involves picking some digits of the hash field value—for instance, the third, fifth, and eighth digits—to form the hash address (for example,

storing 1,000 students with USN of 10 digits into a hash file with 1,000 positions would give the USN 301678923 a hash value of 172 by this hash function). The problem with most hashing functions is that they do not guarantee that distinct values will hash to distinct addresses, because the hash field space—the number of possible values a hash field can take—is usually much larger than the address space—the number of available addresses for records. The hashing function maps the hash field space to the address space.

A collision occurs when the hash field value of a record that is being inserted hashes to an address that already contains a different record. In this situation, we must insert the new record in some other position, since its hash address is occupied. The process of finding another position is called collision resolution. There are numerous methods for collision resolution, including the following:

- Open addressing: Proceeding from the occupied position specified by the hash address, the program checks the subsequent positions in order until an unused (empty) position is found. Algorithm 2.1(b) may be used for this purpose.
- Chaining: For this method, various overflow locations are kept, usually by extending the array with a number of overflow positions. Additionally, a pointer field is added to each record location. A collision is resolved by placing the new record in an unused overflow location and setting the pointer of the occupied hash address location to the address of that overflow location. A linked list of overflow records for each hash address is thus maintained, as shown in Figure 2.1(b).
- Multiple hashing: The program applies a second hash function if the first results in a collision. If another collision results, the program uses open addressing or applies a third hash function and then uses open addressing if necessary. Note that the series of hash functions are used in the same order for retrieval.

Each collision resolution method requires its own algorithms for insertion, retrieval, and deletion of records. The algorithms for chaining are the simplest. Deletion algorithms for open addressing are rather tricky.

The goal of a good hashing function is twofold: first, to distribute the records uniformly over the address space so as to minimize collisions, thus making it possible to locate a record with a given key in a single access. The second, somewhat conflicting, goal is to achieve the above yet occupy

the buckets fully, thus not leaving many unused locations. Simulation and analysis studies have shown that it is usually best to keep a hash file between 70 and 90% full so that the number of collisions remains low and we do not waste too much space. Hence, if we expect to have r records to store in the table, we should choose M locations for the address space such that (r/M) is between 0.7 and 0.9. It may also be useful to choose a prime number for M , since it has been demonstrated that this distributes the hash addresses better over the address space when the mod hashing function is used modulo a prime number. Other hash functions may require M to be a power of 2.

2.3.4.2 External Hashing for Disk Files

Hashing for disk files is called external hashing. To suit the characteristics of disk storage, the target address space is made of buckets, each of which holds multiple records. A bucket is either one disk block or a cluster of contiguous disk blocks. The hashing function maps a key into a relative bucket number rather than assigning an absolute block address to the bucket. A table maintained in the file header converts the bucket number into the corresponding disk block address, as illustrated in Figure 2.2.

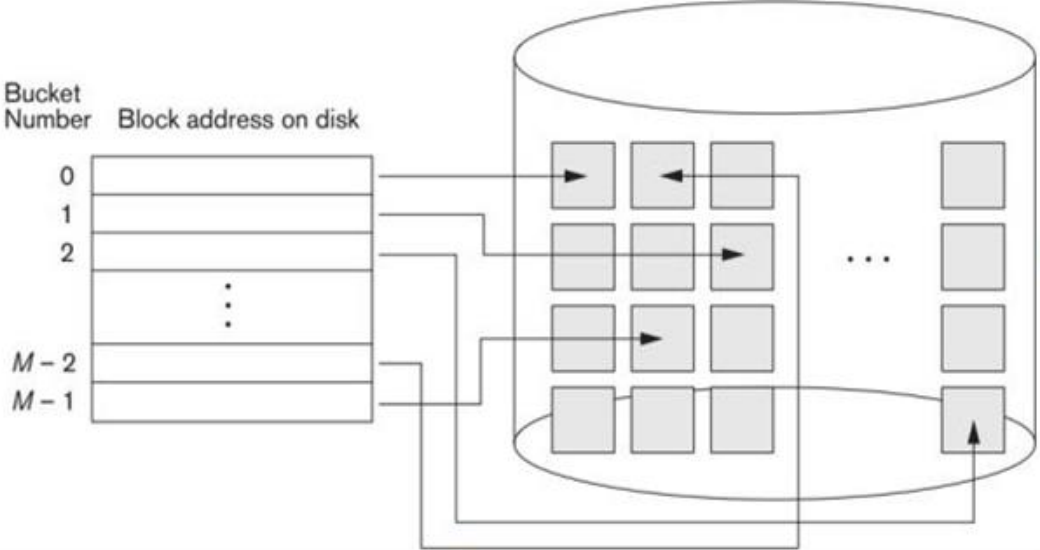


Fig. 2.2. Matching bucket numbers to disk block addresses

The collision problem is less severe with buckets, because as many records as will fit in a bucket can hash to the same bucket without causing problems. However, we must make provisions for

the case where a bucket is filled to capacity and a new record being inserted hashes to that bucket. We can use a variation of chaining in which a pointer is maintained in each bucket to a linked list of overflow records for the bucket, as shown in Figure 2.3. The pointers in the linked list should be record pointers, which include both a block address and a relative record position within the block.

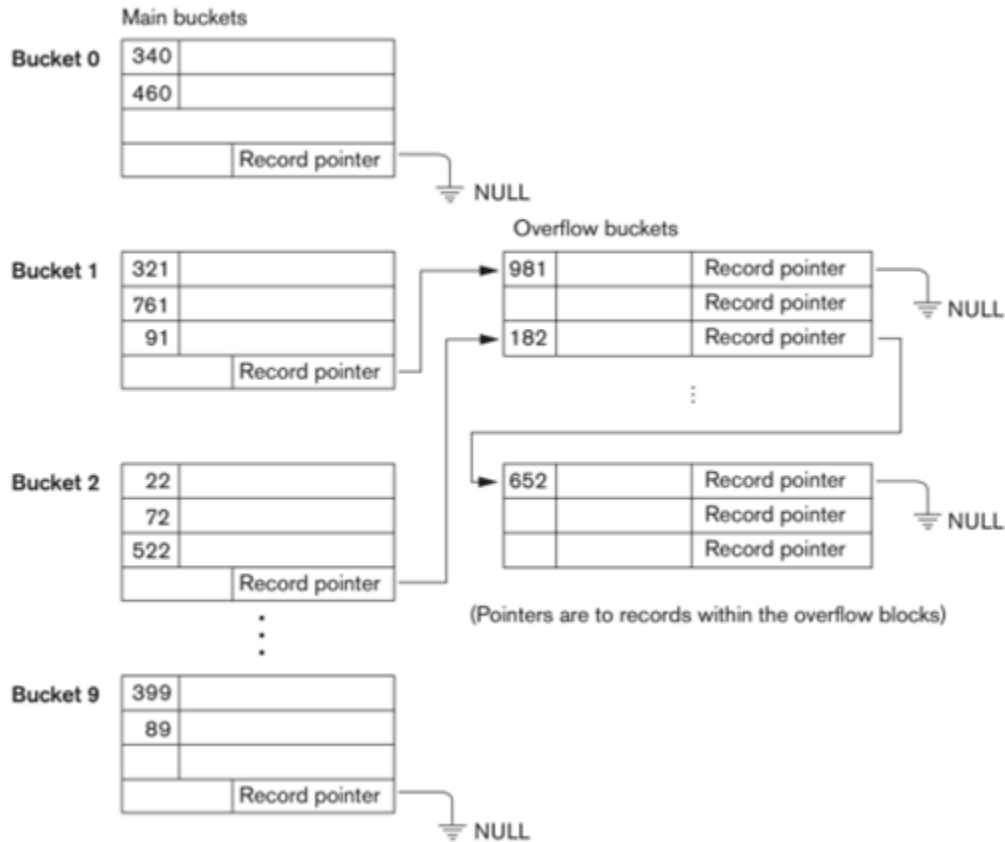


Fig.2.3. Handling overflow for buckets by chaining

The hashing scheme described so far is called static hashing because a fixed number of buckets M is allocated. The function does key-to-address mapping, whereby we are fixing the address space. This can be a serious drawback for dynamic files. Suppose that we allocate M buckets for the address space and let m be the maximum number of records that can fit in one bucket; then at most $(m * M)$ records will fit in the allocated space. If the number of records turns out to be substantially fewer than $(m * M)$, we are left with a lot of unused space. On the other hand, if the number of records increases to substantially more than $(m * M)$, numerous collisions will result and retrieval will be slowed down because of the long lists of overflow records. In either case, we may have to change the number of blocks M allocated and then use a new hashing function

(based on the new value of M) to redistribute the records. These reorganizations can be quite time-consuming for large files. Newer dynamic file organizations based on hashing allow the number of buckets to vary dynamically with only localized reorganization.

When using external hashing, searching for a record given a value of some field other than the hash field is as expensive as in the case of an unordered file. Record deletion can be implemented by removing the record from its bucket. If the bucket has an overflow chain, we can move one of the overflow records into the bucket to replace the deleted record. If the record to be deleted is already in overflow, we simply remove it from the linked list. Notice that removing an overflow record implies that we should keep track of empty positions in overflow. This is done easily by maintaining a linked list of unused overflow locations.

Modifying a specific record's field value depends on two factors: the search condition to locate that specific record and the field to be modified. If the search condition is an equality comparison on the hash field, we can locate the record efficiently by using the hashing function; otherwise, we must do a linear search. A non-hash field can be modified by changing the record and rewriting it in the same bucket. Modifying the hash field means that the record can move to another bucket, which requires deletion of the old record followed by insertion of the modified record.

2.4 INDEXING

An index is a collection of data entries which is used to locate a record in a file. Index table records consist of two parts, the first part consists of value of prime or non-prime attributes of file record known as indexing field and, the second part consists of a pointer to the location where the record is physically stored in memory. In general, index table is like the index of a book, which consists of the name of topic and the page number. During searching of a file record, index is searched to locate the record memory address instead of searching a record in secondary memory. On the basis of properties that affect the efficiency of searching, the indexes can be classified into two categories.

1. Ordered indexing
2. Hashed indexing

2.4.1 Ordered Indexing

In ordered indexing, records of file are stored in some sorted order in physical memory. The values in the index are ordered (sorted) so that binary search can be performed on the index. Ordered indexes can be divided into two categories.

1. Dense indexing
2. Sparse indexing.

2.4.1.1 Dense and Sparse Indexing

Dense index : In dense indexing there is a record in index table for each unique value of the search-key attribute of file and a pointer to the first data record with that value. The other records with the same value of search-key attribute are stored sequentially after the first record. The order of data entries in the index differs from the order of data records as shown in Figure 2.4.

The advantages of dense index are:

- a) It is efficient technique for small and medium sized data files.
- b) Searching is comparatively fast and efficient.

The disadvantages of dense index are:

- a) Index table is large and require more memory space.
- b) Insertion and deletion is comparatively complex.
- c) In-efficient for large data files.

Sparse index : On contrary, in sparse indexing there are only some records in index table for unique values of the search-key attribute of file and a pointer to the first data record with that value. To search a record in sparse index we search for a value that is less than or equal to value in index for which we are looking. After getting the first record, linear search is performed to retrieve the desired record. There is at most one sparse index since it is not possible to build a sparse index that is not clustered.

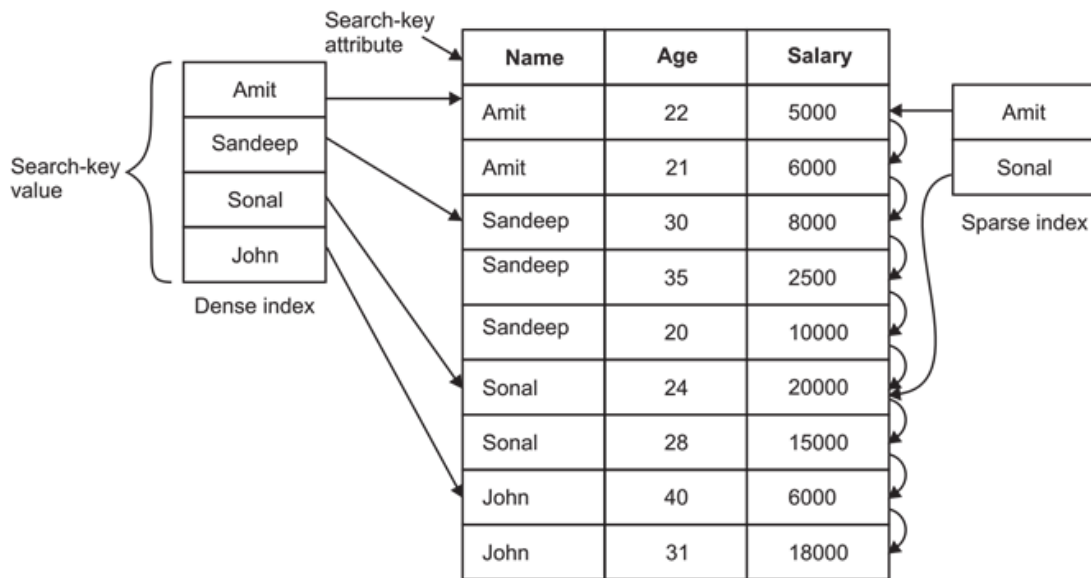


Fig. 2.4. Dense and sparse index.

The advantages of sparse index are:

- a) Index table is small and hence save memory space (specially in large files).
- b) Insertion and deletion is comparatively easy.

The disadvantages of sparse index are:

- a) Searching is comparatively slower, since index table is searched and then linear search is performed inside secondary memory.

2.4.1.2 Clustered and Non-Clustered Indexes

Clustered index: In clustering, index file records are stored physically in order on a non-prime key attribute that does not have a unique value for each record. The non-prime key field is known as clustering field and index is known as clustering index.

It is same as dense index. A file can have at most one clustered index as it can be clustered on at most one search key attribute. It may be sparse.

Non-clustered index: An index that is not clustered is known as non-clustered index. A data file can have more than one non-clustered index.

2.4.1.3 Primary and Secondary Index

Primary index: A primary index consists of all prime-key attributes of a table and a pointer to physical memory address of the record of data file. To retrieve a record on the basis of all primary key attributes, primary index is used for fast searching.

A binary search is done on index table and then directly retrieve that record from physical memory. It may be sparse.

The major advantages of primary index are:

- a) Search operation is very fast.
- b) Index table record is usually smaller.
- c) A primary index is guaranteed not to duplicate.

The major disadvantages of primary index are:

- a) There is only one primary index of a table. To search a record on less than all prime-key attributes, linear search is performed on index table.
- b) To create a primary index of an existing table, records should be in some sequential order otherwise database is required to be adjusted.

Secondary index: A secondary index provides a secondary means of accessing a data file. A secondary index may be on a candidate key field or on non-prime key attributes of a table. To retrieve a record on the basis of non-prime key attributes, secondary index can be used for fast searching. Secondary index must be dense with a index entry for every search key value and a pointer to every record in a file.

The major advantages of secondary index are:

- a) Improve search time if search on non-prime key attributes.
- b) A data file can have more than one secondary index.

The major disadvantages of secondary index are:

- a) A secondary index usually needs more storage space.
- b) Search time is more than primary index.
- c) They impose a significant overhead on the modification of database.

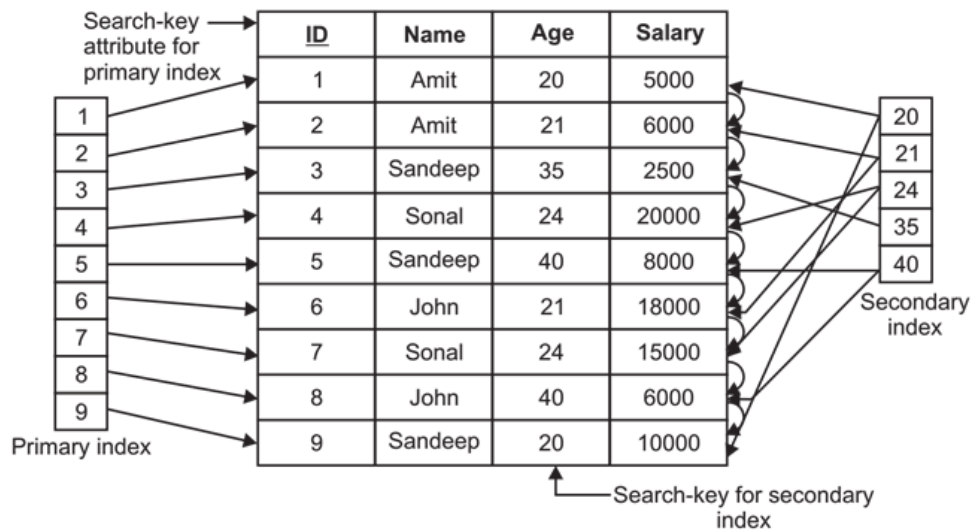


Fig. 2.5. Primary and secondary index.

2.4.1.4 Single and Multilevel Indexes

Single level indexes: A single stage index for a data file is known as single level index. A single level index cannot be divided. It is useful in small and medium size data files. If the file size is bigger, then single level, indexing is not an efficient method. Searching is faster than other indexes for small size data files.

Multilevel indexes: A single index for a large size data file increases the size of index table and increases the search time that results in slower searches. The idea behind multilevel indexes is that, a single level index is divided into multiple levels, which reduces search time.

In multilevel indexes, the first level index consists of two fields, the first field consists of a value of search key attributes and a second field consists of a pointer to the block (or second level index) which consists those values and so on.

To search a record in multilevel index, binary search is used to find the largest of all the small values or equal to the one that needs to be searched. The pointer points to a block of the inner index. After reaching to the desired block, the desired record is searched (in case of two-level indexing) otherwise again the largest of the small values or equal to the one that needs to be searched and so on. Benefits of multilevel indexes are they reduce search time significantly for large size data files.

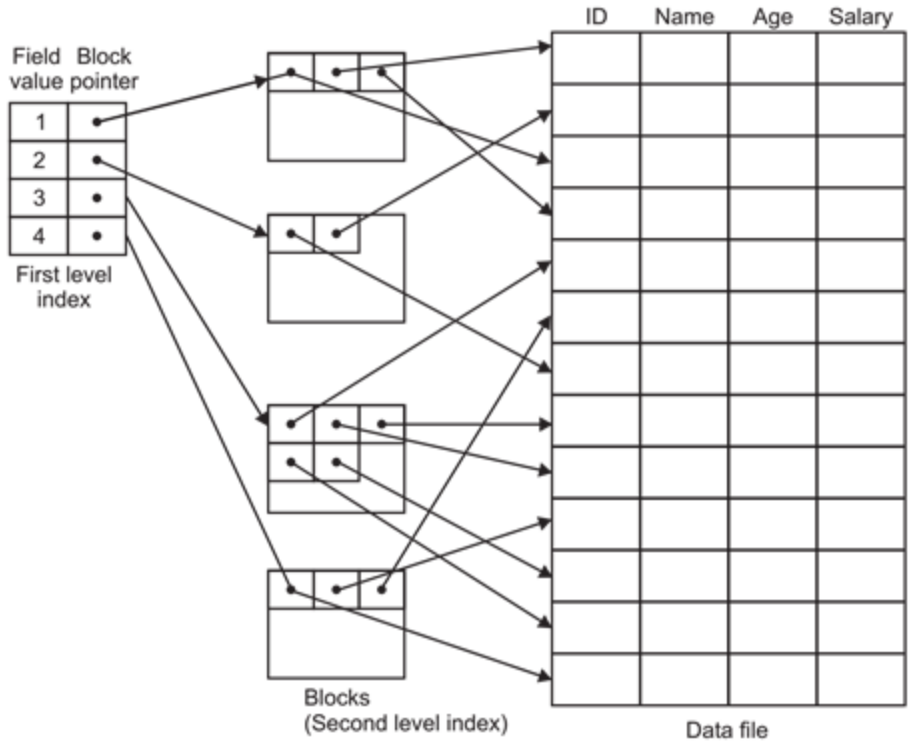


Fig.2.6. Multilevel indexing

2.4.2 Hashed Indexing

To overcome the disadvantages of ordered indexing, a hash index can be created for a data file. Hashing allow us to avoid accessing an index structure. A hashed index consists of two fields, the first field consists of search key attribute values and second field consists of pointer to the hash file structure. Hashed indexing is based on values of records being uniformly distributed using a hashed function.

2.5 COMPARISON OF DIFFERENT FILE ORGANIZATIONS

The main differences between various file organizations are as follows:

Sequential	Indexed	Hashed/Direct
Random retrieval on primary key is impractical.	Random retrieval of primary key is moderately fast.	Random retrieval of primary key is very fast.
There is no wasted space for data.	No wasted space for data but there is extra space for index.	Extra space for addition and deletion of records.

Sequential retrieval on primary key is very fast.	Sequential retrieval on primary key is moderately fast.	Sequential retrieval of primary key is impractical
Multiple key retrieval in sequential file organization is possible	Multiple key retrieval is very fast with multiple indexes.	Multiple key retrieval is not possible
Updating of records generally requires rewriting the file	Updating of records requires maintenance of indexes.	Updating or records is the easiest one.
Addition of new records requires rewriting the file.	Addition of new records is easy and requires maintenance of indexes.	Addition of new records is very easy
Deletion of records can create wasted space.	Deletion of records is easy if space can be allocated dynamically.	Deletion of records is very easy.

2.6 FACTORS AFFECTING CHOICE OF FILE ORGANIZATION

The major factors that affect the choice of file organization are as follows:

- a. Access type: In order to search a record, whether random access or sequential access is required.
- b. Access time: The total time taken by file organization to find a particular record.
- c. File size: Choice is also dependent upon file size. If file size is large then choose direct access otherwise choose sequential access.
- d. Overhead: Each technique has some overhead (it may be space overhead, time overhead etc.). Any technique giving fast access may waste more space than slower techniques.
- e. Updation time: Time required to add, delete and modify any record also plays important role in efficiency.
- f. Complexity: If technique is fast then it is complex and expensive. Whether funds are available to adopt new techniques.
- g. Availability of hardware: The hardware that supports file organization. Example tape reader supports only sequential file organization.

2.7 CHECK YOUR PROGRESS

1. The number of entries in a clustering index is equal to the number of blocks in the data file. (T/F)
2. In _____ file organization, the records are stored in the file in the order in which they are inserted
3. On average, random accesses to N pages are faster than a sequential scan of N pages because random IO's tend to access different cylinders and therefore cause less contention.(T/F)
4. An index is clustered, if (Choose the right option)
 - (a) It is on a set of fields that form a candidate key
 - (b) It is on a set of fields that include the primary key
 - (c) The data records of the file are organized in the same order as the data entries of the index
 - (d) The data records of the file are organized not in the same order as the data entries of the index.
5. A clustering index is defined on the fields which are of type (Choose the right option)
 - (a) non-key and ordering
 - (b) non-key and non-ordering
 - (c) Key and ordering
 - (d) key and non-ordering
6. What is the difference between primary and secondary storage?
7. What is the difference between static and dynamic files?

Answers to check your progress:

1. F
2. heap
3. False. Random I/Os are more expensive than sequential I/Os because every random I/O will incur seek time and rotational delay
4. (c)
5. (a)
6. Primary storage refers to the main storage of the computer or main memory which is the random access memory or RAM. Secondary storage, on the other hand, refers to the external storage devices used to store data on a long-term basis.

Primary storage holds data or applications which can be directly accessed by the processing unit with minimum or no delay. On the contrary, secondary storage is used to store and retrieve data permanently with no delay.

7. Static files – fixed length records. Dynamic files – variable length records.

2.8 SUMMARY

We began this unit by discussing the basic concepts of files. We presented different ways of storing file records on disk. File records are grouped into disk blocks and can be fixed length or variable length and of the same record type or mixed types. Three types of file organizations were then discussed: unordered, ordered, and hashed. Unordered files require a linear search to locate records, but record insertion is very simple. We discussed the deletion problem and the use of deletion markers. Ordered files shorten the time required to read records in order of the ordering field. The time required to search for an arbitrary record, given the value of its ordering key field, is also reduced if a binary search is used. However, maintaining the records in order makes insertion very expensive; thus the technique of using an unordered overflow file to reduce the cost of record insertion was discussed. Overflow records are merged with the master file periodically, and deleted records are physically dropped during file reorganization. Hashing provides very fast access to an arbitrary record of a file, given the value of its hash key. The most suitable method for external hashing is the bucket technique, with one or more contiguous blocks corresponding to each bucket. Collisions causing bucket overflow are handled by open addressing, chaining, or multiple hashing. Access on any non-hash field is slow, and so is ordered access of the records on any field.

We discussed file organizations that involve additional access structures, called indexes, to improve the efficiency of retrieval of records from a data file. These access structures may be used in conjunction with the primary file organization. Three types of ordered single-level indexes were introduced: primary, clustering, and secondary. Each index is specified on a field of the file. Primary and clustering indexes are constructed on the physical ordering field of a file, whereas secondary indexes are specified on non-ordering fields as additional access structures to improve performance of queries and transactions. The field for a primary index must also be a key of the file, whereas it is a non-key field for a clustering index. A single-level index is an ordered file and is searched using a binary search. We learnt how multilevel indexes can be

constructed to improve the efficiency of searching an index. We concluded this unit by discussing Factors which affects choice of file organization.

2.9 KEYWORDS

- **Fixed length record**- every record in the file has exactly the same size (in bytes), the file is said to be made up of fixed-length records.
- **Variable length record** – different records in the file have different sizes, the file is said to be made up of variable-length records..
- **File Organization** - A file organization is a way of arranging the records in a file when the file is stored on secondary/Primary storage (RAM, disk, tape etc)
- **Index** - An index is a collection of data entries which is used to locate a record in a file.

2.10 QUESTIONS FOR SELF-STUDY

1. What is record? What are its types?
2. What are the advantages and disadvantages of fixed length records?
3. What are the reasons for having variable-length records?
4. What are the advantages and disadvantages of fixed length records?
5. What is heap file organization? What are its advantages and disadvantages?
6. What is the difference between static and dynamic files?
7. What is index sequential file organization? What are its advantages and disadvantages?
8. How does multilevel indexing improve the efficiency of searching an index file?

2.11 References

1. Coronel, C., & Morris, S. (2016). Database systems: design, implementation, & management. Cengage Learning.
2. Gupta, S. B., & Mittal, A. (2009). Introduction to database management system. Laxmi Publications, Ltd..
3. Ramakrishnan, R., &Gehrke, J. (2000). Database management systems. McGraw-Hill.

UNIT-3: DATA MODELS

STRUCTURE

- 3.0 Objectives
- 3.1 Data Modeling and Data Models
- 3.2 The Importance of Data Models
- 3.3 Data Model Basic Building Blocks
- 3.4 Business Rules
- 3.5 The Evolution of Data Models
- 3.6 Degrees of Data Abstraction
- 3.7 Check your progress
- 3.8 Summary
- 3.9 Keywords
- 3.10 Questions for self-study
- 3.11 References

3.0 OBJECTIVES

After studying this unit, you will be able to:

- ✓ Describe Data Modeling process and various Data Models
- ✓ Explain The Importance of Data Models in Database design
- ✓ Explain the Basic Building Blocks of Data Models
- ✓ Explain the importance of Business Rules
- ✓ Outline chronological order of Data Models
- ✓ Appraise the end users' view of the data environment.

3.1 DATA MODELING AND DATA MODELS

A data model is a relatively simple representation, usually graphical, of more complex real-world data structures. In general terms, a model is an abstraction of a more complex real-world object or event. A model's main function is to help you understand the complexities of the real-world environment. Within the database environment, a data model represents data

structures and their characteristics, relations, constraints, transformations, and other constructs with the purpose of supporting a specific problem domain.

Data modeling is an iterative, progressive process. You start with a simple understanding of the problem domain, and as your understanding of the problem domain increases, so does the level of detail of the data model. Done properly, the final data model is in effect a “blueprint” containing all the instructions to build a database that will meet all end-user requirements. This blueprint is narrative and graphical in nature, meaning that it contains both text descriptions in plain, unambiguous language and clear, useful diagrams depicting the main data elements.

3.2 DATA MODELING AND DATA MODELS

Data models can facilitate interaction among the designer, the applications programmer, and the end user. A well-developed data model can even foster improved understanding of the organization for which the database design is developed. In short, data models are a communication tool.

When a good database blueprint (data model) is available, it does not matter that an applications programmer’s view of the data is different from that of the manager and/or the end user. Conversely, when a good database blueprint is not available, problems are likely to ensue. For instance, an inventory management program and an order entry system may use conflicting product-numbering schemes, thereby costing the firm dearly.

Data model is akin to a house blueprint. Just as you are not likely to build a good house without a blueprint, you are equally unlikely to create a good database without first creating an appropriate data model.

3.3 DATA MODEL BASIC BUILDING BLOCKS

The basic building blocks of all data models are entities, attributes, relationships, and constraints. An entity, which is a thing or object 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 instance, a company, a job, or a university course).

Each entity has attributes—the particular properties that describe 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.

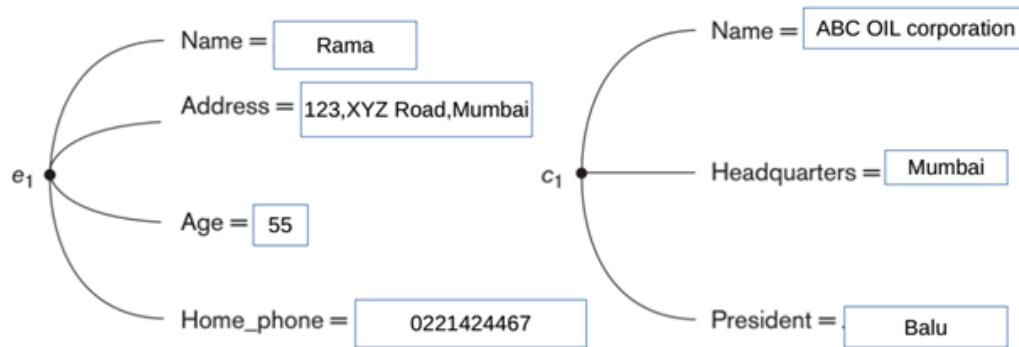


Figure 3.1 Two entities, EMPLOYEE e_1 , and COMPANY c_1 , and their attributes.

Figure 3.1 shows two entities and the values of their attributes. The EMPLOYEE entity e_1 has four attributes: Name, Address, Age, and Home_phone; their values are ‘Rama’, ‘123,XYZ, Road, Mumbai’, ‘55’, and ‘0221424467’, respectively. The COMPANY entity c_1 has three attributes: Name, Headquarters, and President; their values are ‘ABC Oil corporation’, ‘Mumbai’, and ‘Balu’, respectively.

A relationship describes an association among entities. For example, a relationship exists between customers and agents that can be described as follows: an agent can serve many customers, and each customer may be served by one agent. Data models use three types of relationships: one-to-many, many-to-many, and one-to-one. Database designers usually use the shorthand notations 1:M or 1..*, M:N or *..*, and 1:1 or 1..1, respectively.

A constraint is a restriction placed on the data. Constraints are important because they help to ensure data integrity.

Constraints are normally expressed in the form of rules.

For example:

- An employee’s salary must have values that are between 6,000 and 350,000.
- A student’s CGPA must be between 0.00 and 10.00.

How do you properly identify entities, attributes, relationships, and constraints? The first step is to clearly identify the business rules for the problem domain you are modeling.

3.4 BUSINESS RULES

From a database point of view, the collection of data becomes meaningful only when it reflects properly defined business rules. A business rule is a brief, precise, and unambiguous description of a policy, procedure, or principle within a specific organization. Business rules, derived from a detailed description of an organization's operations, help to create and enforce actions within that organization's environment. Business rules must be rendered in writing and updated to reflect any change in the organization's operational environment. Properly written business rules are used to define entities, attributes, relationships, and constraints.

To be effective, business rules must be easy to understand and widely disseminated, to ensure that every person in the organization shares a common interpretation of the rules. Business rules describe, in simple language, the main and distinguishing characteristics of the data as viewed by the company. Examples of business rules are as follows:

- A customer may generate many invoices.
- An invoice is generated by only one customer.
- A training session cannot be scheduled for fewer than 10 employees or for more than 30 employees.

Note that those business rules establish entities, relationships, and constraints. For example, the first two business rules establish two entities (CUSTOMER and INVOICE) and a 1:M relationship between those two entities. The third business rule establishes a constraint (no fewer than 10 people and no more than 30 people), two entities (EMPLOYEE and TRAINING), and a relationship between EMPLOYEE and TRAINING.

3.4.1 Discovering Business Rules

The main sources of business rules are company managers, policy makers, department managers, and written documentation such as a company's procedures, standards, and operations manuals. A faster and more direct source of business rules is direct interviews with end users. Unfortunately, because perceptions differ, end users are sometimes a less reliable source when it comes to specifying business rules. Although end users are crucial contributors to the

development of business rules, it is better to verify end-user perceptions. Too often, interviews with several people who perform the same job yield very different perceptions of what the job components are.

The process of identifying and documenting business rules is essential to database design for several reasons:

- They help to standardize the company's view of data.
- They can be a communications tool between users and designers.
- They allow the designer to understand the nature, role, and scope of the data.
- They allow the designer to understand business processes.
- They allow the designer to develop appropriate relationship participation rules and constraints and to create an accurate data model.

3.4.2 Translating Business Rules into Data Model Components

Business rules helps in for the proper identification of entities, attributes, relationships, and constraints. In the real world, names are used to identify objects. As a general rule, a noun in a business rule will translate into an entity in the model, and a verb (active or passive) associating nouns will translate into a relationship among the entities. For example, the business rule “a customer may generate many invoices” contains two nouns (customer and invoices) and a verb (generate) that associates the nouns. From this business rule, you could deduce that:

- Customer and invoice are objects of interest for the environment and should be represented by their respective entities.
- There is a “generate” relationship between customer and invoice.

To properly identify the type of relationship, you should consider that relationships are bidirectional; that is, they go both ways. For example, the business rule “a customer may generate many invoices” is complemented by the business rule “an invoice is generated by only one customer.” In that case, the relationship is one-to-many (1:M). Customer is the “1” side, and invoice is the “many” side.

As a general rule, to properly identify the relationship type, you should ask two questions:

- How many instances of B are related to one instance of A?
- How many instances of A are related to one instance of B?

For example, you can assess the relationship between student and class by asking two questions:

- In how many classes can one student enroll? Answer: many classes.

- How many students can enroll in one class? Answer: many students.

Therefore, the relationship between student and class is many-to-many (M:N).

3.4.3. Naming Conventions

During the translation of business rules to data model components, you identify entities, attributes, relationships, and constraints. This identification process includes naming the object in a way that makes the object unique and distinguishable from other objects in the problem domain. Therefore, it is important that you pay special attention to how you name the objects you are discovering.

Entity names should be descriptive of the objects in the business environment, and use terminology that is familiar to the users. An attribute name should also be descriptive of the data represented by that attribute. It is also a good practice to prefix the name of an attribute with the name of the entity (or an abbreviation of the entity name) in which it occurs. For example, in the CUSTOMER entity, the customer's credit limit may be called CUS_CREDIT_LIMIT. The CUS indicates that the attribute is descriptive of the CUSTOMER entity, while CREDIT_LIMIT makes it easy to recognize the data that will be contained in the attribute.

3.5 THE EVOLUTION OF DATA MODELS

This section gives an overview of the major data models in roughly chronological order. You will discover that many of the “new” database concepts and structures bear a remarkable resemblance to some of the “old” data model concepts and structures.

3.5.1 Hierarchical and Network Models

Two older, historically important data models, now known as legacy data models, are the network and hierarchical models. The network model represents data as record types and also represents a limited type of 1:N relationship, called a set type. A 1:N, or one-to-many, relationship relates one instance of a record to many record instances using some pointer linking mechanism in these models. The network model, also known as the CODASYL DBTG model, has an associated record-at-a-time language that must be embedded in a host programming language. The network DML was proposed in the 1971 Database Task Group (DBTG) Report as an extension of the COBOL language.

The hierarchical model represents data as hierarchical tree structures. Each hierarchy represents a number of related records. There is no standard language for the hierarchical model. A popular hierarchical DML is DL/1 of the IMS system. It dominated the DBMS market for over 20 years between 1965 and 1985. Its DML, called DL/1, was a de facto industry standard for a long time.

3.5.2 Relational Model

The first commercial implementations of the relational model became available in the early 1980s, such as the SQL/DS system on the MVS operating system by IBM and the Oracle DBMS. Since then, the model has been implemented in a large number of commercial systems, as well as a number of open source systems. Current popular commercial relational DBMSs (RDBMSs) include DB2 (from IBM), Oracle (from Oracle), Sybase DBMS (now from SAP), and SQLServer and Microsoft Access (from Microsoft). In addition, several open source systems, such as MySQL and PostgreSQL are available.

The relational model represents the database as a collection of relations. Informally, each relation resembles a table of values or, to some extent, a flat file of records. It is called a flat file because each record has a simple linear or flat structure. When a relation is thought of as a table of values, each row in the table represents a collection of related data values. A row represents a fact that typically corresponds to a real-world entity or relationship. The table name and column names are used to help to interpret the meaning of the values in each row.

In the formal relational model terminology, a row is called a tuple, a column header is called an attribute, and the table is called a relation. The data type describing the types of values that can appear in each column is represented by a domain of possible values.

3.5.3 The Entity Relationship Model

Entity–Relationship (ER) model is a popular high-level conceptual data model. This model and its variations are frequently used for the conceptual design of database applications and many database design tools employ its concepts. ER models are normally represented in an entity relationship diagram (ERD), which uses graphical representations to model database components.

Peter Chen first introduced the ER data model in 1976; it was the graphical representation of entities and their relationships in a database structure that quickly became popular because it

complemented the relational data model concepts. The relational data model and ERM(Entity–Relationship Model) combined to provide the foundation for tightly structured database design.

The ER model is based on the following components:

Entity: An entity is defined as anything about which data are to be collected and stored. An entity is represented in the ERD by a rectangle, also known as an entity box. The name of the entity, a noun, is written in the center of the rectangle. The entity name is generally written in capital letters and is written in the singular form: PAINTER rather than PAINTERS, and EMPLOYEE rather than EMPLOYEES. Usually, when applying the ERD to the relational model, an entity is mapped to a relational table. Each row in the relational table is known as an entity instance or entity occurrence in the ER model. Each entity is described by a set of attributes that describes particular characteristics of the entity. For example, the entity EMPLOYEE will have attributes such as an Employee ID, a last name, and a first name.

Relationships: Relationships describe associations among data. Most relationships describe associations between two entities. When the basic data model components were introduced, three types of relationships among data were illustrated: one-to-many (1:M), many-to-many (M:N), and one-to-one (1:1). The ER model uses the term connectivity to label the relationship types. The name of the relationship is usually an active or passive verb. For example, a PAINTER paints many PAINTINGs; an EMPLOYEE learns many SKILLs; an EMPLOYEE manages a STORE.

Example ER notations:

A One-to-Many (1:M) Relationship: a PAINTER can paint many PAINTINGs; each PAINTING is painted by one PAINTER.



A Many-to-Many (M:N) Relationship: an EMPLOYEE can learn many SKILLs; each SKILL can be learned by many EMPLOYEEs.



A One-to-One (1:1) Relationship: an EMPLOYEE manages one STORE; each STORE is managed by one EMPLOYEE.



3.5.4. The Object-Oriented (OO) Model

Increasingly complex real-world problems demonstrated a need for a data model that more closely represented the real world. In the object-oriented data model (OODM), both data and their relationships are contained in a single structure known as an object. In turn, the OODM is the basis for the object-oriented database management system (OODBMS)

The OO data model is based on the following components:

- An object is an abstraction of a real-world entity. In general terms, an object may be considered equivalent to an ER model's entity. More precisely, an object represents only one occurrence of an entity.
- Attributes describe the properties of an object. For example, a PERSON object includes the attributes Name, Social Security Number, and Date of Birth.
- Objects that share similar characteristics are grouped in classes. A class is a collection of similar objects with shared structure (attributes) and behavior (methods). In a general sense, a class resembles the ER model's entity set. However, a class is different from an entity set in that it contains a set of procedures known as methods. A class's method represents a real-world action such as finding a selected PERSON's name, changing a PERSON's name, or printing a PERSON's address. In other words, methods are the equivalent of procedures in traditional programming languages. In OO terms, methods define an object's behavior.
- Classes are organized in a class hierarchy. The class hierarchy resembles an upside-down tree in which each class has only one parent. For example, the CUSTOMER class and the EMPLOYEE class share a parent PERSON class.
- Inheritance is the ability of an object within the class hierarchy to inherit the attributes and methods of the classes above it. For example, two classes, CUSTOMER and EMPLOYEE, can be created as subclasses from the class PERSON. In this case, CUSTOMER and EMPLOYEE will inherit all attributes and methods from PERSON.

Object-oriented data models are typically depicted using Unified Modeling Language (UML) class diagrams. Unified Modeling Language (UML) is a language based on OO concepts that describes a set of diagrams and symbols that can be used to graphically model a system. UML class diagrams are used to represent data and their relationships within the larger UML object-oriented system's modeling language.

3.5.5 Newer Data Models: Object/Relational and XML

Facing the demand to support more complex data representations, the relational model's main vendors evolved the model further and created the extended relational data model (ERDM). The ERDM adds many of the OO model's features within the inherently simpler relational database structure. The ERDM gave birth to a new generation of relational databases supporting OO features such as objects (encapsulated data and methods), extensible data types based on classes and inheritance. That's why a DBMS based on the ERDM is often described as an object/relational database management system (O/R DBMS).

The use of complex objects received a boost with the Internet revolution. When organizations integrated their business models with the Internet, they realized the potential of the Internet to access, distribute and exchange critical business information. This resulted in the widespread adoption of the Internet as a business communication tool. It is in this environment that Extensible Markup Language (XML) emerged as the de facto standard for the efficient and effective exchange of structured, semi structured and unstructured data. Organizations using XML data soon realized there was a need to manage the large amounts of unstructured data such as word-processing documents, Web pages, e-mails, diagrams, etc., found in most of today's organizations. To address this need, XML databases emerged to manage unstructured data within a native XML format.

3.6 Degrees of Data Abstraction

In the early 1970s, the American National Standards Institute (ANSI) Standards Planning and Requirements Committee (SPARC) defined a framework for data modeling based on degrees of data abstraction. The ANSI/SPARC architecture (as it is often referred to) defines three levels of data abstraction: external, conceptual, and internal. You can use this framework to better understand database models, as shown in Figure 3.6.

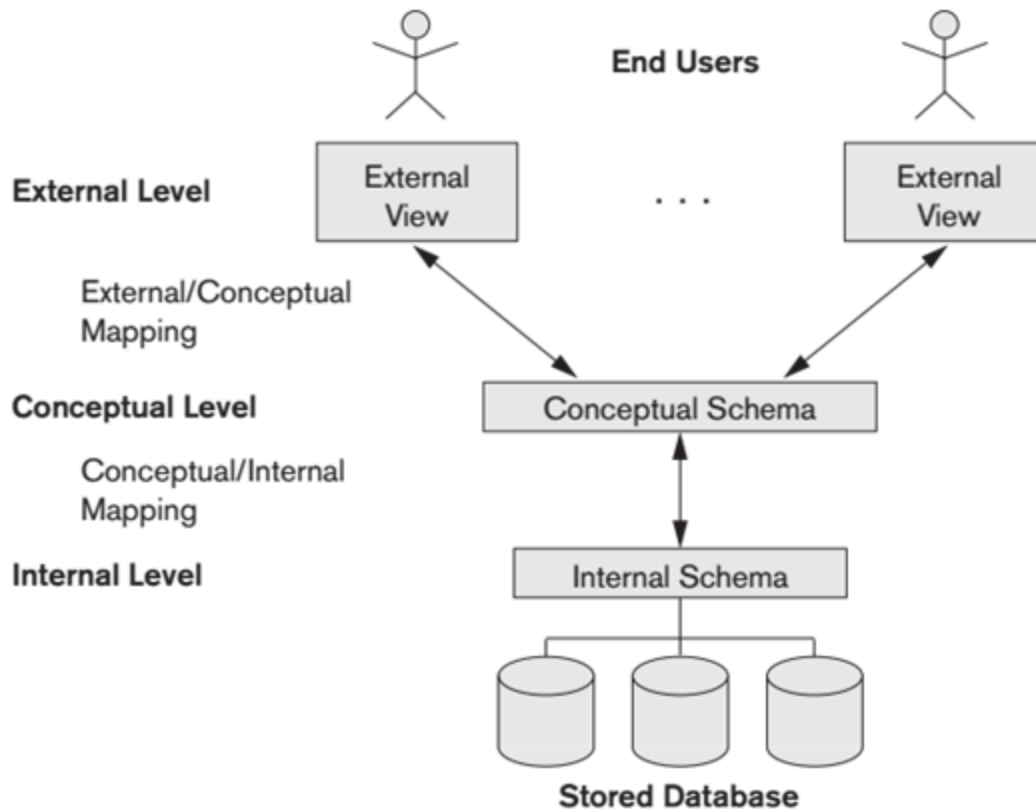


Fig. 3.6 Data abstraction levels

Fig 3.6 illustrates a three-schema architecture whose purpose is to separate the user applications from the physical database. In this architecture, schemas can be defined at the following three levels:

1. 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.
2. The conceptual level has a conceptual schema, which describes the structure of the whole database for a community of users. 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 of conceptual schema is often based on a conceptual schema design in a high-level data model.
3. 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. As in the previous level, each external schema is typically implemented using a representational data model, possibly based on an external schema design in a high-level conceptual data model.

3.6.1 Data Independence

The three-schema architecture can be used to further explain the concept of data independence, which can be defined as the capacity to change the schema at one level of a database system without having to change the schema at the next higher level. We can define two types of data independence:

1. Logical data independence is the capacity to change the conceptual schema without having to change external schemas or application programs. We may change the conceptual schema to expand the database (by adding a record type or data item), to change constraints, or to reduce the database (by removing a record type or data item). In the last case, external schemas that refer only to the remaining data should not be affected. Only the view definition and the mappings need to be changed in a DBMS that supports logical data independence. After the conceptual schema undergoes a logical reorganization, application programs that reference the external schema constructs must work as before.
2. Physical data independence is the capacity to change the internal schema without having to change the conceptual schema. Hence, the external schemas need not be changed as well. Changes to the internal schema may be needed because some physical files were reorganized—for example, by creating additional access structures—to improve the performance of retrieval or update. If the same data as before remains in the database, we should not have to change the conceptual schema.

3.7 CHECK YOUR PROGRESS

1. A data model is usually graphical. (T/F)
2. A simple representation of complex real-world data structures is called -----
3. When you can change the internal model without affecting the conceptual model, it is said to have _____ independence.

4. In relational schema, each tuple is divided in fields called
(a) Relations (b) Domains
(c) Queries (d) All of the above
5. The relational model represents the database as a collection of _____.
(a) files (b) relations (c) applications (d) systems
6. What are the advantages of hierarchical model?
7. What is Network model?
8. What is tuple?
9. Name some commercially available relational database systems.
10. Give an example of relational model.

Answers to check your progress:

1. T
2. data model
3. logical
4. b
5. b
6. The model allows you to easily add and delete new information. Data at the top of the hierarchy can be accessed quickly. This model works well with linear data storage mediums such as tapes.
7. The network model is a database model conceived as a flexible way of representing objects and their relationships. Its distinguishing feature is that the schema, viewed as a graph in which object types are nodes and relationship types are arcs, is not restricted to being a hierarchy
8. A tuple is simply a row contained in a table in the table space.
9. Oracle, Sybase
10. student table with 3 columns ID, Name and Age

3.8SUMMARY

A data model is an abstraction of a complex real-world data environment. Database

designers use data models to communicate with applications programmers and end users. The basic data-modeling components are entities, attributes, relationships, and constraints. Business rules are used to identify and define the basic modeling components within a specific real-world environment.

The hierarchical and network data models were early data models that are no longer used. The relational model is the current database implementation standard. In the relational model, the end user perceives the data as being stored in tables. Tables are related to each other by means of common values in common attributes. The entity relationship (ER) model is a popular graphical tool for data modeling that complements the relational model. The ER model allows database designers to visually present different views of the data—as seen by database designers, programmers, and end users—and to integrate the data into a common framework.

The object-oriented data model (OODM) uses objects as the basic modeling structure. An object resembles an entity in that it includes the facts that define it. But unlike an entity, the object also includes information about relationships between the facts, as well as relationships with other objects, thus giving its data more meaning. The relational model has adopted many object-oriented (OO) extensions to become the extended relational data model (ERDM). Object/relational database management systems (O/R DBMS) were developed to implement the ERDM. At this point, the OODM is largely used in specialized engineering and scientific applications, while the ERDM is primarily used in business applications.

Data-modeling requirements are a function of different data views (global vs. local) and the level of data abstraction. The American National Standards Institute Standards Planning and Requirements Committee (ANSI/SPARC) describes three levels of data abstraction: external, conceptual, and internal.

3.9 KEYWORDS

- **Attribute**- An attribute is a characteristic of an entity.
- **Business rule** – A business rule is a brief, precise, and unambiguous description of a policy, procedure, or principle within a specific organization.
- **Data model** - A data model is a relatively simple representation, usually graphical, of more complex real-world data structures.
- **Entity**- which is a thing or object in the real world with an independent existence

3.10 QUESTION FOR SELF STUDY

1. Discuss the importance of data modeling.
2. Describe the basic features of the relational data model and discuss their importance to the end user and the designer.
3. Explain how the entity relationship (ER) model helped produce a more structured relational database design environment.
4. In terms of data and structural independence, compare file system data management with the five data models discussed in this unit.
5. What is meant by Data Independence? State its importance in database technology
6. What is a relational diagram? Give an example.
7. What is logical independence? What is physical independence?

3.11 REFERENCES

1. Coronel, C., & Morris, S. (2016). Database systems: design, implementation, & management. Cengage Learning.
2. Gupta, S. B., & Mittal, A. (2009). Introduction to database management system. Laxmi Publications, Ltd..
3. Ramakrishnan, R., & Gehrke, J. (2000). Database management systems. McGraw-Hill.

UNIT-4: THE RELATIONAL DATABASE MODEL

STRUCTURE

- 4.0 Objectives
- 4.1 Logical View of Data
- 4.2 Keys
- 4.3 Integrity Rules
- 4.4 Relational Algebra
- 4.5 The Data Dictionary and the System Catalog
- 4.6 Relationships within the Relational Database
- 4.7 Data Redundancy Revisited
- 4.8 Indexes
- 4.9 Codd's Relational Database Rules
- 4.10 Check your progress
- 4.11 Summary
- 4.12 Keywords
- 4.13 Questions for self-study
- 4.14 References

4.0 OBJECTIVES

After studying this unit, you will be able to:

- ✓ Learn that the relational database model offers a logical view of data
- ✓ Explain about the relational model's basic component: relations
- ✓ Explain that relations are logical constructs composed of rows (tuples) and columns (attributes)
- ✓ Describe that relations are implemented as tables in a relational DBMS
- ✓ Discuss about relational database operators, the data dictionary, and the system catalog
- ✓ Explain how data redundancy is handled in the relational database model
- ✓ Explain why indexing is important.

4.1 LOGICAL VIEW OF DATA

The practical significance of taking the logical view of data is that it serves as a reminder of the simple file concept of data storage. Although the use of a table, quite unlike that of a file, has the advantages of structural and data independence, a table does resemble a file from a conceptual point of view. Because you can think of related records as being stored in independent tables, the relational database model is much easier to understand than the hierarchical and network models. Logical simplicity tends to yield simple and effective database design methodologies.

4.1.1 Tables and Their Characteristics

The relational data model represents a database as a collection of tables, where each table can be stored as a separate file. The logical view of the relational database is facilitated by the creation of data relationships based on a logical construct known as a relation. Because a relation is a mathematical construct, end users find it much easier to think of a relation as a table. A table is perceived as a two-dimensional structure composed of rows and columns. A table is also called a relation because the relational model's creator, E. F. Codd, used the term relation as a synonym for table. You can think of a table as a persistent representation of a logical relation, that is, a relation whose contents can be permanently saved for future use. As far as the table's user is concerned, a table contains a group of related entity occurrences, that is, an entity set. For example, a STUDENT table contains a collection of entity occurrences, each representing a student. For that reason, the terms entity set and table are often used interchangeably.

The characteristics of a relational table are summarized in Table 4.1

Table 4.1: The characteristics of a relational table

1	A table is perceived as a two-dimensional structure composed of rows and columns.
2	Each table row (tuple) represents a single entity occurrence within the entity set.
3	Each table column represents an attribute, and each column has a distinct name.
4	Each row/column intersection represents a single data value.
5	All values in a column must conform to the same data format.
6	Each column has a specific range of values known as the attribute domain.
7	The order of the rows and columns is immaterial to the DBMS.
8	Each table must have an attribute or a combination of attributes that uniquely identifies each row.

An example table is shown below

EID	Name	Salary	Department
1	Amit	6,000	Accounts
2	Sumit	10,000	Computer
3	Lalit	15,000	Accounts
4	Deepak	9,000	Electrical
5	Sandeep	4,000	Civil

Table 4.1 Above table is named as Employee, which has the following attributes EID, Name, Salary, Department.

Each row in the above table is an entity within the entity set. Each column represents an attribute. The column's range of permissible values is known as its domain and each table must have a primary key. In general terms, the primary key (PK) is an attribute (or a combination of attributes) that uniquely identifies any given row. In the above example EID is the primary key.

4.2 KEYS

An important constraint on the entities of an entity type is the key or uniqueness constraint on attributes. An entity type usually has one or more attributes 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 Name attribute is a key of the COMPANY entity type because no two companies are allowed to have the same name. 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 composite attribute and designate it as a key 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. Superfluous attributes must not be included in a key. In ER diagrammatic notation, each key attribute has its name underlined inside the oval, as illustrated in Figure 4.2.

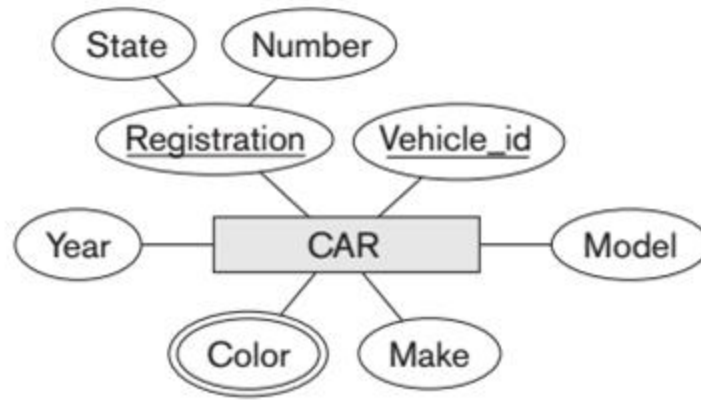


Fig 4.2. Entity CAR and its attributes

Specifying that an attribute is a key of an entity type means that the 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 entity set; rather, it is a constraint on any entity set of the entity type at any point in time. This key constraint is derived from the constraints of the mini world that the database represents.

Some entity types have more than one key attribute. For example, each of the *Vehicle_id* and *Registration* attributes of the entity type *CAR* (Figure 4.2) is a key in its own right. The *Registration* attribute is an example of a composite key formed from two simple component attributes, *State* and *Number*, 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.

The key's role is based on a concept known as determination. In the context of a database table, the statement "A determines B" indicates that if you know the value of attribute A, you can look up (determine) the value of attribute B. The principle of determination is very important because it is used in the definition of a central relational database concept known as functional dependence. A functional dependency is a constraint between two sets of attributes from the database. Suppose that our relational database schema has n attributes A_1, A_2, \dots, A_n ; let us think of the whole database as being described by a single universal relation schema $R = \{A_1, A_2, \dots, A_n\}$. We do not imply that we will actually store the database as a single universal table; we use this concept only in developing the formal theory of data dependencies.

A functional dependency, denoted by $X \rightarrow Y$, between two sets of attributes X and Y that are subsets of R specifies a constraint on the possible tuples that can form a relation state r of R . The constraint is that, for any two tuples t_1 and t_2 in r that have $t_1[X] = t_2[X]$, they must also have $t_1[Y] = t_2[Y]$.

This means that the values of the Y component of a tuple in r depend on, or are determined by, the values of the X component; alternatively, the values of the X component of a tuple uniquely (or functionally) determine the values of the Y component. We also say that there is a functional dependency from X to Y , or that Y is functionally dependent on X . A functional dependency

$X \rightarrow Y$ is a full functional dependency if removal of any attribute A from X means that the dependency does not hold anymore; that is, for any attribute $A \in X$, $(X - \{A\})$ does not functionally determine Y . A functional dependency $X \rightarrow Y$ is a partial dependency if some attribute $A \in X$ can be removed from X and the dependency still holds; that is, for some $A \in X$, $(X - \{A\}) \rightarrow Y$.

Within the broad key classification, several specialized keys can be defined. For example, a superkey is any key that uniquely identifies each row. Formally A superkey of a relation schema $R = \{A_1, A_2, \dots, A_n\}$ is a set of attributes $S \subseteq R$ with the property that no two tuples t_1 and t_2 in any legal relation state r of R will have $t_1[S] = t_2[S]$. A key K is a superkey with the additional property that removal of any attribute from K will cause K not to be a superkey anymore.

If a relation schema has more than one key, each is called a candidate key. One of the candidate keys is arbitrarily designated to be the primary key, and the others are called secondary keys. In a practical relational database, each relation schema must have a primary key. If no candidate key is known for a relation, the entire relation can be treated as a default super key. Within a table, each primary key value must be unique to ensure that each row is uniquely identified by the primary key. In that case, the table is said to exhibit entity integrity. To maintain entity integrity, a null (that is, no data entry at all) is not permitted in the primary key.

In some cases, a particular entity may not have an applicable value for an attribute. For example, the `Apartment_number` 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 `College_degrees` attribute applies only to people 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 Apartment_number attribute, and a person with no college degree would have NULL for College_degrees. 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 number of ‘Rama’ in the Employee table it will be null. 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 instance, 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 Home_phone attribute of a person is NULL.

A relational database can also be represented by a relational schema. A relational schema is a textual representation of the database tables where each table is listed by its name followed by the list of its attributes in parentheses. The primary key attribute(s) is (are) underlined.

VENDOR (VEND_CODE, VEND_CONTACT, VEND_AREACODE, VEND_PHONE)

PRODUCT (PROD_CODE, PROD_DESCRIPT, PROD_PRICE, PROD_ON_HAND, VEND_CODE)

For example, consider the following relational schema

A foreign key (FK) is an attribute whose values match the primary key values in the related table. The VEND_CODE is the primary key in the VENDOR table, and it occurs as a foreign key in the PRODUCT table.

Finally, a secondary key is defined as a key that is used strictly for data retrieval purposes. Suppose customer data are stored in a CUSTOMER table in which the customer number is the primary key. Do you suppose that most customers will remember their numbers? Data retrieval for a customer can be facilitated when the customer’s last name and phone number are used. In that case, the primary key is the customer number; the secondary key is the combination of the customer’s last name and phone number. Keep in mind that a secondary key does not necessarily yield a unique outcome. For example, a customer’s last name and home telephone number could easily yield several matches where one family lives together and shares a phone line.

4.3 INTEGRITY RULES

Relational database integrity rules are very important to good database design. Many (but by no means all) RDBMSs enforce integrity rules automatically. However, it is much safer to make sure that your application design conforms to the entity and referential integrity rules mentioned below.

The entity integrity constraint states that no primary key value can be NULL. This is because the primary key value is used to identify individual tuples in a relation. Having NULL values for the primary key implies that we cannot identify some tuples. For example, if two or more tuples had NULL for their primary keys, we may not be able to distinguish them if we try to reference them from other relations.

Key constraints and entity integrity constraints are specified on individual relations. The referential integrity constraint is specified between two relations and is used to maintain the consistency among tuples in the two relations. Informally, the referential integrity constraint states that a tuple in one relation that refers to another relation must refer to an existing tuple in that relation. Below examples illustrate entity integrity and referential integrity rules

EID	Name	Salary	Department
1	Amit	6,000	Accounts
2	Sumit	10,000	Computer
3	Lalit	15,000	Accounts
4	Deepak	9,000	Electrical
5	Sandeep	4,000	Civil

This is not allowed because EID is a primary key

Fig. 4.3. Entity integrity

EID	Name	Salary	Dept-ID
1	Amit	6,000	1A
2	Sumit	10,000	2C
3	Lalit	15,000	4F
4	Deepak	9,000	3F
5	Sandeep	4,000	—

Dept-ID	Dept-Name
1A	Accounts
2C	Computer
3E	Electrical
4C	Civil

Null value is allowed

This is not allowed because Dept-ID is a foreign key and the value 4F is not present in attribute Dept-ID of relation Department.

Fig.4.4. Referential integrity

4.4 RELATIONAL ALGEBRA

A data model must include a set of operations to manipulate the database, in addition to the data model's concepts for defining the database's structure and constraints. The basic set of operations for the formal relational model is the relational algebra. These operations enable a user to specify basic retrieval requests as relational algebra expressions. The result of a retrieval query is a new relation. The algebra operations thus produce new relations, which can be further manipulated using operations of the same algebra. A sequence of relational algebra operations forms a relational algebra expression, whose result will also be a relation that represents the result of a database query (or retrieval request).

The relational algebra is very important for several reasons. First, it provides a formal foundation for relational model operations. Second, and perhaps more important, it is used as a basis for implementing and optimizing queries in the query processing and optimization modules that are integral parts of relational database management systems (RDBMSs) Third, some of its concepts are incorporated into the SQL standard query language for RDBMSs. Although most commercial RDBMSs in use today do not provide user interfaces for relational algebra queries, the core operations and functions in the internal modules of most relational systems are based on relational algebra operations.

The relational algebra is often considered to be an integral part of the relational data model. Its operations can be divided into two groups. One group includes set operations from mathematical set theory; these are applicable because each relation is defined to be a set of tuples in the formal relational model. Set operations include UNION, INTERSECTION, SET DIFFERENCE, and CARTESIAN PRODUCT (also known as CROSS PRODUCT). The other group consists of operations developed specifically for relational databases—these include SELECT, PROJECT, and JOIN, among others.

We discuss the different operators in detail, with examples, in subsequent sections. All the examples are based on the EMPLOYEE-STUDENT database shown in Figure 4.5. This database contain two Tables EMPLOYEE and STUDENT and the relationship is that an employee can also a student and vice versa.

Employee			Student		
EID	Name	Salary	SID	Name	Fees
1E	John	10,000	1S	Smith	1,000
2E	Ramesh	5,000	2S	Vijay	950
3E	Smith	8,000	3S	Gaurav	2,000
4E	Jack	6,000	4S	Nile	1,500
5E	Nile	15,000	5S	John	950

Fig.4.5. Employee and Student relations

4.4.1 Unary Relational Operations: SELECT and PROJECT

The selection operation is a unary operation. This is used to find horizontal subset of relation or tuples of relation. It is denoted by sigma (σ).

Ex. If you want all the employees having salary more than 9,000 from relation Employee. The query, is

$$\sigma_{\text{salary} > 9,000} (\text{Employee})$$

The result is shown in Figure 4.6.

EID	Name	Salary
1E	John	10,000
5E	Nile	15,000

Fig.4.6. Result of select operation

The projection operation is a unary operation which applies only on a single relation at a time. Project operation is used to select vertical subset of relation (i.e., columns of table). It is denoted by pi (π).

Consider Figure 4.5. If you want all the names of employees and their salary from relation employee. Then query is

$$\pi_{\text{name, salary}} (\text{Employee})$$

The result is shown in Figure 4.7.

Name	Salary
John	10,000
Ramesh	5,000
Smith	8,000
Jack	6,000
Nile	15,000

Fig.4.7. Result of project operation

4.4.2. Relational Algebra Operations from Set Theory

4.4.2.1. The UNION, INTERSECTION, and MINUS Operations

The union operation is a binary operation that is used to find union of relations. Here relations are considered as sets. So, duplicate values are eliminated. It is denoted by (\cup).

Conditions for union operation: There are two necessary conditions for union operation.

- (i) Both the relations have same number of attributes.
- (ii) Data types of their corresponding attributes must be same.

Two relations are said to be union compatible if they follow the above two conditions.

Ex. If you want to find the names of all employees and names of all students together then the query is

$$\pi_{\text{Name}} (\text{Employee}) \cup \pi_{\text{Name}} (\text{Student})$$

The result is shown in Figure 4.8

Name
John
Ramesh
Smith
Jack
Nile
Vijay
Gaurav

Fig.4.8. Result of set union

Set intersection is used to find common tuples between two relations. It is denoted by (\cap). If you want to find all the employees from Relation Employee those are also students. Rules of set union operations are also applicable here. Then the query, is

$$\pi_{\text{Name}} (\text{Employee}) \cap \pi_{\text{Name}} (\text{Student})$$

The result is shown in Figure. 4.9

Name
John
Smith
Nile

Fig.4.9. Result of set intersection operational

Set-difference operation is a binary operation which is used to find tuples that are present in one relation but not in other relation. It is denoted by ($-$). It removes the common tuples of two relations and produce a new relation having rest of the tuples of first relation.

Ex. If you want the names of those employees that are not students, then the query, is

$$\pi_{\text{Name}} (\text{Employee}) - \pi_{\text{Name}} (\text{Student})$$

The result is shown in Figure 4.10

Name
Ramesh
Jack

Fig.4.10. Result of set difference operation

4.4.3. The CARTESIAN PRODUCT (CROSS PRODUCT) Operation

Let us look at the CARTESIAN PRODUCT operation—also known as CROSS PRODUCT or CROSS JOIN—which is denoted by \times . This is also a binary set operation, but the relations on which it is applied do not have to be union compatible. In its binary form, this set operation produces a new element by combining every member (tuple) from one relation (set) with every member (tuple) from the other relation (set). In general, the result of $R(A_1, A_2, \dots, A_n) \times S(B_1, B_2, \dots, B_m)$ is a relation Q with degree $n + m$ attributes $Q(A_1, A_2, \dots, A_n, B_1, B_2, \dots, B_m)$, in that order. The resulting relation Q has one tuple for each combination of tuples—one from R

and one from S. Hence, if R has nR tuples (denoted as $|R| = nR$), and S has nS tuples, then $R \times S$ will have $nR * nS$ tuples.

4.4.4. Binary Relational Operations: JOIN and DIVISION

Natural join is used to join two relations having any number of attributes. It is denoted by symbol \bowtie . Join is used to combine related tuples from two relations into single “longer” tuples. This operation is very important for any relational database with more than a single relation because it allows us to process relationships among relations. data-type. Consider the relations in Figure 4.11.

Employee			
EID	Name	Salary	Dept-ID
1	Amit	5,000	10
2	Sachin	8,000	20
3	Vinay	2,000	20
4	Vivek	6,000	10

Department	
Dept_ID	Dept_Name
10	Sales
20	Purchase

Ex. Find the names of all employees from relation employee with their respective department names. The query is

$$\pi_{\text{Name, Dept_name}} \left[\begin{array}{c} \text{Employee} \bowtie \text{Department} \\ \text{Employee} \cdot \text{EID} = \text{Department} \cdot \text{Dept_ID} \end{array} \right]$$

Outer Join is an extension of natural join operations. It deals with the missing information caused by natural join operation.

There are three types of outer joins are:

- Left outer join: It is used to take all tuples of relation that are on the left side whether they are matching with tuples of right side relation or not.
- Right outer join: It is used to take all tuples of relation that are on the right side whether they are matching with tuples of left side relation or not.
- Full outer join: It is used to take all tuples from left and right relation whether they match with each other or did not match.

The DIVISION operation, denoted by \div , is useful for a special kind of query that sometimes occurs in database applications. In general, the DIVISION operation is applied to two relations $R(Z) \div S(X)$, where the attributes of S are a subset of the attributes of R ; that is, $X \subseteq Z$. Let Y be the set of attributes of R that are not attributes of S ; that is, $Y = Z - X$ (and hence $Z = X \cup Y$). The result of DIVISION is a relation $T(Y)$ that includes a tuple t if tuples tR appear in R with $tR [Y] = t$, and with $tR [X] = tS$ for every tuple tS in S . This means that, for a tuple t to appear in the result T of the DIVISION, the values in t must appear in R in combination with every tuple in S . Note that in the formulation of the DIVISION operation, the tuples in the denominator relation S restrict the numerator relation R by selecting those tuples in the result that match all values present in the denominator.

For instance if the DIVIDE operation uses one single-column table (e.g., column “a”) as the divisor and one 2-column table (i.e., columns “a” and “b”) as the dividend. The tables must have a common column (e.g., column “a”). The output of the DIVIDE operation is a single column with the values of column “a” from the dividend table rows where the value of the common column (i.e., column “a”) in both tables matches.

4.5 THE DATA DICTIONARY AND THE SYSTEM CATALOG

The data dictionary provides a detailed description of all tables found within the user/designer-created database. Thus, the data dictionary contains at least all of the attribute names and characteristics for each table in the system. In short, the data dictionary contains metadata: data about data.

The data dictionary gives the human view of the entities, attributes, and relationships. The purpose of the data dictionary is to ensure that all members of database design and implementation teams use the same table and attribute names and characteristics. The DBMS’s internally stored data dictionary contains additional information about relationship types, entity and referential integrity checks and enforcement, and index types and components. This additional information is generated during the database implementation stage. The data dictionary is sometimes described as “the database designer’s database” because it records the design decisions about tables and their structures.

Like the data dictionary, the system catalog contains metadata. The system catalog can be described as a detailed system data dictionary that describes all objects within the database, including data about table names, the table's creator and creation date, the number of columns in each table, the data type corresponding to each column, index filenames, index creators, authorized users, and access privileges. Because the system catalog contains all required data dictionary information, the terms system catalog and data dictionary are often used interchangeably. In fact, current relational database software generally provides only a system catalog, from which the designer's data dictionary information may be derived. The system catalog is actually a system-created database whose tables store the user/designer-created database characteristics and contents. Therefore, the system catalog tables can be queried just like any user/designer-created table.

4.6 Relationships within the Relational Database

Relationship types usually have certain constraints that limit the possible combinations of entities that may participate in the corresponding relationship set. These constraints are determined from the mini world situation that the relationships represent. For example, if the company has a rule that each employee must work for exactly one department, then we would like to describe this constraint in the schema.

4.6.1. Cardinality Ratios for Binary Relationships.

The cardinality ratio for a binary relationship specifies the maximum number of relationship instances that an entity can participate in. For example, in the WORKS_FOR binary relationship type, DEPARTMENT:EMPLOYEE is of cardinality ratio 1:N, meaning that each department can be related to (that is, employs) any number of employees (N), but an employee can be related to (work for) at most one department (1). This means that for this particular relationship type WORKS_FOR, a particular department entity can be related to any number of employees (N indicates there is no maximum number). On the other hand, an employee can be related to a maximum of one department. The possible cardinality ratios for binary relationship types are 1:1, 1:N, N:1, and M:N.

An example of a 1:1 binary relationship is MANAGES (Figure 4.6.1), which relates a department entity to the employee who manages that department. This represents the miniworld constraints that—at any point in time—an employee can manage at most one department and a

department can have at most one manager. The relationship type WORKS_ON (Figure 4.6.2) is of cardinality ratio M:N, because the mini world rule is that an employee can work on several projects and a project can have several employees.

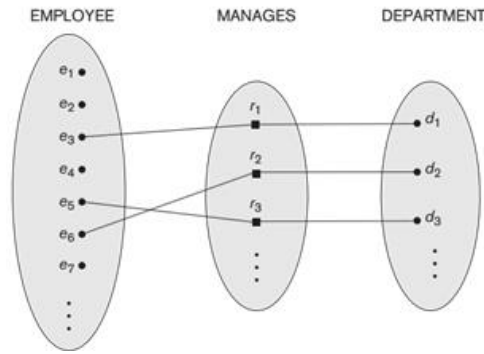


Fig.4.12. A 1:1 relationship, MANAGES

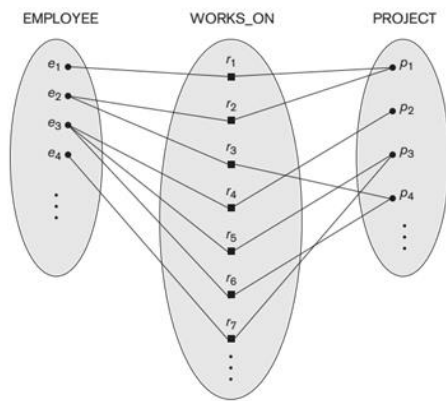


Fig:4.13. An M:N relationship WORKS_ON

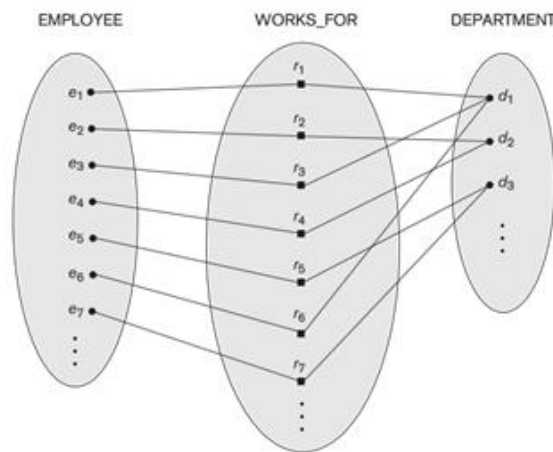


Fig: 4.14. An N:1 relation Works_For

4.7 DATA REDUNDANCY REVISITED

Data redundancy exists when the same data are stored unnecessarily at different places

Uncontrolled data redundancy sets the stage for:

- **Poor data security:** Having multiple copies of data increases the chances for a copy of the data to be susceptible to unauthorized access.
- **Data inconsistency:** Data inconsistency exists when different and conflicting versions of the same data appear in different places. Data entry errors are more likely to occur when complex entries (such as 10-digit phone numbers) are made in several different files and/or recur frequently in one or more files. A data entry error such as an incorrectly spelled name or an incorrect phone number yields the same kind of data integrity problems.
- **Data anomalies:** The dictionary defines anomaly as “an abnormality.” Ideally, a field value change should be made in only a single place. Data redundancy, however, fosters an abnormal condition by forcing field value changes in many different locations.

The proper use of foreign keys is crucial to controlling data redundancy. Although the use of foreign keys does not totally eliminate data redundancies, because the foreign key values can be repeated many times, the proper use of foreign keys minimizes data redundancies, thus minimizing the chance that destructive data anomalies will develop.

The real test of redundancy is not how many copies of a given attribute are stored, but whether the elimination of an attribute will eliminate information. Therefore, if you delete an attribute and the original information can still be generated through relational algebra, the inclusion of that attribute would be redundant. Given that view of redundancy, proper foreign keys are clearly not redundant in spite of their multiple occurrences in a table.

However, even when you use this less restrictive view of redundancy, keep in mind that controlled redundancies are often designed as part of the system to ensure transaction speed and/or information requirements. Exclusive reliance on relational algebra to produce required information may lead to elegant designs that fail the test of practicality.

4.8 INDEXES

An index is an orderly arrangement used to logically access rows in a table. From a conceptual point of view, an index is composed of an index key and a set of pointers. The index key is, in effect, the index's reference point. More formally, an index is an ordered arrangement of keys and pointers. Each key points to the location of the data identified by the key. DBMSs use indexes for many different purposes. An index can be used to retrieve data more efficiently. Indexes can also be used by a DBMS to retrieve data ordered by a specific attribute or attributes. For example, creating an index on a customer's last name will allow you to retrieve the customer data alphabetically by the customer's last name. Indexes play an important role in DBMSs for the implementation of primary keys. When you define a table's primary key, the DBMS automatically creates a unique index on the primary key column(s) you declared.

4.9 CODD'S RELATIONAL DATABASE RULES

Dr. Codd's 12 Relational Database Rules

RULE	RULE NAME	DESCRIPTION
1	Information	All information in a relational database must be logically represented as column values in rows within tables.
2	Guaranteed Access	Every value in a table is guaranteed to be accessible through a combination of table name, primary key value, and column name.
3	Systematic Treatment of Nulls	Nulls must be represented and treated in a systematic way, independent of data type.
4	Dynamic Online Catalog Based on the Relational Model	The metadata must be stored and managed as ordinary data, that is, in tables within the database. Such data must be available to authorized users using the standard database relational language.
5	Comprehensive Data Sublanguage	The relational database may support many languages. However, it must support one well-defined, declarative language with support for data definition, view definition, data manipulation (interactive and by program), integrity constraints, authorization, and transaction management (begin, commit, and rollback)
6	View Updating	Any view that is theoretically updatable must be updatable through the system.
7	High-Level Insert,	The database must support set-level inserts, updates, and deletes.

	Update, and Delete	
8	Physical Data Independence	Application programs and ad hoc facilities are logically unaffected when physical access methods or storage structures are changed.
9	Logical Data Independence	Application programs and ad hoc facilities are logically unaffected when changes are made to the table structures that preserve the original table values (changing order of columns or inserting columns)
10	Integrity Independence	All relational integrity constraints must be definable in the relational language and stored in the system catalog, not at the application level
11	Distribution Independence	The end users and application programs are unaware and unaffected by the data location (distributed vs. local databases).
12	Non subversion	If the system supports low-level access to the data, there must not be a way to bypass the integrity rules of the database.
	Rule Zero	All preceding rules are based on the notion that in order for a database to be considered relational, it must use its relational abilities exclusively to manage the database

4.10 CHECK YOUR PROGRESS

1. A composite key is a primary key composed of more than one attribute(T/F)
2. An attribute value can be derived from another attribute.(T/F)
3. Attributes that are not divisible are called _____.
4. _____ specifies the maximum number of relationship instances that an entity can participate
5. Which of the following criteria should be considered when selecting an identifier?
 - (a) Choose an identifier that will not be null.
 - (b) Choose an identifier that doesn't have large composite attributes.
 - (c) Choose an identifier that is stable.
 - (d) All of the above
6. An entity has
 - (i) a set of properties
 - (ii) a set of properties and values for all the properties
 - (iii) a set of properties and the values for some set of properties may non-uniquely identify an entity
 - (iv) a set of properties and the values for some set of properties may uniquely identify an entity.

Which of the above are valid?

(a) (i) only (b) (ii) only (c) (iii) only (d) (iv) only

7. What is domain? Give an example.

8. What is simple attribute? Give an example.

9. What is relationship?

10. What is derived attribute? Give an example.

Answers to check your progress:

1. T

2. T

3. atomic

4. Cardinality

5. d

6. d

7. a domain is the set of possible values for a given attribute

8. A simple attribute is an attribute that cannot be subdivided. For example, age, sex, and marital status

9. Relationship links entities

10. A derived attribute is an attribute whose value is calculated (derived) from other attributes. . For example, an employee's age, EMP_AGE, may be found by computing the integer value of the difference between the current date and the EMP_DOB.

4.11 SUMMARY

Tables are the basic building blocks of a relational database. A grouping of related entities, known as an entity set, is stored in a table. Conceptually speaking, the relational table is composed of intersecting rows (tuples) and columns. Each row represents a single entity, and each column represents the characteristics (attributes) of the entities. Keys are central to the use of relational tables. Keys define functional dependencies; that is, other attributes are dependent on the key and can, therefore, be found if the key value is known. A key can be classified as a super key, a candidate key, a primary key, a secondary key, or a foreign key. Each table row must have a primary key. The primary key is an attribute or a combination of attributes that

uniquely identifies all remaining attributes found in any given row. Because a primary key must be unique, no null values are allowed if entity integrity is to be maintained. Although the tables are independent, they can be linked by common attributes. Thus, the primary key of one table can appear as the foreign key in another table to which it is linked. Referential integrity dictates that the foreign key must contain values that match the primary key in the related table or must contain nulls. The relational model supports relational algebra functions: SELECT, PROJECT, JOIN, INTERSECT, UNION, DIFFERENCE, PRODUCT, and DIVIDE. A relational database performs much of the data manipulation work behind the scenes. For example, when you create a database, the RDBMS automatically produces a structure to house a data dictionary for your database. Each time you create a new table within the database, the RDBMS updates the data dictionary, thereby providing the database documentation. Good design begins by identifying appropriate entities and their attributes and then the relationships among the entities. Those relationships (1:1, 1:M, and M:N) can be represented using ERDs. The use of ERDs allows you to create and evaluate simple logical design. The 1:M relationship is most easily incorporated in a good design; you just have to make sure that the primary key of the “1” is included in the table of the “many”.

4.12 KEYWORDS

- **Candidate key**- A candidate key can be described as a superkey without unnecessary attributes, that is, a minimal superkey.
- **Superkey**- A minimal (irreducible) superkey. A superkey that does not contain a subset of attributes that is itself a superkey.
- **Primary key** - A candidate key selected to uniquely identify all other attribute values in any given row. Cannot contain null entries.
- **Foreign key** - An attribute (or combination of attributes) in one table whose values must either match the primary key in another table or be null.
- **Schema** - is the description of the database
- **Functional dependence** - The attribute B is functionally dependent on the attribute A if each value in column A determines one and only one value in column B.
- **Referential integrity** - referential integrity means that if the foreign key contains a value, that value refers to an existing valid tuple (row) in another relation.

- **Relational algebra-** Relational algebra defines the theoretical way of manipulating table contents using the eight relational operators: SELECT, PROJECT, JOIN, INTERSECT, UNION, DIFFERENCE, PRODUCT, and DIVIDE.

4.13 QUESTIONS FOR SELF STUDY

1. “All candidate keys can be primary keys”. Do you agree with the statement? Justify your answer.
2. What is the difference between a database and a table?
3. What does it mean to say that a database displays both entity integrity and referential integrity?
4. Why are entity integrity and referential integrity important in a database?
5. What are the requirements that two relations must satisfy in order to be considered union-compatible?
6. Explain why the data dictionary is sometimes called “the database designer’s database.”
7. How would you implement a 1:M relationship in a database composed of two tables? Give an example.
8. Differentiate between the following with examples:
 - (a) Entity and attributes
 - (b) Primary and foreign key
 - (c) Candidate key and primary key.

4.14 REFERENCES

1. Coronel, C., & Morris, S. (2016). Database systems: design, implementation, & management. Cengage Learning.
2. Gupta, S. B., & Mittal, A. (2009). Introduction to database management system. Laxmi Publications, Ltd..
3. Ramakrishnan, R., & Gehrke, J. (2000). Database management systems. McGraw-Hill.




Karnataka State Open University
Mukthagangothri, Mysore – 570 006.
Dept. of Studies and Research in Management

MBA IT Specialization
III Semester

Database Management System



Block 2


Karnataka State Open University

Mukthagangothri, Mysore – 570 006.

Dept. of Studies and Research in Management

MBA. IT Specialization

III Semester

DATABASE MANAGEMENT SYSTEM

BLOCK 2-ER MODEL AND SQL

UNIT NO.	TITLE	PAGE NUMBERS
UNIT 5	E-R MODELS	1-24
UNIT 6	NORMALIZATION OF DATABASE	25-44
UNIT 7	INTRODUCTION TO STRUCTURED QUERY LANGUAGE (SQL)	45-67
UNIT 8	DATA WAREHOUSES AND DATA MINING	68-89

BLOCK 2 INTRODUCTION

In Unit 5 we discuss ER Model. The E-R model is mainly used for communication between database designers and end users during the analysis phase of database development. This E-R model is a representation of the structure and constraints of a database that is independent of the DBMS and its associated data model that will be used to implement the database. Good database design must be matched to good table structures. In unit 6, you will learn to evaluate and design good table structures to control data redundancies, thereby avoiding data anomalies. The process that yields such desirable results is known as normalization. In unit 7, you will learn the basics of Structured Query Language (SQL). SQL, pronounced S-Q-L by some and “sequel” by others, is composed of commands that enable users to create database and table structures, perform various types of data manipulation and data administration, and query the database to extract useful information. All relational DBMS software supports SQL, and many software vendors have developed extensions to the basic SQL command set. In unit 8, you will learn about the data mining and data warehousing concepts along with advantages and disadvantages.

This block consists of 4 units and is organized as follows:

- Unit 5-** E-R Models: Introduction, Basic Concepts, Types of Attributes, Relationship Sets, Mapping Constraints, Keys, Entity-Relationship Diagram, Types of Entity Sets, EER model,.
- Unit 6-** Normalization of Database: Database Tables and Normalization, The benefits Normalization, The Normalization Proces:1NF, 2NF, 3NF, BCNF and 4NF, Normalization and Database Design, Denormalization.
- Unit 7-** Introduction to Structured Query Language (SQL): Introduction to SQL, DDL, DML: Basic SELECT Queries, SELECT Statement Options, WHERE Clause Options, Aggregate Processing and DCL.
- Unit 8-** Data Warehouses and Data Mining: Data Warehouse, OLTP, OLAP, Data Mining Comparison of Data Mining and Structured Query Language (SQL) and also with data warehousing.

UNIT-5: E-R MODELS

STRUCTURE

- 5.0 Objectives
- 5.1 Introduction
- 5.2 Basic Concepts
- 5.3 Types of Attributes
 - 5.3.1 Simple and Composite Attributes
 - 5.3.2 Single Valued and Multi-valued Attributes
 - 5.3.3 Derived Attributes and Stored Attributes
- 5.4 Relationship Sets
- 5.5 Mapping Constraints
- 5.6 Keys
- 5.7 Entity—Relationship Diagram
- 5.8 Types of Entity Sets
- 5.9 Enhanced Entity-Relationship (EER) Model
 - 5.9.1 Superclass and Subclass Entity Types
 - 5.9.2 Specialization
 - 5.9.3 Generalization
 - 5.9.4 Attribute Inheritance
 - 5.9.5 Aggregation
- 5.11 Check your progress
- 5.12 Summary
- 5.13 Questions for self-study
- 5.14 References

5.0 OBJECTIVES

After studying this unit, you will be able to:

- ✓ The main characteristics of entity relationship components
- ✓ How relationships between entities are defined, refined, and incorporated into the database design process
- ✓ How ERD components affect database design and implementation
- ✓ That real-world database design often requires the reconciliation of conflicting goals

5.1 INTRODUCTION

In this unit, we follow the traditional approach of concentrating on the database structures and constraints during conceptual database design. We present the modelling concepts of the **entity–relationship (ER) model**, which is a popular high-level conceptual data model. This model and its variations are frequently used for the conceptual design of database applications and many database design tools employ its concepts. We describe the basic data-structuring concepts and constraints of the ER model and discuss their use in the design of conceptual schemas for database applications. We also present the diagrammatic notation associated with the ER model, known as **ER diagrams**.

5.2 BASIC CONCEPTS

The entity-relationship (E-R) model was introduced by Chen in 1976. He described the main constructs of the E-R model i.e., entities, relationships and their associated attributes. The E-R model continues to evolve but there is not yet a standard notation for E-R modeling. E-R model is mainly used for conceptual data modeling. It is popular due to the factors such as relative ease of use, CASE tool support, and the belief that entities and relationships are natural modeling concepts in the real world.

The E-R model is mainly used for communication between database designers and end users during the analysis phase of database development. This E-R model is a representation of the structure and constraints of a database that is independent of the DBMS and its associated data model that will be used to implement the database.

In 1980's many new database applications like Computer Aided Manufacturing (CAM), Computer Aided Design (CAD), Computer Aided Software Engineering (CASE), Digital publishing, World Wide Web (WWW), Telecommunication applications etc., were introduced. The basic E-R modeling concepts were no longer sufficient to represent the requirement of these newer and complex applications. Basic E-R model was not capable to represent additional semantic modeling concepts. Database designers and practitioners introduced a new model named as Enhanced Entity-Relationship Model (EER) which includes the basic E-R model concepts with additional semantic concepts.

1 Enterprise

Enterprise refers to any kind of organization.

Ex. Colleges, schools, banks, any company etc.

2 Entity

Entity refers to an “object” or “thing” in real world. Object may be any person, place, event etc.

Ex. Students of colleges and schools, loans in banks, employees in any company etc.

3 Attributes

These are the characteristics of any entity.

Ex., (i) A student can be described by his name, age, address, height, class etc.

(ii) Loans can be described by their types such as house loan, car loan etc.

(iii) Employees in any company can be described by their Employee ID, name, department, designation etc.

(iv) A car can be described by his color, model, company etc.

4 Value

Value is the information or data which is stored in attributes of any entity.

5 Entity Sets

All the entities having same attributes make an entity set.

6 Domain

Domain or value set is the set of all values or information about any attribute.

Ex. Consider the *student* table shown in Figure 5.1. It describes the basic concepts.

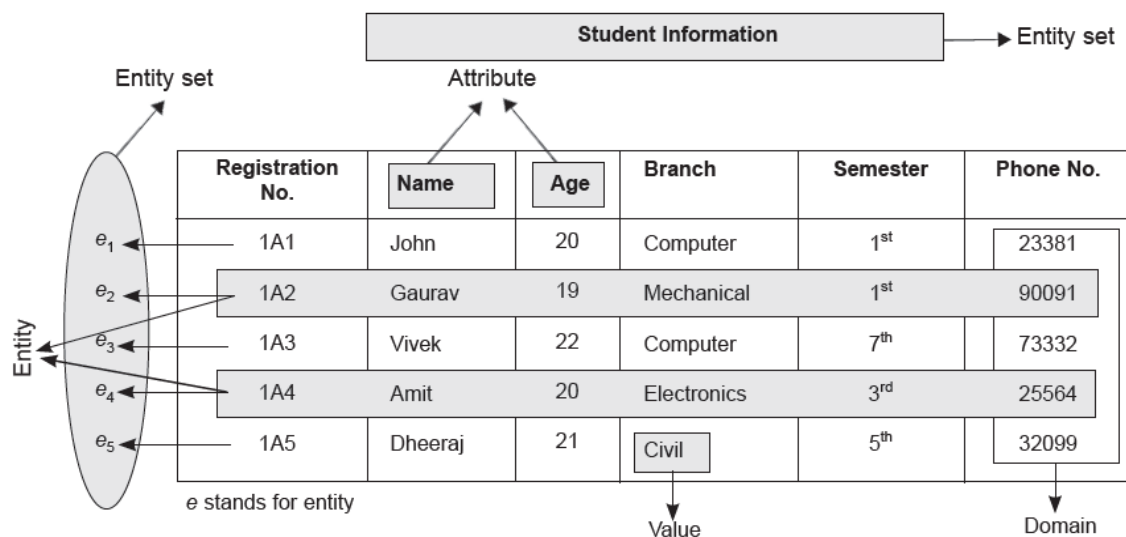


Fig. 5.1. Table showing basic concepts of E-R model.

(a) **Enterprise** : Here, enterprise is college where students are studied.

(b) **Entity** : Here, entity refers to any single student with all his values.

Ex.

1A1	John	20	Computer	1 st	23381
-----	------	----	----------	-----------------	-------

(c) **Attributes** :The students are described by Registration No., Name, Age, Branch, Semester, Phone No. These are the attributes of students.

(d) **Value** :The values are 1A1, 21, Civil, Gaurav, 90091, 5th etc.

(e) **Entity Set** :All students are described by same set of attributes. So, all these students combine together to make an entity set “Student Information”.

(f) **Domain** : (Value set) for, attribute, *Name* it is John, Gaurav, Vivek, Amit, Dheeraj and for *Age*, it is 20, 19, 21, 22.

5.3 TYPES OF ATTRIBUTES

Attributes can be characterized by the following threemajor types:

5.3.1 Simple and Composite Attributes

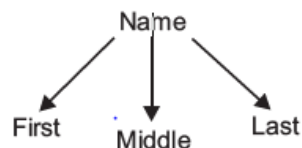
Simple Attributes are those which cannot be divided into subparts.

Ex. Age of student Age



Composite Attributes are those which can be divided into subparts.

Ex. Name of a student can be divided into First Name, Middle Name, and Last Name.



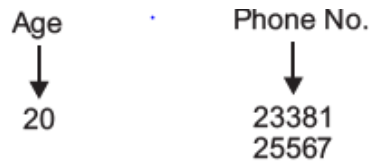
5.3.2 Single Valued and Multi-valued Attributes

Single Valued Attribute: An attribute having only single value for a particular entity is known as single value attribute.

Ex. Age of student.

Multi-Valued Attributes: An attribute having more than one possible value for a particular entity is known as multi-valued attribute.

Ex. Phone number of a student. A student may have more than one phone.



5.3.3 Derived Attributes and Stored Attributes

Derived Attributes: An attribute that can be derived from other known attributes is known as derived attribute.

Ex. Age of employees can be derived if you know date of birth and system date.

$$\text{Age} = \text{System date} - \text{Date of birth}$$

Stored Attributes: An attribute which cannot be derived by other known attributes is known as stored attribute.

Ex. Date of birth of any employee.

NULL Value: Null stands for nothing. An attribute have a null value if either the value of that attribute is not known or the value is not applicable.

Caution: NULL is not equal to Zero (0). But you can say that NULL is blank as shown in Figure 5.2.

Ex.

Name	Subject	Marks
abc	Maths	92
def	Science	—
ghi	Maths	0

→ This is null

Fig.5.2. Table with null value.

5.4 RELATIONSHIP SETS

1. **Relationship:** A relationship is the association among several entities. It connects different entities through a meaningful relation.

2. **Relationship Set:** A relationship set is a set of relationships of the same type.

Consider an example, employees work in different departments. Then relationship exists between employees and departments because each employee must belong to some department. Relation of all employees with department when combined makes the relationship set because each employee has same kind of relation with departments.

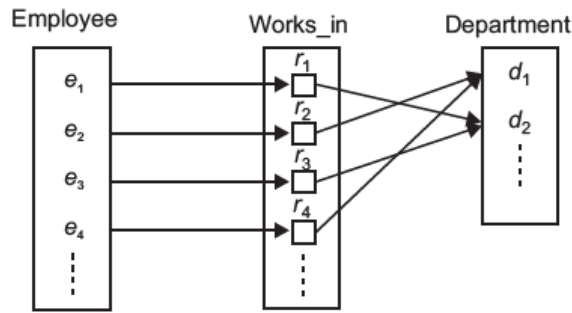


Fig.5.3. Binary relationship set.

Here, Employee and Department are two entity sets. r stands for relationship between Employee and Department. Works_in is the relationship set as shown in Figure 5.3.

- **Descriptive Attributes:** Attributes of any relationship set are known as descriptive attributes.

5.4.1 Degree of Relationship Sets

Total number of entity sets participate in a relationship set is known as degree of that relationship set.

1. Binary Relationship Set

A relationship set in which only two entity sets are involved is known as binary relationship set. Ex. The Figure 5.3 shows the Binary relationship set.

2. Ternary Relationship Set

A relationship set in which three entity sets are involved is known as ternary relationship set or a relationship set having degree three.

Ex. The Figure 5.4 shows the relationship set works_in, which is a ternary relationship set.

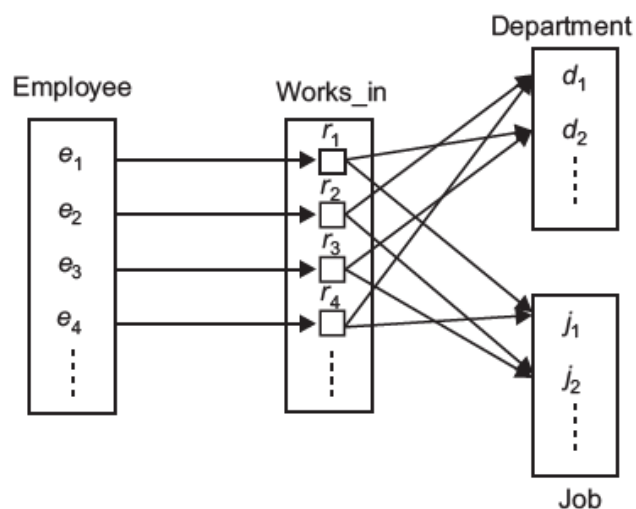


Fig.5.4. Ternary relationship set.

5.4.2 Role and Recursive Relationship Set

- **Role** : The function of any entity which it plays in relationship set is called that entity's role. e.g., employee plays the role of worker in his department in Figure 2.4.
- **Recursive Relationship Set** : When the same entity sets participate in same relationship set more than once with different roles each time, then this type of recursive relationship set is known as Recursive Relationship set. e.g., consider an example of relationship set works_in and two entity set student and college. A student who attends weekend classes in college as student may also be lecturer in that college. Then this person plays two roles (student, faculty) in same relationship set work_in.

5.5 MAPPING CONSTRAINTS

There are certain constraints in E-R model. Data in the database must follow the constraints. Constraints act as rules to which the contents of database must conform. There are two types of mapping constraints: (a) Mapping cardinalities, (b) Participation constraints.

5.5.1 Mapping Cardinalities (Cardinality Ratios)

It specifies the number of entities of an entity set that is associated with entities of another entity set through a relationship set.

Mapping Cardinalities are helpful in describing binary relationship sets.

Two entity sets X and Y having binary relationship set R must have one of the following mapping cardinality:

1. One to One (1 : 1) : An entity in X is associated with at most one entity in Y and an entity in Y is associated with at most one entity in X.

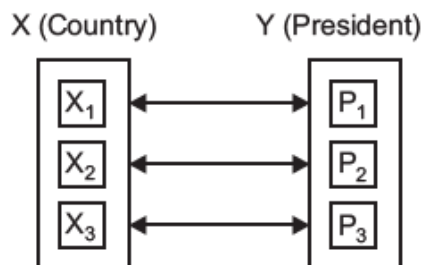


Fig. 5.5. One to one cardinality ratio.

A country has only one president. Any person may be the president of at most one country.

2. One to Many (1 : N) : An entity in X is associated with any number of entities in Y. An entity in Y is associated with at most one entity in X.

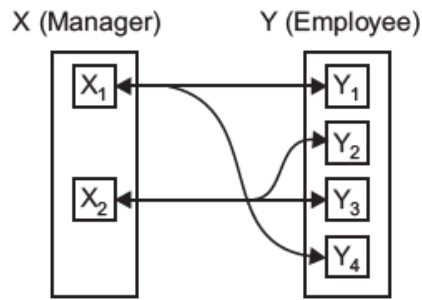


Fig. 5.6. One to many cardinality ratio.

A manager has many employees under it but an employee works under only one manager.

3. Many to One (N : 1) : An entity in X is associated with at most one entity in Y. An entity in Y is associated with any number of entities in X.

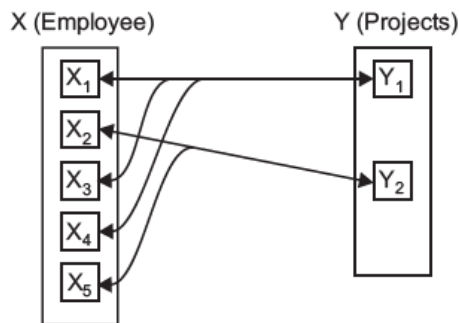


Fig 5.7. Many to one cardinality ratio.

An employee can work on single project while any project can be assigned to more than one employee.

4. Many to Many (M : N) : An entity in X is associated with any number (zero or more) of entities in Y and vice versa

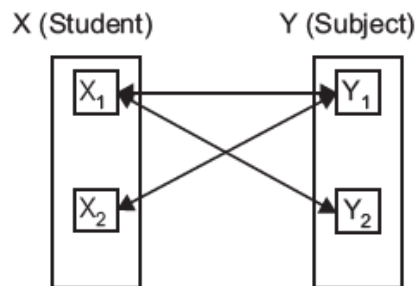


Fig 5.8. Many to many cardinality ratio.

A student can have more than one subject and one subject can be given to more than one student.

5.6 KEYS

A key is an attribute or set of attributes that is used to identify data in entity sets. The attributes which are used as key are known as key attributes. Rest of all is known as Non-key attributes.

5.6.1 Types of Keys

There are many keys that are used in the different tables. These are as follows:

1. **Super Key:** A super key is a set of collection of one or more than one attributes that can identify data uniquely. Any entity set has more than one super key.

Employee

Reg. No	ID	Name	Salary	Dept-ID
S1D	1	Mohan	1500	10
A25	2	Sohan	2000	30
33Z	3	Vikas	3000	20
Z4X	4	Madhu	1000	10
A5C	5	Sonal	5000	20

(a)

Department

Dept-ID	Dept-Name
10	Sales
20	Marketing
30	Development

(b)

Fig.5.9. Entity sets employee and department.

Ex. In entity set Employee, shown in Figure 2.9(a), Super Keys are

- (a) (ID, Name, Salary, Reg. No.)
- (b) (ID, Name, Reg. No.)
- (c) (ID) etc.

All combinations can identify data uniquely.

2. Candidate Key: The minimal super key is known as candidate key. Consider a super key and then take all of its proper subsets. If no one of the proper subsets are super key. Then this super key is taken as candidate key.

Ex. ID and Reg. No. are candidate key

Example: Find all possible candidate keys for the following relation based on its current tuples:

<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>
<i>a1</i>	<i>b1</i>	<i>c1</i>	<i>d1</i>
<i>a1</i>	<i>b2</i>	<i>c2</i>	<i>d1</i>
<i>a2</i>	<i>b2</i>	<i>c1</i>	<i>d1</i>
<i>a2</i>	<i>b1</i>	<i>c2</i>	<i>d1</i>

Ans. There are three candidate keys of this relation

{A, B}, {A, C}, {B, C}

Example: Given a relation Students as follows:

Students (SSN, Name, Home_Address, Birthdate, GPA).

(a) Determine some candidate keys of this relation?

(b) Determine a super key that is not a candidate key?

Ans. (a) {SSN}, {Name, Home_Address, Birthdate} are candidate keys.

(b) {SSN, Name} is a super key but not a candidate key.

3. Primary Key: An attribute which identifies data uniquely is known as Primary Key.

OR

- The term Primary Key is used to denote Candidate key.
- Any entity set can have more than one Candidate key but only one Primary Key.
- Ex. In entity set Employee, either Reg. No. is primary key or ID is primary key.

4. Alternate Keys : All the candidate keys other than Primary Key are known as Alternate Keys. Ex. If you take ID as Primary Key. Then, Reg. No. is an alternate key.

5. Secondary Key : An attribute or set of attributes which doesn't identify data uniquely but identifies a group of data is known as secondary key.

Ex. Name, Salary and Department No. are all secondary keys.













6. Foreign Key : A foreign key is an attribute in any entity set which is also a Primary Key in any other entity set.

Ex. Dept_ID: This is an attribute in entity set Employee and also a primary key in entity set Department. Thus, it is a foreign key in Employee.

5.7 ENTITY—RELATIONSHIP DIAGRAM

E-R diagrams represents the logical structure of a database. Symbols used in E-R diagrams are shown in Table 5.1.

Table 5.1. Symbols in E-R diagram

S.No.	Name of Symbol	Symbol	Meaning
1.	Rectangle		Entity Set (Strong)
2.	Double Rectangle		Entity Set (Weak)
3.	Ellipse		Attribute
4.	Diamond		Relationship Set
5.	Double Diamond		Identifying Relationship Type
6.	Double Ellipses		Multi-valued attributes
7.	Dashed Ellipses		Derived attributes
8.	Ellipse with line inside it		Key attribute
9.	Ellipse joined with other ellipses		Composite attributes
10.	Double lines		Total Participation
11.	Single line		Partial Participation
12.	Triangle		Specialization or Generalization

Examples:

1. Make an E-R diagram having two entity sets, Customer and Item.

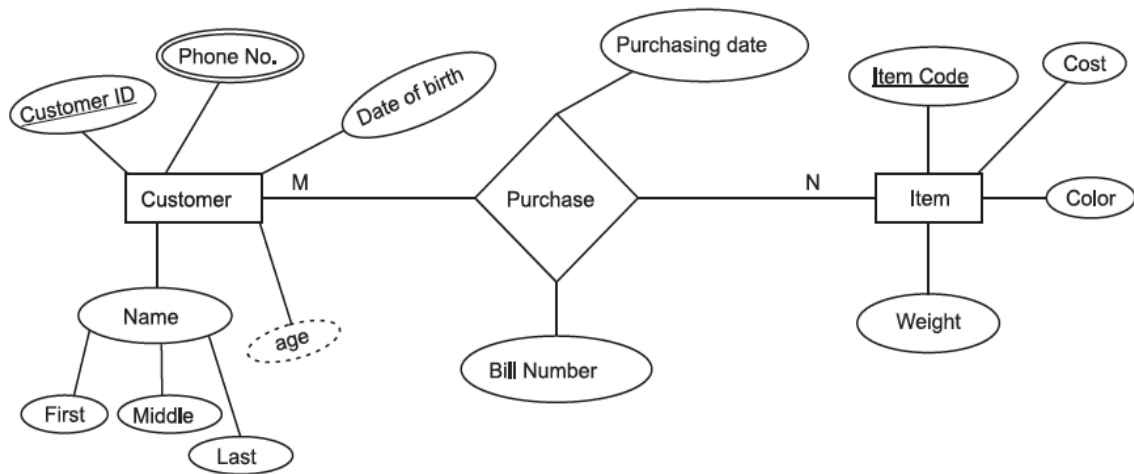


Fig. 5.10. E-R diagram with customer and item entity sets.

- Cardinality Ratio is many to many because a customer can buy any number of items and same item can be purchased by more than one customer.

2. Make an E-R diagram with entities Customer, Loan and Payment

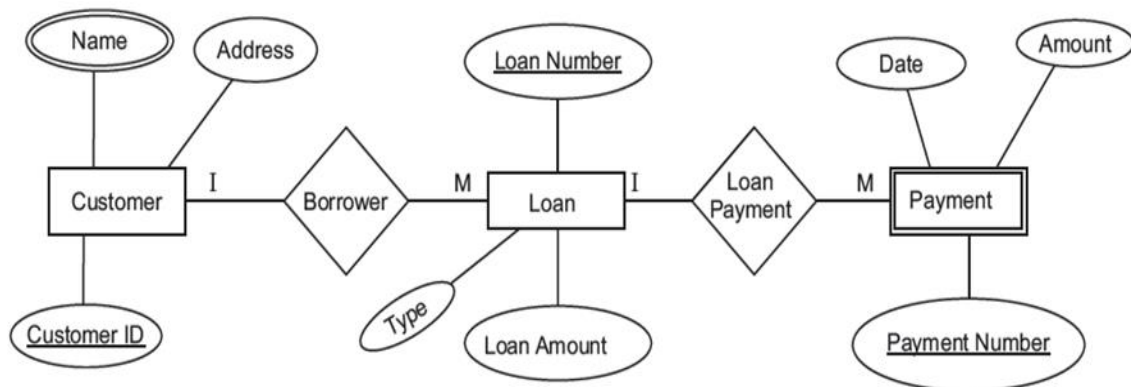


Fig.5.11 E-R diagram with entities Customer, Loan and Payment

5.7.1 Advantages of E-R model

The major advantages of E-R model are as follows:

1. Straightforward relation representation: The relation representation of the database model using E-R diagram are relatively more straightforward than other models.
2. Mapping with relational model: It can be easily mapped onto the relational model. The E-R diagrams used in the E-R model can easily be transformed into relational tables. The entities and attributes of E-R model can easily be transformed into relations (tables) and columns (fields) in a relational model.

3. Communication tool: It is very simple and easy to understand with a minimum of training efforts required. Therefore, the model can be used by the database designer to communicate the design to the end user.
4. Design tool: E-R model can also be used as a design plan by the database developer to implement a data model in specific database management software.
5. Easy conversion to other models: E-R diagrams can be easily converted to a network or hierarchical data model.
6. Graphical representation: E-R model provides graphical and diagrammatical representation of various entities, their attributes and relationships between entities.
7. Easy to modify: Modifications to E-R diagram at later stage is relatively easier than in other models.

5.7.2 Limitation of E-R Model

Limitation of E-R Model : E-R model cannot express relationships between relationships. In other words E-R model is not capable to express relationship set between relationship sets. This limitation can be overcome by using EER model.

5.8 TYPES OF ENTITY SETS

There are two types of entity sets as given below:

5.8.1 Strong Entity Sets

Entity set having any key attributes are known as Strong Entity sets.

Ex. The Figure 5.12 shows strong entity set Student having key attribute Reg_No.

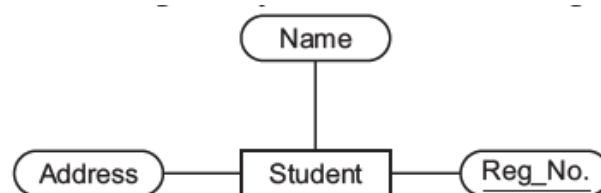


Fig.5.12. Strong entity set.

5.8.2 Weak Entity Sets

Entity sets having no key attributes are known as Weak Entity sets.

Ex. The Figure 5.13 shows weak entity set Table having no key attributes to identify the tuples uniquely.

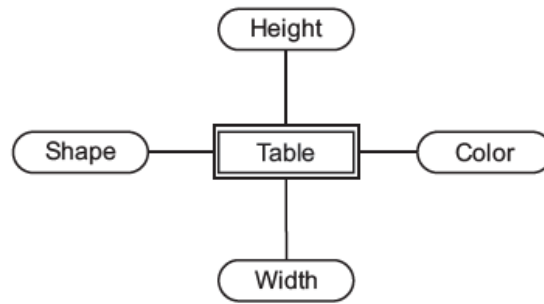


Fig 5.13. Weak entity set.

A weak entity set is always associated with another entity set to make it meaningful. This entity set is known as Identifying entity set.

5.9 ENHANCED ENTITY-RELATIONSHIP (EER) MODEL

EER model is basically the enhanced version of E-R model which includes all the basic concepts of E-R model with capability to support additional semantic concepts of complex applications. These additional concepts are:

- Specialization
- Generalization
- Categorization.

Before discussing the concepts of specialization, generalization, and categorization two other entity types super class (super type) and subclass (subtype) are described.

5.9.1 Superclass and Subclass Entity Types

The most important new modeling construct introduced by EER was superclass and subclass entity types. These are also known as super type and subtype entities respectively. By using these two entity types E-R model can be divided into more specialized sub-models or can join some sub-models to make a generalized E-R model.

- **Super class Entity Type (Super type):** A super class entity type is a generic entity type that includes one or more distinct subclasses that require to be represented in a data model. It means members belong to subclass are same as the entity in the superclass. The relationship between a superclass and subclass is a one-to-one (1 : 1) relationship. In some cases, a superclass can have overlapping subclasses.
- **Subclass Entity Type (Subtype):** A subclass entity type is a more specialized entity type that has a distinct role in the organisation. A subclass is a member of superclass. It is one of the data-modeling abstractions used in EER. A subclass may be further divided and in that case it acts as superclass for its subclasses.

The superclass/subclass relationship is shown in Figure 5.14.

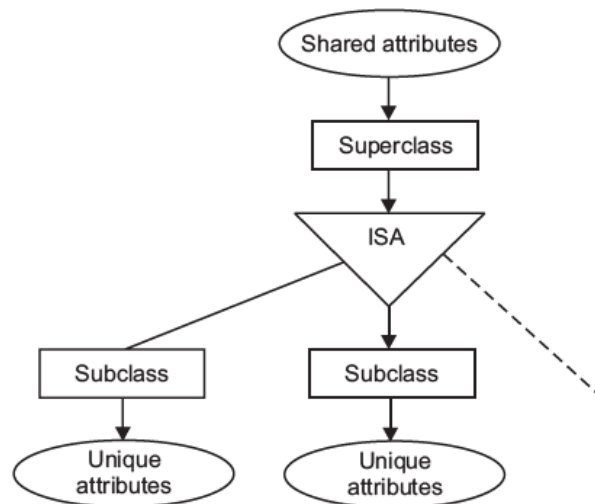


FIGURE 5.14. Superclass/subclass relationship.

Consider the example of a Bank as shown in Figure 5.15 in which PERSON entity is superclass entity type which is further divided into EMPLOYEE and CUSTOMER entities. Here EMPLOYEE and CUSTOMER entities are subclass entity type.

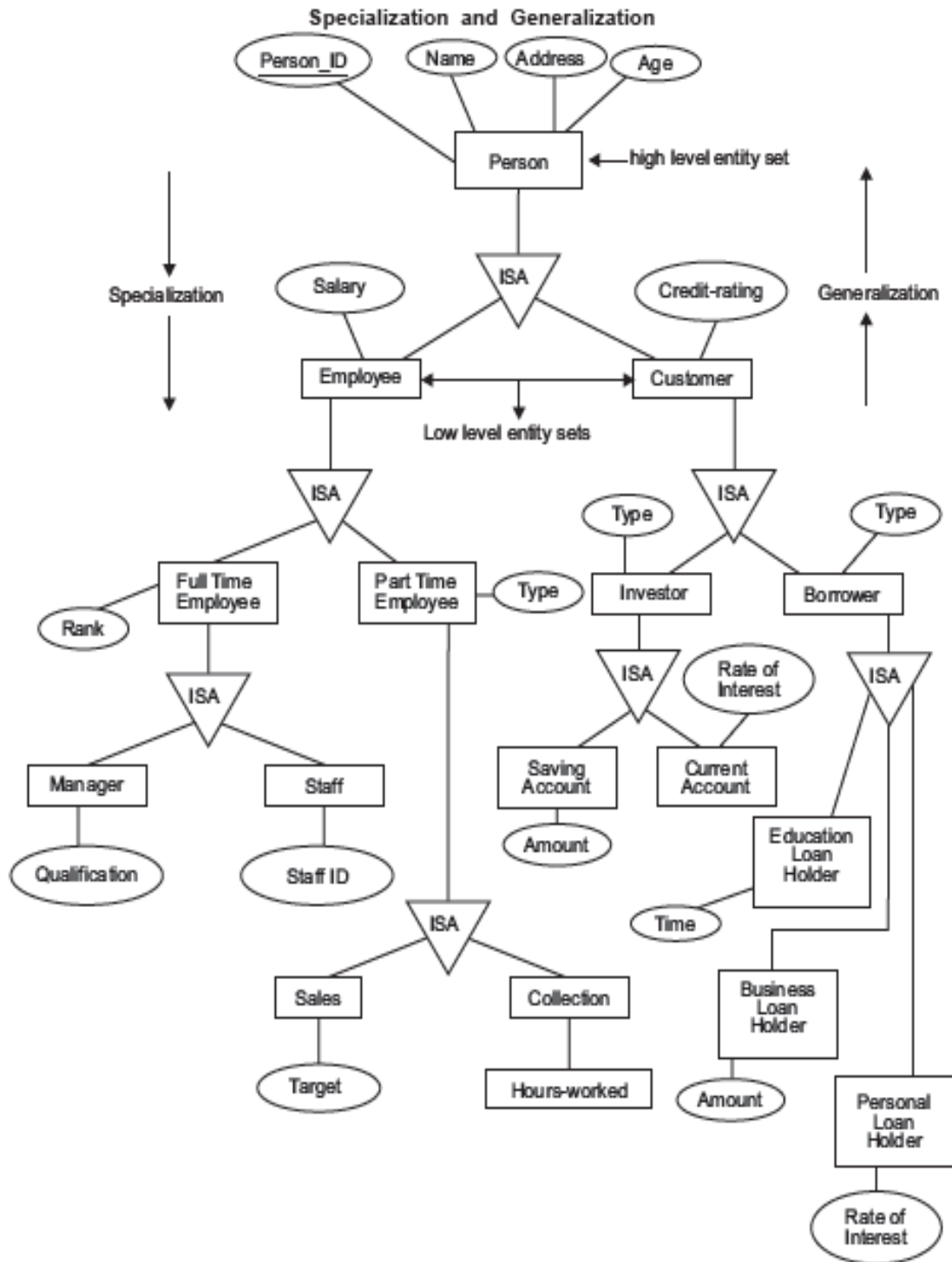


Fig. 5.15. Specialization and generalization.

Basic concept is that by E-R model, a person belongs to a Bank is known, and by EER, how it belongs to Bank is known means, what is the exact relationship between Bank and that person? A person may be an employee or customer which can be further categorized into Manager, Staff or Investor, Borrower and so on.

5.9.2 Specialization

Specialization includes subgrouping of entities within an entity set having some distinct nature than other entities. If deep information is needed then go towards specialization. In other words Specialization is a process by which any existing entity set is divided into smaller entity sets according to the distinct or different nature of entities.

Consider the example of Bank in Figure 5.15. Person is an entity set of all people who belongs to bank. Further Person is classified into Employees and Customers of bank. So, Person entity set is divided into Employee entity set and Customer entity set. Employees are further classified into two categories full time employees and part time employees and so on. Customers are also classified into Investors and Borrowers and so on.

5.9.3 Generalization

Generalization is a process by which two or more entity sets can be combined into a single entity set by determining similarities between the entities. It's an abstract view of any Enterprise. Generalization proceeds from the recognition that a number of entities set share some common features. If an abstract view of information is needed then go towards generalization.

Consider the example in Figure 5.15. Here Investor and Borrower are two entity sets. They have common feature that both are Customer of the Bank. Similarly, Employee entity set and Customer entity set can be combined into Person entity set.

5.9.4 Attribute Inheritance

Specialization and generalization leads to attribute inheritance between higher level entity set and lower level entity set. Inheritance is a process by which lower level entity set inherits (or taken) some properties of its higher level entity set.

Consider the Figure 5.15. Here entity sets Employee and Customer inherits attributes Person_ID, Name, Address, Age from Person entity set.

5.9.5 Aggregation

Aggregation is an abstraction process in which a relationship set is considered as higher level entity set.

Consider an example of ternary relationship having three entity sets Employee, Job and Branch with relationship set works-on as shown in Figure 5.16. The information about Managers on employes, managers of particular jobs and of different branches can be taken easily.

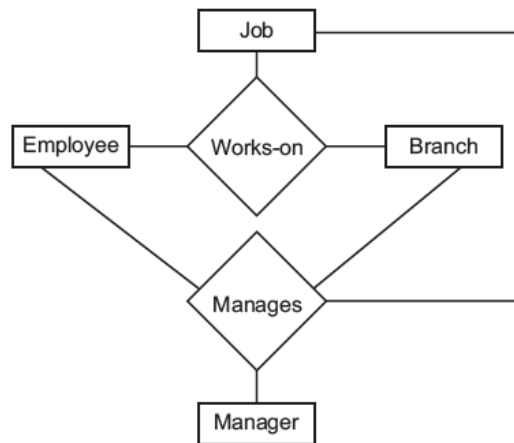


Fig 5.16.E-R model.

5.10 CHECK YOUR PROGRESS

1. All simple attributes are also single-valued (T/F)
 2. A weak entity has a primary key that is partially or totally derived from the parent entity in the relationship.(T/F)
 3. When the value of an attribute A is obtained from the value of an attribute B, then the attribute A is called _____.
 4. The partial key attribute is underlined with a _____ line.
 5. The entity type on which the _____type depends is called the identifying owner.
(a) Strong entity (b) Relationship (c) Weak entity (d) E-R
 6. The E-R model is expressed in terms of
 - (i) Entities
 - (ii) The relationship among entities
 - (iii) The attributes of the entities.
- Then
- (a) (i) and (iii) (b) (i) and (ii) (c) (ii) and (iii) (d) none of these
7. What is the degree of relationship set?
 8. What are three types of data relationships?
 9. What is mapping constraint?
 10. What is an integrity constraint?

Answers to check your progress:

1. T
2. T
3. Derived

4. Dotted
5. c
6. b
7. The degree of a relationship type is the number of participating entity types.
8. one to one (1 : 1), one to many (1 : N) and many to many (N : N)
9. There are two types of mapping constraints: (a) Mapping cardinalities, (b) Participation constraints.
Mapping cardinalities: It specifies the number of entities of an entity set that are associated with entities of another entity set through a relationship set.
Participation constraints: It tells the participation of entity set in relationship sets.
10. These are the rules or constraints applied to the database to keep data stable, accurate or consistent.

5.11 SUMMARY

The ER Modelling(ERM) uses ER Diagrams(ERD) to represent the conceptual database as viewed by the end user. The ERM's main components are entities, relationships and attributes. The ERD also includes connectivity and cardinality notations. An ERD can also show relationship strength, relationship participation (optional or mandatory) and degree of relationship (unary, binary, ternary, etc.). Connectivity describes the relationship classification (1:1, 1:M, or M: N). Cardinality expresses the specific number of entity occurrences associated with an occurrence of a related entity. Connectivities and cardinalities are usually based on business rules. ERDs may be based on many different ERMs. However, regardless of which model is selected, the modelling logic remains the same. Because no ERM can accurately portray all real-world data and action constraints, application software must be used to augment the implementation of at least some of the business rules. Unified Modelling Language (UML) class diagrams are used to represent the static data structures in a data model. The symbols used in the UML class and ER diagrams are very similar. The UML class diagrams can be used to depict data models at the conceptual or implementation abstraction levels. Database designers, no matter how well they are able to produce designs that conform to all applicable modelling conventions are often forced to make design compromises. Those compromises are required when end users have vital transaction-speed and/or information requirements that prevent the use of "perfect" modelling logic and adherence to all modelling conventions. Therefore, database designers must use their

professional judgment to determine how and to what extent the modelling conventions are subject to modification. To ensure that their professional judgments are sound, database designers must have detailed and in-depth knowledge of data-modelling conventions. It is also important to document the design process from beginning to end, which helps keep the design process on track and allows for easy modifications down the road.

5.12 KEYWORDS

- **Binary relationship** - A binary relationship exists when two entities are associated in a relationship.
- **Cardinality** – Cardinality expresses the minimum and maximum number of entity occurrences associated with one occurrence of the related entity.
- **Composite attribute** - A composite attribute, not to be confused with a composite key, is an attribute that can be further subdivided to yield additional attributes.
- **Derived attribute** - A derived attribute is an attribute whose value is calculated (derived) from other attributes.
- **Relationship degree**- A relationship degree indicates the number of entities or participants associated with a relationship.
- **Strong entity** - If an entity can exist apart from all of its related entities (it is existence-independent), then that entity is referred to as a strong entity or regular entity.
- **Weak entity** - a weak entity is one that meets two conditions:
 1. The entity is existence-dependent; that is, it cannot exist without the entity with which it has a relationship.
 2. The entity has a primary key that is partially or totally derived from the parent entity in the relationship.

5.13 QUESTIONS FOR SELF-STUDY

1. Discuss the conventions for displaying an ER schema as an ER diagram.
2. Explain the difference between a weak and a strong entity set. Why do we have the concept of weak entity set?
3. “All candidate keys can be primary keys”. Do you agree with the statement? Justify your answer.
4. Design the ER diagram for the Educational Institute System. Make your own assumptions about

5. What is meant by a recursive relationship type? Give some examples of recursive relationship types.

5.14 QUESTIONS FOR SELF-STUDY

1. Coronel, C., & Morris, S. (2016). Database systems: design, implementation, & management. Cengage Learning.
2. Gupta, S. B., & Mittal, A. (2009). Introduction to database management system. Laxmi Publications, Ltd..
3. Ramakrishnan, R., & Gehrke, J. (2000). Database management systems. McGraw-Hill.

UNIT-6: NORMALIZATION OF DATABASE

STRUCTURE

- 6.0 Objectives
- 6.1 Introduction
- 6.2 Database Tables and Normalization
- 6.3 The benefits of Normalization
- 6.4 The Normalization Process
- 6.5 Various normal forms
 - 6.5.1 1NF
 - 6.5.2 2NF
 - 6.5.3 3NF
 - 6.5.4 BCNF
 - 6.5.5 4NF
- 6.6 Normalization and Database Design
- 6.7 Denormalization
- 6.8 Check Your Progress
- 6.9 Summary
- 6.10 Keywords
- 6.11 Questions for self-study
- 6.12 References

6.0 OBJECTIVES

After studying this unit, you will be able to:

- ✓ What normalization is and what role it plays in the database design process.
- ✓ About the normal forms 1NF, 2NF, 3NF, BCNF, and 4NF.
- ✓ Explain the need of normalization.
- ✓ That normalization and ER modelling are used concurrently to produce a good database
- ✓ That some situations require denormalization to generate information efficiently.

6.1 INTRODUCTION

Normalization is based on the analysis of functional dependencies. A functional dependency is a constraint between two attributes or two sets of attributes. The purpose of the database design is to arrange the various data items into an organized structure so that it

generates set of relationships and stores the information without any repetition. A bad database design may result into redundant and spurious data and information.

Normalization is a process by which we can decompose or divide any relation into more than one relation to remove anomalies in relational database. It is a step by step process and each step is known as Normal Form. Normalization is a reversible process.

Normalization is a process for deciding which attributes should be grouped together in a relation. It is a tool to validate and improve a logical design, so that it satisfies certain constraints that avoid redundancy of data. Furthermore, Normalization is defined as the process of decomposing relations with anomalies to produce smaller, well-organized relations. Thus, in normalization process, a relation with redundancy can be refined by decomposing it or replacing it with smaller relations that contain the same information, but without redundancy.

Functional dependencies

Functional dependencies are the result of interrelationship between attributes or in between tuples in any relation.

Definition : In relation R, X and Y are the two subsets of the set of attributes, Y is said to be functionally dependent on X if a given value of X (all attributes in X) uniquely determines the value of Y (all attributes in Y).

It is denoted by $X \rightarrow Y$ (Y depends upon X).

Determinant : Here X is known as determinant of functional dependency.

Consider the example of Employee relation:

Employee		
EID	Name	Salary
1	Aditya	15,000
2	Manoj	16,000
3	Sandeep	9,000
4	Vikas	10,000
5	Manoj	9,000

Fig. 6.1. Employee relation.

In Employee relation, EID is primary key. Suppose you want to know the name and salary of any employee. If you have EID of that employee, then you can easily find information of that employee. So, Name and Salary attributes depend upon EID attribute.

Here, X is (EID) and Y is (Name, Salary)

X (EID) : Y (Name, Salary)

The determinant is EID

Suppose X has value 5 then Y has value (Manoj, 9,000)

Functional Dependency Chart/Diagram

It is the graphical representation of function dependencies among attributes in any relation.

The following four steps are followed to draw FD chart.

1. Find out the primary key attributes.
2. Make a rectangle and write all primary key attributes inside it.
3. Write all non-prime key attributes outside the rectangle.
4. Use arrows to show functional dependency among attributes.

Consider the example of Figure 6.1. Its functional dependency chart is shown in Figure 6.2.

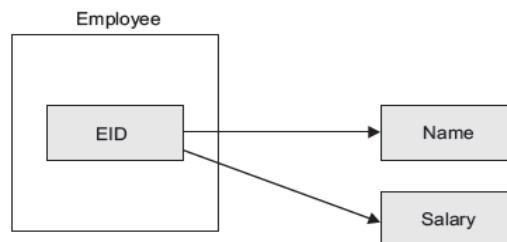


Fig. 6.2. Functional dependency chart of employee relation.

It is easier to remember all dependencies by making FD charts.

Example. Consider the following relation:

Professor (Pfcode, Dept, Head, Time) It is assumed that

- (i) A professor can work in more than one dept.
- (ii) The time he spends in each dept is given.
- (iii) Each dept has only one head.

Draw the dependency diagram for the given relation by identifying the dependencies.

Sol. The Figure 6.3 shows the corresponding dependency diagram.

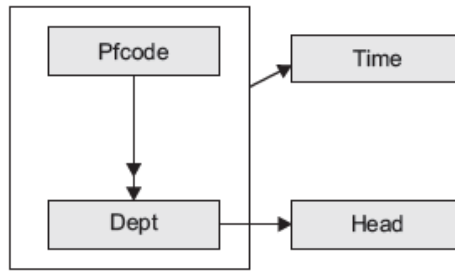


Fig. 6.3 Dependency diagram

Types of Functional Dependencies

There are four major types of FD's.

1. Partial Dependency and Fully Functional Dependency

- Partial dependency : Suppose you have more than one attributes in primary key. Let A be the non-prime key attribute. If A is not dependent upon all prime key attributes then partial dependency exists.
- Fully functional dependency : Let A be the non-prime key attribute and value of A is dependent upon all prime key attributes. Then A is said to be fully functional dependent. Consider a relation student having prime key attributes (RollNo and Game) and non-prime key attributes (Grade, Name and Fee).

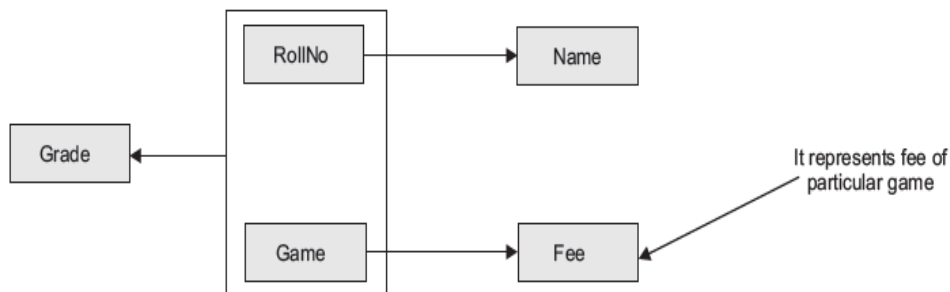


Fig. 6.4. Functional dependency chart showing partial and fully functional dependency of student relation.

As shown in Figure 6.4, Name and Fee are partially dependent because you can find the name of student by his RollNo. and fee of any game by name of the game.

Grade is fully functionally dependent because you can find the grade of any student in a particular game if you know RollNo. and Game of that student. Partial dependency is due to more than one prime key attribute.

2. Transitive Dependency and Non-transitive Dependency

- Transitive dependency : Transitive dependency is due to dependency between non-prime key attributes. Suppose in a relation R, $X \rightarrow Y$ (Y depends upon X), $Y \rightarrow Z$ (Z depends upon Y), then $X \rightarrow Z$ (Z depends upon X). Therefore, Z is said to be transitively dependent upon X.
- Non-transitive dependency : Any functional dependency which is not transitive is known as Non-transitive dependency. Non-transitive dependency exists if there is no dependency between non-prime key attributes.

Consider a relation student (whose functional dependency chart is shown in Figure 6.5) having prime key attribute (RollNo) and non-prime key attributes (Name, Semester, Hostel).

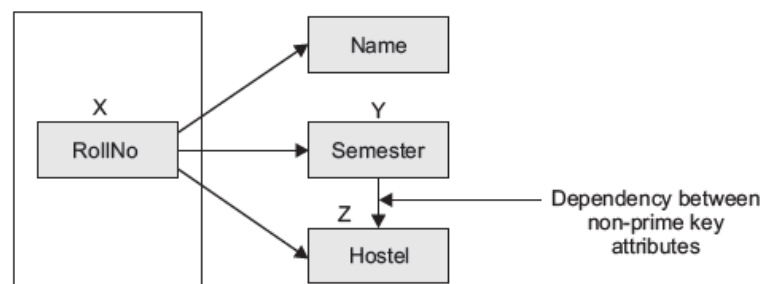


Fig.6.5. Functional dependency chart showing transitive and non-transitive dependency on relation student.

For each semester there is different hostel

Here Hostel is transitively dependent upon RollNo. Semester of any student can be find by his RollNo. Hostel can be find out by semester of student.

Here, Name is non-transitively dependent upon RollNo.

3. Single Valued Dependency and Multivalued Dependency

- Single valued dependency : In any relation R, if for a particular value of X, Y has single value then it is known as single valued dependency.
- Multivalued dependency (MVD) : In any relation R, if for a particular value of X, Y has more then one value, then it is known as multivalued dependency. It is denoted by $X \twoheadrightarrow Y$.

Consider the relation Teacher shown in Figure 6.6(a) and its FD chart shown in Figure 6.6(b).

ID	Teacher	Class	Days
1Z	Sam	Computer	1
2Z	John	Computer	6
1Z	Sam	Electronics	3
2Z	John	Mechanical	5
3Z	Nick	Mechanical	2

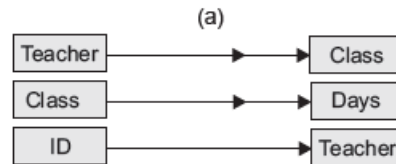


Fig. 6.6. Functional dependency chart showing single valued and multivalued dependency on relation teacher.

6.2 DATABASE TABLES AND NORMALIZATION

Having good relational database software is not enough to avoid the data redundancy in Database Systems. If the database tables are treated as though they are files in a file system, the relational database management system (RDBMS) never has a chance to demonstrate its superior data-handling capabilities. The table is the basic building block of database design. Consequently, the table's structure is of great interest. Ideally, Entity Relationship (ER) Modeling, yields good table structures. Yet it is possible to create poor table structures even in a good database design. So how do you recognize a poor table structure and how do you produce a good table? The answer to both questions involves normalization.

Normalization is a process for evaluating and correcting table structures to minimize data redundancies, thereby reducing the likelihood of data anomalies. The normalization process involves assigning attributes to tables based on the concept of determination. Normalization works through a series of stages called normal forms. The first three stages are described as first normal form (1NF), second normal form (2NF), and third normal form (3NF). From a structural point of view, 2NF is better than 1NF, and 3NF is better than 2NF. For most purposes in business database design, 3NF is as high as you need to go in the normalization process. However, you will discover that properly designed 3NF structures also meet the requirements of fourth normal form (4NF).

Although normalization is a very important database design ingredient, you should not assume that the highest level of normalization is always the most desirable. Generally, the higher the normal form, the more relational join operations are required to produce a

specified output and the more resources are required by the database system to respond to end-user queries. A successful design must also consider end-user demand for fast performance. Therefore, you will occasionally be expected to denormalize some portions of a database design in order to meet performance requirements. Denormalization produces a lower normal form; that is, a 3NF will be converted to a 2NF through denormalization. However, the price you pay for increased performance through denormalization is greater data redundancy.

6.3 THE BENEFITS OF NORMALIZATION

The benefits of normalization include:

- Normalization produces smaller tables with smaller rows, this means more rows per page and hence less logical I/O.
- Searching, sorting, and creating indexes are faster, since tables are narrower, and more rows fit on a data page.
- The normalization produces more tables by splitting the original tables. Thus there can be more clustered indexes and hence there is more flexibility in tuning the queries.
- Index searching is generally faster as indexes tend to be narrower and shorter.
- The more tables allow better use of segments to control physical placement of data.
- There are fewer indexes per table and hence data modification commands are faster.
- There are small number of null values and less redundant data. This makes the database more compact.
- Data modification anomalies are reduced.
- Normalization is conceptually cleaner and easier to maintain and change as the needs change

6.4 THE NORMALIZATION PROCESS

In this section, you will learn how to use normalization to produce a set of normalized tables to store the data that will be used to generate the required information. The objective of normalization is to ensure that each table conforms to the concept of well-formed relations—that is, tables that have the following characteristics:

- Each table represents a single subject. For example, a course table will contain only data that directly pertain to courses. Similarly, a student table will contain only student data.
- No data item will be unnecessarily stored in more than one table (in short, tables have minimum controlled redundancy). The reason for this requirement is to ensure that the data are updated in only one place.
- All nonprime attributes in a table are dependent on the primary key—the entire primary key and nothing but the primary key. The reason for this requirement is to ensure that the data are uniquely identifiable by a primary key value.
- Each table is void of insertion, update or deletion anomalies. This is to ensure the integrity and consistency of the data.

To accomplish the objective, the normalization process takes you through the steps that lead to successively higher normal forms.

The concept of keys is central to the discussion of normalization. A candidate key is a minimal (irreducible) superkey. The primary key is the candidate key that is selected to be the primary means used to identify the rows in the table. Although normalization is typically presented from the perspective of candidate keys, for the sake of simplicity while initially explaining the normalization process, we will make the assumption that for each table there is only one candidate key and therefore that candidate key is the primary key.

From the data modeller's point of view, the objective of normalization is to ensure that all tables are at least in third normal form (3NF). Even higher-level normal forms exist. However, normal forms such as the fifth normal form (5NF) and domain-key normal form (DKNF) are not likely to be encountered in a business environment and are mainly of theoretical interest. More often than not, such higher normal forms increase joins (slowing performance) without adding any value in the elimination of data redundancy. Some very specialized applications, such as statistical research, might require normalization beyond the 4NF, but those applications fall outside the scope of most business operations.

6.5 VARIOUS NORMALFORMS

The different normal forms are as follows: Each of which has its importance and are more desirable than the previous one.

6.5.1 First Normal Form (1NF)

A relation is in first normal form if domain of each attribute contains only atomic values. It means atomicity must be present in relation.

Consider the relation Employee as shown in Figure 6.8. It is not in first normal form because attribute Name is not atomic. So, divide it into two attributes First Name and Last Name as shown in Figure 6.7.

Employee

EID	First Name	Second Name	Salary	Dept. No.	Dept. Name
1	Shivi	Goyal	10,000	2	Accounts
2	Amit	Chopra	9,000	2	Accounts
3	Deepak	Gupta	11,000	1	Sales
4	Sandeep	Sharma	8,500	5	Marketing
5	Vikas	Malik	7,000	5	Marketing
6	Gaurav	Jain	15,000	2	Accounts
7	Lalit	Parmar	14,000	5	Marketing
8	Vishal	Bamel	10,500	2	Accounts

Fig. 6.7. Employee relation in 1NF.

Now, relation Employee is in 1NF.

Anomalies in First Normal Form: First Normal form deals only with atomicity. Anomalies described earlier are also applicable here.

Example: Given the relation PART (Part_ID, Descr, Price, Comp_ID, No) having the following dependencies

Part_ID → Descr

Part_ID → Price

Part_ID, Comp_ID → No

Determine the highest normal form of this relation.

Sol. There exists multi-value attributes. The attributes Comp_ID and No are not determined by the primary key. Hence the relation is not in 1NF.

6.5.2 Second Normal Form (2NF)

A relation is in second normal form if it is in 1NF and all non-primary key attributes must be fully functionally dependent upon primary key attributes.

Consider the relation Student as shown in Figure 6.8(a) :

Student

RollNo.	Game	Name	Fee	Grade
1	Cricket	Amit	200	A
2	Badminton	Dheeraj	150	B
3	Cricket	Lalit	200	A
4	Badminton	Parul	150	C
5	Hockey	Jack	100	A
6	Cricket	John	200	C

(a)

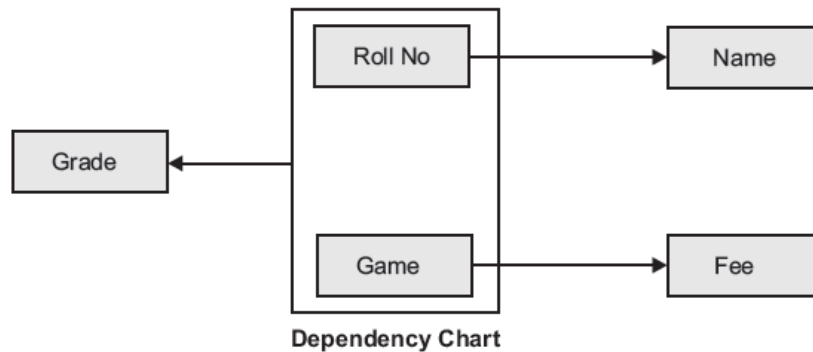


Fig 6.8.Student relation with anomalies.

The Primary Key is (RollNo., Game). Each Student can participate in more than one game.

Relation Student is in 1NF but still contains anomalies.

1. **Deletion anomaly:** Suppose you want to delete student Jack. Here you lose information about game Hockey because he is the only player participated in hockey.
2. **Insertion anomaly:** Suppose you want to add a new game Basket Ball having no student participated in it. You cannot add this information unless there is a player for it.
3. **Updation anomaly:** Suppose you want to change Fee of Cricket. Here, you have to search all the students participated in cricket and update fee individually otherwise it produces inconsistency.

The solution of this problem is to separate Partial dependencies and fully functional dependencies. So, divide Student relation into three relations Student(RollNo., Name), Games (Game, Fee) and Performance(RollNo., Game, Grade) as shown in Figure 6.9.

Student		Games	
RollNo.	Name	Game	Fee
1	Amit	Cricket	200
2	Dheeraj	Badminton	150
3	Lalit	Hockey	100
4	Parul		
5	Jack		
6	John		

Performance		
RollNo.	Game	Grade
1	Cricket	A
2	Badminton	B
3	Cricket	A
4	Badminton	C
5	Hockey	A
6	Cricket	C

Fig. 6.9 Student Relation

Now, Deletion, Insertion and Updation operations can be performed without causing inconsistency.

6.5.3 Third Normal Form (3NF)

A relation is in Third Normal Form if it is in 2NF and non-primary key attributes must be non-transitively dependent upon primary key attributes. In other words a relation is in 3NF if it is in 2NF and having no transitive dependency.

Consider the relation Student as shown in Figure 6.10(a).

Student			
RollNo.	Name	Semester	Hostel
1	Lalit	1	H1
2	Gaurav	2	H2
3	Vishal	1	H1
4	Neha	4	H4
5	John	3	H3

(a)

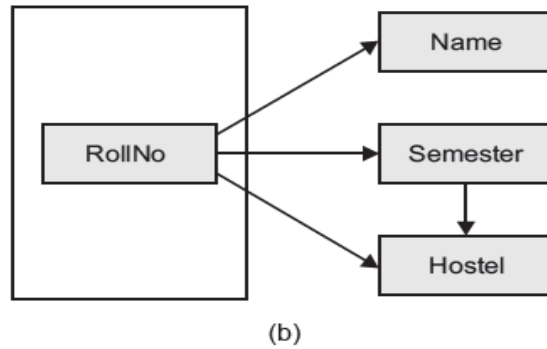


Fig. 6.10.Student relation.

The Primary Key is (RollNo.). The condition is different Hostel is allotted for different semester. Student relation is in 2NF but still contains anomalies.

1. Deletion anomaly: If you want to delete student Gaurav. You loose information about Hostel H2 because he is the only student staying in hostel H2.
2. Insertion anomaly: If you want to add a new Hostel H8 and this is not allotted to any student. You cannot add this information.
3. Updation anomaly: If you want to change hostel of all students of first semester. You have to search all the students of first semester and update them individually otherwise it causes inconsistency.

The solution of this problem is to divide relation Student into two relations Student(RollNo, Name, Semester) and Hostels(Semester, Hostel) as shown in Figure 6.11.

Student		
RollNo.	Name	Semester
1	Lalit	1
2	Gaurav	2
3	Vishal	1
4	Neha	4
5	John	3

Hostels	
Semester	Hostel
1	H1
2	H2
3	H3
4	H4

Fig. 6.11.Relations in 3NF.

Now, deletion, insertion and updation operations can be performed without causing inconsistency.

Example. Given the relation BANK (Account#, Cust_No, Cust_Name, Balance). Determine whether the relation is in 1NF, 2NF, 3NF or unnormalizd. If it is not in 3NF, convert it into 3NF relations.

Sol. Since there does not exist multi-value attributes, hence the relation is at least 1NF.

- There does not exist any partial dependency as the primary key has only one attribute.
- There exists a transitive dependency i.e. $Cust_No \rightarrow Cust_Name$

Hence the relation is not in 3NF. To convert this relation into 3NF, make a new relation Customers with attributes Cust_Name and Cust_No on which it depends. Therefore, the relations in 3NF are Customers (Cust_No, Cust_Name) BANK (Account#, Cust_No, Balance).

6.5.4 Boyce Codd Normal Form (BCNF)

BCNF is a strict format of 3NF. A relation is in BCNF if and only if all determinants are candidate keys. BCNF deals with multiple candidate keys. Relations in 3NF also contains anomalies. Consider the relation Student as shown in Figure 6.12(a).

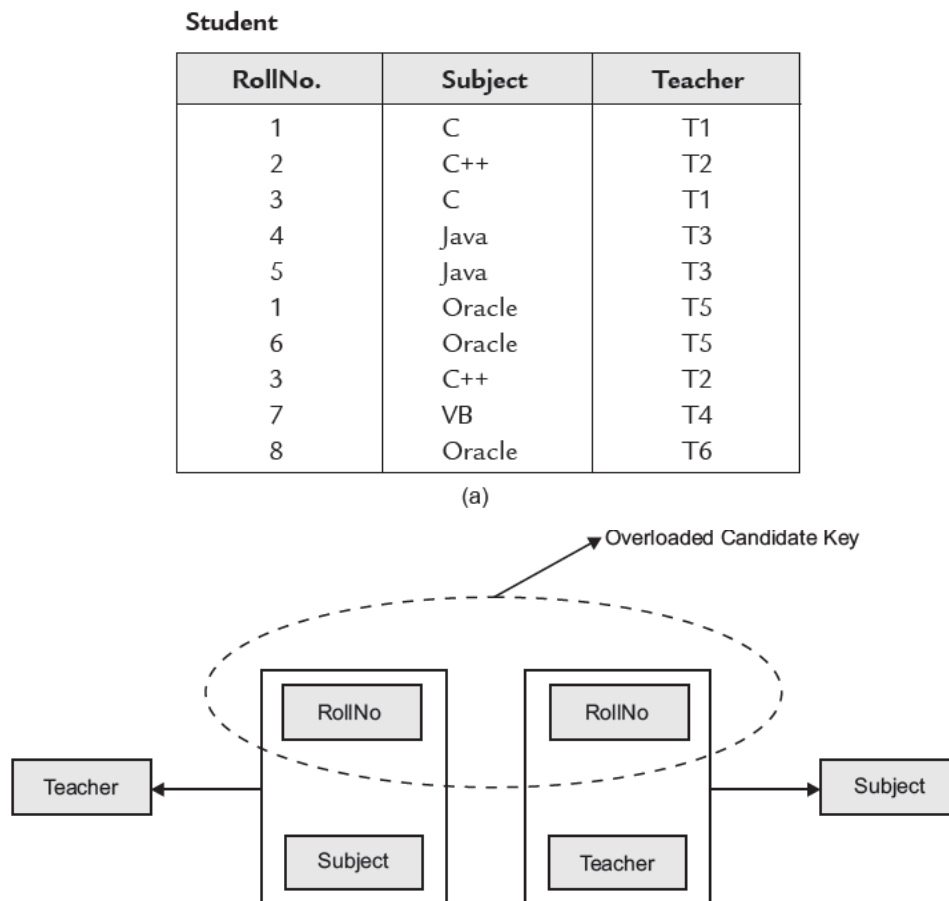


Fig. 6.12.Student relation.

Assumptions:

- Student can have more than 1 subject.
- A Teacher can teach only 1 subject.
- A subject can be taught by more than 1 teacher

There are two candidate keys (RollNo., Subject) and (RollNo., Teacher)

Relation Student is in 3NF but still contain anomalies.

1. Deletion anomaly: If you delete student whose RollNo. is 7. You will also loose information that Teacher T4 is teaching the subject VB.
2. Insertion anomaly: If you want to add a new Subject VC++, you cannot do that until a student chooses subject VC++ and a teacher teaches subject VC++.
3. Updation anomaly: Suppose you want to change Teacher for Subject C. You have to search all the students having subject C and update each record individually otherwise it causes inconsistency.

In relation Student, candidate key is overloaded. You can find Teacher by RollNo. and Subject. You can also find Subject by RollNo. and Teacher. Here RollNo. is overloaded. You can also find Subject by Teacher.

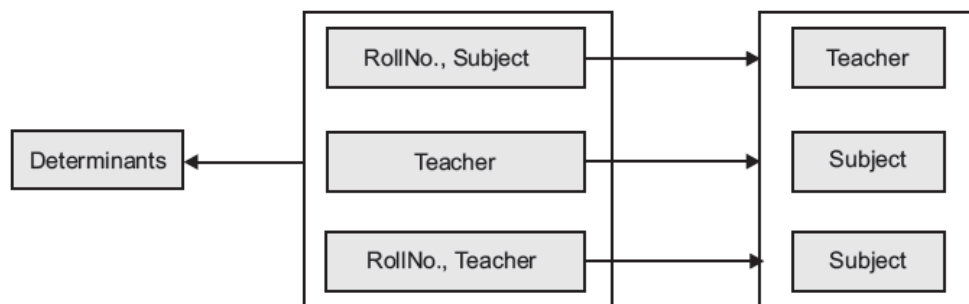


Fig 6.13.Determinants in relation student.

The solution of this problem is to divide relation Student in two relations Stu-Teac and Teac-Sub as shown in Figure 6.14.

Stu-Teac	
RollNo.	Teacher
1	T1
2	T2
3	T1
4	T3
5	T3
1	T5
6	T5
3	T2
7	T4
8	T6

Teac-Sub	
Teacher	Subject
T1	C
T2	C++
T3	Java
T4	VB
T5	Oracle
T6	Oracle

Teac-Sub
Candidate Key (Teacher)

Stu - Teac (RollNo., Teacher)
Candidate Key (RollNo., Teacher)

Fig 6.14.Relations in BCNF.

In this solution all determinants are candidate keys.

6.5.5 Fourth Normal Form (4NF)

Definition: A relation is in fourth normal form (4NF) if and only if it is in BoyceCodd normal form and there are no nontrivial multivalued dependencies.

A relation is in 4NF if it is in BCNF and for all Multivalued Functional Dependencies (MVD) of the form $X \twoheadrightarrow Y$ either $X \rightarrow Y$ is a trivial MVD or X is a super key of relation.

Relations in BCNF also contains anomalies. Consider the relation Project-Work as shown in Figure 6.15.

Programmer	Project	Module
P1	1	M1
P2	1	M2
P3	2	M1
P1	3	M1
P4	3	M2

Fig. 6.15. Project-work relation.

Assumptions:

- A Programmer can work on any number of projects.
- A project can have more than one module.

Relation Project-work is in BCNF but still contains anomalies.

1. Deletion anomaly: If you delete project 2. You will lose information about Programmer P3.
2. Insertion anomaly: If you want to add a new project 4. You cannot add this project until it is assigned to any programmer.
3. Updation anomaly: If you want to change name of project 1. Then you have to search all the programmers having project 1 and update them individually otherwise it causes inconsistency.

Dependencies in Relation Project-work are

Programmer \twoheadrightarrow Project

Project \twoheadrightarrow Module

The solution of this problem is to divide relation Project-Work into two relations Prog-Prj (Programmer, Project) and Prj-Module (Project, Module) as shown in Figure 6.16.

Programmer	Project
P1	1
P2	1
P3	2
P1	3
P4	3

Here Programmer is the super key

Project	Module
1	M1
1	M2
2	M1
3	M1
3	M2

Here Project is the super key

Fig.6.16. Relations are in 4NF.

6.6 NORMALIZATION AND DATABASE DESIGN

We know how normalization procedures can be used to produce good tables from poor ones. You will likely have ample opportunity to put this skill into practice when you begin to work with real-world databases. Normalization should be part of the design process. Therefore, make sure that proposed entities meet the required normal form before the table structures are created. Keep in mind that if you follow the design procedures discussed earlier, the likelihood of data anomalies will be small. But even the best database designers are known to make occasional mistakes that come to light during normalization checks. However, many of the real-world databases you encounter will have been improperly designed or burdened with anomalies if they were improperly modified over the course of time and that means you might be asked to redesign and modify existing databases that are, in effect, anomaly traps. Therefore, you should be aware of good design principles and procedures as well as normalization procedures.

First, an ERD is created through an iterative process. You begin by identifying relevant entities, their attributes, and their relationships. Then you use the results to identify additional entities and attributes. The ERD provides the big picture or macro view of an organization's data requirements and operations.

Second, normalization focuses on the characteristics of specific entities; that is, normalization represents a micro view of the entities within the ERD and as you learned in the previous sections, the normalization process might yield additional entities and attributes to be incorporated into the ERD. Therefore, it is difficult to separate the normalization process from the ER modelling process; the two techniques are used in an iterative and incremental process.

6.7 DENORMALIZATION

It's important to remember that the optimal relational database implementation requires that all tables be at least in third normal form (3NF). A good relational DBMS excels at managing normalized relations; that is, relations void of any unnecessary redundancies that might cause data anomalies. Although the creation of normalized relations is an important database design goal, it is only one of many such goals. Good database design also considers processing (or reporting) requirements and processing speed. The problem with normalization is that as tables are decomposed to conform to normalization requirements, the number of database tables expands. Therefore, in order to generate information, data must be put together from various tables. Joining a large number of tables takes additional input/output (I/O) operations and processing logic thereby reducing system speed. Most relational database systems are able to handle joins very efficiently. However, rare and occasional circumstances may allow some degree of denormalization so processing speed can be increased. Furthermore, the database design process could, in some cases, introduce some small degree of redundant data in the model. This, in effect, creates "denormalized" relations.

6.8 CHECK YOUR PROGRESS

1. Every relation which is in 3NF is in BCNF.(T/F)
2. Dependencies that are based on only a part of a composite primary key are called transitive dependencies..(T/F)
3. A functional dependency of the form $x \rightarrow y$ is _____ if $y \subset x$
4. If R is a 3NF then every non-prime attribute is _____ and non-transitively dependent on any key.
5. If a relation is in 2NF and 3NF forms then
 - (a) No non-prime attribute is functionally dependent on other non-prime attributes.
 - (b) No non-prime attribute is functionally dependent on prime attributes.
 - (c) All attribute are functionally independent.
 - (d) Prime attribute is functionally independent of all non-prime attributes.
6. Match the following:
 1. 2NF (a) Transitive dependencies eliminated
 2. 3NF (b) multivalued attribute removed
 3. 4NF (c) contains no partial functional dependencies
 4. 5NF (d) contains no join dependency

(a) 1. (a), 2. (), 3. (b), 4. (d)

(b) 1. (d), 2. (c), 3. (a), 4. (b)

(c) 1. (c), 2. (a), 3. (b), 4. (d)

(d) 1. (a), 2. (b), 3. (c), 4. (d)

7. What is a canonical cover??

8. What is multi-valued dependency?

9. What is a functional dependency?

10. What is de-normalization?

Answers to Check your progress:

1. F

2. F

3. trivial

4. Independently

5. b

6. c

7. A cover F_c for a set of FD's F is known as canonical cover for F if F logically implies all dependencies in F_c , and F_c logically implies all dependencies

8. Let R be a relation having attributes or sets of attributes A , B , and C . There is a multivalued dependency of attribute B on attribute A if and only if the set of B values associated with a given A value is independent of the C values.

9. Functional Dependency (FD) is a constraint that determines the relation of one attribute to another attribute in a Database Management System (DBMS). Functional Dependency helps to maintain the quality of data in the database. It plays a vital role to find the difference between good and bad database design.

10. Denormalization is the process of adding precomputed redundant data to an otherwise normalized relational database to improve read performance of the database. Normalizing a database involves removing redundancy so only a single copy exists of each piece of information.

6.9 SUMMARY

Normalization is a technique used to design tables in which data redundancies are minimized. The first three normal forms (1NF, 2NF, and 3NF) are most commonly encountered. From a structural point of view, higher normal forms are better than lower normal forms, because higher normal forms yield relatively fewer data redundancies in the database. Almost all business designs use 3NF as the ideal normal form. A special, more restricted 3NF known as Boyce-Codd normal form, or BCNF, is also used. A table is in 1NF

when all key attributes are defined and when all remaining attributes are dependent on the primary key. However, a table in 1NF can still contain both partial and transitive dependencies. (A partial dependency is one in which an attribute is functionally dependent on only a part of a multiattribute primary key. A transitive dependency is one in which one attribute is functionally dependent on another non-key attribute.) A table with a single-attribute primary key cannot exhibit partial dependencies. A table is in 2NF when it is in 1NF and contains no partial dependencies. Therefore, a 1NF table is automatically in 2NF when its primary key is based on only a single attribute. A table in 2NF may still contain transitive dependencies. A table is in 3NF when it is in 2NF and contains no transitive dependencies. Given that definition of 3NF, the Boyce-Codd normal form (BCNF) is merely a special 3NF case in which all determinant keys are candidate keys. When a table has only a single candidate key, a 3NF table is automatically in BCNF. A table that is not in 3NF may be split into new tables until all of the tables meet the 3NF requirements. Normalization is an important part—but only a part—of the design process. As entities and attributes are defined during the ER modelling process, subject each entity (set) to normalization checks and form new entity (sets) as required. Incorporate the normalized entities into the ERD and continue the iterative ER process until all entities and their attributes are defined and all equivalent tables are in 3NF. A table in 3NF might contain multivalued dependencies that produce either numerous null values or redundant data.

Therefore, it might be necessary to convert a 3NF table to the fourth normal form (4NF) by splitting the table to remove the multivalued dependencies. Thus, a table is in 4NF when it is in 3NF and contains no multivalued dependencies. The larger the number of tables, the more additional I/O operations and processing logic required to join them. Therefore, tables are sometimes denormalized to yield less I/O in order to increase processing speed. Unfortunately, with larger tables, you pay for the increased processing speed by making the data updates less efficient, by making indexing more cumbersome and by introducing data redundancies that are likely to yield data anomalies. In the design of production databases, use denormalization sparingly and cautiously.

6.10 KEYWORDS

- **Atomic attribute** - Attributes that are not divisible are called simple or atomic attributes.
- **Boyce-Codd normal form**– Boyce-Codd normal form (BCNF) is merely a special 3NF case in which all determinant keys are candidate keys.

- **First normal form (1NF)** - 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.
- **Fourth normal form(4NF)** - a table is in 4NF when it is in 3NF and contains no multivalued dependencies.
- **Second normal form (2NF)**- A relation schema R is in 2NF if every nonprime attribute A in R is fully functionally dependent on the primary key of R.
- **Third normal form(3NF)** - 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.
- **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.
- **Prime attribute**- An attribute of relation schema R is called a prime attribute of R if it is a member of some candidate key of R.

6.11 QUESTIONS FOR SELF-STUDY

1. What do you understand by functional dependency? Explain with examples.
2. Compare and contrast Full/Partial/Transitive dependencies.
3. Describe pros and cons of database normalization.
4. Explain the concept of the following forms with suitable examples : 1NF, 2NF, 3NF, BCNF, 4NF.

6.12 REFERENCES

1. Coronel, C., & Morris, S. (2016). Database systems: design, implementation, & management. Cengage Learning.
2. Gupta, S. B., & Mittal, A. (2009). Introduction to database management system.Laxmi Publications, Ltd..
3. Ramakrishnan, R., &Gehrke, J. (2000). Database management systems.McGraw-Hill.

UNIT-7: INTRODUCTION TO STRUCTURED QUERY LANGUAGE (SQL)

STRUCTURE

- 7.0 Objectives
- 7.1 Introduction
- 7.2 Structured Query Language (SQL)
- 7.3 Characteristics of SQL
- 7.4 Advantages of SQL
- 7.5 Parts (Components) of SQL Language
- 7.6 Basic Data Types
- 7.7 Data Manipulation in SQL
- 7.8 Data Definition Language (DDL)
- 7.9 Aggregate functions
- 7.10 Data Control Language (DCL)
- 7.11 Check Your Progress
- 7.12 Summary
- 7.13 Keywords
- 7.14 Questions for self-study
- 7.15 References

7.0 OBJECTIVES

After studying this unit, you will be able to:

- ✓ The basic commands and functions of SQL
- ✓ Various option in SELECT statements
- ✓ Discuss basic data types available
- ✓ Differentiate DDL and DML
- ✓ Explain the use of DCL
- ✓ How to use SQL to query a database for useful information

7.1 INTRODUCTION

In the unit 4, we have discussed about the relational algebra which is used for relational query. It provides a powerful set of operations to specify queries. But this type of language is expensive to implement and use. In this chapter, we discuss two query languages

that are widely used in the various commercial RDBMS's. The first one is the Structured Query Language (SQL) and the second one is the Query-By-Example (QBE).

7.2 STRUCTURED QUERY LANGUAGE(SQL)

The name SQL pronounced as “ess-cue-ell” or ‘sequel’ is the abbreviation for structured query language. The SQL consists of a set of facilities for defining, accessing and managing relational databases. All tasks related to relational data management-creating tables, querying the database, deleting, granting access to users etc., can be done using SQL. It has been accepted as an American standard by American National Standards Institute (ANSI) and is a Federal Information Processing Standard (FIPS). It is also an international standard recognized by the ISO. The first commercial DBMS that supported SQL was Oracle in 1979. SQL statements can be invoked either interactively in a terminal session or by embedding them in application programs.

7.3 CHARACTERISTICS OF SQL

The following are the important characteristics of SQL.

1. SQL is extremely flexible.
2. SQL uses a free form syntax that gives the user the ability to structure SQL statements in a way best suited.
3. It is a free formatted language, i.e., there is no need to start SQL statements in a particular column or to be finished in a single line.
4. It has relatively few commands.
5. It is a non-procedural language.

7.4 ADVANTAGES OF SQL

The advantages of SQL are as follows:

1. SQL is a high level language that provides a greater degree of abstraction than procedural languages. The programmer has to specify what data is needed but need not to specify, how to retrieve it.
2. SQL is a unified language. The same language can be used to define data structures, querying data, control access to the data, insert, delete and modify occurrences of the data and so on.

3. All the programs written in SQL are portable, thus they can be moved from one database to another with very little modification. Such porting could be required when DBMS needs to be upgraded or changed.
4. The language is simple and easy to learn. It can handle complex situations very efficiently.
5. The language has sound theoretical base and there is no ambiguity about the way a query will interpret the data and produce the results. Thus the results to be expected are well defined.
6. SQL processes sets-of-records rather than just one record-at-a time. This set-at-a time feature of the SQL makes it more powerful.
7. SQL as a language is independent of the way it is implemented internally. This is because SQL specifies what is required and not how it should be done.
8. SQL enables its users to deal with a number of database management systems where it is available.

7.5 PARTS (COMPONENTS) OF SQL LANGUAGE

The SQL language is mainly divided into four major parts. The four parts are further divided into subparts. The major parts and subparts are as follows:

7.5.1 Data-Definition Language (DDL)

The SQL DDL provides commands for defining the relations, deleting the relations and modifying the existing relation schemas.

- View Definition Language (VDL) : The SQL DDL provide commands for defining and dropping the views.
- Integrity: The SQL DDL provide commands for specifying integrity constraints that must be satisfied by the data stored in the database.
- Authorization: The SQL DDL provide commands for specifying access rights to the relations and views.

7.5.2 Data Manipulation Language (DML)

The SQL DML provides a query language. This query language is based on relational algebra and tuple relational calculus. This contains commands to insert tuples into the database, to delete tuples from the database and to modify/update tuples in the database.

7.5.3 Data Control Language or Transaction Control Language (DCL or TCL)

The SQL DCL provide commands that help the DBA to control the database such as commands to grant or revoke privileges to access the database and to store or remove transactions that would affect the database.

7.5.4 Embedded SQL and Dynamic SQL

- Embedded SQL defines the way the SQL statements can be embedded within general purpose programming languages like C, C++, Cobol, Pascal etc. The language in which SQL queries are embedded is referred to as a host language. The SQL queries embedded in the host language constitute embedded SQL.
- Dynamic SQL allows programs to construct and submit SQL queries at run time.

To show the working of DML, DDL and DCL commands, the company database is used. The relational schema is shown in Figure 7.1 with descriptions and primary key attributes underlined. The corresponding tables are shown in Figure 7.2 and Figure 7.3, where DID is the foreign key in Emp table.

(i) Emp. (Employee) with attributes <u>EID</u> (Employee ID), Name, Salary, Hire_date, Job, <u>DID</u> (Department ID), MID (Manager ID).
(ii) Dept. (Department) with attributes <u>DID</u> (Department ID), DName (Department Name), Loc (Location), MID (Manager ID).

Fig. 7.1.Relational schema.

Emp (Employee)

EID	Name	Salary	Hire-date	Job	DID	MID
701	Deepak	8000	5-Jan-2001	Analyst	30	707
702	Naresh	9000	10-Jan-2001	Manager	10	707
703	Sumesh	7000	5-Feb-2001	Salesman	20	705
704	Aditya	9000	27-Nov-2003	Analyst	30	707
705	Lalit	6500	8-Oct-2002	Manager	20	707
706	Amit		4-Nov-2004	Clerk	10	702
707	Vishal	9500	1-Jan-2001	Manager	30	
708	Sumit	8000	6-Jan-2006	Accountant	10	702

Fig. 7.2.Employee table.

Dept (Department)

DID	DName	Loc	MID
10	Accounts	Bangalore	702
20	Sales	Delhi	705
30	Research	Pune	707
40	Developing	Hyderabad	

Fig. 7.3.Department table.

7.6 BASIC DATA TYPES

The SQL supports a variety of data types as shown in Table 7.1.

Table 7.1. Basic data types

Datatype	Description	Size
Number(p, s)	It is used to store numeric data types. p stands for precision (total number of decimal digits) and s stands for scale (total number of digits after decimal point).	Range of p is from 1 to 38. And s is from -84 to 127.
Date	It is used to store date and time values.	Range of date is from Jan 1, 47 B.C. to Dec. 31, 9999 A.D.
Char(size)	It is used to store fixed size character data.	Range of char is 1 (By default) to 2000 bytes.
Varchar2(size)	It is used to store variable size character data.	Range of varchar2 is 1 (By default) to 4000 bytes.
Long	It is used to store variable size character data.	Range of long is upto 2 GB.
Clob	It is used to store variable size character data.	Range of clob is upto 4 GB
Raw(size)	It is used to store fixed binary data.	Maximum size is upto 2000 bytes.
Long raw	It is used to store variable binary data.	Maximum size is upto 2 GB.

7.7 DATA MANIPULATION IN SQL

SQL has one basic statement for retrieving information from the database: The SELECT statement. SQL also provides three other DML statements to modify the database. These statements are: update, delete and insert.

The basic form of the SELECT statement is formed of three clauses SELECT, FROM, and WHERE, having the following form:

```
SELECT < attribute list >
FROM < table list >
WHERE < condition >
```

In this form,

- <attribute list > is the list of attribute names whose values are to be retrieved by the query.
- <table list > is the list of relation names required to process the query.
- <condition> is a conditional expression that identifies the tuples to be retrieved by the query.

The following examples show the working of SELECT statement with different options.

The INSERT, UPDATE and DELETE statements are also described in the following subsections.

7.7.1 Select Statement

Select statement is used to retrieve information from table.

```
Syntax :          select < column list >
from< table name >.
```

Example 1: To display all department ID and the Department name, the query is

```
select DID, DName from Dept ;
```

DID	DName
10	Accounts
20	Sales
30	Research
40	Developing

Example 2: To select all columns use “*”.

```
select * from Dept;
```

DID	DName	Loc	Manager-ID
10	Accounts	Bangalore	702
20	Sales	Delhi	705
30	Research	Pune	707
40	Developing	Hyderabad	

(i) Column Alias: You can give name to columns of your choice by using keyword “As” (gives column name in upper-case letter) or by using “ ” (gives column name as specified in query).

Example 3:

```
select DID As Department_ID, DName from Dept ;
```

Department-ID	DName
10	Accounts
20	Sales
30	Research
40	Developing

(ii) Concatenation Operator (||): It is used to concatenate two or more columns.

Example 4:

```
select DName || Loc As department from Dept ;
```

DEPARTMENT
AccountsBangalore
SalesDelhi
ResearchPune
DevelopingHyderabad

(iii) Literal Character Strings: A literal is a number, character or date that can be included in columns. Character and date literals must be enclosed with single quotation marks (‘ ‘).

Example 5:

```
select DName || branch is situated at ' || Loc As department from Dept ;
```

DEPARTMENT
Accounts branch is situated at Bangalore
Sales branch is situated at Delhi
Research branch is situated at Pune
Developing branch is situated at Hyderabad

(iv) Eliminating Duplicate Rows : To eliminate duplicate rows, the keyword 'DISTINCT' is used.

Example 6 : SELECT salary

FROM Emp;

↓

Salary
8000
9000
7000
9000
6500
9500
8000

SELECT DISTINCT salary

FROM Emp;

↓

Salary
8000
9000
7000
6500
9500

(v) Arithmetic Operators and NULL Values : SQL provides arithmetic operators to perform calculations. Arithmetic operators with their precedence are

Description	Operator
Multiply	*
Divide	/
Add	+
Subtract	-

A null value is unknown value and it is different from zero. Any arithmetic operation with null value gives null results.

Example 7 : Suppose you want to increase salary of each employee by 500.

```
SELECT EID, salary + 500 "New Salary"
FROM Emp;
```

EID	New salary
701	8500
702	9500
703	7500
704	9500
705	7000
706	
707	10000
708	8500

(vi) **Where Clause :** WHERE clause is used to select particular rows from table.

Syntax: select <column list>from <table name>where <condition>.

Example 8: List the name of employees having salary ` 9000.

```
select name from empwhere salary = 9000;
```

Name
Naresh
Aditya

(a) Comparison or relational operators : The relational operators provided by SQL areas follows.

Operator	Description
=	Equal to
>	Greater than
<	Less than
>=	Greater than equal to
=<	Less than equal to
<>	Not equal to

Example 9 : List the name of employees having salary not equal to ` 9000.

```
select name from empwhere salary <> 9000;
```

Name
Deepak
Sumesh
Lalit
Amit
Vishal
Sumit

(b) Special operators: Special operators provided by SQL are as follows:

Operator	Description
IN	Checking the value within a set
BETWEEN	Checking the value within a range
LIKE	Matching the pattern of characters
IS NULL	Checking the null value

Example 10 : List name and EID of employees having salary ` 8000 or ` 9500.

select EID, name from Empwhere salary IN (8000, 9500);

EID	Name
701	Deepak
707	Vishal
708	Sumit

Example 11 : List the EID and names of employees who were hired by company from 5–Feb–2001 to 1–Jan–2006.

SELECT EID, Name FROM Emp WHERE Hire_Date in (5-Feb-2001, 1-Jan-2006);

EID	Name
703	Sumesh
704	Aditya
705	Lalit
706	Amit

Example 12 : List the EID and names of employees having MID equal to null.

select EID, name from Empwhere MID IS NULL;

EID	Name
707	Vishal

7.7.2 Insert Statement

Insert statement is used to insert or add new rows in table.

Syntax : INSERT INTO <table name> (column 1, column 2,....., column n)
VALUES (value 1, value 2,....., value n);

*Only a single row is inserted at a time

*Name of columns can be given in any order.

7.7.3 Update Statement

Update statement is used to modify the values of existing rows.

Syntax : UPDATE <table name>
SET <(column 1 = value 1), (column 2 = value 2),....., (column n =value n)>
WHERE <condition>;

*All rows in the table satisfies the condition are updated.

7.7.4 Delete Statement

Delete statement is used to remove existing rows from table.

Syntax : DELETE FROM <table name>WHERE <condition>;

Example 12: Consider table Dept. (Figure 7.3). Suppose it is empty.

12 (a) Insert new rows in Dept. table.

```
Insert INTO Dept (DID, DName, Loc, MID)
VALUES (10; 'Accounts'; 'Bangalore', 708);
Insert INTO Dept (DName, Loc, DID, MID)
VALUES ('Accounts', Hyderabad; 40, NULL);
Insert INTO Dept (DID, DName, MID, Loc)
VALUES (50, 'Testing', 709, 'Delhi');
```

12 (b) Update the MID of DName Accounts to 702
UPDATE Dept
SET MID = 702
WHERE DName = 'Accounts';

12 (c) Delete row from Dept having DName = Testing
DELETE FROM Dept
WHERE DName = 'Testing';

12 (d) Select *from Dept;

DID	DName	LOC	MID
10	Accounts	Bangalore	702
40	Developing	Hyderabad	

7.8 DATA DEFINITION LANGUAGE (DDL)

SQL DDL commands are used to create, modify or remove database structures including tables. These commands have an immediate effect on the database, and also record information in the data dictionary. The following examples show working of some of the DDL commands.

7.8.1 Create Table

Create table statement is used to create new tables in database.

```
Syntax : CREATE TABLE <table name>
        (<column name <data type (size)>,
         ----- );
```

Example 51 : Create a table Dept with attributes DID (Department ID), DName (Department Name), Loc (Location), MID (Manager ID).

```
CREATE TABLE Dept
(DID Number(4),
DName Varchar2(20),
Loc Varchar2(20),
MID Number(4));
```

1. **Column Constraints:** Constraints are rules which are forced on database to follow them for consistency purpose

Constraint	Description
Not Null	By using this constraint, null value to a particular attribute cannot be assigned.
Unique	Each value of an attribute must be unique
Primary Key	Value at each column must be unique and Not Null
Foreign Key	Particular attribute must follow referential integrity.
Check	Specified condition must be true for attribute

*Attribute or Column is synonyms.

2. **DEFAULT Value:** It assigns a default value to an attribute if at the time of insertion of new row, no value is given to that attribute.

Example : Create a table student with attributes SID (student ID), Name, Fee and default value of fee is zero.

```

CREATE TABLE Student
( SID Number(4),
  Name Varchar2(20),
  Fee Number(4) DEFAULT 0);

```

It is better to give name to constraints so that later you can drop them easily.

Example 53 : Repeat ex-2 with one more constraint, that the name of student must not be null.

```

CREATE TABLE Student
( SID Number(4),
  Name Varchar2(20) NOT NULL,
  Fee Number(4) DEFAULT 0);

```

← System named

OR

```

CREATE TABLE Student
(
  SID Number(4),
  Name Varchar2(20)
  CONSTRAINT Name_not_null
  NOT NULL,
  Fee Number(4) DEFAULT 0);

```

← User named

Type of constraint Name of constraint

Example : Create a table, Dept, with attributes DID, DName, Loc and MID. DName must be unique.

```

CREATE TABLE Dept
( DID Number(4),
  DNameVarchar2(20),
  Constraint dname-unique
  U NIQUE
  LocVarchar2(20),
  MID Number(4));

```

OR

```

CREATE TABLE Dept
( DID Number(4),
  DNameVarchar2(20),
  LocVarchar2(20),

```

```
MID Number(4),  
CONSTRAINT dname_unique UNIQUE (DName));
```

7.8.2 Alter Table Statement

Alter table statement is used to add or drop a constraint. It can also be used to disable or enable any constraint.

Syntax:

(i) To add a constraint:

```
ALTER Table <table name>  
ADD CONSTRAINT <condition>;
```

(ii) To drop a constraint:

```
ALTER Table <table name>  
DROP CONSTRAINT <constraint name> CASCADE CONSTRAINTS;
```

(iii) To enable a constraint:

```
ALTER Table <table name>  
ENABLE CONSTRAINT <constraint name>;
```

(iv) To disable a constraint:

```
ALTER Table <table name>  
DISABLE CONSTRAINT <constraint name> CASCADE;
```

Alter table statement is also used to add or drop columns of tables and to modify name and attributes of an existing column.

(v) To add new column:

```
ALTER TABLE <table name>  
ADD (<column name><data type(size)>);
```

(vi) To drop a column:

```
ALTER TABLE <table name>  
DROP COLUMN <column name>;
```

(vii) To modify a column:

```
ALTER TABLE <table name>  
MODIFY (<column name><new data type |new size| new default value>);
```

7.8.3 Describe Statement

It describes the structure of table.

Syntax : DESCRIBE <table name>;

Example: Describe the structure of table Emp.

DESCRIBE Emp;

Name	NULL	Type
EID	NOT NULL	NUMBER(4)
NAME		VARCHAR2(30)
SALARY		NUMBER(6)
HIRE-DATE	NOT NULL	DATE
JOB		VARCHAR2(20)
DID		NUMBER(4)
MID		NUMBER(4)

Example :

(a) Create a table student with attributes SID, Name and Address.

```
CREATE TABLE Student
(
  SID Number(4),
  NAME Varchar2(20),
  ADDRESS VARchar2(25));
```

(b) Add a new column fee in table student.

```
ALTER TABLE Student
ADD (FEE NUMBER(4));
```

(c) Drop column address from table student

```
ALTER TABLE Student
DROP Column      address;
```

(d) Modify column NAME (increase size to 30)

```
ALTER TABLE Student
Modify      (name Varchar2(30));
```

(e) Describe the structure of table student.

DESCRIBE Student;

Name	NULL	Type
SID		NUMBER(4)
NAME		VARCHAR2(30)
FEE		NUMBER(4)

7.8.4 Drop Statement

Drop table statement is used to remove table from database.

Syntax : DROP TABLE <table name>;

Example: Remove table student from database.

DROP TABLE Student;

7.9 AGGREGATE PROCESSING

SQL can perform various mathematical summaries for you, such as counting the number of rows that contain a specified condition, finding the minimum or maximum values for some specified attribute, summing the values in a specified column, and averaging the values in a specified column.

Some Basic SQL Aggregate functions

FUNCTION	OUTPUT
COUNT	The number of rows containing non-null values
MIN	The minimum attribute value encountered in a given column
MAX	The maximum attribute value encountered in a given column
SUM	The sum of all values for a given column
AVG	The arithmetic mean (average) for a specified column

To illustrate another standard SQL command format, most of the remaining input and output sequences are presented using the Oracle RDBMS.

COUNT

The COUNT function is used to tally the number of non-null values of an attribute. COUNT can be used in conjunction with the DISTINCT clause.

MAX and MIN

The MAX and MIN functions help you find answers to problems such as the:

- Highest (maximum) price in the PRODUCT table.
- Lowest (minimum) price in the PRODUCT table.

The MAX and MIN aggregate functions can also be used with date columns. For example, to find out the product that has the oldest date, you would use MIN(P_INDATE). In the same manner, to find out the most recent product, you would use MAX(P_INDATE).

SUM

The SUM function computes the total sum for any specified attribute, using whatever condition(s) you have imposed.

AVG

The AVG function format is similar to those of MIN and MAX and is subject to the same operating restrictions.

7.10 DATA CONTROL LANGUAGE (DCL)

The SQL specifies that a transaction begins automatically when an SQL statement is executed.

The following four statements show the different ways to end the transaction and the next two statements show different privileges on database to users.

7.9.1 Commit Statement

A transaction is completed successfully after commit. Commit statement is used to make data changes permanent to database.

Syntax : COMMIT;

7.9.2 Rollback

Rollback statement is used to terminate current transaction and discarding all data change pending due to that transaction.

Syntax : ROLLBACK;

7.9.3 Savepoint

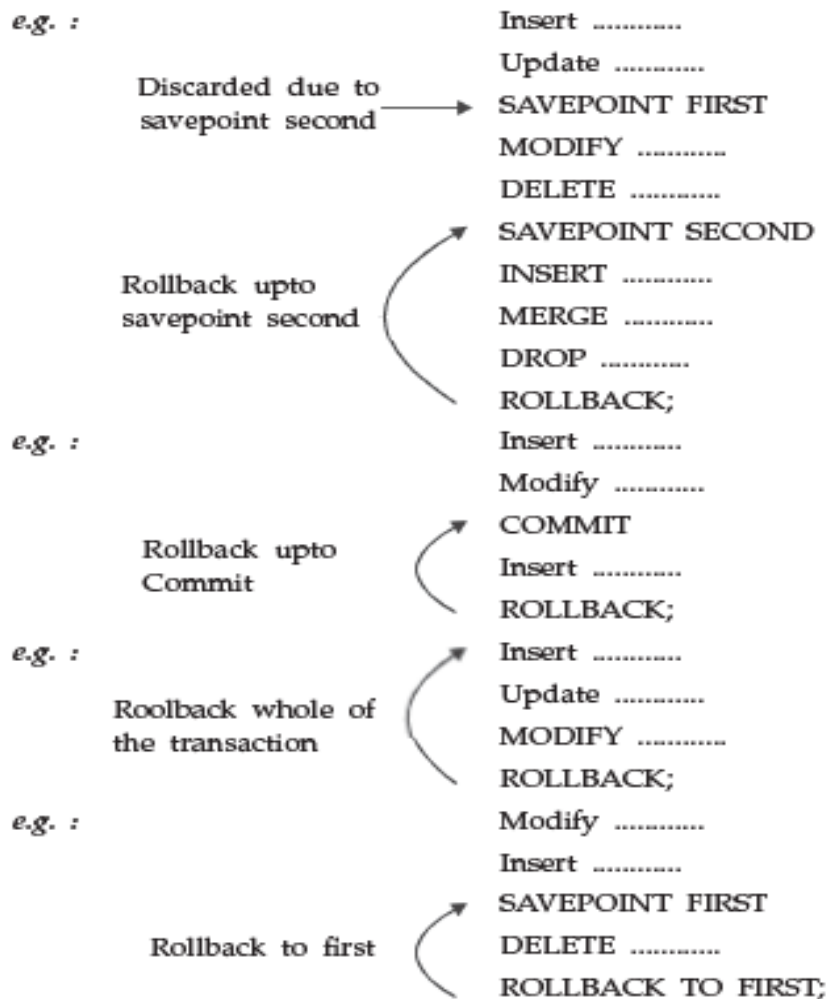
It is used to partially commit the current transaction and put a savepoint at that position.

Syntax : SAVEPOINT <name>;

If second savepoint will be created within same transaction then earlier savepoint is automatically discarded by database.

7.9.4 Rollback to Savepoint

It is used to partially rollback any transaction and pending data changes upto specified savepoint are discarded.



7.9.5 Grant Statement

Grant statement is used to give different permissions on different portions of database to different users. In a multi-user database management system, it is required to grant different permissions for security purposes.

7.9.6 REVOKE Statement

REVOKE statement is used to take away any authority from a user that was granted earlier.

Syntax: REVOKE <privilege-list>| ALL
 ON <table name> [(column-comma-list)]
 FROM <user-list>| PUBLIC

Ex.: REVOKE the UPDATE permission on table Dept from Rohan.

```
REVOKE U PDATE
ON Dept
FROM Rohn;
```

Ex.: REVOKE INSERT and UPDATE permission on Name and EID columns of tableEmp from all users.

REVOKE INSERT, UPDATE (Name, EID)

ON Emp

FROM PUBLIC;

7.11 CHECK YOUR PROGRESS

1. In SQL, attributes declared UNIQUE cannot have NULL values.(T/F)
2. A single query cannot have WHERE, GROUP BY, HAVING and ORDER BY clauses simultaneously (T/F)
3. The _____ operator is used to specify a range of values
4. To provide a condition on group of tuples associated with each value of the Grouping attribute, _____ clause is used.
5. Database table by name Loan_Records is given below.

Borrower	Bank_Manager	Loan_Amount
Ramesh	Sunderajan	10000.00
Suresh	Ramgopal	5000.00
Mahesh	Sunderajan	7000.00

What is the output of the following SQL query?

```
SELECT Count(*)
FROM ( (SELECT Borrower, Bank_Manager
FROM Loan_Records) AS S
NATURAL JOIN (SELECT Bank_Manager,
Loan_Amount
FROM Loan_Records) AS T );
```

A. 3 B. 9 C. 5 D. 6

6. The relation book (title, price) contains the titles and prices of different books. Assuming that no two books have the same price, what does the following SQL query list?

Select title from book as B where (select count(*) from book as T where T.price > B.price) < 5

- (a) Titles of the four most expensive books
 - (b) Title of the fifth most inexpensive book
 - (c) Title of the fifth most expensive book
 - (d) Titles of the five most expensive books
7. What are the Characteristics of SQL?
 8. What are the basic datatypes of SQL?
 9. What is a select statement? Name the two required components of a select statement.
 10. What are various special operators provided by SQL?

Answers to check your progress:

1. T
2. F
3. BETWEEN
4. GROUP BY
5. C
6. D

7.Characteristics of SQL are:

- SQL is extremely flexible.
- SQL uses a free form syntax that gives the user the ability to structure SQL statements
- It is a free formatted language, i.e., there is no need to start SQL statements in a particular column or to be finished in a single line.
- It has relatively few commands.
- It is a non-procedural language.

8. Number, Date, Char, Varchar2, Long

9. Select statement is used to retrieve information from table.

select< column list >from < table name >.

10. IN, ANY, ALL, BETWEEN, LIKE, IS NULL

7.12 SUMMARY

- The SQL commands can be divided into two overall categories: data definition language (DDL) commands and data manipulation language (DML) commands.
- The ANSI standard data types are supported by all RDBMS vendors in different ways. The basic data types are NUMBER, INTEGER, CHAR, VARCHAR, and DATE.
- The SELECT statement is the main data retrieval command in SQL. A SELECT statement has the following syntax:

```
SELECT columnlist  
FROM tablelist  
[WHERE conditionlist ]  
[GROUP BY columnlist ]  
[HAVING conditionlist ]  
[ORDER BY columnlist [ASC | DESC] ] ;
```

- The column list represents one or more column names separated by commas. The column list may also include computed columns, aliases, and aggregate functions. A computed column is represented by an expression or formula (for example, P_PRICE * P_QOH). The FROM clause contains a list of table names or view names
- The WHERE clause can be used with the SELECT, UPDATE, and DELETE statements to restrict the rows affected by the DDL command. The condition list represents one or more conditional expressions separated by logical operators (AND, OR, and NOT). The conditional expression can contain any comparison operators (=, >, <, >=, <=, and <>) as well as special operators (BETWEEN, IS NULL, LIKE, IN, and EXISTS)..
- Aggregate functions (COUNT, MIN, MAX, and AVG) are special functions that perform arithmetic computations over a set of rows. The aggregate functions are usually used in conjunction with the GROUP BY clause to group the output of aggregate computations by one or more attributes. The HAVING clause is used to restrict the output of the GROUP BY clause by selecting only the aggregate rows that match a given condition.
- The ORDER BY clause is used to sort the output of a SELECT statement. The ORDER BY clause can sort by one or more columns and can use either ascending or descending order.

- You can join the output of multiple tables with the SELECT statement. The join operation is performed every time you specify two or more tables in the FROM clause and use a join condition in the WHERE clause to match the foreign key of one table to the primary key of the related table. If you do not specify a join condition, the DBMS will automatically perform a Cartesian product of the tables you specify in the FROM clause.

7.13 KEYWORDS

- **EXISTS-** The EXISTS special operator can be used whenever there is a requirement to execute a command based on the result of another query. That is, if a subquery returns any rows, run the main query; otherwise, don't.
- **GROUP BY** - Frequency distributions can be created quickly and easily using the GROUP BY clause within the SELECT statement.
- **HAVING-** A particularly useful extension of the GROUP BY feature is the HAVING clause. The HAVING clause operates very much like the WHERE clause in the SELECT statement. However, the WHERE clause applies to columns and expressions for individual rows, while the HAVING clause is applied to the output of a GROUP BY operation.
- **Inner query** - is a query that is embedded (or nested) inside another query.
- **LIKE** - Used to check whether an attribute value matches a given string pattern.

7.14 QUESTIONS FOR SELF STUDY

1. Write a note on SQL.
2. Mention characteristics of SQL.
3. List the advantages of SQL.
4. What are the data types supported by SQL.
5. Explain with suitable examples the various commands used in querying and manipulating the table through DML.
6. Explain with suitable examples the various commands used in querying and manipulating the table through DDL.
7. Differentiate DDL and DML.
8. Write a note on DCL.

7.15 REFERENCES

1. Coronel, C., & Morris, S. (2016). Database systems: design, implementation, & management. Cengage Learning.
2. Gupta, S. B., & Mittal, A. (2009). Introduction to database management system. Laxmi Publications, Ltd..
3. Ramakrishnan, R., & Gehrke, J. (2000). Database management systems. McGraw-Hill.

UNIT-8: DATA WAREHOUSE AND DATA MINING

STRUCTURE

8.0 Objectives

8.1 Introduction

8.2 Data warehouse

8.2.1 Distinctive Characteristics of Data Warehouses

8.2.2 Difference between Database and Data Warehouse

8.2.3 Data Warehouse Architecture

8.2.4 Data Warehouse Components

8.2.5 Advantages of Data Warehouse

8.2.6 Disadvantages/Limitations of Data Warehouse

8.3 OLTP (On-line Transaction Processing)

8.3.1 Limitations of OLTP

8.4 OLAP (On-Line Analytical Processing)

8.4.1 Codd's OLAP Characteristics

8.4.2 Difference between OLTP and OLAP

8.5 Data Mining

8.6 Comparison of Data Mining and Structured Query Language (SQL)

8.7 Comparison of Data Mining and Data Warehousing

8.8 Check your progress

8.9 Summary

8.10 Keywords

8.11 Questions for self-study

8.12 References

8.0 OBJECTIVES

After studying this unit, you will be able to learn:

- ✓ Analyse data warehouse architecture and its components
- ✓ Explain advantages and disadvantages of data warehouse
- ✓ Analyse data mining tools, models and techniques
- ✓ Differentiate OLAP and OLTP
- ✓ Compare data mining and SQL

8.1 INTRODUCTION

It is general consensus among the Business gurus that in today's competitive market readily available high quality information is vital in business. Consider the comments from a management expert.

“Information is pivotal in today's business environment. Success is dependent on its early and decisive use. A lack of information is a sure sign for failure. The rapidly changing environment in which business operates demands ever more immediate access to data”.

The above comments strongly emphasis on information and the recent advances in I.T., we might expect most organizations to have highly developed systems for delivering information to managers and other users. There are two major reasons for the information gap that has been created in most organizations.

The first reason is that the organizations have developed fragmented information systems and their supporting databases for many years. In this environment, it is very difficult for managers to locate and use accurate information.

The second reason is that most systems are developed to support operational processing (It captures, stores, and manipulates data to support daily operations of the organization), with little or no thought given to the information processing (It is the analysis of summarized data or other forms of information to support decision making) or analytical tools needed for decision making.

The data warehouses bridging this information gap and consolidate and integrate information from many different sources and arrange it in a meaningful format for making accurate business decisions. It meets these needs without disturbing operational processing.

While the benefits of the data warehouse have been accepted for some years, the major successes mostly have been in large organizations. The reality is that there is still the need for a database focused upon the operational processing needs of the organization. At the same time, it is important to design and construct a database system or we can say data warehouse, which will answer the growing, present and future managerial needs of the organization.

8.2 DATA WAREHOUSE

Definition: Data warehouse is a collection of data designed to support management decision making.

Another definition is: Data warehousing is the process, whereby, organizations extract meaning from their informational assets through the use of data warehouses. (Barguin, 1996).

Another definition is: A data warehouse is a subject oriented, integrated, time variant, nonvolatile collection of data used in support of management decision making processes. (Inmon and Hackathorn, 1994). The meaning of the key terms in this definition is as follows :

- (i) **Subject oriented:** In data warehouse, data is organized and optimized according to specific subjects or areas of interest of the organization rather than simply as computer files. Examples of major subject areas include Customers, Products, Accounts, and Transactions etc. It provides capability to provide answers to various queries coming from various functional areas within an organization.
- (ii) **Integrated:** Data warehouse is integrated. A single source of information for and about understanding multiple areas of interest. It provides information about a variety of subjects at one place. The input data comes from various sources in inconsistent form. Data warehouse refines the data to make it consistent and provides a unified view of overall organizational data to users. So, data warehouse is a centralized depository of data of the entire organization which helps in better understanding of organisation's operations for strategic business opportunities and hence increase decision making capabilities.
- (iii) **Non-volatile :** Data warehouse contains stable information that doesn't change each time an operational process is executed. The data in the data warehouse are loaded and refreshed from operational systems, but cannot be changed by end users. New data is always added as a supplement to database, rather than a replacement.
- (iv) **Time-variant :** The data in Data warehouse is only accurate and valid at some point in time or over some time interval. Data contains a time-dimension so that they may be used as a historical record of business'. i.e., sales statistics of previous week.
- (v) **Accessible :** The primary purpose of a data warehouse is to provide readily accessible information to end users.

A number of separate technologies have come together to make Data warehousing possible to implement. However, it may be deployed physically, the data warehouse may be viewed as a single, consistent state of information with appropriate tools to provide valuable information about a business.

8.2.1 Distinctive Characteristics of Data Warehouses

The data warehouses have many characteristics that make them different from others.

- It typically integrates several resources e.g., sales databases from various regions/states/years.
- It requires more historical data than generally maintained in operational databases.
- It must be optimized for access to very large amounts of data.
- It is mostly read-accessed and rarely write-accessed.
- Data may be more coarse grained than in operational databases.
- Data warehouses are maintained separately from operational data.
- It is based on client-server architecture.
- It provides multi-user support.
- It is capable of handling dynamic sparse matrices.
- It provides multidimensional conceptual view.
- It supports unrestricted cross-dimensional operations.
- It maintains transparency.
- It provides consistent and flexible reporting performance.
- Its having unlimited dimensions and aggregation levels.

8.2.2 Difference between Database and Data Warehouse

There are many differences in database and data warehouse. These differences are in the organization and data stored in both. The various differences are as follows:

S.No.	Database	Data Warehouse
1.	A database is a collection of related data. The database system is a collection of database and DBMS.	A data warehouses is a collection of information as well as a supporting system.
2.	The databases maintain a balance between efficiency in transaction processing and supporting query requirements <i>i.e.</i> , they cannot be further optimized for the applications such as OLAP, DSS and data mining.	A data warehouse is generally optimized to access from a decision maker's needs. These are designed specifically to support efficient extraction, processing and presentation for analytical and decision-making purpose.
3.	Database are generally small as compared to Data warehouses and contain data from single source.	Data warehouses generally contain very large amounts of data from multiple sources that may include databases from different data models and sometimes files acquired from independent systems and platforms.
4.	Multidatabases provide access to disjoint and usually heterogeneous databases and are volatile.	Data warehouse is generally a store of integrated data from multiple sources, processed for storage in a multi-dimensional model and nonvolatile.

5.	They do not support time series and trend analysis.	Data warehouses support time-series and trend analysis, both of which require more historical data.
6.	In databases, transactions are the unit and are the agent of change to the database.	The information in the data warehouses is much more course-grained and is refreshed according to a careful choice of incremental refresh policy.
7.	Data in the databases can be changed regularly.	Data can only be added. Once data are stored, no changes are allowed.
8.	Data represent current view.	Data are historic in nature.
9.	Same data can have different representations or meanings.	It provides a unified view of all data elements with a common definition and representation.
10.	Data are stored with a functional or process orientation.	Data are stored with a subject orientation.

8.2.3 Data Warehouse Architecture

The hardware, software and data resources required to construct the data warehouse depends upon the organization that wants to construct it. The needs and resources available, forces the decisions of the organization regarding the architecture of a particular data warehouse. There are many phases, that are common to all the data warehouses regardless of the organization or

the design selected. The most common phases are acquisition of data, storage of data and data access. The general architecture of a data warehouse is shown in Figure 8.1.

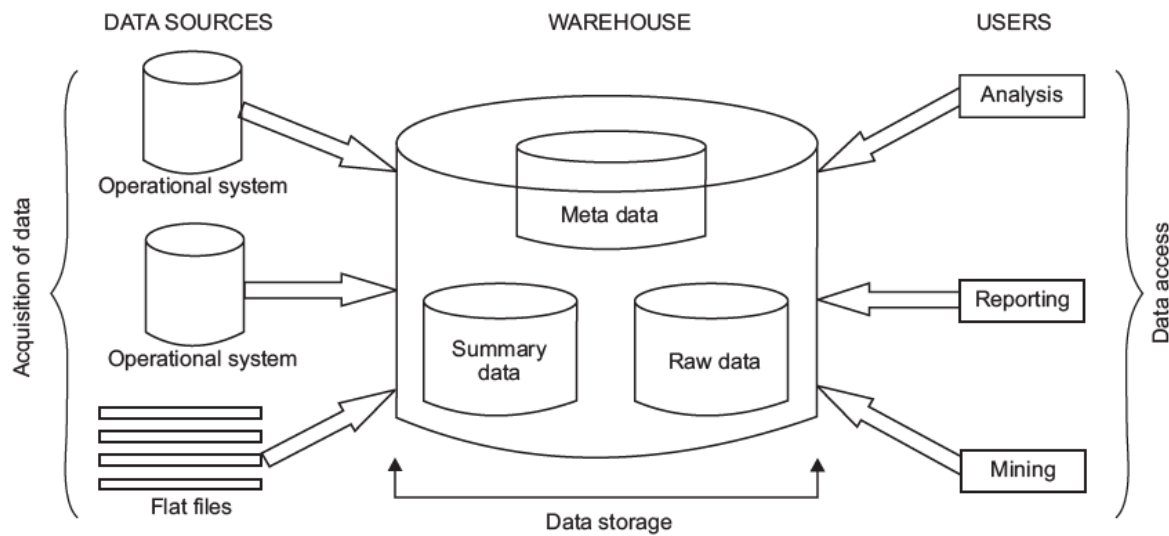


Fig. 8.1. Data warehouse architecture.

Acquisition of Data: All the data warehouses must have a source from where the data is acquired. Most of the data in the data warehouse is derived from the operational data of the organization. The required data is extracted, filtered, translated and integrated into the data storage environment.

Storage of Data: The large amounts of operational data that is historical in nature are defined, indexed and then partitioned to allow for economic and efficient access.

Data Access: A number of data mining applications allow many users throughout the organization to retrieve, analyze, query and generate reports. The ability to access data is fundamental to the concept of data warehouse in the organization.

8.2.4 Data Warehouse Components

There are mainly six components of a data warehouse. These are as follows:

- (i) Summarized data (ii) Operational data-store
- (iii) Integration/Transformation programs (iv) Detailed data
- (v) Meta data (vi) Archives

(i) **Summarized data:** The raw data generated by a transaction-processing system may be too large to store online. However, many queries can be answered by just maintaining the summary data obtained by aggregation on a relation, rather than maintain the entire

relation. Summary data is classified into two categories-Lightly summarized and highly summarized.

- (a) **Lightly summarized data:** This represents data distilled from current detailed data. It is summarized according to some unit of time and always resides on disk.
- (b) **Highly summarized data:** This represents data distilled from lightly summarized data. It is more compact and easily accessible and resides on disk.
- (ii) **Operational data store:** Operational databases are the source data for the data warehouse. Operational data store is a repository of operational data.
- (iii) **Integration/transformation programs:** The integration and transformation programs convert the operational data that is applications specific into enterprise data. The major functions performed by these programs are as follows :
 - Reformatting, re-evaluation or changing key structures.
 - Adding time elements.
 - Default values identification.
 - Providing logic to choose between multiple data sources.
 - Summarizing, tallying and merging data from multiple sources.

These programs are modified when operational or data warehouse environments change to reflect the changes.

- (iv) **Detailed data:** Detailed data is of two types—Older detail data and current detail data. The older detail data represent data that is not very recent, may be as old as ten years or longer. It is voluminous and most frequently stored on mass storage such as tape. The *current detail data* represent data of a recent nature and always has a shorter time horizon than older detail data. It can be voluminous; it is almost always stored on disk to permit faster access.
- (v) **Meta data:** Meta data is data about data. Meta data for data warehouse users are part of the data warehouse itself and controls access and analysis of the data warehouse contents. The Meta data repository is a key data warehouse component. It contains both **technical** and **business** Meta data. The **technical Meta data** cover details about acquisition, processing, storage structure, data descriptions, warehouse operations and maintenance and access support functionality. The **business Meta data** covers the relevant business rules and organizational details supporting the warehouse.
- (vi) **Archives:** These contain old or historical data of significant interest and have value to the enterprise. It is generally used for forecasting and trend analysis, thus, these archives store old data and the meta data that describe the characteristics of the old data.

8.2.5 Advantages of Data Warehouse

The data warehouse has many advantages for an enterprise. The most important one are as follows.

1. **Effective decision making:** The major benefit of a data warehouse is its ability to analyze and execute business decisions based on data from multiple sources. By using data warehouse, one can look at past trends and may be do some predictions of what is going to happen in the future.
2. **Increases the productivity of business analysts:** The data warehouse can provide analysts with pre-calculated reports and graphs, that increase the productivity of business analysts.
3. **An enterprise can maintain better customer relationships by correlating all customer data through a single data warehouse.**
4. **It provides supplementing disaster recovery plans with another data backup source.**
5. **Business and information re-engineering:** By knowing what information is important to the enterprise, that is possible by using data warehousing, the re-engineering efforts become more directional and have priorities. Also the data warehouse development is the effective first step in re-engineering the enterprise's legacy system.

8.2.6 Disadvantages/Limitations of Data Warehouse

The data warehouse have some limitations, these are as follows:

1. The data warehouse is very expensive solution and generally found in large firms.
2. Performance tuning is hard due to very large size of the data warehouse.
3. The cost of maintaining the data warehouse is very high.
4. A data warehouse has a high demand of various resources.
5. Scalability can be a problem with the data warehouse.
6. Complexity of integration in data warehouse.
7. Data warehouse is query intensive.

8.3 OLTP (ON-LINE TRANSACTION PROCESSING)

The term OLTP covers applications that work with transactional or atomic data i.e., the individual records contained within a database. OLTP applications usually just retrieve groups of records and present them to the end-user, for example, the list of computer software sold at a particular store during one day. These applications typically use relational

databases, with a fact or data table containing individual transactions linked to meta tables that store data about customers and product details.

8.3.1 Limitations of OLTP

The following are the major limitations of OLTP:

1. **Increasing Data Storage:** The companies are storing more and more data about their business and retrieving many thousands of records for immediate analysis is a time and resource consuming process, particularly when many users are using an application at the same time. Database engines that can quickly retrieve thousands of records for 5–7 users have to struggle when they have to return the results of large queries to thousands of users that are accessing simultaneously.
2. **Caching frequently requested data in temporary tables and data stores can help a lot, but solves the problem partly, particularly if each user requires a slightly different set of data.** In current data warehouses where the required data might be spread across multiple tables, the complexity of the query may also cause time delays and require more system resources which means more money must be spent on database servers in order to keep up with user demands.
3. **Data versus Information:** Business users need both data and information. Users who make business decisions based on events that are happening need the information contained within their company's data. Database engines were not primarily designed to retrieve groups of records and then sum them together mathematically and they tend not to perform well when asked to do so. An OLTP application would always be able to provide the answers, but not in the typical few-seconds response times demanded by users.
4. **Caching results doesnot help here either, because in order to be effective, every possible aggregation must be cached, or the benefit won't always be realized.** Caching on this scale would require enormous sets of temporary tables and enormous amounts of disk space to store them.
5. **Data Layout:** The relational database model was designed for transactional processing and is not always the best way to store data when attempting to answer business questions such as “Sales of Mobile phones by region” or “Volume of credit-card transactions by month”. These types of queries require vast amounts of data to be retrieved and aggregated on-demand, something that will require time and system resources to achieve.

The answer to the limitations of OLTP is to use a different approach altogether to the problem and that approach is OLAP. OLAP applications store data in a different way from the traditional relational model, allowing them to work with data sets designed to serve greater numbers of users in parallel. OLAP data stores are designed to work with aggregated data, allowing them to quickly answer high-level questions about a company's data and still allowing users to access the original transactional data when required.

OLAP applications differ from OLTP applications in the way that they store data, the way that they analyze data and the way that they present data to the end-user. It is these fundamental differences that allow OLAP applications to answer more sophisticated business questions.

8.4 OLAP (ON-LINE ANALYTICAL PROCESSING)

OLAP can be defined as the interactive process of creating, managing and analyzing data, as well as reporting on the data. This data being usually perceived and manipulated as though it were stored in a multi-dimensional array. OLAP allows users to perform quick and effective analysis on large amounts of data. The data are stored in a multi-dimensional fashion that more closely models real business data. OLAP also allows users to access summary data faster and easier. OLAP applications present the end user with information rather than just data. They make it easy for users to identify patterns or trends in the data very quickly, without the need for them to search through huge data.

OLAP systems are data warehouse front-end software tools to make aggregate data available efficiently, for advanced analysis, to managers of an enterprise. The analysis often requires resource intensive aggregations processing and therefore it becomes necessary to implement a special database (data warehouse) to improve OLAP response time. It is essential that an OLAP system provides facilities for a manager to pose adhoc complex queries to obtain the information that he/she requires. OLAP applications move into areas such as forecasting and data mining, allowing users to answer questions such as "What are our predicted costs for next year?" and "Show me our most successful salesman".

Definition: OLAP is the dynamic enterprise analysis required to create, manipulate, animate and synthesize information from exegetical, contemplative and formulaic data analysis models.

Or

OLAP is a fast analysis of shared multidimensional information for advanced analysis.

Or

OLAP, which is software technology that enables analysts, managers and executives to gain insight into data through fast, consistent, interactive access to a wide variety of possible views of information that, has been transformed from raw data to reflect that real dimensional of the enterprise as understood by the user.

8.4.1 Codd's OLAP Characteristics

The most important characteristics OLAP systems listed by Codd are as follows:

1. **Multidimensional conceptual view:** As noted above, this is central characteristic of an OLAP system. By requiring a multidimensional view, it is possible to carry out operations like slice and dice.
2. **Accessibility:** The OLAP software should be sitting between data sources (e.g. data warehouse) and an OLAP front-end.
3. **Batch extraction vs interpretive:** An OLAP system should provide multidimensional data staging plus precalculation of aggregates in large multidimensional databases.
4. **Multi-user support:** Since the OLAP system is shared, the OLAP software should provide many normal database operations including retrieval, update, concurrency control, integrity and security.
5. **Storing OLAP results:** OLAP results data should be kept separate from source data. Read-write OLAP applications should not be implemented directly on live transaction data if OLAP source systems are supplying information to the OLAP system directly.
6. **Extraction of missing values:** The OLAP system should distinguish missing values from zero values. A large data cube may have a large number of zeros as well as some missing values. If a distinction is not made between zero values and missing values, the aggregates are likely to be computed incorrectly.
7. **Treatment of missing values:** An OLAP system should ignore all missing values regardless of their source. Correct aggregate values will be computed once the missing values are ignored.
8. **Uniform reporting performance:** Increasing the number of dimensions or database size should not significantly degrade the reporting performance of the OLAP system. This is a good objective although it may be difficult to achieve in practice.
9. **Generic dimensionality:** An OLAP system should treat each dimension as equivalent in both its structure and operational capabilities. Additional operational capabilities may be

granted to selected dimensions but such additional functions should be grantable to any dimension.

10. Unlimited dimensions and aggregation levels: An OLAP system should allow unlimited dimensions and aggregation levels. Practically, the number of dimensions is rarely more than 10 and the number of hierarchies rarely more than six.

8.4.2 Difference between OLTP and OLAP

OLTP and OLAP are complementing technologies. The major differences between two OLTP and OLAP are as follows:

	OLTP	OLAP
Application	The applications of OLTP are ERP, CRM and legacy apps.	The applications of OLAP are Management information system (MIS), decision support system(DSS).
Functions	Its function is mission-critical.	Its function is management-critical.
Nature of queries	Mostly queries are simple.	Mostly the queries are complex.
Typical users	The users of OLTP are the staff members and are thousands in numbers.	The users of the OLAP are managers and executives and are dozens in numbers.
Horizon	The information processed and stored for weeks or months.	The information processed and stored for years.
Nature of data	The data is current, detailed and relational in nature.	The data is historical, summarized and multidimensional in nature.
Refresh	The refreshing is immediate.	The refreshing is periodic.
Data model	It uses the E-R model.	It uses multi-dimensional tables.
Nature of design	The design of OLTP is application oriented.	The design of OLAP is subject oriented.
Schema	It uses normalized schema.	It uses star schema.
Emphasis	The emphasis is more on update of data.	The emphasis is more on retrieval of data.

8.4.3 OLAP Operations

The most common operations of OLAP systems are as follows:

- (a) Roll-up (b) Drill-down, drill-up and Drill-across (c) Slice and dice (d) Pivot or rotate
- (a) **Roll-up:** Roll-up is like zooming out on the data cube. It is required when the user needs further abstraction or less detail. This operation performs further aggregations on the data.
- (b) **Drill-down, drill-up and Drill-across:** Drill-down is like zooming in on the data and is therefore the reverse of roll-up. It is an appropriate operation when the user needs further

details or when the user wants to partition more finely or wants to focus on some particular values of certain dimensions. Drill-down adds more details to the data. The hierarchy defined on a dimension may be involved in drill-down. Roll-up and drill-down operations do not remove any events but change the level of granularity of a particular dimension.

Drill-up refers to the process of selecting a child field and displaying its parent field.

Drill-across describes any changes to the sectioned fields.

- (c) **Slice and dice:** This term is used to describe the process used to retrieve and view data stored in an OLAP cube. Since data can be displayed in a two-dimensional format, the multi-dimensional cube must be restricted into flat “slices” of data. To pick a particular orientation of data in a cube, the user is actually “slicing and dicing” the data in order to view a simple flat layout.

Slice: A slice is a subset of the cube corresponding to a single value for one or more members of the dimensions. For example, a slice operation is performed when the user wants a selection on one dimension of a three-dimensional cube resulting in a two-dimensional site.

In slicing, first step is to make a selection on one dimension of the given cube is performed which resulted in a sub-cube. The second step reduces the dimensionality of the cubes. The Third step sets one or more dimensions to specific values and keeps a subset of dimensions for selected values

Dice: The dice operation is similar to slice but dicing does not involve reducing the number of dimensions. A dice is obtained by performing a selection on two or more dimensions.

In dicing, the first step is to define a sub-cube by performing a selection of one or more dimensions. The second step refers to range select condition on one dimension, or to select condition on more than one dimension. The third step reduces the number of member values of one or more dimensions.

- (d) **Pivot or Rotate:** The pivot operation is used when the user wishes to re-orient the view of the data cube. It may involve swapping the rows and columns, or moving one of the row dimensions into the column dimension.

8.5 DATA MINING

Data mining is a collection of techniques, which is used to find undiscovered patterns by manipulating large volumes of data. It is a process of mining or discovering of new information.

It is used in conjunction of data warehousing to help in certain types of decisions. It is applied to operational database with individual transactions.

Data mining involves the use of sophisticated data analysis tools to discover previously unknown, valid patterns and actionable information from vast amount of data and using it to make crucial business decisions. It is a strong tool, which allows end users to directly access, and manipulate the data within data warehousing environment without the need of any other tool.

In simple words we can say data mining is a step by step process in which first we extract large amount of data from database and refine it in a way that results into discovering of new facts. So, we can say that data mining is a process, which turns our data into knowledge.

8.5.1 Data Mining Process as a Part of Knowledge Discovery Process

Data mining helps to determine previously unknown data patterns, actionable information from large databases. So, data mining process help in discovering new facts from data, also called knowledge discovery in database (KDD). KDD is a six step process. The steps are as follows:

- (i) **Data selection:** In data selection step large amount of data are extracted from database and try to categorize it to identify target datasets and relevant attributes.
- (ii) **Data cleansing or data pre-processing:** As the name suggest, in this step, all the corrupted and unwanted data is removed to avoid inconsistencies.
- (iii) **Data enrichment or data transformation:** In this step, you can add some additional information, transformation of fields can be done (For example, you extract a field “Name” and it is 40 characters long but you want it to be 25 characters long only. So, it needs to be transformed) or existing fields can be combined to generate new fields etc. In simple words here you can make data useful according to your use. This data is used as input for data mining.
- (iv) **Data mining:** In this step, the process tries to explore new previously unknown patterns and presented them into understandable form to end user.

- (v) **Human interpretation:** In this process human interference is required to study the patterns provided by previous step.
- (vi) **Knowledge discovery:** It is the last step in which actual knowledge is discovered. During data mining process you can undo any of its steps and use new knowledge or data to redo the step. So, you can say that data mining process also true to optimize the data processing. Data mining process is shown in Figure 8.2.

8.5.2 Goals of Data Mining

Goals of data mining fall into the following classes:

- (i) **Prediction:** Data mining helps in prediction of behavior of certain data attributes in the future. It is very helpful in complex data scenarios. Take the example of sales department, in which by analyzing buying transactions they can predict the behavior and buying capacity of customers. By analyzing and prediction they can categorize their sales activity according to area, needs of customer, life standard of customers etc., to enhance the sales. In a scientific context, certain seismic wave patterns may predict an earthquake with high probability.

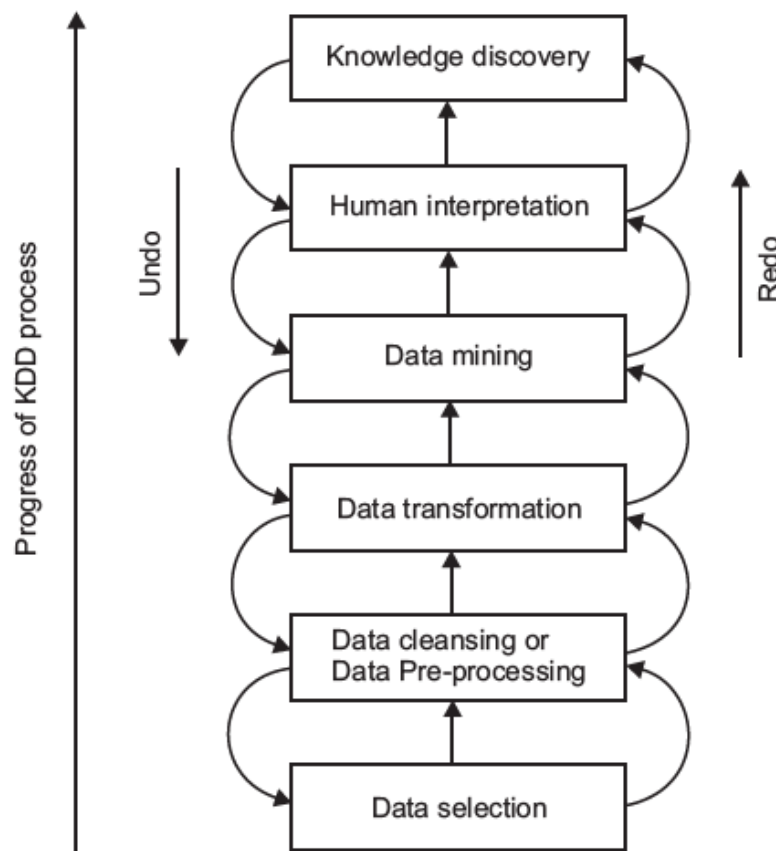


Fig. 8.2.Data mining process (KDD).

- (ii) **Identification:** One major goal of data mining is to identify the existence of an item, an event, or an activity on the bases of analysis made on different data patterns. For example, scientists are trying to identify the life on Mars by analyzing different soil patterns. Authentication is also a form of identification.
- (iii) **Classification:** Data mining is helpful in classifying the data into different categories on the basis of certain parameters. For example, in a company HR department can grade their employees on the basis of their performance parameters like their technical knowledge, functional knowledge, discipline etc.
- (iv) **Optimization:** Last and the most important goal of data mining is to optimize the use of limited resources like time, cost, space, manpower and machine power in such a way that it will make a boom in output such as profits, increase in sales, cutting in expenditure etc.

8.5.3 Elements of Data Mining

Data mining consists of five major elements:

1. Extract, transform, and load transaction data on to the data warehouse system.
2. Store and manage the data in a multi-dimensional database system.
3. Provide data access to business analysts and information technology.
4. Analyze the data by application software.
5. Present the data in a useful format, such as graphs or tables.

8.5.4 Types of Knowledge Discovered during Data Mining

Data mining process results into discovery of new knowledge. The discovered knowledge can be categorized into different forms as given below:

- (i) **Association rules:** Data can be mined to find relations or associations. Association rules correlate the presence of a set of items with another range of values for another set of variables e.g. (a) If a customer buys a shirt he or she also buy a matching trouser. (b) If a customer buys a PC, he may also buy some CD's.
- (ii) **Classification hierarchies:** The classification process is used to create different hierarchy of classes on the basis of existing set of events or transactions. e.g. (a) Customers may be divided into several ranges of credit worthiness based on the history of previous credit transactions. (b) Stocks may be given priority in stock market on the basis of their past performance such as growth, income, and stability. (c) Employees can be graded according to their past performance.

- (iii) **Sequential patterns:** Data can be mixed to anticipate the behaviour of patterns. e.g. If a patient underwent cardiac bypass surgery for blocked arteries and an aneurysm and later on developed high blood urea within a year of surgery then the patient is likely to suffer from kidney failure within the next 17 months.
- (iv) **Pattern within time series:** If we analyze the data taken at regular intervals then we may find similarities within positions of a time series of data. E.g. Sales of woolen clothes are increased in winter.
- (v) **Clustering:** we can group data items together according to logical relationship known as clusters. E.g. we can categorize websites into groups from “most likely to access” to “least likely to access”. Different groups or clusters are dissimilar and records within the group are similar to each other.

8.5.5 Models of Data Mining

The three proposed models of data mining are as follows:

1. **CRISP (Cross-Industry Standard Process for Data mining):** It was proposed in the mid 1990’s by European consortium of companies to serve as a non-proprietary standard process model for data mining. The sequence of steps of data mining in this model is shown in Figure 8.3.

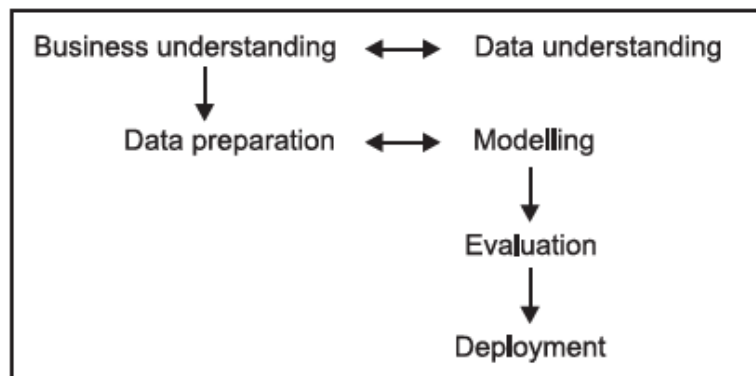


Fig. 8.3. CRISP.

2. **Six-sigma methodology:** It is a well-structured, data-driven methodology for eliminating defects, waste, quality control problem of all kinds of business activities. The sequence of steps of data mining in this model is shown in Figure 8.4.

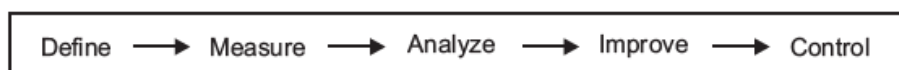


Fig. 8.4. Six-sigma methodology.

3. **SEMMA** : It is somewhat similar to six-sigma methodology. It was proposed by SAS institute. The sequence of steps of data mining in this model is shown in Figure 8.5. It focuses more on technical activities, involved in data mining.

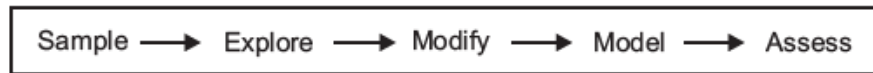


Fig 8.5.SEMMA.

8.5.6 Techniques used in Data Mining

The most commonly used data mining techniques are as follows:

1. **Genetic algorithms:** Optimization techniques that use processes such as genetic mutation, combination and natural selection are design based on concepts of evolution.
2. **Artificial neural networks :** Non-linear predictive models that learn through training and resemble biological neural networks in structure.
2. **Rule induction:** The extraction of useful if-then rules from data based on statistical significance.
3. **Decision trees:** They are tree shaped structures that represent sets of decisions. These decisions generate rules for the classification of a dataset.

8.5.7 Data Mining Tools

Various data mining tools are as follows:

1. Text term searching tools.
2. Sequence similarity searching tools.
3. Sequence submission tools.
4. Computer assisted passenger prescreening system (CAPPS II).
5. Terrorism information awareness program (TIA).
6. Query managers and report writers.
7. Multidimensional databases tools.
8. Exploration and discovery tools.

8.5.9 Advantages of Data Mining

Following are the advantages of data mining:

1. **Marketing/Retailing:** Data mining provides marketers and retailers with useful and accurate trends about their customers purchasing behavior.
2. **Banking/Finance:** Data mining can assist financial institutions in areas such as credit reporting and loan information.

3. Law enforcement: Data mining helps law enforcers in identifying criminal suspects as well as capturing them by examining trends in location, crime type etc.
4. Researches: It helps researches by speeding up their data analyzing process, which helps them to do more work within some time limits.

8.5.10 Disadvantages of Data Mining

1. Privacy issues: With the widespread use of technology, personal privacy has always been a major concern of any country. It is possible that any fraud company can sold their customers information.
2. Security issues: Data mining gives access directly to database to different users, which may cause leakage of secured data.
3. Inaccurate information: Data mining is not 100% accurate. It may contain inaccurate information that leads to inconsistency.

8.5.11 Scope of Improvement in Data Mining

1. Scaling Up for high dimensional data or high-speed streams.
2. Developing a unifying theory of data mining.
3. Mining complex knowledge from complex data.
4. Mining sequence data and time series data.
5. Data mining in a network setting.
6. Distributed data mining and mining multi-agent data.
7. Security, privacy and data integrity.
8. Data mining for biological and environmental problems.
9. Dealing with Non-static, unbalanced and cost-sensitive data.
10. Data mining process related problems.

8.6 COMPARISON OF DATA MINING AND SQL

The major differences and similarities between Data mining and SQL are as follows:

S.No.	Data Mining	Structured Query Language
1.	Data mining is a key member in the Business Intelligence (BI) product family, together with Online Analytical processing (OLAP), enterprise reporting and ETL. Data mining is about analyzing data and finding hidden patterns using automatic or semiautomatic means.	SQL (Structured Query Language) is a database computer language designed for the retrieval and management of data in relational database management systems (RDBMS), database schema creation and modification, and database object access control management.

2.	Financial applications, Enterprise Resource Management (ERP), Customer Relationship Management (CRM), and Web logs from its input.	It requires relational database. It can't act on any database.
3.	Discovery-based approaches in which pattern-matching and other algorithms are employed to determine the key relationships in the data.	Verification-based approach, in which the user hypothesized about specific data interrelationships and then uses the tools to verify or refute those hypothesis.
4.	It is not dependent on the Analyst but Data mining algorithms can look at numerous multidimensional data relationships concurrently, highlighting those that are dominant or exceptional.	It depends on the ability of the analyst to pose appropriate questions and quickly return results, manage the complexity of the attribute space.
5.	Data mining works on algorithms.	No algorithms are used for queries.
6.	Patterns are discovered by iteration of each subset until the algorithm is applied fully. For example, in making decision trees.	Queries are applied to gain a number of combinations, but no definite pattern is recognized.
7.	Data mining constructs models of the data in question.	Query simply returns the data that fulfills certain constraints.
8.	A data mining Query with SQL. IF AGE > 35 AND CAR = MINIVAN THEN TOTAL SPENT > \$100 Or IF SEX= M AND ZIP= 05566 THEN TOTAL SPENT >\$100	A SQL query. SELECT * FROM CUSTOMER _TABLE WHERE TOTAL _ SPENT> \$100;

8.7 COMPARISON OF DATA MINING AND DATA WAREHOUSING

The major differences between Data mining are Data warehousing are as follows:

S.No.	Data Mining	Data Warehousing
1.	Data mining is about analyzing data and finding hidden patterns using automatic or semiautomatic means. It is a key member in the Business Intelligence (BI), together with OLAP, enterprise reporting and ETL.	Data warehouse is a repository of an organization's electronically stored data. It is designed to facilitate reporting and analysis.
2.	In data mining, the analyst is looking to support a hypothesis based on correlations, patterns and cause and effects relationships in statistical models.	Data warehousing is concerned with answering a broader question and slicing and dicing data based on experience to add value to organization.
3.	Data mining is intended for user who are statistically inclined.	Data warehouse users tend to be data experts who analyze by business dimensions directly.

4.	Data miners engage in question formulation based mainly on 'law of large numbers' to identify potentially useful relationships between data elements.	Data warehousing analysts are concerned with customer experience and can help the customer by improving the customer experience.
5.	Data mining predicts the future.	Data warehouse describes the present and past.
6.	Prescriptive algorithms are used in data mining.	Query and reporting functions are used in data warehousing.
7.	Data mining rely on Historical data to derive conclusions.	Data warehousing also rely on historical data to derive conclusions.
8.	Data mining fall short of delivering a predictive model.	Data warehousing also fall short of delivering a predictive model.

8.8 CHECK YOUR PROGRESS

1. Data-mining tools are similar to QBE tools, SQL, and report generators in the typical database environment.(True/False)
2. A data mart is a subset of a data warehouse in which only a focused portion of the data warehouse information is kept.(True/False)
3. _____ are software tools used to query information in a data warehouse.
4. _____ are databases that support OLTP.
5. What are the phases of data warehouse?

Answers to check your progress:

1. False
2. True
3. Data-mining tools
4. Operational databases
5. Acquisition of data, storage of data, data access

8.9 SUMMARY

In this unit Distinctive Characteristics of Data Warehouses along with its architecture and components are discussed. Also the merits and demerits are highlighted. Data mart benefits are detailed. OLTP and OLAP characteristics are analyzed with differences. Data Mining Process as a Part of Knowledge Discovery Process is detailed along with goals and elements. The merits and demerits of data mining are analyzed.

8.10 KEYWORDS

- Data warehouse: A data warehouse is a type of data management system that is designed to enable and support business intelligence (BI) activities, especially analytics.
- Data mining: Data mining is the process of finding anomalies, patterns and correlations within large data sets to predict outcomes.
- OLAP: Online analytical processing (OLAP) is a technology that organizes large business databases and supports complex analysis.
- OLTP: OLTP (online transaction processing) is a class of software programs capable of supporting transaction-oriented applications.
- Data Mart : A data mart is a simple form of data warehouse focused on a single subject or line of business

8.11 QUESTIONS FOR SELF STUDY

1. What is a data warehouse? Explain its characteristics.
2. What is the difference between data warehouse and database?
3. What are the components of data warehouse?
4. What are the advantages and disadvantages of data warehouse?
5. What is data mining? Explain data mining as a knowledge discovery process.
6. What are the goals of data mining? Explain various data mining tools.
7. What are the advantages and disadvantages of data mining?
8. What are the difference between OLAP and OLTP?
9. Compare and contrast data mining and SQL.
10. Compare and contrast data warehouse and database.
11. Compare and contrast data warehouse and data mining.
12. Explain the architecture of data warehouse.

8.13 References

1. Coronel, C., & Morris, S. (2016). Database systems: design, implementation, & management. Cengage Learning.
2. Gupta, S. B., & Mittal, A. (2009). Introduction to database management system. Laxmi Publications, Ltd..
3. Ramakrishnan, R., & Gehrke, J. (2000). Database management systems. McGraw-Hill.



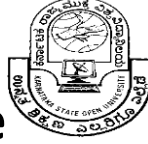
Karnataka State Open University
Mukthagangothri, Mysore – 570 006.
Dept. of Studies and Research in Management

MBA IT Specialization
III Semester

Database Management System



Block 3



Karnataka State Open University

Mukthagangothri, Mysore – 570 006.

Dept. of Studies and Research in Management

MBA. IT Specialization

III Semester

DATABASE MANAGEMENT SYSTEM

BLOCK3-DATABASE DESIGN AND CONTROL

UNIT NO.	TITLE	PAGE NUMBERS
UNIT 9	DATABASE DESIGN	1-27
UNIT 10	TRANSACTION MANAGEMENT AND CONCURRENCY CONTROL	28-54
UNIT 11	DISTRIBUTED DATABASE MANAGEMENT SYSTEMS (DDBMS)	55-71
UNIT 12	BUSINESS INTELLIGENCE AND DATA WAREHOUSES	72-88

BLOCK 3 INTRODUCTION

Information systems are the product of a carefully staged development process. Systems analysis is used to determine the need for an information system and to establish its limits. Within systems analysis, the actual information system is created through a process known as systems development. In Unit 9, you will be introduced to some classical approaches to database design: top-down vs. bottom-up and centralized vs. decentralized. In unit 10, you will learn about the most common algorithms for concurrency control: locks, time stamping, and optimistic methods. Because locks are the most widely used method, you will examine various levels and types of locks. Locks can also create deadlocks, so you will learn about strategies for managing deadlocks. Database contents can be damaged or destroyed by critical operational errors, including transaction management failures. Therefore, in this unit you will also learn how database recovery management maintains a database's contents. In unit 11, you will learn that a single database can be divided into several fragments. The fragments can be stored on different computers within a network. Processing, too, can be dispersed among several different network sites, or nodes. Unit 12 explores the main concepts and components of business intelligence and decision support systems that gather, generate, and present information for business decision makers

This block consists of 4 units and is organized as follows:

Unit 9: Database Design: The Information System, The Systems Development Life Cycle, The Database Life Cycle, Conceptual Design, DBMS Software Selection, Logical Design, Physical Design, Database Design Strategies, Centralized versus Decentralized Design

Unit 10: Transaction Management and Concurrency Control: What Is a Transaction? Concurrency Control, Concurrency Control with Locking Methods, Concurrency Control with Time Stamping Methods, Concurrency Control with Optimistic Methods, ANSI Levels of Transaction Isolation, Database Recovery Management

Unit 11: Distributed Database Management Systems (DDBMS): The Evolution of DDBMS, DDBMS Advantages and Disadvantages, Distributed Processing and Distributed Databases, Characteristics of DDBMS, DDBMS Components, Levels of Data and Process Distribution, Distributed Database Transparency Features

Unit 12: Business Intelligence and Data Warehouses: The Need for Data Analysis, Business Intelligence, Decision Support Data, Data Analytics, And Data Visualization

UNIT-9: DATABASE DESIGN

STRUCTURE

- 9.0 Objectives
- 9.1 The Information System
- 9.2 The Systems Development Life Cycle
- 9.3 The Database Life Cycle
- 9.4 Conceptual Design
- 9.5 DBMS Software Selection
- 9.6 Logical Design
- 9.7 Physical Design
- 9.8 Database Design Strategies
- 9.9 Centralized versus Decentralized Design
- 9.10 Check your progress
- 9.11 Summary
- 9.12 Key words
- 9.13 Questions for self-study
- 9.14 References

9.0 OBJECTIVES

After studying this unit, you will be able to:

- ✓ Explain that successful database design must reflect the information system of which the database is a part.
- ✓ Discuss that successful information systems are developed within a framework known as the Systems Development Life Cycle (SDLC)
- ✓ Describe that within the information system, the most successful databases are subject to frequent evaluation and revision within a framework known as the Database Life Cycle (DBLC)
- ✓ Explain how to conduct evaluation and revision within the SDLC and DBLC frameworks
- ✓ Discuss about database design strategies: top-down vs. bottom-up design and centralized vs. decentralized design

9.1 THE INFORMATION SYSTEM

Basically, a database is a carefully designed and constructed repository of facts. The database is a part of a larger whole known as an information system, which provides for data collection, storage, and retrieval. The information system also facilitates the transformation of data into information and it allows for the management of both data and information. Thus, a complete information system is composed of people, hardware, software, the database(s), application programs, and procedures. Systems analysis is the process that establishes the need for and the extent of an information system. The process of creating an information system is known as systems development. One key characteristic of current information systems is the strategic value of information in the age of global business. Therefore, information systems should always be aligned with the strategic business goals; the view of isolated and independent information systems is no longer valid. Current information systems should always be integrated with the company's enterprise-wide information systems architecture.

The performance of an information system depends on three factors:

- Database design and implementation.
- Application design and implementation.
- Administrative procedures.

This unit emphasizes the database design and implementation segment of the triad—arguably the most important of the three.

Creating a sound information system is hard work: systems analysis and development require much planning to ensure that all of the activities will interface with each other, that they will complement each other, and that they will be completed on time.

In a broad sense, the term database development describes the process of database design and implementation. The primary objective in database design is to create complete, normalized, non-redundant (to the extent possible), and fully integrated conceptual, logical and physical database models. The implementation phase includes creating the database storage structure, loading data into the database, and providing for data management.

To make the procedures discussed in this unit broadly applicable, the unit focuses on the elements that are common to all information systems. Most of the processes and procedures

described in this unit do not depend on the size, type or complexity of the database being implemented.

9.2 THE SYSTEMS DEVELOPMENT LIFE CYCLE

The Systems Development Life Cycle (SDLC) is a general framework through which you can track and understand the activities required to develop and maintain information systems. Within that framework, there are several ways to complete various tasks specified in the SDLC. For example, this unit focuses on ER modelling and on relational database design and implementation issues.

The Systems Development Life Cycle (SDLC) traces the history (life cycle) of an information system. Perhaps more important to the system designer, the SDLC provides the big picture within which the database design and application development can be mapped out and evaluated.

As illustrated in Figure 9.1, the traditional SDLC is divided into five phases: planning, analysis, detailed systems design, implementation, and maintenance. The SDLC is an iterative rather than a sequential process.

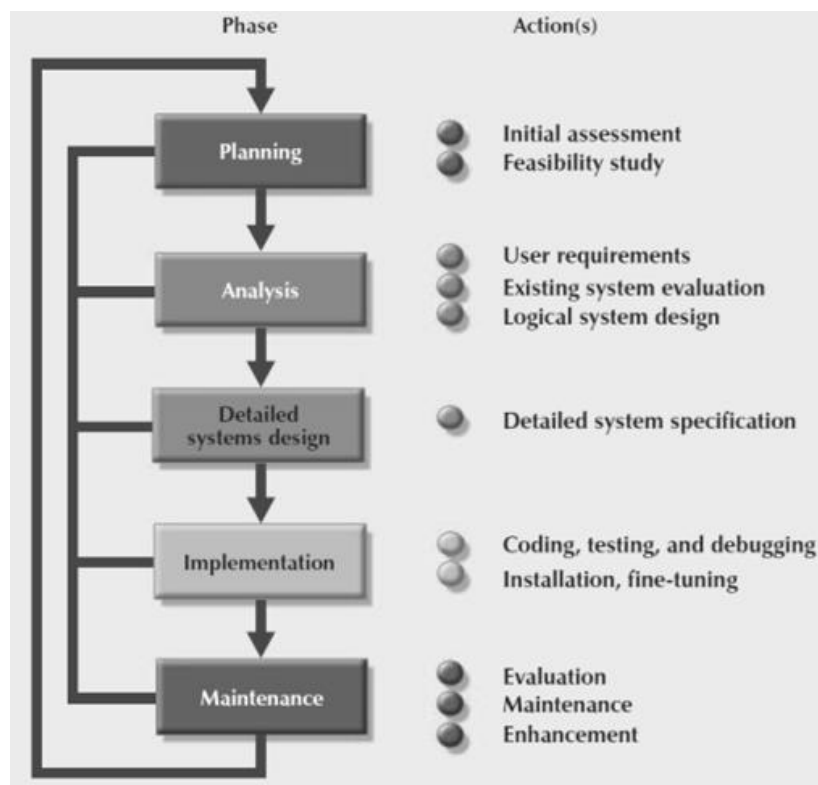


Fig. 9.1 The Systems Development Life Cycle(SDLC)

Because the Database Life Cycle (DBLC) fits into and resembles the Systems Development Life Cycle (SDLC), a brief description of the SDLC is in order.

PLANNING

The SDLC planning phase yields a general overview of the company and its objectives. An initial assessment of the information flow-and-extent requirements must be made during this discovery portion of the SDLC. Such an assessment should answer some important questions:

- Should the existing system be continued?
- Should the existing system be modified?
- Should the existing system be replaced?

If it is decided that a new system is necessary, the next question is whether it is feasible. The feasibility study must address the following:

- The technical aspects of hardware and software requirements.
- The system cost.
- The operational cost.

ANALYSIS

Problems defined during the planning phase are examined in greater detail during the analysis phase. A macro analysis must be made of both individual needs and organizational needs, addressing questions such as:

- What are the requirements of the current system's end users?
- Do those requirements fit into the overall information requirements?

The analysis phase of the SDLC is, in effect, a thorough audit of user requirements.

The existing hardware and software systems are also studied during the analysis phase. The result of analysis should be a better understanding of the system's functional areas, actual and potential problems, and opportunities. End users and the system designer(s) must work together to identify processes and to uncover potential problem areas.

Along with a study of user requirements and the existing systems, the analysis phase also includes the creation of a logical systems design. The logical design must specify the appropriate conceptual data model, inputs, processes, and expected output requirements.

The database design's data-modelling activities take place at this point to discover and describe all entities and their attributes and the relationships among the entities within the

database. Defining the logical system also yields functional descriptions of the system's components (modules) for each process within the database environment. All data transformations (processes) are described and documented, using such systems analysis tools as DFDs. The conceptual data model is validated against those processes.

DETAILED SYSTEMS DESIGN

In the detailed systems design phase, the designer completes the design of the system's processes. The design includes all the necessary technical specifications for the screens, menus, reports, and other devices that might be used to help make the system a more efficient information generator. The steps are laid out for conversion from the old to the new system. Training principles and methodologies are also planned and must be submitted for management's approval.

IMPLEMENTATION

During the implementation phase, the hardware, DBMS software, and application programs are installed, and the database design is implemented. During the initial stages of the implementation phase, the system enters into a cycle of coding, testing, and debugging until it is ready to be delivered. The actual database is created, and the system is customized by the creation of tables and views, user authorizations, and so on.

The database contents might be loaded interactively or in batch mode, using a variety of methods and devices:

- Customized user programs.
- Database interface programs.
- Conversion programs that import the data from a different file structure, using batch programs, a database utility, or both.

The system is subjected to exhaustive testing until it is ready for use. Traditionally, the implementation and testing of a new system took 50 to 60 percent of the total development time. However, the advent of sophisticated application generators and debugging tools has substantially decreased coding and testing time. After testing is concluded, the final documentation is reviewed and printed and end users are trained. The system is in full operation at the end of this phase but will be continuously evaluated and fine-tuned.

MAINTENANCE

Almost as soon as the system is operational, end users begin to request changes in it. Those changes generate system maintenance activities, which can be grouped into three types:

- Corrective maintenance in response to systems errors.
- Adaptive maintenance due to changes in the business environment.
- Perfective maintenance to enhance the system.

Because every request for structural change requires retracing the SDLC steps, the system is, in a sense, always at some stage of the SDLC.

9.3 THE DATABASE LIFE CYCLE

Within the larger information system, the database, too, is subject to a life cycle. The Database Life Cycle (DBLC) contains six phases, as shown in Figure 9.2: database initial study, database design, implementation and loading, testing and evaluation, operation, and maintenance and evolution.

THE DATABASE INITIAL STUDY

If a designer has been called in, chances are the current system has failed to perform functions deemed vital by the company. So, in addition to examining the current system's operation within the company, the designer must determine how and why the current system fails. That means spending a lot of time talking with (but mostly listening to) end users. Although database design is a technical business, it is also people-oriented. Database designers must be excellent communicators, and they must have finely tuned interpersonal skills.

Depending on the complexity and scope of the database environment, the database designer might be a lone operator or part of a systems development team composed of a project leader, one or more senior systems analysts, and one or more junior systems analysts. The word designer is used generically here to cover a wide range of design team compositions.

The overall purpose of the database initial study is to:

- Analyse the company situation.
- Define problems and constraints.
- Define objectives.
- Define scope and boundaries.

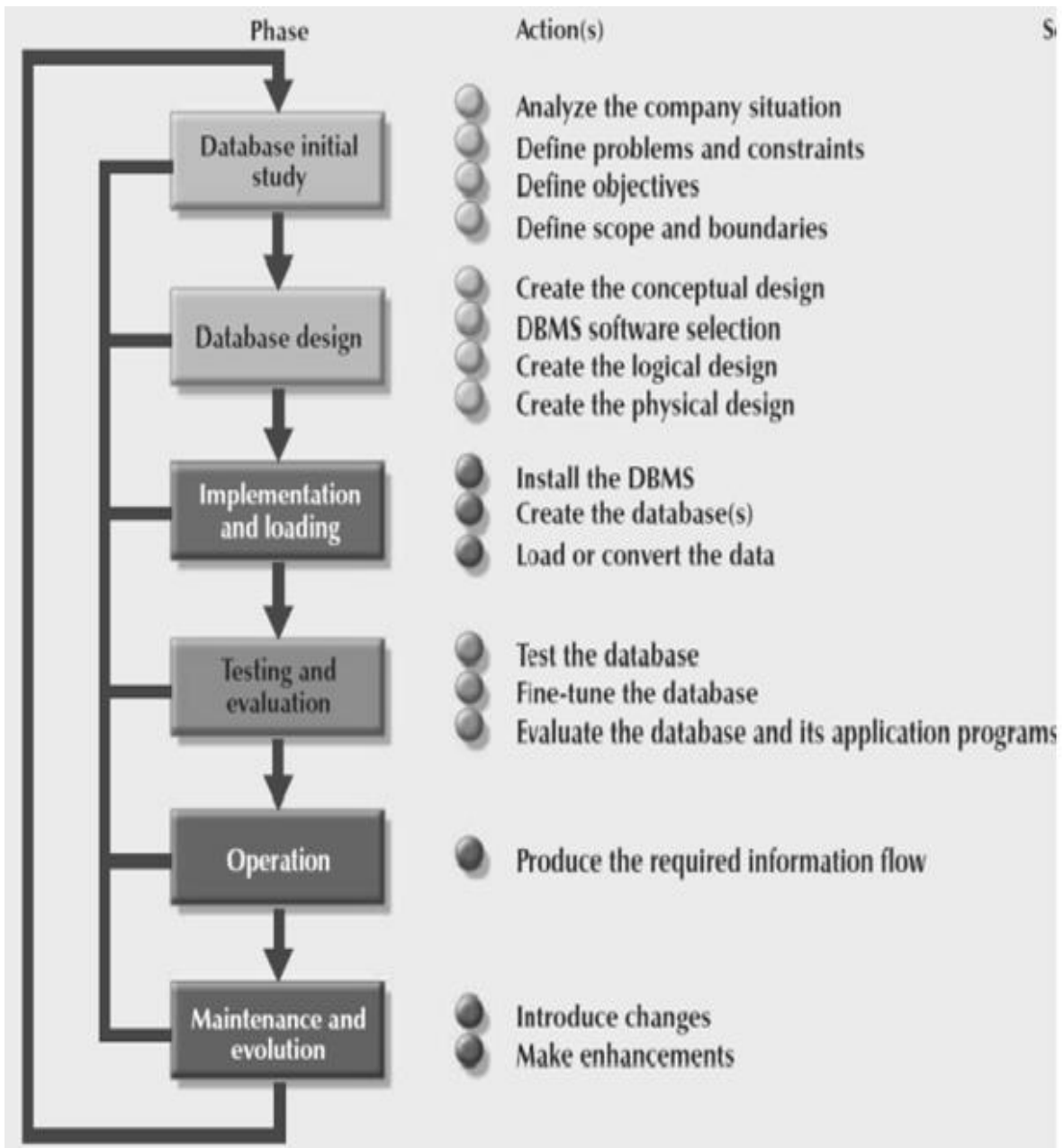


Fig. 9.2. The Database Life Cycle(DBLC)

Figure 9.3 depicts the interactive and iterative processes required to complete the first phase of the DBLC successfully. As you examine Figure 9.3, note that the database initial study phase leads to the development of the database system objectives. Using Figure 9.3 as a discussion template, let's examine each of its components in greater detail

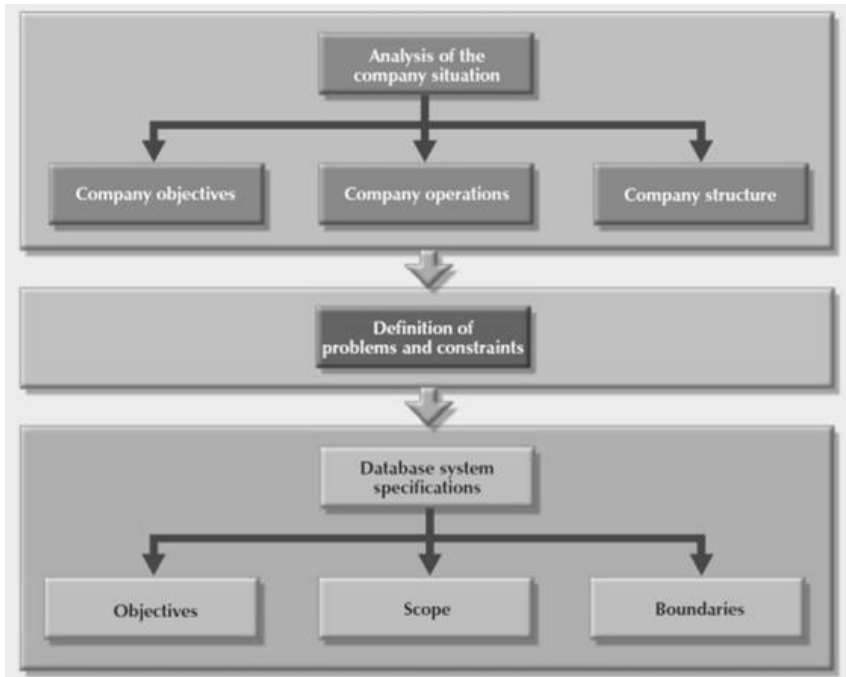


Fig. 9.3. A summary of activities in the database initial study

ANALYZE THE COMPANY SITUATION

The company situation describes the general conditions in which a company operates its organizational structure, and its mission. To analyse the company situation, the database designer must discover what the company's operational components are, how they function, and how they interact.

These issues must be resolved:

- What is the organization's general operating environment, and what is its mission within that environment? The design must satisfy the operational demands created by the organization's mission.
- What is the organization's structure? Knowing who controls what and who reports to whom is quite useful when you are trying to define required information flows, specific report and query formats, and so on.

DEFINE PROBLEMS AND CONSTRAINTS

The designer has both formal and informal sources of information. If the company has existed for any length of time, it already has some kind of system in place (either manual or computer-based). How does the existing system function?

What input does the system require? What documents does the system generate? By whom and how is the system output used? Studying the paper trail can be very informative. In

addition to the official version of the system's operation, there is also the more informal, real version; the designer must be shrewd enough to see how these differ.

The process of defining problems might initially appear to be unstructured. Company end users are often unable to describe precisely the larger scope of company operations or to identify the real problems encountered during company operations. Often the managerial view of a company's operation and its problems is different from that of the end users, who perform the actual routine work.

During the initial problem definition process, the designer is likely to collect very broad problem descriptions.

DEFINE OBJECTIVES

A proposed database system must be designed to help solve at least the major problems identified during the problem discovery process. As the list of problems unfolds, several common sources are likely to be discovered.

Note that the initial study phase also yields proposed problem solutions. The designer's job is to make sure that the database system objectives, as seen by the designer, correspond to those envisioned by the end user(s). In any case, the database designer must begin to address the following questions:

- What is the proposed system's initial objective?
- Will the system interface with other existing or future systems in the company?
- Will the system share the data with other systems or users?

DEFINE SCOPE AND BOUNDARIES

The designer must recognize the existence of two sets of limits: scope and boundaries. The system's scope defines the extent of the design according to operational requirements. Will the database design encompass the entire organization, one or more departments within the organization, or one or more functions of a single department? The designer must know the scope which helps in defining the required data structures, the type and number of entities, the physical size of the database, and so on.

The proposed system is also subject to limits known as boundaries, which are external to the system. Budget and time imposes boundaries on the designer. Boundaries are also imposed by existing hardware and software. Ideally, the designer can choose the hardware and

software that will best accomplish the system goals. The scope and boundaries become the factors that force the design into a specific mold, and the designer's job is to design the best system possible within those constraints.

DATABASE DESIGN

The second phase focuses on the design of the database model that will support company operations and objectives. This is arguably the most critical DBLC phase: making sure that the final product meets user and system requirements.

In the process of database design, you must concentrate on the data characteristics required to build the database model. At this point, there are two views of the data within the system: the business view of data as a source of information and the designer's view of the data structure, its access and the activities required to transform the data into information. Figure 9.4 contrasts those views. Note that you can summarize the different views by looking at the terms what and how. Defining data is an integral part of the DBLC's second phase.

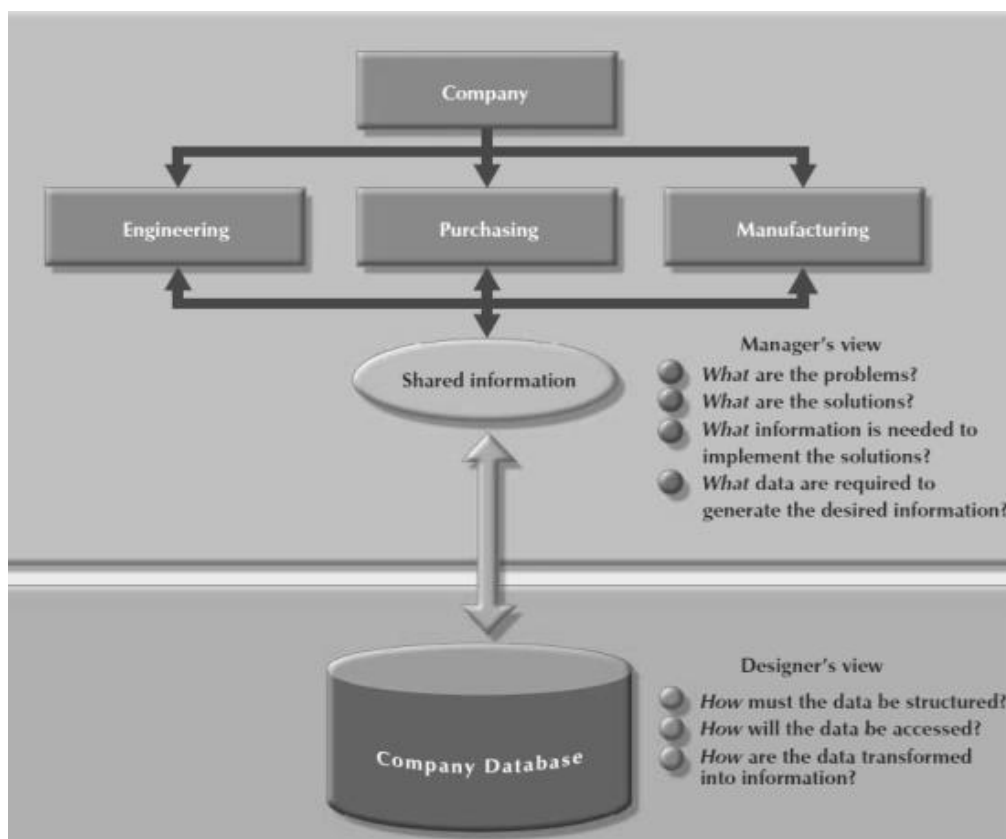


Fig 9.4 Two views of data: Business manager and database manager

The database design process is depicted in Figure 9.5. Looking at Figure 9.5, you can see that there are four essential stages: conceptual, logical, and physical design, plus the DBMS

selection stage, which is critical to determine the type of logical and physical design to be performed. The design process starts with conceptual design and moves to the logical and physical design stages. At each stage, more details about the data model design are determined and documented. You could think of the conceptual design as the overall data as seen by the end user, the logical design as the data as seen by the DBMS, and the physical design as the data as seen by the operating system's storage management devices.

It is important to note that the overwhelming majority of database designs and implementations are based on the relational model and, therefore, use the relational model constructs and techniques. At the completion of the database design activities, you will have a complete database design ready to be implemented.

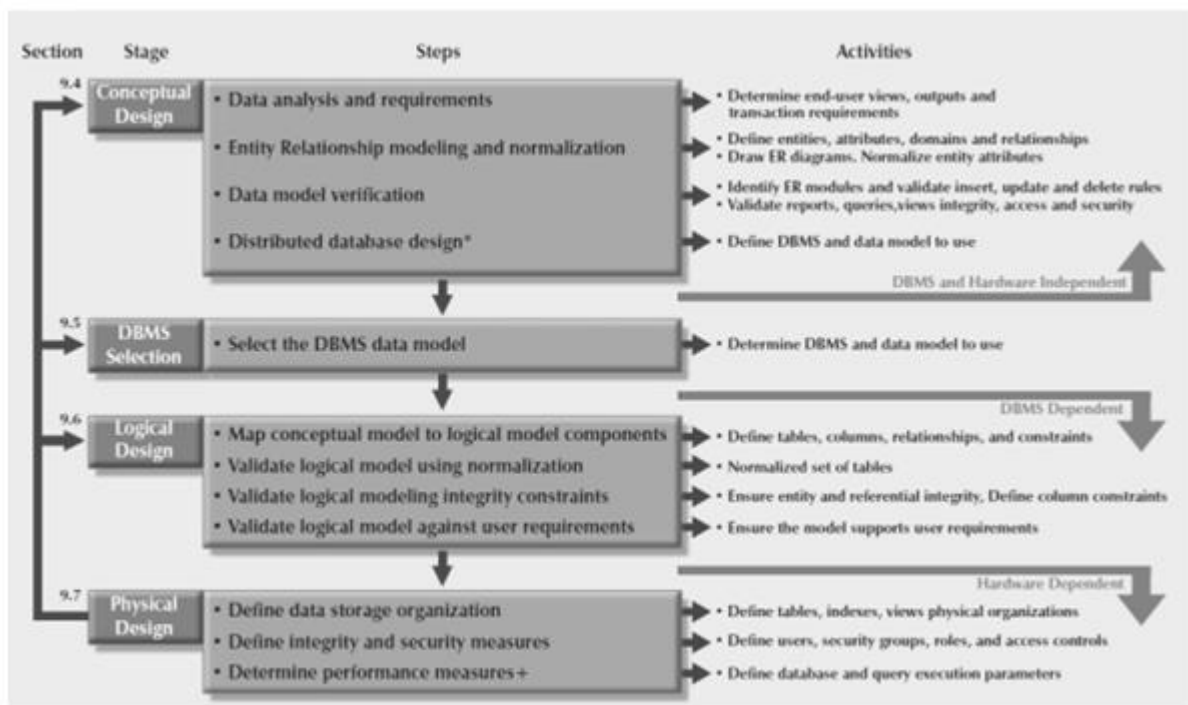


Fig. 9.5 Database design process

IMPLEMENTATION AND LOADING

The output of the database design phase is a series of instructions detailing the creation of tables, attributes, domains, views, indexes, security constraints, and storage and performance guidelines. In this phase, you actually implement all these design specifications.

INSTALL THE DBMS

This step is required only when a new dedicated instance of the DBMS is necessary for the system. In many cases, the organization will have standardized on a particular DBMS in order to leverage investments in the technology and the skills that employees have already developed. The DBMS may be installed on a new server or it may be installed on existing servers. One current trend is called virtualization. Virtualization is a technique that creates logical representations of computing resources that are independent of the underlying physical computing resources. This technique is used in many areas of computing such as the creation of virtual servers, virtual storage, and virtual private networks. In a database environment, database virtualization refers to the installation of a new instance of the DBMS on a virtual server running on shared hardware. This is normally a task that involves system and network administrators to create appropriate user groups and services in the server configuration and network routing.

CREATE THE DATABASE(S)

In most modern relational DBMSs a new database implementation requires the creation of special storage-related constructs to house the end-user tables. The constructs usually include the storage group (or file groups), the table spaces, and the tables.

LOAD OR CONVERT THE DATA

After the database has been created, the data must be loaded into the database tables. Typically, the data will have to be migrated from the prior version of the system. Often, data to be included in the system must be aggregated from multiple sources. In a best-case scenario, all of the data will be in a relational database so that it can be readily transferred to the new database. Unfortunately, this is not always the case. Data may have to be imported from other relational databases, non-relational databases, flat files, legacy systems or even manual paper-and-pencil systems. If the data format does not support direct importing into the new database, conversion programs may have to be created to reformat the data so that it can be imported. In a worst-case scenario, much of the data may have to be manually entered into the database. Once the data has been loaded, the DBA works with the application developers to test and evaluate the database.

TESTING AND EVALUATION

In the design phase, decisions were made to ensure integrity, security, performance, and recoverability of the database. During implementation and loading, these plans were put into

place. In testing and evaluation, the DBA tests and fine-tunes the database to ensure that it performs as expected. This phase occurs in conjunction with applications programming. Programmers use database tools to prototype the applications during the coding of the programs. Tools such as report generators, screen painters, and menu generators are especially useful to the applications programmers.

TEST THE DATABASE

During this step, the DBA tests the database to ensure that it maintains the integrity and security of the data. Data integrity is enforced by the DBMS through the proper use of primary and foreign key rules. Many DBMS also support the creation of domain constraints, and database triggers. Testing will ensure that these constraints were properly designed and implemented. In addition, data integrity is also the result of properly implemented data management policies. Such policies are part of a comprehensive data administration framework.

You must test for the following:

- Physical security allows only authorized personnel physical access to specific areas.
- Access rights can be established through the use of database software. The assignment of access rights may restrict operations (CREATE, UPDATE, DELETE, and so on) on predetermined objects such as databases, tables, views, queries, and reports.
- Audit trails are usually provided by the DBMS to check for access violations. Although the audit trail is an after-the-fact device, its mere existence can discourage unauthorized use.
- Data encryption can be used to render data useless to unauthorized users who might have violated some of the database security layers.
- Diskless workstations allow end users to access the database without being able to download the information from their workstations.

FINE-TUNE THE DATABASE

Although database performance can be difficult to evaluate because there are no standards for database performance measures, it is typically one of the most important factors in database implementation. Different systems will place different performance requirements on the database. Systems to support rapid transactions will require the database to be implemented in such a way so as to provide superior performance during high volumes of inserts, updates,

and deletes. Other systems, like decision support systems, may require superior performance on complex data retrieval tasks. Many factors can impact the database's performance on various tasks. Environmental factors, such as the hardware and software environment in which the database exists, can have a significant impact on database performance. Naturally, the characteristics and volume of the data in the database also affect database performance: a search of 10 tuples will be faster than a search of 100,000 tuples. Other important factors in database performance include system and database configuration parameters such as data placement, access path definition, the use of indexes, and buffer size.

EVALUATE THE DATABASE AND ITS APPLICATION PROGRAMS

As the database and application programs are created and tested, the system must also be evaluated from a more holistic approach. Testing and evaluation of the individual components should culminate in a variety of broader system tests to ensure that all of the components interact properly to meet the needs of the users. At this time, integration issues and deployment plans are refined, user training is conducted, and system documentation is finalized. Once the system receives final approval, it must be a sustainable resource for the organization. To ensure that the data contained in the database are protected against loss, backup and recovery plans are tested.

Timely data availability is crucial for almost every database. Unfortunately, the database can be subject to data loss through unintended data deletion, power outages, and other causes. Data backup and recovery procedures create a safety valve, ensuring the availability of consistent data. Typically, database vendors encourage the use of fault-tolerant components such as uninterruptible power supply (UPS) units, RAID storage devices, clustered servers, and data replication technologies to ensure the continuous operation of the database in case of a hardware failure. Even with these components, backup and restore functions constitute a very important component of daily database operations.

Some DBMSs provide functions that allow the database administrator to schedule automatic database backups to permanent storage devices such as disks, DVDs, tapes, and online storage. Database backups can be performed at different levels:

- A full backup of the database, or dump of the entire database. In this case, all database objects are backed up in their entirety.

- A differential backup of the database, in which only the last modifications to the database (when compared with a previous full backup copy) are copied. In this case, only the objects that have been updated since the last full backup are backed up.
- A transaction log backup, which backs up only the transaction log operations that are not reflected in a previous backup copy of the database. In this case, only the transaction log is backed up; no other database objects are backed up.

The database backup is stored in a secure place, usually in a different building from the database itself, and is protected against dangers such as fire, theft, flood, and other potential calamities. The main purpose of the backup is to guarantee database restoration following system (hardware/software) failures.

Failures that plague databases and systems are generally induced by software, hardware, programming exemptions, transactions, or external factors. Table 9.1 summarizes the most common sources of database failure.

Table 9.1 Common Sources of Database Failures

SOURCE	DESCRIPTION	EXAMPLE
Software	Software-induced failures may be traceable to the operating system, the DBMS software, application programs, or viruses.	The SQL.Slammer worm affected many unpatched MS SQL Server systems in 2003 causing damages valued in millions of dollars.
Hardware	Hardware-induced failures may include memory chip errors, disk crashes, bad disk sectors, and "disk full" errors.	A bad memory module or a multiple hard disk failure in a database system can bring a database system to an abrupt stop.
Programming exemptions	Application programs or end users may roll back transactions when certain conditions are defined. Programming exemptions can also be caused by malicious or improperly tested code that can be exploited by hackers.	Hackers constantly searching for exploits in unprotected Web database systems.
Transactions	The system detects deadlocks and aborts one of the transactions. (See Chapter 10.)	Deadlock occurs when executing multiple simultaneous transactions.
External factors	Backups are especially important when a system suffers complete destruction from fire, earthquake, flood, or other natural disaster.	In 2005, Hurricane Katrina in New Orleans caused data losses in the millions of dollars.

Depending on the type and extent of the failure, the recovery process ranges from a minor short-term inconvenience to a major long-term rebuild. Regardless of the extent of the required recovery process, recovery is not possible without a usable backup.

The database recovery process generally follows a predictable scenario. First, the type and extent of the required recovery are determined. If the entire database needs to be recovered to a consistent state, the recovery uses the most recent backup copy of the database in a known consistent state. The backup copy is then rolled forward to restore all subsequent transactions by using the transaction log information. If the database needs to be recovered but the committed portion of the database is still usable, the recovery process uses the transaction log to “undo” all of the transactions that were not committed.

At the end of this phase, the database completes an iterative process of continuous testing, evaluation, and modification that continues until the system is certified as ready to enter the operational phase.

OPERATION

Once the database has passed the evaluation stage, it is considered to be operational. At that point, the database, its management, its users, and its application programs constitute a complete information system.

The beginning of the operational phase invariably starts the process of system evolution. As soon as all of the targeted end users have entered the operations phase, problems that could not have been foreseen during the testing phase begin to surface. Some of the problems are serious enough to warrant emergency “patchwork,” while others are merely minor annoyances.

MAINTENANCE AND EVOLUTION

The database administrator must be prepared to perform routine maintenance activities within the database. Some of the required periodic maintenance activities include:

- Preventive maintenance (backup).
- Corrective maintenance (recovery).
- Adaptive maintenance (enhancing performance, adding entities and attributes, and so on).
- Assignment of access permissions and their maintenance for new and old users.
- Generation of database access statistics to improve the efficiency and usefulness of system audits and to monitor system performance.
- Periodic security audits based on the system-generated statistics.
- Periodic (monthly, quarterly, or yearly) system-usage summaries for internal billing or budgeting purposes.

A summary of the parallel activities that take place within the SDLC and the DBLC is shown in Figure 9.6.

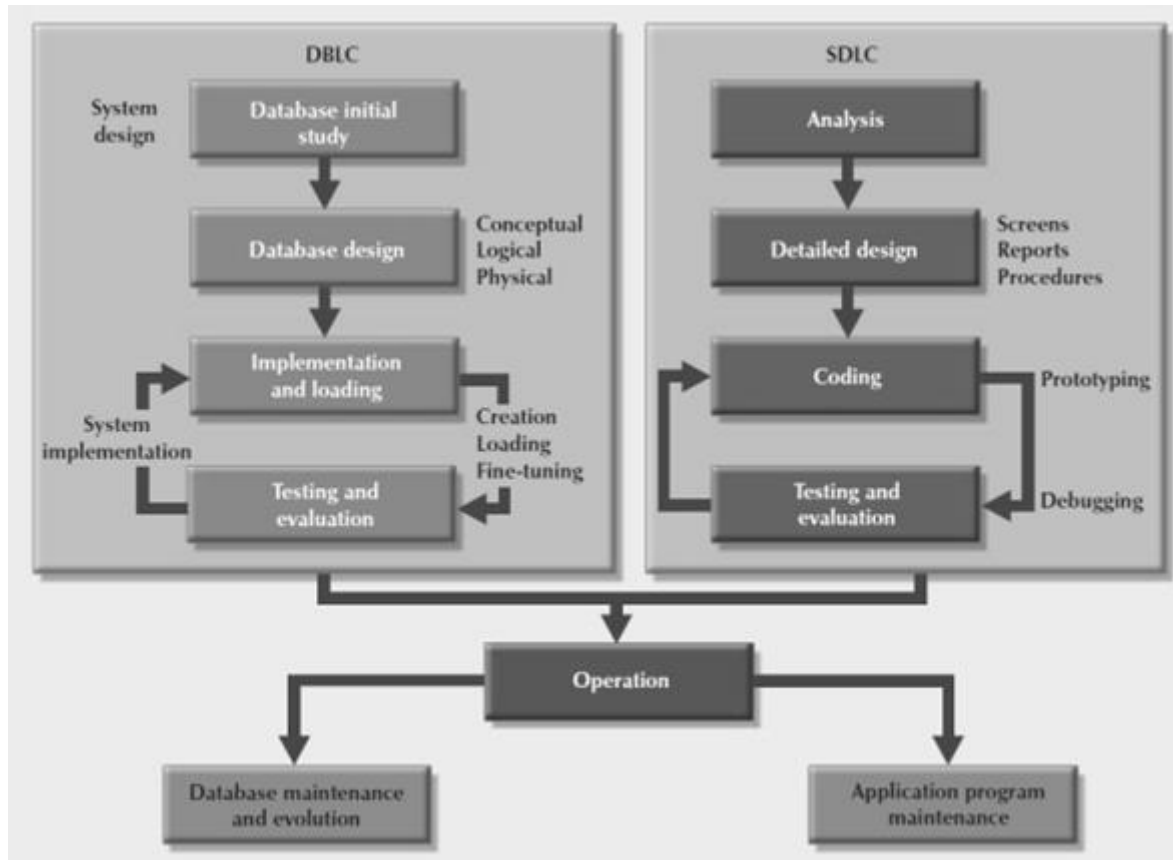


Fig 9.6 Parallel activities in the DBLC and the SDLC

9.4 CONCEPTUAL DESIGN

Recall that the second phase of the DBLC is database design, and that database design comprises four stages: conceptual design, DBMS selection, logical design, and physical design.

Conceptual design is the first stage in the database design process. The goal at this stage is to design a database that is independent of database software and physical details. The output of this process is a conceptual data model that describes the main data entities, attributes, relationships, and constraints of a given problem domain. This design is descriptive and narrative in form. That is, it is generally composed of a graphical representation as well as textual descriptions of the main data elements, relationships, and constraints. In this stage, data modelling is used to create an abstract database structure that represents real-world objects in the most realistic way possible. The conceptual model must embody a clear understanding of the business and its functional areas. At this level of abstraction, the type of

hardware and/or database model to be used might not have been identified yet. Therefore, the design must be software and hardware independent so that the system can be set up within any hardware and software platform chosen later.

Keep in mind the following minimal data rule:

All that is needed is there, and all that is there is needed. In other words, make sure that all data needed are in the model and that all data in the model are needed. All data elements required by the database transactions must be defined in the model, and all data elements defined in the model must be used by at least one database transaction.

However, as you apply the minimal data rule, avoid an excessive short-term bias. Focus not only on the immediate data needs of the business but also on the future data needs. Thus, the database design must leave room for future modifications and additions, ensuring that the business's investment in information resources will endure. The conceptual design has four steps, which are depicted in Table 9.2.

Table 9.2. Conceptual Design Steps

STEP	ACTIVITY
1	Data analysis and requirements
2	Entity relationship modeling and normalization
3	Data model verification
4	Distributed database design

Data Analysis and Requirements

The first step in conceptual design is to discover the characteristics of the data elements. An effective database is an information factory that produces key ingredients for successful decision making. Appropriate data element characteristics are those that can be transformed into appropriate information.

Entity Relationship Modelling and Normalization

Before creating the ER model, the designer must communicate and enforce appropriate standards to be used in the documentation of the design. The standards include the use of diagrams and symbols, documentation writing style, layout, and any other inventions to be followed during documentation. Designers often overlook this very important requirement, especially when they are working as members of a design team. Failure to standardize documentation often means a failure to communicate later, and communications failures often lead to poor design work. In contrast, well-defined and enforced standards make design

work easier and promise (but do not guarantee) a smooth integration of all system components.

Because the business rules usually define the nature of the relationship(s) among the entities, the designer must incorporate them into the conceptual model. The process of defining business rules and developing the conceptual model using ER diagrams can be described using the steps shown in Table 9.3.

Table 9.3 Developing the Conceptual Model Using ER Diagrams

STEP	ACTIVITY
1	Identify, analyze, and refine the business rules.
2	Identify the main entities, using the results of Step 1.
3	Define the relationships among the entities, using the results of Steps 1 and 2.
4	Define the attributes, primary keys, and foreign keys for each of the entities.
5	Normalize the entities. (Remember that entities are implemented as tables in an RDBMS.)
6	Complete the initial ER diagram.
7	Validate the ER model against the end users' information and processing requirements.
8	Modify the ER model, using the results of Step 7.

Some of the steps listed in Table 9.3 take place concurrently. And some, such as the normalization process, can generate a demand for additional entities and/or attributes, thereby causing the designer to revise the ER model.

9.5 DBMS SOFTWARE SELECTION

The selection of DBMS software is critical to the information system's smooth operation. Consequently, the advantages and disadvantages of the proposed DBMS software should be carefully studied. To avoid false expectations, the end user must be made aware of the limitations of both the DBMS and the database.

Although the factors affecting the purchasing decision vary from company to company, some of the most common are:

- Cost. This includes the original purchase price, along with maintenance, operational, license, installation, training, and conversion costs.
- DBMS features and tools. Some database software includes a variety of tools that facilitate the application development task. For example, the availability of query by example (QBE), screen painters, report generators, application generators, data dictionaries, and so on, helps to create a more pleasant work environment for both the end user and the application programmer. Database administrator facilities,

query facilities, ease of use, performance, security, concurrency control, transaction processing, and third-party support also influence DBMS software selection.

- Underlying model. This can be hierarchical, network, relational, object/relational, or object-oriented.
- Portability. A DBMS can be portable across platforms, systems, and languages.
- DBMS hardware requirements. Items to consider include processor(s), RAM, disk space, and so on.

9.6 LOGICAL DESIGN

Logical design is the second stage in the database design process. The logical design goal is to design an enterprise-wide database based on a specific data model but independent of physical-level details. Logical design requires that all objects in the conceptual model be mapped to the specific constructs used by the selected databasemodel.

The logical design is generally performed in four steps, which are depicted in Table 9.4.

Table 9.4 Logical Design Steps

STEP	ACTIVITY
1	Map conceptual model to logical model components
2	Validate logical model using normalization
3	Validate logical model integrity constraints
4	Validate logical model against user requirements

Such steps, like most of the data-modelling process, are not necessarily performed sequentially, but in an iterative fashion. The following sections cover these steps in more detail.

9.7 PHYSICAL DESIGN

Physical design is the process of determining the data storage organization and data access characteristics of the database in order to ensure its integrity, security, and performance. This is the last stage in the database design process. The storage characteristics are a function of the types of devices supported by the hardware, the type of data access methods supported by the system, and the DBMS. Physical design could become a very technical job that affects not only the accessibility of the data in the storage device(s) but also the performance of the system.

The physical design stage is composed of the steps depicted in Table 9.5.

Table 9.5 Physical Design Steps

STEP	ACTIVITY
1	Define data storage organization
2	Define integrity and security measures
3	Determine performance measurements

The following sections cover these steps in more detail.

Define Data Storage Organization

Before you can define the data storage organization, you must determine the volume of data to be managed and the data usage patterns.

- Knowing the data volume will help you determine how much storage space to reserve for the database. To do this, the designer follows a process similar to the one used during the ER model verification process. For each table, identify all possible transactions, their frequency, and volume. For each transaction, you determine the amount of data to be added or deleted from the database. This information will help you determine the amount of data to be stored in the related table.
- Conversely, knowing how frequently the new data is inserted, updated, and retrieved will help the designer to determine the data usage patterns. Usage patterns are critical, in particular in distributed database design. For example, are there any weekly batch uploads or monthly aggregation reports to be generated? How frequently is new data added to the system?

Equipped with the two previous pieces of information, the designer must:

- Determine the location and physical storage organization for each table. In this step the designer assigns which tables will use what table spaces and the location of the table spaces.
- Identify what indexes and the type of indexes to be used for each table. Indexes are useful for ensuring the uniqueness of data values in a column and to facilitate data lookups. You also know that the DBMS automatically creates a unique index for the primary key of each table.
- Identify what views and the type of views to be used on each table. A view is useful to limit access to data based on user or transaction needs. Views can also be used to simplify processing and end-user data access. In this step the designer must ensure that all views can be implemented and that they provide only the required data. At

this time, the designer must also get familiar with the types of views supported by the DBMS and how those types of views could help meet system goals.

Define Integrity and Security Measures

Once the physical organization of the tables, indexes, and views are defined, the database is ready to be used by the end users. But before a user can access the data in the database, he or she must be properly authenticated. In this step of physical design, two tasks must be addressed:

Define user and security groups and roles.

User management is more a function of database administration than database design. But, as a designer you must be aware of the different types of users and group of users in order to properly enforce database security. Most DBMS implementations support the use of database roles.

Determine Performance Measures

Physical design becomes more complex when data are distributed at different locations because the performance is affected by the communication media's throughput. Given such complexities, it is not surprising that designers favour database software that hides as many of the physical-level activities as possible. In spite of the fact that relational models tend to hide the complexities of the computer's physical characteristics, the performance of relational databases is affected by physical storage characteristics. For example, performance can be affected by the characteristics of the storage media, such as seek time, sector and block (page) size, buffer pool size, and the number of disk platters and read/write heads. In addition, factors such as the creation of an index can have a considerable effect on the relational database's performance, that is, data access speed and efficiency.

In summary, physical design performance measurement deals with fine-tuning the DBMS and queries to ensure that they will meet end-user performance requirements. The preceding sections have separated the discussions of logical and physical design activities. In fact, logical and physical design can be carried out in parallel, on a table-by-table basis. Such parallel activities require the designer to have a thorough understanding of the software and hardware in order to take full advantage of both software and hardware characteristics.

9.8 DATABASE DESIGN STRATEGIES

There are two classical approaches to database design:

- Top-down design starts by identifying the data sets and then defines the data elements for each of those sets. This process involves the identification of different entity types and the definition of each entity's attributes.
- Bottom-up design first identifies the data elements (items) and then groups them together in data sets. In other words, it first defines attributes, and then groups them to form entities.

The two approaches are illustrated in Figure 9.7. The selection of a primary emphasis on top-down or bottom-up procedures often depend on the scope of the problem or on personal preferences. Although the two methodologies are complementary rather than mutually exclusive, a primary emphasis on a bottom-up approach may be more productive for small databases with few entities, attributes, relations, and transactions. For situations in which the number, variety, and complexity of entities, relations, and transactions is overwhelming, a primarily top-down approach may be more easily managed. Most companies have standards for systems development and database design already in place.

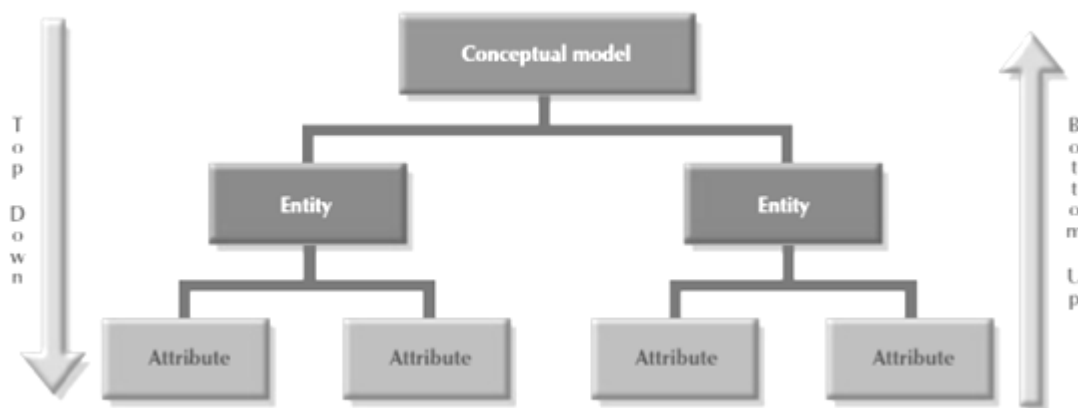


Fig. 9.7 Top-down vs. bottom-up design sequencing

9.9 CENTRALIZED VERSUS DECENTRALIZED DESIGNS

The two general approaches (bottom-up and top-down) to database design can be influenced by factors such as the scope and size of the system, the company's management style, and the company's structure (centralized or decentralized). Depending on such factors,

the database design may be based on two very different design philosophies: centralized and decentralized.

Centralized design is productive when the data component is composed of a relatively small number of objects and procedures. The design can be carried out and represented in a fairly simple database. Centralized design is typical of relatively simple and/or small databases and can be successfully done by a single person (database administrator) or by a small, informal design team. The company operations and the scope of the problem are sufficiently limited to allow even a single designer to define the problem(s), create the conceptual design, verify the conceptual design with the user views, define system processes and data constraints to ensure the efficacy of the design, and ensure that the design will comply with all the requirements.

Figure 9.8 summarizes the centralized design option. Note that a single conceptual design is completed and then validated in the centralized design approach.

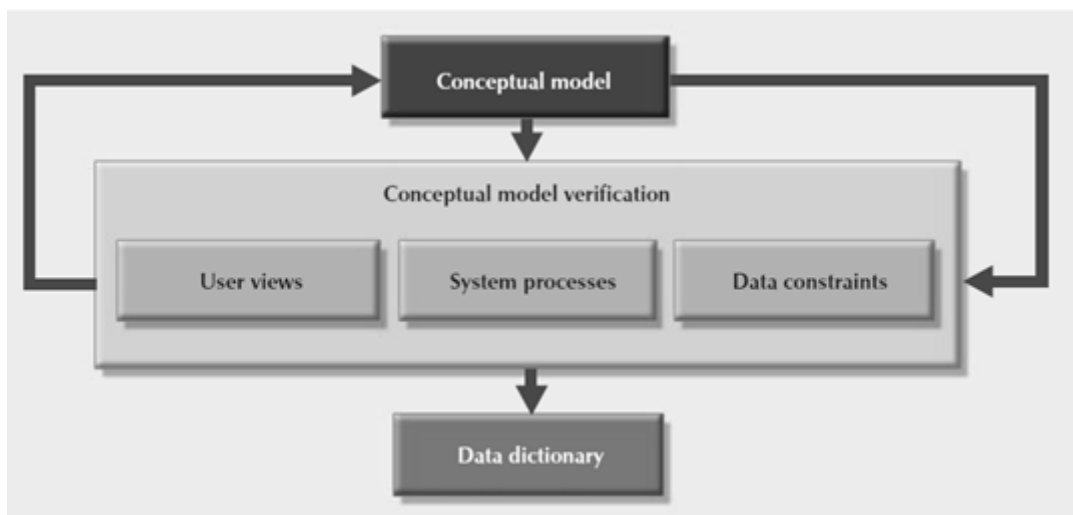


Fig. 9.8 Centralized design

Decentralized design might be used when the data component of the system has a considerable number of entities and complex relations on which very complex operations are performed. Decentralized design is also likely to be employed when the problem itself is spread across several operational sites and each element is a subset of the entire data set. (See Figure 9.9.)

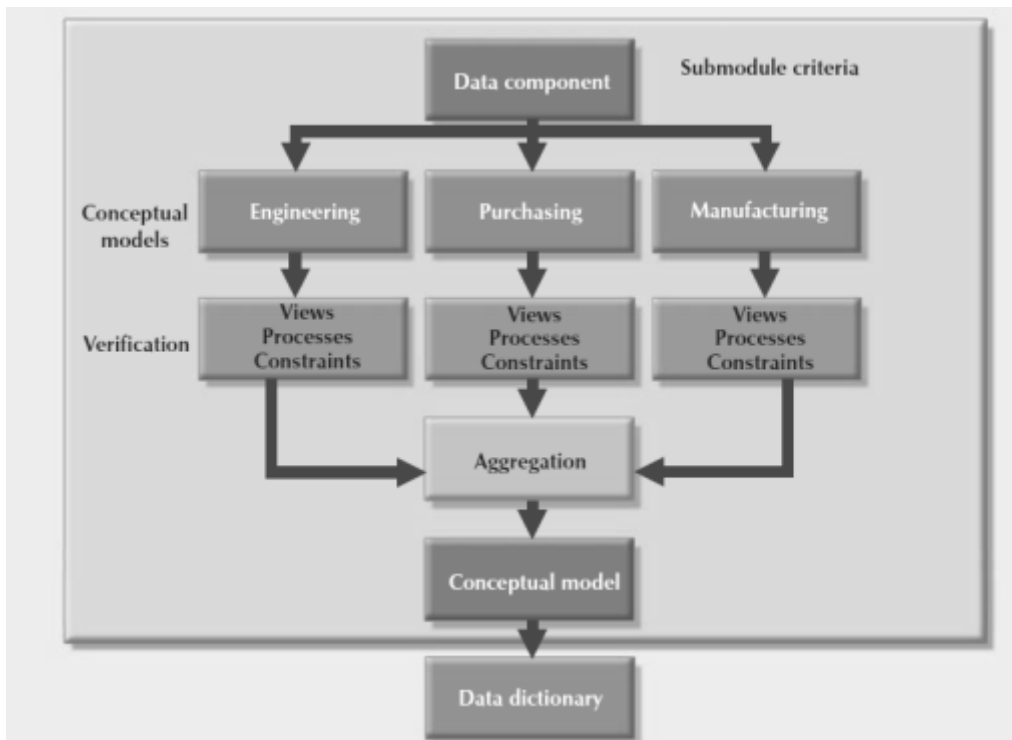


Fig. 9.9. Decentralized design

In large and complex projects, the database design typically cannot be done by only one person. Instead, a carefully selected team of database designers is employed to tackle a complex database project. Within the decentralized design framework, the database design task is divided into several modules. Once the design criteria have been established, the lead designer assigns design subsets or modules to design groups within the team. Because each design group focuses on modelling a subset of the system, the definition of boundaries and the interrelation among data subsets must be very precise. Each design group creates a conceptual data model corresponding to the subset being modelled. Each conceptual model is then verified individually against the user views, processes, and constraints for each of the modules. After the verification process has been completed, all modules are integrated into one conceptual model. Because the data dictionary describes the characteristics of all objects within the conceptual data model, it plays a vital role in the integration process. Naturally, after the subsets have been aggregated into a larger conceptual model, the lead designer must verify that the combined conceptual model is still able to support all of the required transactions.

9.10 CHECK YOUR PROGRESS

1. What is an information system?
2. What does the acronym SDLC mean?

3. What does the acronym DBLC mean?
4. What is the minimal data rule in conceptual design?
5. What are business rules?
6. What is the data dictionary's function in database design?

Answers to check your progress:

1. An information system is designed to facilitate the transformation of data into information and to manage both data and information.
2. The Systems Development Life Cycle
3. The Database Life Cycle
4. All that is needed is there, and all that is there is needed.
5. Using simple language, business rules describe the main and distinguishing characteristics of the data as viewed by the company.
6. The data dictionary is sometimes described as “the database designer's database” because it records the design decisions about tables and their structures.

9.11 SUMMARY

- An information system is designed to facilitate the transformation of data into information and to manage both data and information. Thus, the database is a very important part of the information system. Systems analysis is the process that establishes the need for and the extent of an information system. Systems development is the process of creating an information system.
- The Systems Development Life Cycle (SDLC) traces the history (life cycle) of an application within the information system. The SDLC can be divided into five phases: planning, analysis, detailed systems design, implementation, and maintenance. The SDLC is an iterative rather than a sequential process.
- The Database Life Cycle (DBLC) describes the history of the database within the information system. The DBLC is composed of six phases: database initial study, database design, implementation and loading, testing and evaluation, operation, and maintenance and evolution. Like the SDLC, the DBLC is iterative rather than sequential.
- The database design and implementation process moves through a series of well-defined stages: database initial study, database design, implementation and loading, testing and evaluation, operation, and maintenance and evolution.

- The conceptual portion of the design may be subject to several variations based on two basic design philosophies: bottom-up vs. top-down and centralized vs. decentralized.

9.12 KEYWORDS

- **Bottom-up design** –It first identifies the data elements (items) and then groups them together in data sets. In other words, it first defines attributes, and then groups them to form entities.
- **Centralized design** - is typical of relatively simple and/or small databases and can be successfully done by a single person (database administrator) or by a small, informal design team.
- **Clustered tables** – is a storage technique stores related rows from two related tables in adjacent data blocks on disk.
- **Cohesivity** - describes the strength of the relationships found among the module’s entities.
- **Conceptual design** - is the first stage in the database design process. The goal at this stage is to design a database that is independent of database software and physical details.

9.13 QUESTIONS FOR SELF-STUDY

1. Discuss the distinction between centralized and decentralized conceptual database design.
2. Discuss the distinction between top-down and bottom-up approaches in database design.
3. What factors are important in a DBMS software selection?
4. What is the data dictionary’s function in database design?
5. What three levels of backup may be used in database recovery management? Briefly describe what each of those three backup levels does

9.14 REFERENCES

1. Coronel, C., & Morris, S. (2016). Database systems: design, implementation, & management. Cengage Learning.
2. Gupta, S. B., & Mittal, A. (2009). Introduction to database management system.Laxmi Publications, Ltd..
3. Ramakrishnan, R., &Gehrke, J. (2000). Database management systems.McGraw-Hill.

UNIT -10: TRANSACTION MANAGEMENT AND CONCURRENCY CONTROL

STRUCTURE

- 10.0 Objectives
- 10.1 What Is a Transaction? Concurrency Control
- 10.2 Concurrency Control
- 10.3 Concurrency Control with Locking Methods
- 10.4 Concurrency Control with Time Stamping Methods
- 10.5 Concurrency Control with Optimistic Methods
- 10.6 ANSI Levels of Transaction Isolation
- 10.7 Database Recovery Management
- 10.8 Check your progress
- 10.9 Summary
- 10.10 Keywords
- 10.11 Questions for self-study
- 10.12 References

10.0 OBJECTIVES

After studying this unit, you will be able to:

- ✓ Explain about database transactions and their properties
- ✓ Define what concurrency control is and what role it plays in maintaining the database's integrity
- ✓ Describe what locking methods are and how they work
- ✓ Discuss how stamping methods are used for concurrency control
- ✓ Discuss how optimistic methods are used for concurrency control
- ✓ Explain how database recovery management is used to maintain database integrity

10.1 What Is a Transaction?

To illustrate what transactions are and how they work, let's use a sample database. The relational diagram for that database is shown in Figure 10.1.

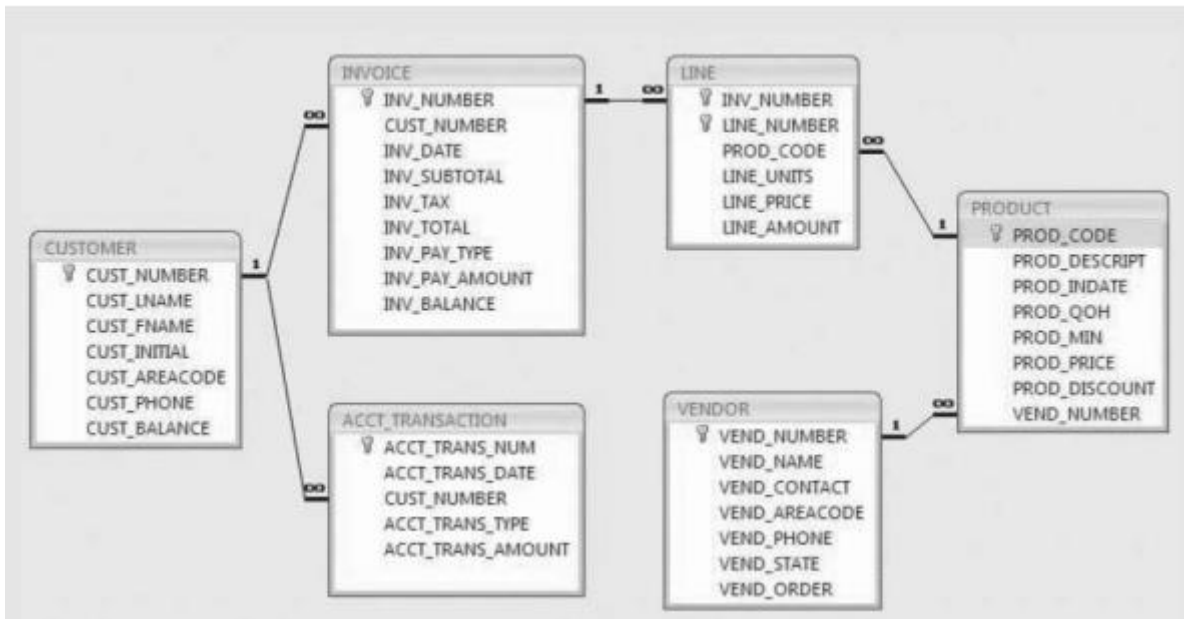


Fig. 10.1 SaleCo a sample database relational diagram

As you examine the relational diagram in Figure 10.1, note the following features:

- The design stores the customer balance (CUST_BALANCE) value in the CUSTOMER table to indicate the total amount owed by the customer. The CUST_BALANCE attribute is increased when the customer makes a purchase on credit, and it is decreased when the customer makes a payment. Including the current customer account balance in the CUSTOMER table makes it very easy to write a query to determine the current balance for any customer and to generate important summaries such as total, average, minimum, and maximum balances.
- The ACCT_TRANSACTION table records all customer purchases and payments to track the details of customer account activity.

To understand the concept of a transaction, suppose that you sell a product to a customer. Furthermore, suppose that the customer may charge the purchase to his or her account. Given that scenario, your sales transaction consists of at least the following parts:

- You must write a new customer invoice.
- You must reduce the quantity on hand in the product's inventory.
- You must update the account transactions.
- You must update the customer balance.

The preceding sales transaction must be reflected in the database. In database terms, a transaction is any action that reads from and/or writes to a database. A transaction may

consist of a simple SELECT statement to generate a list of table contents; it may consist of a series of related UPDATE statements to change the values of attributes in various tables; it may consist of a series of INSERT statements to add rows to one or more tables or it may consist of a combination of SELECT, UPDATE, and INSERT statements. The sales transaction example includes a combination of INSERT and UPDATE statements.

Given the preceding discussion, you can now augment the definition of a transaction. A transaction is a logical unit of work that must be entirely completed or entirely aborted; no intermediate states are acceptable. In other words, a multicomponent transaction, such as the previously mentioned sale, must not be partially completed. Updating only the inventory or only the accounts receivable is not acceptable. All of the SQL statements in the transaction must be completed successfully. If any of the SQL statements fail, the entire transaction is rolled back to the original database state that existed before the transaction started. A successful transaction changes the database from one consistent state to another. A consistent database state is one in which all data integrity constraints are satisfied.

To ensure consistency of the database, every transaction must begin with the database in a known consistent state. If the database is not in a consistent state, the transaction will yield an inconsistent database that violates its integrity and business rules. For that reason, subject to limitations discussed later, all transactions are controlled and executed by the DBMS to guarantee database integrity.

Most real-world database transactions are formed by two or more database requests. A database request is the equivalent of a single SQL statement in an application program or transaction. For example, if a transaction is composed of two UPDATE statements and one INSERT statement, the transaction uses three database requests. In turn, each database request generates several input/output (I/O) operations that read from or write to physical storage media.

Evaluating Transaction Results

Not all transactions update the database.

A transaction may consist of a single SQL statement or a collection of related SQL statements.

Now suppose that there are four SQL statements in a transaction and DBMS completes the first three SQL statements. Furthermore, suppose that during the execution of the fourth

statement the computer system experiences a loss of electrical power. If the computer does not have a backup power supply, the transaction cannot be completed.

Suppose the first three statements have modified the database then the database is in an inconsistent state, and it is not usable for subsequent transactions. Assuming that the DBMS supports transaction management, the DBMS will roll back the database to a previous consistent state. Although the DBMS is designed to recover a database to a previous consistent state when an interruption prevents the completion of a transaction, the transaction itself is defined by the end user or programmer and must be semantically correct. The DBMS cannot guarantee that the semantic meaning of the transaction truly represents the real-world event.

Clearly, improper or incomplete transactions can have a devastating effect on database integrity. Some DBMSs—especially the relational variety—provide means by which the user can define enforceable constraints based on business rules. Other integrity rules, such as those governing referential and entity integrity, are enforced automatically by the DBMS when the table structures are properly defined, thereby letting the DBMS validate some transactions. For example, if a transaction inserts a new customer number into a customer table and the customer number being inserted already exists, the DBMS will end the transaction with an error code to indicate a violation of the primary key integrity rule.

Transaction Properties

Each individual transaction must display atomicity, consistency, isolation, and durability. These properties are sometimes referred to as the ACID test. In addition, when executing multiple transactions, the DBMS must schedule the concurrent execution of the transaction's operations. The schedule of such transaction's operations must exhibit the property of serializability. Let's look briefly at each of the properties.

- **Atomicity** requires that all operations (SQL requests) of a transaction be completed; if not, the transaction is aborted. If a transaction T1 has four SQL requests, all four requests must be successfully completed; otherwise, the entire transaction is aborted. In other words, a transaction is treated as a single, indivisible, logical unit of work.
- **Consistency** indicates the permanence of the database's consistent state. A transaction takes a database from one consistent state to another consistent state. When a transaction is completed, the database must be in a consistent state; if any

of the transaction parts violates an integrity constraint, the entire transaction is aborted.

- **Isolation** means that the data used during the execution of a transaction cannot be used by a second transaction until the first one is completed. In other words, if a transaction T1 is being executed and is using the data item X, that data item cannot be accessed by any other transaction (T2 ... Tn) until T1 ends. This property is particularly useful in multiuser database environments because several users can access and update the database at the same time.
- **Durability** ensures that once transaction changes are done (committed), they cannot be undone or lost, even in the event of a system failure.

Serializability ensures that the schedule for the concurrent execution of the transactions yields consistent results. This property is important in multiuser and distributed databases, where multiple transactions are likely to be executed concurrently. Naturally, if only a single transaction is executed, serializability is not an issue.

A single-user database system automatically ensures serializability and isolation of the database because only one transaction is executed at a time. The atomicity, consistency, and durability of transactions must be guaranteed by the single-user DBMSs. (Even a single-user DBMS must manage recovery from errors created by operating-system-induced interruptions, power interruptions, and improper application execution.)

Multiuser databases are typically subject to multiple concurrent transactions. Therefore, the multiuser DBMS must implement controls to ensure serializability and isolation of transactions—in addition to atomicity and durability—to guard the database's consistency and integrity. For example, if several concurrent transactions are executed over the same data set and the second transaction updates the database before the first transaction is finished, the isolation property is violated and the database is no longer consistent. The DBMS must manage the transactions by using concurrency control techniques to avoid such undesirable situations.

Transaction Management with SQL

The American National Standards Institute (ANSI) has defined standards that govern SQL database transactions. Transaction support is provided by two SQL statements: COMMIT and ROLLBACK. The ANSI standards require that when a transaction sequence is initiated by a

user or an application program, the sequence must continue through all succeeding SQL statements until one of the following four events occurs:

- A COMMIT statement is reached, in which case all changes are permanently recorded within the database. The COMMIT statement automatically ends the SQL transaction.
 - A ROLLBACK statement is reached, in which case all changes are aborted and the database is rolled back to its previous consistent state.
 - The end of a program is successfully reached, in which case all changes are permanently recorded within the database. This action is equivalent to COMMIT.
 - The program is abnormally terminated, in which case the changes made in the database are aborted and the database is rolled back to its previous consistent state. This action is equivalent to ROLLBACK.

A transaction begins implicitly when the first SQL statement is encountered. Not all SQL implementations follow the ANSI standard; some (such as SQL Server) use transaction management statements such as:

BEGIN TRANSACTION;

To indicate the beginning of a new transaction. Other SQL implementations allow you to assign characteristics for the transactions as parameters to the BEGIN statement. For example, the Oracle RDBMS uses the SET TRANSACTION statement to declare a new transaction start and its properties.

The Transaction Log

A DBMS uses a transaction log to keep track of all transactions that update the database. The information stored in this log is used by the DBMS for a recovery requirement triggered by a ROLLBACK statement, a program's abnormal termination, or a system failure such as a network discrepancy or a disk crash. Some RDBMSs use the transaction log to recover a database forward to a currently consistent state. After a server failure, for example, Oracle automatically rolls back uncommitted transactions and rolls forward transactions that were committed but not yet written to the physical database. This behaviour is required for transactional correctness and is typical of any transactional DBMS.

While the DBMS executes transactions that modify the database, it also automatically updates the transaction log. The transaction log stores:

- A record for the beginning of the transaction.
- For each transaction component (SQL statement):
 - The type of operation being performed (update, delete, insert).
 - The names of the objects affected by the transaction (the name of the table).
 - The “before” and “after” values for the fields being updated.
 - Pointers to the previous and next transaction log entries for the same transaction.
- The ending (COMMIT) of the transaction.

Although using a transaction log increases the processing overhead of a DBMS, the ability to restore a corrupted database is worth the price.

10.2 CONCURRENCY CONTROL

The coordination of the simultaneous execution of transactions in a multiuser database system is known as concurrency control. The objective of concurrency control is to ensure the serializability of transactions in a multiuser database environment. Concurrency control is important because the simultaneous execution of transactions over a shared database can create several data integrity and consistency problems. The three main problems are lost updates, uncommitted data, and inconsistent retrievals.

Lost Updates

The lost update problem occurs when two concurrent transactions, T1 and T2, are updating the same data element and one of the updates is lost (overwritten by the other transaction). To see an illustration of lost updates, let’s examine a simple PRODUCT table. One of the PRODUCT table’s attributes is a product’s quantity on hand (PROD_QOH). Assume that you have a product whose current PROD_QOH value is 35. Also assume that two concurrent transactions, T1 and T2, occur that update the PROD_QOH value for some item in the PRODUCT table. The transactions are shown in Table 10.2:

Table 10.2 Two concurrent transactions to update QOH

TRANSACTION	COMPUTATION
T1: Purchase 100 units	$PROD_QOH = PROD_QOH + 100$
T2: Sell 30 units	$PROD_QOH = PROD_QOH - 30$

Table 10.3 shows the serial execution of those transactions under normal circumstances, yielding the correct answer $PROD_QOH = 105$.

Table 10.3 Serial execution of two transactions

TIME	TRANSACTION	STEP	STORED VALUE
1	T1	Read PROD_QOH	35
2	T1	PROD_QOH = 35 + 100	
3	T1	Write PROD_QOH	135
4	T2	Read PROD_QOH	135
5	T2	PROD_QOH = 135 - 30	
6	T2	Write PROD_QOH	105

But suppose that a transaction is able to read a product’s PROD_QOH value from the table before a previous transaction (using the same product) has been committed. The sequence depicted in Table 10.4 shows how the lost update problem can arise. Note that the first transaction (T1) has not yet been committed when the second transaction (T2) is executed. Therefore, T2 still operates on the value 35, and its subtraction yields 5 in memory. In the meantime, T1 writes the value 135 to disk, which is promptly overwritten by T2. In short, the addition of 100 units is “lost” during the process.

Table 10.4 lost updates

TIME	TRANSACTION	STEP	STORED VALUE
1	T1	Read PROD_QOH	35
2	T2	Read PROD_QOH	35
3	T1	PROD_QOH = 35 + 100	
4	T2	PROD_QOH = 35 - 30	
5	T1	Write PROD_QOH (Lost update)	135
6	T2	Write PROD_QOH	5

Uncommitted Data

The phenomenon of uncommitted data occurs when two transactions, T1 and T2, are executed concurrently and the first transaction (T1) is rolled back after the second transaction (T2) has already accessed the uncommitted data—thus violating the isolation property of transactions. To illustrate that possibility, let’s use the same transactions described during the lost updates discussion. T1 has two atomic parts to it, one of which is the update of the inventory, the other possibly being the update of the invoice total (not shown). T1 is forced to roll back due to an error during the updating of the invoice’s total; hence, it rolls back all the way, undoing the inventory update as well. This time the T1 transaction is rolled back to eliminate the addition of the 100 units. Because T2 subtracts 30 from the original 35 units, the correct answer should be 5.

Table 10.5 Transactions creating an uncommitted data problem

TRANSACTION	COMPUTATION
T1: Purchase 100 units	PROD_QOH = PROD_QOH + 100 (Rolled back)
T2: Sell 30 units	PROD_QOH = PROD_QOH - 30

Table 10.6 shows how, under normal circumstances, the serial execution of those transactions yields the correct Answer.

Table 10.6 Correct execution of two transactions

TIME	TRANSACTION	STEP	STORED VALUE
1	T1	Read PROD_QOH	35
2	T1	PROD_QOH = 35 + 100	
3	T1	Write PROD_QOH	135
4	T1	*****ROLLBACK *****	35
5	T2	Read PROD_QOH	35
6	T2	PROD_QOH = 35 - 30	
7	T2	Write PROD_QOH	5

Table 10.7 shows how the uncommitted data problem can arise when the ROLLBACK is completed after T2 has begun its execution.

Table 10.7 An uncommitted data problem

TIME	TRANSACTION	STEP	STORED VALUE
1	T1	Read PROD_QOH	35
2	T1	PROD_QOH = 35 + 100	
3	T1	Write PROD_QOH	135
4	T2	Read PROD_QOH (Read uncommitted data)	135
5	T2	PROD_QOH = 135 - 30	
6	T1	***** ROLLBACK *****	35
7	T2	Write PROD_QOH	105

Inconsistent Retrievals

Inconsistent retrievals occur when a transaction accesses data before and after another transaction(s) finish working with such data. For example, an inconsistent retrieval would occur if transaction T1 calculated some summary (aggregate) function over a set of data while another transaction (T2) was updating the same data. The problem is that the transaction might read some data before they are changed and other data after they are changed, thereby yielding inconsistent results.

To illustrate that problem, assume the following conditions:

1. T1 calculates the total quantity on hand of the products stored in the PRODUCT table.
2. At the same time, T2 updates the quantity on hand (PROD_QOH) for two of the PRODUCT table's products.

The two transactions are shown in Table 10.8.

Table 10.8 retrieval during update

TRANSACTION T1		TRANSACTION T2	
SELECT	SUM(PROD_QOH)	UPDATE	PRODUCT
FROM	PRODUCT	SET	PROD_QOH = PROD_QOH + 10
		WHERE	PROD_CODE = '1546-QQ2'
		UPDATE	PRODUCT
		SET	PROD_QOH = PROD_QOH - 10
		WHERE	PROD_CODE = '1558-QW1'
		COMMIT;	

While T1 calculates the total quantity on hand (PROD_QOH) for all items, T2 represents the correction of a typing error: the user added 10 units to product 1558-QW1's PROD_QOH but meant to add the 10 units to product 1546-QQ2's PROD_QOH. To correct the problem, the user adds 10 to product 1546-QQ2's PROD_QOH and subtracts 10 from product 1558-QW1's PROD_QOH. (See the two UPDATE statements in Table 10.7.) The initial and final PROD_QOH values are reflected in Table 10.9. (Only a few of the PROD_CODE values for the PRODUCT table are shown. To illustrate the point, the sum for the PROD_QOH values is given for those few products.)

Table 10.9 Transaction results: data entry correction

	BEFORE	AFTER
PROD_CODE	PROD_QOH	PROD_QOH
11QER/31	8	8
13-Q2/P2	32	32
1546-QQ2	15	(15 + 10) → 25
1558-QW1	23	(23 - 10) → 13
2232-QTY	8	8
2232-QWE	6	6
Total	92	92

Although the final results shown in Table 10.8 are correct after the adjustment, Table 10.10 demonstrates that inconsistent retrievals are possible during the transaction execution, making the result of T1's execution incorrect. The "After" summation shown in Table 10.9 reflects the fact that the value of 25 for product 1546-QQ2 was read after the WRITE statement was completed. Therefore, the "After" total is $40 + 25 = 65$. The "Before" total reflects the fact that the value of 23 for product 1558-QW1 was read before the next WRITE

statement was completed to reflect the corrected update of 13. Therefore, the “Before” total is $65 + 23 = 88$.

Table 10.10 Inconsistent retrievals

TIME	TRANSACTION	ACTION	VALUE	TOTAL
1	T1	Read PROD_QOH for PROD_CODE = '11QER/31'	8	8
2	T1	Read PROD_QOH for PROD_CODE = '13-Q2/P2'	32	40
3	T2	Read PROD_QOH for PROD_CODE = '1546-QQ2'	15	
4	T2	PROD_QOH = 15 + 10		
5	T2	Write PROD_QOH for PROD_CODE = '1546-QQ2'	25	
6	T1	Read PROD_QOH for PROD_CODE = '1546-QQ2'	25	(After) 65
7	T1	Read PROD_QOH for PROD_CODE = '1558-QW1'	23	(Before) 88
8	T2	Read PROD_QOH for PROD_CODE = '1558-QW1'	23	
9	T2	PROD_QOH = 23 - 10		
10	T2	Write PROD_QOH for PROD_CODE = '1558-QW1'	13	
11	T2	***** COMMIT *****		
12	T1	Read PROD_QOH for PROD_CODE = '2232-QTY'	8	96
13	T1	Read PROD_QOH for PROD_CODE = '2232-QWE'	6	102

The computed answer of 102 is obviously wrong because you know from Table 10.9 that the correct answer is 92.

Unless the DBMS exercises concurrency control, a multiuser database environment can create havoc within the information system.

The Scheduler

You now know that severe problems can arise when two or more concurrent transactions are executed. You also know that a database transaction involves a series of database I/O operations that take the database from one consistent state to another. Finally, you know that database consistency can be ensured only before and after the execution of transactions. A database always moves through an unavoidable temporary state of inconsistency during a transaction’s execution if such transaction updates multiple tables/rows. (If the transaction contains only one update, then there is no temporary inconsistency.) That temporary inconsistency exists because a computer executes the operations serially, one after another. During this serial process, the isolation property of transactions prevents them from accessing the data not yet released by other transactions. The job of the scheduler is even more important today, with the use of multicore processors that have the capability of executing several instructions at the same time. What would happen if two transactions executed concurrently and they were accessing the same data?

In previous examples, the operations within a transaction were executed in an arbitrary order. As long as two transactions, T1 and T2, access unrelated data, there is no conflict and the order of execution is irrelevant to the final outcome. But if the transactions operate on related

(or the same) data, conflict is possible among the transaction components and the selection of one execution order over another might have some undesirable consequences. So how is the correct order determined, and who determines that order? Fortunately, the DBMS handles that tricky assignment by using a built-in scheduler.

The scheduler is a special DBMS process that establishes the order in which the operations within concurrent transactions are executed. The scheduler interleaves the execution of database operations to ensure serializability and isolation of transactions. To determine the appropriate order, the scheduler bases its actions on concurrency control algorithms, such as locking or time stamping methods, which are explained in the next sections. However, it is important to understand that not all transactions are serializable. The DBMS determines what transactions are serializable and proceeds to interleave the execution of the transaction's operations. Generally, transactions that are not serializable are executed on a first-come, first-served basis by the DBMS. The scheduler's main job is to create a serializable schedule of a transaction's operations. A serializable schedule is a schedule of a transaction's operations in which the interleaved execution of the transactions (T1, T2, T3, etc.) yields the same results as if the transactions were executed in serial order (one after another).

The scheduler also makes sure that the computer's central processing unit (CPU) and storage systems are used efficiently. If there were no way to schedule the execution of transactions, all transactions would be executed on a first-come, first-served basis. The problem with that approach is that processing time is wasted when the CPU waits for a READ or WRITE operation to finish, thereby losing several CPU cycles. In short, first-come, first-served scheduling tends to yield unacceptable response times within the multiuser DBMS environment. Therefore, some other scheduling method is needed to improve the efficiency of the overall system.

Additionally, the scheduler facilitates data isolation to ensure that two transactions do not update the same data element at the same time. Database operations might require READ and/or WRITE actions that produce conflicts. For example, Table 10.11 shows the possible conflict scenarios when two transactions, T1 and T2, are executed concurrently over the same data. Note that in Table 10.11, two operations are in conflict when they access the same data and at least one of them is a WRITE operation.

Table 10.11 Read/Write conflict scenarios: conflicting database operations matrix

	TRANSACTIONS		RESULT
	T1	T2	
Operations	Read	Read	No conflict
	Read	Write	Conflict
	Write	Read	Conflict
	Write	Write	Conflict

Several methods have been proposed to schedule the execution of conflicting operations in concurrent transactions.

Those methods have been classified as locking, time stamping, and optimistic. Locking methods, discussed next, are used most frequently.

10.3 CONCURRENCY CONTROL WITH LOCKING METHODS

A lock guarantees exclusive use of a data item to a current transaction. In other words, transaction T2 does not have access to a data item that is currently being used by transaction T1. A transaction acquires a lock prior to data access; the lock is released (unlocked) when the transaction is completed so that another transaction can lock the data item for its exclusive use. This series of locking actions assumes that there is a likelihood of concurrent transactions attempting to manipulate the same data item at the same time. The use of locks based on the assumption that conflict between transactions is likely to occur is often referred to as pessimistic locking. data consistency cannot be guaranteed during a transaction; the database might be in a temporary inconsistent state when several updates are executed. Therefore, locks are required to prevent another transaction from reading inconsistent data. Most multiuser DBMSs automatically initiate and enforce locking procedures. All lock information is managed by a lock manager, which is responsible for assigning and policing the locks used by the transactions.

Lock Granularity

Lock granularity indicates the level of lock use. Locking can take place at the following levels: database, table, page, row, or even field (attribute).

Database Level

In a database-level lock, the entire database is locked, thus preventing the use of any tables in the database by transaction T2 while transaction T1 is being executed. This level of locking is good for batch processes, but it is unsuitable for multiuser DBMSs. You can imagine how

slow the data access would be if thousands of transactions had to wait for the previous transaction to be completed before the next one could reserve the entire database.

Table Level

In a table-level lock, the entire table is locked, preventing access to any row by transaction T2 while transaction T1 is using the table. If a transaction requires access to several tables, each table may be locked. However, two transactions can access the same database as long as they access different tables. Table-level locks, while less restrictive than database-level locks, because traffic jams when many transactions are waiting to access the same table. Such a condition is especially irksome if the lock forces a delay when different transactions require access to different parts of the same table, that is, when the transactions would not interfere with each other. Consequently, table-level locks are not suitable for multiuser DBMSs.

Page Level

In a page-level lock, the DBMS will lock an entire diskpage. A diskpage, or page, is the equivalent of a diskblock, which can be described as a directly addressable section of a disk. A page has a fixed size, such as 4K, 8K, or 16K.

For example, if you want to write only 73 bytes to a 4K page, the entire 4K page must be read from disk, updated in memory, and written back to disk. A table can span several pages, and a page can contain several rows of one or more tables. Page-level locks are currently the most frequently used multiuser DBMS locking method.

Row Level

A row-level lock is much less restrictive than the locks discussed earlier. The DBMS allows concurrent transactions to access different rows of the same table even when the rows are located on the same page. Although the row-level locking approach improves the availability of data, its management requires high overhead because a lock exists for each row in a table of the database involved in a conflicting transaction. Modern DBMSs automatically escalate a lock from a row-level to a page-level lock when the application session requests multiple locks on the same page.

Field Level

The field-level lock allows concurrent transactions to access the same row as long as they require the use of different fields (attributes) within that row. Although field-level locking clearly yields the most flexible multiuser data access, it is rarely implemented in a DBMS

because it requires an extremely high level of computer overhead and because the row-level lock is much more useful in practice.

Lock Types

Regardless of the level of locking, the DBMS may use different lock types: binary or shared/exclusive.

Binary Locks

A binary lock has only two states: locked (1) or unlocked (0). If an object—that is, a database, table, page, or row—is locked by a transaction, no other transaction can use that object. If an object is unlocked, any transaction can lock the object for its use.

However, binary locks are now considered too restrictive to yield optimal concurrency conditions. For example, the DBMS will not allow two transactions to read the same database object even though neither transaction updates the database, and therefore, no concurrency problems can occur. Concurrency conflicts occur only when two transactions execute concurrently and one of them updates the database.

Shared/Exclusive Locks

The labels “shared” and “exclusive” indicate the nature of the lock. An exclusive lock exists when access is reserved specifically for the transaction that locked the object. The exclusive lock must be used when the potential for conflict exists. A shared lock exists when concurrent transactions are granted read access on the basis of a common lock. A shared lock produces no conflict as long as all the concurrent transactions are read-only. A shared lock is issued when a transaction wants to read data from the database and no exclusive lock is held on that data item. An exclusive lock is issued when a transaction wants to update (write) a data item and no locks are currently held on that data item by any other transaction. Using the shared/exclusive locking concept, a lock can have three states: unlocked, shared (read), and exclusive (write). Two transactions conflict only when at least one of them is a Write transaction. Because the two Read transactions can be safely executed at once, shared locks allow several Read transactions to read the same data item concurrently. For example, if transaction T1 has a shared lock on data item X and transaction T2 wants to read data item X, T2 may also obtain a shared lock on data item X.

If transaction T2 updates data item X, an exclusive lock is required by T2 over data item X. The exclusive lock is granted if and only if no other locks are held on the data item.

Therefore, if a shared or exclusive lock is already held on data item X by transaction T1, an exclusive lock cannot be granted to transaction T2 and T2 must wait to begin until T1 commits. This condition is known as the mutual exclusive rule: only one transaction at a time can own an exclusive lock on the same object.

Although the use of shared locks renders data access more efficient, a shared/exclusive lock schema increases the lock manager's overhead, for several reasons:

- The type of lock held must be known before a lock can be granted.
- Three lock operations exist: `READ_LOCK` (to check the type of lock), `WRITE_LOCK` (to issue the lock), and `UNLOCK` (to release the lock).
- The schema has been enhanced to allow a lock upgrade (from shared to exclusive) and a lock downgrade (from exclusive to shared). Although locks prevent serious data inconsistencies, they can lead to two major problems:
- The resulting transaction schedule might not be serializable.
- The schedule might create deadlocks. A deadlock occurs when two transactions wait indefinitely for each other to unlock data. A database deadlock, which is equivalent to trafficgridlock in a big city, is caused when two or more transactions wait for each other to unlock data.

Fortunately, both problems can be managed: serializability is guaranteed through a locking protocol known as two-phase locking and deadlocks can be managed by using deadlock detection and prevention techniques. Those techniques are examined in the next two sections.

Two-Phase Locking to Ensure Serializability

Two-phase locking defines how transactions acquire and relinquish locks. Two-phase locking guarantees serializability, but it does not prevent deadlocks. The two phases are:

1. A growing phase, in which a transaction acquires all required locks without unlocking any data. Once all locks have been acquired, the transaction is in its locked point.
2. A shrinking phase, in which a transaction releases all locks and cannot obtain any new lock.

The two-phase locking protocol is governed by the following rules:

- Two transactions cannot have conflicting locks.
- No unlock operation can precede a lock operation in the same transaction.
- No data are affected until all locks are obtained—that is, until the transaction is in its locked point.

Two-phase locking increases the transaction processing cost and might cause additional undesirable effects. One undesirable effect is the possibility of creating deadlocks.

Deadlocks

A deadlock occurs when two transactions wait indefinitely for each other to unlock data. For example, a deadlock occurs when two transactions, T1 and T2, exist in the following mode:

T1 = access data items X and Y

T2 = access data items Y and X

If T1 has not unlocked data item Y, T2 cannot begin; if T2 has not unlocked data item X, T1 cannot continue. Consequently, T1 and T2 each wait for the other to unlock the required data item. Such a deadlock is also known as a deadly embrace. Table 10.13 demonstrates how a deadlock condition is created.

TIME	TRANSACTION	REPLY	LOCK STATUS	
0			Data X	Data Y
1	T1:LOCK(X)	OK	Unlocked	Unlocked
2	T2: LOCK(Y)	OK	Locked	Unlocked
3	T1:LOCK(Y)	WAIT	Locked	Locked
4	T2:LOCK(X)	WAIT	Locked	Locked
5	T1:LOCK(Y)	WAIT	Locked	Locked
6	T2:LOCK(X)	WAIT	Locked	Locked
7	T1:LOCK(Y)	WAIT	Locked	Locked
8	T2:LOCK(X)	WAIT	Locked	Locked
9	T1:LOCK(Y)	WAIT	Locked	Locked
...
...
...
...




Table 10.13. How a deadlock condition is created

The preceding example used only two concurrent transactions to demonstrate a deadlock condition. In a real-world DBMS, many more transactions can be executed simultaneously, thereby increasing the probability of generating deadlocks. Note that deadlocks are possible only when one of the transactions wants to obtain an exclusive lock on a data item; no deadlock condition can exist among shared locks.

The three basic techniques to control deadlocks are:

- **Deadlock prevention.** A transaction requesting a new lock is aborted when there is the possibility that a deadlock can occur. If the transaction is aborted, all changes made by this transaction are rolled back and all locks obtained by the transaction are released. The transaction is then rescheduled for execution. Deadlock prevention works because it avoids the conditions that lead to deadlocking.

- Deadlock detection. The DBMS periodically tests the database for deadlocks. If a deadlock is found, one of the transactions (the “victim”) is aborted (rolled back and restarted) and the other transaction continues.
- Deadlock avoidance. The transaction must obtain all of the locks it needs before it can be executed. This technique avoids the rolling back of conflicting transactions by requiring that locks be obtained in succession.

10.4 CONCURRENCY CONTROL WITH TIME STAMPING METHODS

The time stamping approach to scheduling concurrent transactions assigns a global, unique time stamp to each transaction. The time stamp value produces an explicit order in which transactions are submitted to the DBMS. Time stamps must have two properties: uniqueness and monotonicity. Uniqueness ensures that no equal time stamp values can exist, and monotonicity ensures that time stamp values always increase. All database operations (Read and Write) within the same transaction must have the same time stamp. The DBMS executes conflicting operations in time stamp order, thereby ensuring serializability of the transactions. If two transactions conflict, one is stopped, rolled back, rescheduled, and assigned a new time stamp value.

The disadvantage of the time stamping approach is that each value stored in the database requires two additional time stamp fields: one for the last time the field was read and one for the last update. Time stamping thus increases memory needs and the database’s processing overhead. Time stamping demands a lot of system resources because many transactions might have to be stopped, rescheduled, and restamped.

Wait/Die and Wound/Wait Schemes

You have learned that time stamping methods are used to manage concurrent transaction execution. In this section, you will learn about two schemes used to decide which transaction is rolled back and which continues executing: the wait/die scheme and the wound/wait scheme. An example illustrates the difference. Assume that you have two conflicting transactions: T1 and T2, each with a unique time stamp. Suppose T1 has a time stamp of 11548789 and T2 has a time stamp of 19562545. You can deduce from the time stamps that T1 is the older transaction (the lower time stamp value) and T2 is the newer transaction. Given that scenario, the four possible outcomes are shown in Table 10.14.

TRANSACTION REQUESTING LOCK	TRANSACTION OWNING LOCK	WAIT/DIE SCHEME	WOUND/WAIT SCHEME
T1 (11548789)	T2 (19562545)	<ul style="list-style-type: none"> • T1 waits until T2 is completed and T2 releases its locks. 	<ul style="list-style-type: none"> • T1 preempts (rolls back) T2. • T2 is rescheduled using the same time stamp.
T2 (19562545)	T1 (11548789)	<ul style="list-style-type: none"> • T2 dies (rolls back). • T2 is rescheduled using the same time stamp. 	<ul style="list-style-type: none"> • T2 waits until T1 is completed and T1 releases its locks.

Table 10.14 Wait/Die and Wound/Wait concurrency control schemes

Using the wait/die scheme:

- If the transaction requesting the lock is the older of the two transactions, it will wait until the other transaction is completed and the locks are released.
- If the transaction requesting the lock is the younger of the two transactions, it will die (roll back) and is rescheduled using the same time stamp.
- In short, in the wait/die scheme, the older transaction waits for the younger to complete and release its locks.

In the wound/wait scheme:

- If the transaction requesting the lock is the older of the two transactions, it will preempt (wound) the younger transaction (by rolling it back). T1 pre-empts T2 when T1 rolls back T2. The younger, pre-empted transaction is rescheduled using the same time stamp.
- If the transaction requesting the lock is the younger of the two transactions, it will wait until the other transaction is completed and the locks are released.
- In short, in the wound/wait scheme, the older transaction rolls back the younger transaction and reschedules it.

10.5 Concurrency Control with Optimistic Methods

The optimistic approach is based on the assumption that the majority of the database operations do not conflict. The optimistic approach requires neither locking nor time stamping techniques. Instead, a transaction is executed without restrictions until it is committed. Using an optimistic approach, each transaction moves through two or three phases, referred to as read, validation, and write.

- During the read phase, the transaction reads the database, executes the needed computations, and makes the updates to a private copy of the database values. All

update operations of the transaction are recorded in a temporary update file, which is not accessed by the remaining transactions.

- During the validation phase, the transaction is validated to ensure that the changes made will not affect the integrity and consistency of the database. If the validation test is positive, the transaction goes to the write phase. If the validation test is negative, the transaction is restarted and the changes are discarded.
- During the write phase, the changes are permanently applied to the database.
- The optimistic approach is acceptable for most read or query database systems that require few update transactions.

10.6 ANSI LEVELS OF TRANSACTION ISOLATION

In ANSI SQL, there are four standard isolation levels: Serializable, Repeatable Reads, Read Committed, and Read Uncommitted.

The default for many databases is Read Committed, which only guarantees that you won't see data from a transaction while that transaction is in progress. It does this by briefly acquiring locks during reads, while maintaining write locks until the transaction is committed.

If you need to repeat the same read multiple times during a transaction, and want to be reasonably certain that it always returns the same value, you need to hold a read lock for the entire duration. This is automatically done for you when using the Repeatable Reads isolation level.

We say "reasonably certain" for Repeatable Reads because of the possibility of "phantom reads". A phantom read can occur when you perform a query using a where clause such as "WHERE Status = 1". Those rows will be locked, but nothing prevents a new row matching the criteria from being added. The term "phantom" applies to the rows that appear the second time the query is executed.

To be absolutely certain that two reads in the same transaction return the same data, you can use the Serializable isolation level. This uses "range-locks", which prevent new rows from being added if they match a WHERE clause in an open transaction.

Generally speaking, the higher your isolation level the worse your performance is due to lock contention. So to improve read performance, some databases also support Read Uncommitted. This isolation level ignores locks (and is in fact called NOLOCK in SQL Server). As a result, it can perform dirty reads.

10.7 DATABASE RECOVERY MANAGEMENT

Database recovery restores a database from a given state (usually inconsistent) to a previously consistent state. Recovery techniques are based on the atomic transaction property: all portions of the transaction must be treated as a single, logical unit of work in which all operations are applied and completed to produce a consistent database.

If, for some reason, any transaction operation cannot be completed, the transaction must be aborted and any changes to the database must be rolled back (undone). In short, transaction recovery reverses all of the changes that the transaction made to the database before the transaction was aborted.

Although this unit has emphasized the recovery of transactions, recovery techniques also apply to the database and to the system after some type of critical error has occurred. Critical events can cause a database to become non-operational and compromise the integrity of the data. Examples of critical events are:

- Hardware/software failures. A failure of this type could be a hard disk media failure, a bad capacitor on a motherboard, or a failing memory bank. Other causes of errors under this category include application program or operating system errors that cause data to be overwritten, deleted, or lost. Some database administrators argue that this is one of the most common sources of database problems.
- Human-caused incidents. This type of event can be categorized as unintentional or intentional.
 - An unintentional failure is caused by carelessness by end users. Such errors include deleting the wrong rows from a table, pressing the wrong key on the keyboard, or shutting down the main database server by accident.
 - Intentional events are of a more severe nature and normally indicate that the company data are at serious risk. Under this category are security threats caused by hackers trying to gain unauthorized access to data resources and virus attacks caused by disgruntled employees trying to compromise the database operation and damage the company.
- Natural disasters. This category includes fires, earthquakes, floods, and power failures. Whatever the cause, a critical error can render the database in an inconsistent state. The following section introduces the various techniques used to recover the database from an inconsistent state to a consistent state.

Transaction Recovery

Database transaction recovery uses data in the transaction log to recover a database from an inconsistent state to a consistent state.

Before continuing, let's examine four important concepts that affect the recovery process:

- The write-ahead-log protocol ensures that transaction logs are always written before any database data are actually updated. This protocol ensures that, in case of a failure, the database can later be recovered to a consistent state, using the data in the transaction log.
- Redundant transaction logs (several copies of the transaction log) ensure that a physical disk failure will not impair the DBMS's ability to recover data.
- Database buffers are temporary storage areas in primary memory used to speed up disk operations. To improve processing time, the DBMS software reads the data from the physical disk and stores a copy of it on a "buffer" in primary memory. When a transaction updates data, it actually updates the copy of the data in the buffer because that process is much faster than accessing the physical disk every time. Later on, all buffers that contain updated data are written to a physical disk during a single operation, thereby saving significant processing time.
- Database checkpoints are operations in which the DBMS writes all of its updated buffers to disk. While this is happening, the DBMS does not execute any other requests. A checkpoint operation is also registered in the transaction log. As a result of this operation, the physical database and the transaction log will be in sync. This synchronization is required because update operations update the copy of the data in the buffers and not in the physical database. Checkpoints are automatically scheduled by the DBMS several times per hour.
- The database recovery process involves bringing the database to a consistent state after a failure. Transaction recovery procedures generally make use of deferred-write and write-through techniques.

When the recovery procedure uses a deferred-write technique (also called a deferred update), the transaction operations do not immediately update the physical database. Instead, only the transaction log is updated. The database is physically updated only after the transaction reaches its commit point, using information from the transaction log.

If the transaction aborts before it reaches its commit point, no changes (no ROLLBACK or undo) need to be made to the database because the database was never updated. The recovery process for all started and committed transactions (before the failure) follows these steps:

1. Identify the last checkpoint in the transaction log. This is the last time transaction data was physically saved to disk.
2. For a transaction that started and was committed before the last checkpoint, nothing needs to be done because the data are already saved.
3. For a transaction that performed a commit operation after the last checkpoint, the DBMS uses the transaction log records to redo the transaction and to update the database, using the “after” values in the transaction log. The changes are made in ascending order, from oldest to newest.
4. For any transaction that had a ROLLBACK operation after the last checkpoint or that was left active (with neither a COMMIT nor a ROLLBACK) before the failure occurred, nothing needs to be done because the database was never updated.

When the recovery procedure uses a write-through technique (also called an immediate update), the database is immediately updated by transaction operations during the transaction’s execution, even before the transaction reaches its commit point. If the transaction aborts before it reaches its commit point, a ROLLBACK or undo operation needs to be done to restore the database to a consistent state. In that case, the ROLLBACK operation will use the transaction log “before” values. The recovery process follows these steps:

1. Identify the last checkpoint in the transaction log. This is the last time transaction data were physically saved to disk.
2. For a transaction that started and was committed before the last checkpoint, nothing needs to be done because the data are already saved.
3. For a transaction that was committed after the last checkpoint, the DBMS redoes the transaction, using the “after” values of the transaction log. Changes are applied in ascending order, from oldest to newest.
4. For any transaction that had a ROLLBACK operation after the last checkpoint or that was left active (with neither a COMMIT nor a ROLLBACK) before the failure occurred, the DBMS uses the transaction log records to ROLLBACK or undo the operations, using the “before” values in the transaction log. Changes are applied in reverse order, from newest to oldest.

10.8 CHECK YOUR PROGRESS

1. What is a database transaction?
2. What is a consistent database state?
3. What is DBMS ACID test?
4. What is concurrency control?
5. What is the DBMS scheduler?
6. What is pessimistic locking?
7. What is Lock granularity?
8. What are the three ways of handling deadlock in a data base
9. How many standard isolation levels are there In ANSI SQL?
10. Give two Examples of critical events.

Answers to check your progress:

1. A transaction is a logical unit of work that must be entirely completed or entirely aborted; no intermediate states are acceptable.
2. A consistent database state is one in which all data integrity constraints are satisfied.
3. Atomicity, consistency, isolation, and durability.
4. The coordination of the simultaneous execution of transactions in a multiuser database system is known as concurrency control.
5. The scheduler is a special DBMS process that establishes the order in which the operations within concurrent transactions are executed. The scheduler interleaves the execution of database operations to ensure serializability and isolation of transactions.
6. The use of locks based on the assumption that conflict between transactions is likely to occur is often referred to as pessimistic locking.
7. Lock granularity indicates the level of lock use. Locking can take place at the following levels: database, table, page, row, or even field (attribute).
8. Prevention, detection and avoidance
9. In ANSI SQL, there are four standard isolation levels: Serializable, Repeatable Reads, Read Committed, and Read Uncommitted.
10. Hardware/software failures, Human caused incidents

10.9 SUMMARY

- Transactions have four main properties: atomicity (all parts of the transaction are executed; otherwise, the transaction is aborted), consistency (the database's consistent state is maintained), isolation (data used by one transaction cannot be accessed by another transaction until the first transaction is completed), and durability (the changes made by a transaction cannot be rolled back once the transaction is committed). In addition, transaction schedules have the property of serializability (the result of the concurrent execution of transactions is the same as that of the transactions being executed in serial order).
- SQL provides support for transactions through the use of two statements: COMMIT (saves changes to disk) and ROLLBACK (restores the previous database state).
- SQL transactions are formed by several SQL statements or database requests. Each database request originates several I/O database operations.
- The transaction log keeps track of all transactions that modify the database. The information stored in the transaction log is used for recovery (ROLLBACK) purposes.
- Concurrency control coordinates the simultaneous execution of transactions. The concurrent execution of transactions can result in three main problems: lost updates, uncommitted data, and inconsistent retrievals.
- The scheduler is responsible for establishing the order in which the concurrent transaction operations are executed. The transaction execution order is critical and ensures database integrity in multiuser database systems. Locking, time stamping, and optimistic methods are used by the scheduler to ensure the serializability of transactions.
- A lock guarantees unique access to a data item by a transaction. The lock prevents one transaction from using the data item while another transaction is using it. There are several levels of locks: database, table, page, row, and field.
- Two types of locks can be used in database systems: binary locks and shared/exclusive locks. A binary lock can have only two states: locked (1) or unlocked (0). A shared lock is used when a transaction wants to read data from a database and no other transaction is updating the same data. Several shared or "read" locks can exist for a particular item. An exclusive lock is issued when a transaction wants to update (write to) the database and no other locks (shared or exclusive) are held on the data.
- Serializability of schedules is guaranteed through the use of two-phase locking. The two-phase locking schema has a growing phase, in which the transaction acquires all of the locks

that it needs without unlocking any data, and a shrinking phase, in which the transaction releases all of the locks without acquiring new locks.

- When two or more transactions wait indefinitely for each other to release a lock, they are in a deadlock, also called a deadly embrace. There are three deadlock control techniques: prevention, detection, and avoidance.
- Concurrency control with time stamping methods assigns a unique time stamp to each transaction and schedules the execution of conflicting transactions in time stamp order. Two schemes are used to decide which transaction is rolled back and which continues executing: the wait/die scheme and the wound/wait scheme.
- Concurrency control with optimistic methods assumes that the majority of database transactions do not conflict and that transactions are executed concurrently, using private, temporary copies of the data. At commit time, the private copies are updated to the database.
- Database recovery restores the database from a given state to a previous consistent state. Database recovery is triggered when a critical event occurs, such as a hardware error or application error.

10.10 KEYWORDS

- **Atomicity**- all parts of the transaction are executed; otherwise, the transaction is aborted
- **Binary lock** – A binary lock can have only two states: locked (1) or unlocked (0).
- **Checkpoints** - are operations in which the DBMS writes all of its updated buffers to disk.
- **Concurrency control** - The coordination of the simultaneous execution of transactions in a multiuser database system is known as concurrency control.
- **Deadlock** - When two or more transactions wait indefinitely for each other to release a lock, they are in a deadlock
- **Scheduler** - The scheduler is responsible for establishing the order in which the concurrent transaction operations are executed.
- **Serializability**- the result of the concurrent execution of transactions is the same as that of the transactions being executed in serial order.

10.11 QUESTIONS FOR SELF-STUDY

1. List and discuss the four transaction properties.
2. What is a scheduler, what does it do, and why is its activity important to concurrency control?

3. What is a deadlock, and how can it be avoided? Discuss several strategies for dealing with deadlocks.
4. Why might it take a long time to complete transactions when an optimistic approach to concurrency control is used?
5. What are the three types of database-critical events that can trigger the database recovery process? Give some examples for each one.

10.12 REFERENCES

1. Coronel, C., & Morris, S. (2016). Database systems: design, implementation, & management. Cengage Learning.
2. Gupta, S. B., & Mittal, A. (2009). Introduction to database management system. Laxmi Publications, Ltd..
3. Ramakrishnan, R., & Gehrke, J. (2000). Database management systems. McGraw-Hill.

UNIT -11: DISTRIBUTED DATABASE MANAGEMENT SYSTEMS

STRUCTURE

- 11.0 Objectives
- 11.1 The Evolution of DDBMS
- 11.2 DDBMS Advantages and Disadvantages
- 11.3 Distributed Processing and Distributed Databases
- 11.4 Characteristics of DDBMS
- 11.5 DDBMS Components
- 11.6 Levels of Data and Process Distribution
- 11.7 Distributed Database Transparency Features
- 11.8 Check your progress
- 11.9 Summary
- 11.10 Keywords
- 11.11 Questions for self-study
- 11.12 References

11.0 OBJECTIVES

After studying this unit, you will be able to:

- ✓ What a distributed database management system (DDBMS) is and what its components are
- ✓ How database implementation is affected by different levels of data and process distribution
- ✓ How transactions are managed in a distributed database environment
- ✓ How database design is affected by the distributed database environment

11.1 THE EVOLUTION OF DDBMS

A distributed database management system (DDBMS) governs the storage and processing of logically related data over interconnected computer systems in which both data and processing are distributed among several sites. To understand how and why the DDBMS is different from the DBMS, it is useful to briefly examine the changes in the business environment that set the stage for the development of the DDBMS.

During the 1970s, corporations implemented centralized database management systems to meet their structured information needs. Structured information is usually presented as regularly issued formal reports in a standard format. Such information, generated by procedural programming languages, is created by specialists in response to precisely channelled requests. Thus, structured information needs are well served by centralized systems. The use of a centralized database required that corporate data be stored in a single central site, usually a mainframe computer. Data access was provided through dumb terminals. The centralized approach worked well to fill the structured information needs of corporations, but it fell short when quickly moving events required faster response times and equally quick access to information.

The slow progression from information request to approval to specialist to user simply did not serve decision makers well in a dynamic environment. What was needed was quick, unstructured access to databases, using ad hoc queries to generate on-the-spot information. Database management systems based on the relational model could provide the environment in which unstructured information needs would be met by employing ad hoc queries. End users would be given the ability to access data when needed. Unfortunately, the early relational model implementations did not yet deliver acceptable throughput when compared to the well-established hierarchical or network database models.

The last two decades gave birth to a series of crucial social and technological changes that affected database development and design. Among those changes were:

- Business operations became decentralized.
- Competition increased at the global level.
- Customer demands and market needs favoured a decentralized management style.
- Rapid technological change created low-cost computers with mainframe-like power, impressive multifunction handheld portable wireless devices with cellular phone and data services, and increasingly complex and fast networks to connect them. As a consequence, corporations have increasingly adopted advanced network technologies as the platform for their computerized solutions.
- The large number of applications based on DBMSs and the need to protect investments in centralized DBMS software made the notion of data sharing attractive. Data realms are converging in the digital world more and more. As a result, single applications manage multiple different types of data (voice, video, music, images, etc.), and such data are accessed from multiple geographically dispersed locations.

Those factors created a dynamic business environment in which companies had to respond quickly to competitive and technological pressures. As large business units restructured to form leaner, quickly reacting, dispersed operations, two database requirements became obvious:

- Rapid ad hoc data access became crucial in the quick-response decision-making environment.
- The decentralization of management structures based on the decentralization of business units made decentralized multiple-access and multiple-location databases a necessity.
- During recent years, the factors just described became even more firmly entrenched. However, the way those factors were addressed was strongly influenced by:
 - The growing acceptance of the Internet as the platform for data access and distribution. The World Wide Web (the Web) is, in effect, the repository for distributed data.
 - The wireless revolution. The widespread use of wireless digital devices, such as smart phones like the iPhone and BlackBerry and personal digital assistants (PDAs), has created high demand for data access. Such devices access data from geographically dispersed locations and require varied data exchanges in multiple formats (data, voice, video, music, pictures, etc.) Although distributed data access does not necessarily imply distributed databases, performance and failure tolerance requirements often make use of data replication techniques similar to the ones found in distributed databases.
 - The accelerated growth of companies providing “application as a service” type of services. This new type of service provides remote application services to companies wanting to outsource their application development, maintenance, and operations. The company data is generally stored on central servers and is not necessarily distributed. Just as with wireless data access, this type of service may not require fully distributed data functionality; however, other factors such as performance and failure tolerance often require the use of data replication techniques similar to the ones found in distributed databases.
 - The increased focus on data analysis that led to data mining and data warehousing. Although a data warehouse is not usually a distributed database, it does rely on techniques such as data replication and distributed queries that facilitate data extraction and integration.

- The decentralized database is especially desirable because centralized database management is subject to problems such as:
- Performance degradation because of a growing number of remote locations over greater distances.
- High costs associated with maintaining and operating large central (mainframe) database systems.
- Reliability problems created by dependence on a central site (single point of failure syndrome) and the need for data replication.
- Scalability problems associated with the physical limits imposed by a single location (power, temperature conditioning, and power consumption.)
- Organizational rigidity imposed by the database might not support the flexibility and agility required by modern global organizations.

11.2 DDBMS ADVANTAGES AND DISADVANTAGES

Distributed database management systems deliver several advantages over traditional systems. At the same time, they are subject to some problems. Table 11.1 summarizes the advantages and disadvantages associated with a DDBMS.

Table 11.1 DDBMS Advantages and Disadvantages

ADVANTAGES	DISADVANTAGES
<ul style="list-style-type: none"> • <i>Data are located near the greatest demand site.</i> The data in a distributed database system are dispersed to match business requirements. • <i>Faster data access.</i> End users often work with only a locally stored subset of the company's data. • <i>Faster data processing.</i> A distributed database system spreads out the systems workload by processing data at several sites. • <i>Growth facilitation.</i> New sites can be added to the network without affecting the operations of other sites. • <i>Improved communications.</i> Because local sites are smaller and located closer to customers, local sites foster better communication among departments and between customers and company staff. • <i>Reduced operating costs.</i> It is more cost-effective to add workstations to a network than to update a mainframe system. Development work is done more cheaply and more quickly on low-cost PCs than on mainframes. • <i>User-friendly interface.</i> PCs and workstations are usually equipped with an easy-to-use graphical user interface (GUI). The GUI simplifies training and use for end users. • <i>Less danger of a single-point failure.</i> When one of the computers fails, the workload is picked up by other workstations. Data are also distributed at multiple sites. • <i>Processor independence.</i> The end user is able to access any available copy of the data, and an end user's request is processed by any processor at the data location. 	<ul style="list-style-type: none"> • <i>Complexity of management and control.</i> Applications must recognize data location, and they must be able to stitch together data from various sites. Database administrators must have the ability to coordinate database activities to prevent database degradation due to data anomalies. • <i>Technological difficulty.</i> Data integrity, transaction management, concurrency control, security, backup, recovery, query optimization, access path selection, and so on, must all be addressed and resolved. • <i>Security.</i> The probability of security lapses increases when data are located at multiple sites. The responsibility of data management will be shared by different people at several sites. • <i>Lack of standards.</i> There are no standard communication protocols at the database level. (Although TCP/IP is the de facto standard at the network level, there is no standard at the application level.) For example, different database vendors employ different—and often incompatible—techniques to manage the distribution of data and processing in a DDBMS environment. • <i>Increased storage and infrastructure requirements.</i> Multiple copies of data are required at different sites, thus requiring additional disk storage space. • <i>Increased training cost.</i> Training costs are generally higher in a distributed model than they would be in a centralized model, sometimes even to the extent of offsetting operational and hardware savings. • <i>Costs.</i> Distributed databases require duplicated infrastructure to operate (physical location, environment, personnel, software, licensing, etc.)

11.3 DISTRIBUTED PROCESSING AND DISTRIBUTED DATABASES

In distributed processing, a database's logical processing is shared among two or more physically independent sites that are connected through a network. For example, the data input/output (I/O), data selection, and data validation might be performed on one computer, and a report based on that data might be created on another computer.

A basic distributed processing environment is illustrated in Figure 11.1, which shows that a distributed processing system shares the database processing chores among three sites connected through a communications network.

Although the database resides at only one site, each site can access the data and update the database. The database is located on Computer A, a network computer known as the database server.

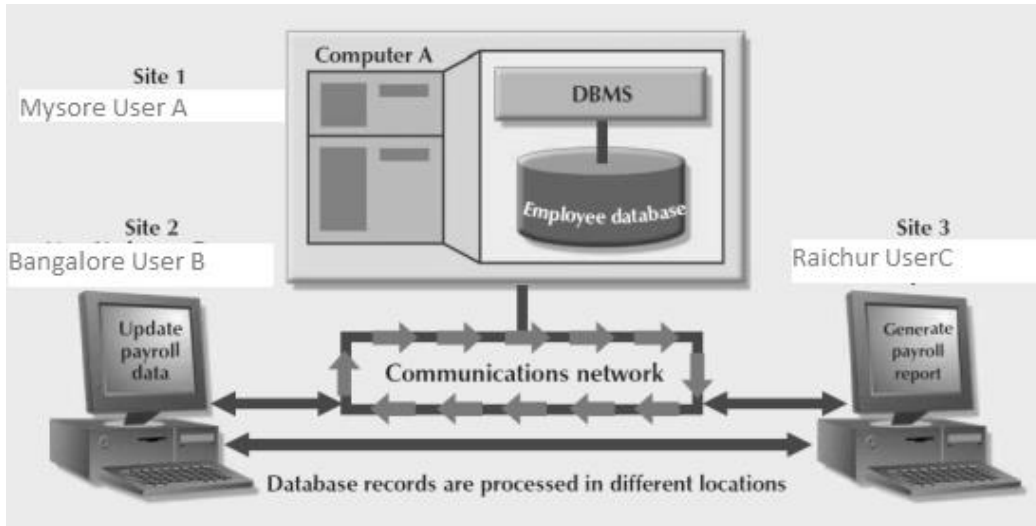


Fig 11.1. Distributed processing environment

A distributed database, on the other hand, stores a logically related database over two or more physically independent sites. The sites are connected via a computer network. In contrast, the distributed processing system uses only a single-site database but shares the processing chores among several sites. In a distributed database system, a database is composed of several parts known as database fragments. The database fragments are located at different sites and can be replicated among various sites. Each database fragment is, in turn, managed by its local database process. An example of a distributed database environment is shown in Figure 11.2.

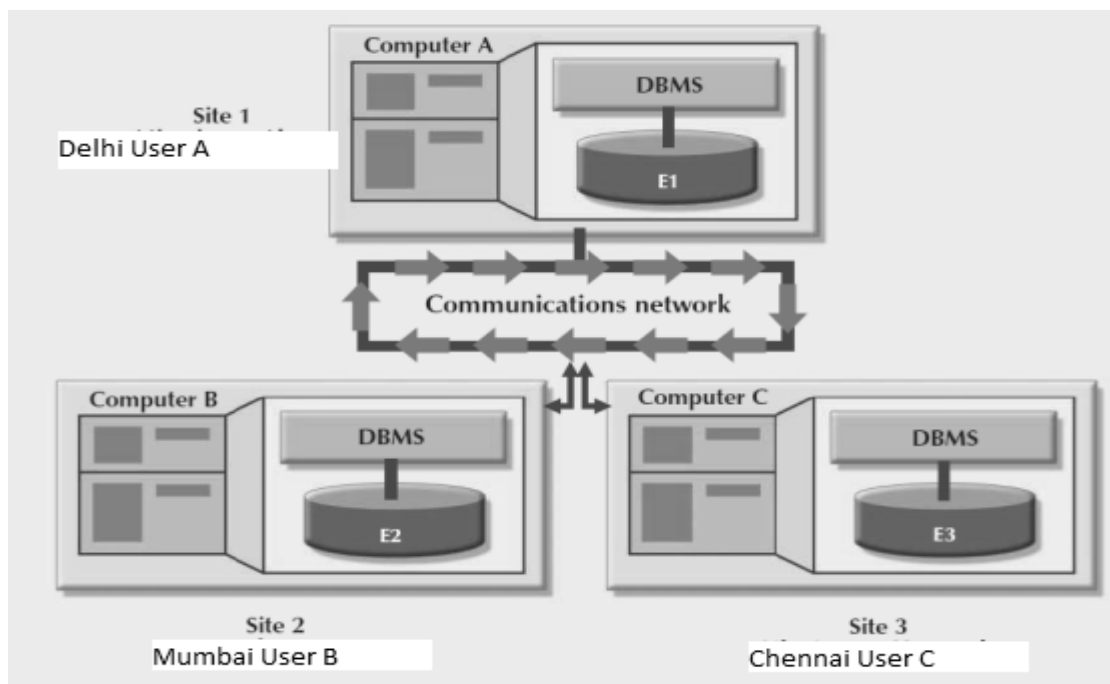


Fig. 11.2 Distributed database environment

The database in Figure 11.2 is divided into three database fragments (E1, E2, and E3) located at different sites. The computers are connected through a network system. In a fully distributed database, the users A, B, and C do not need to know the name or location of each database fragment in order to access the database. Also, the users might be located at sites other than Delhi, Mumbai or Chennai and still be able to access the database as a single logical unit.

As you examine Figures 11.1 and 11.2, you should keep the following points in mind:

- Distributed processing does not require a distributed database, but a distributed database requires distributed processing (each database fragment is managed by its own local database process).
- Distributed processing may be based on a single database located on a single computer. For the management of distributed data to occur, copies or parts of the database processing functions must be distributed to all data storage sites.
- Both distributed processing and distributed databases require a network to connect all components.

11.4 CHARACTERISTICS OF DDBMS

A DDBMS governs the storage and processing of logically related data over interconnected computer systems in which both data and processing functions are distributed among several sites. A DBMS must have at least the following functions to be classified as distributed:

- Application interface to interact with the end user, application programs, and other DBMSs within the distributed database.
- Validation to analyse data requests for syntax correctness.
- Transformation to decompose complex requests into atomic data request components.
- Query optimization to find the best access strategy. (Which database fragments must be accessed by the query, and how must data updates, if any, be synchronized?)
- Mapping to determine the data location of local and remote fragments.
- I/O interface to read or write data from or to permanent local storage.
- Formatting to prepare the data for presentation to the end user or to an application program.
- Security to provide data privacy at both local and remote databases.

- Backup and recovery to ensure the availability and recoverability of the database in case of a failure.
- DB administration features for the database administrator.
- Concurrency control to manage simultaneous data access and to ensure data consistency across database fragments in the DDBMS.
- Transaction management to ensure that the data moves from one consistent state to another. This activity includes the synchronization of local and remote transactions as well as transactions across multiple distributed segments.

A fully distributed database management system must perform all of the functions of a centralized DBMS, as follows:

1. Receive an application's (or an end user's) request.
2. Validate, analyze, and decompose the request. The request might include mathematical and/or logical operations such as the following: Select all customers with a balance greater than Rs. 1,000. The request might require data from only a single table, or it might require access to several tables.
3. Map the request's logical-to-physical data components.
4. Decompose the request into several disk I/O operations.
5. Search for, locate, read, and validate the data.
6. Ensure database consistency, security, and integrity.
7. Validate the data for the conditions, if any, specified by the request.
8. Present the selected data in the required format.

11.5 DDBMS COMPONENTS

The DDBMS must include at least the following components:

- Computer workstations or remote devices (sites or nodes) that form the network system. The distributed database system must be independent of the computer system hardware.
- Network hardware and software components that reside in each workstation or device. The network components allow all sites to interact and exchange data. Because the components—computers, operating systems, network hardware, and so on—are likely to be supplied by different vendors, it is best to ensure that distributed database functions can be run on multiple platforms.

- Communications media that carry the data from one node to another. The DDBMS must be communications media-independent; that is, it must be able to support several types of communications media.
- The transaction processor (TP), which is the software component found in each computer or device that requests data. The transaction processor receives and processes the application's data requests (remote and local). The TP is also known as the application processor (AP) or the transaction manager (TM).
- The data processor (DP), which is the software component residing on each computer or device that stores and retrieves data located at the site. The DP is also known as the data manager (DM). A data processor may even be a centralized DBMS.

.DPs and TPs can be added to the system without affecting the operation of the other components. A TP and a DP can reside on the same computer, allowing the end user to access local as well as remote data transparently. In theory, a DP can be an independent centralized DBMS with proper interfaces to support remote access from other independent DBMSs in the network.

11.6 LEVELS OF DATA AND PROCESS DISTRIBUTION

Current database systems can be classified on the basis of how process distribution and data distribution are supported. For example, a DBMS may store data in a single site (centralized DB) or in multiple sites (distributed DB) and may support data processing at a single site or at multiple sites. Table 12.2 uses a simple matrix to classify database systems according to data and process distribution. These types of processes are discussed in the sections that follow.

Table 11.2 Database Systems: Levels of Data and process distribution

	SINGLE-SITE DATA	MULTIPLE-SITE DATA
Single-site process	Host DBMS	Not applicable (Requires multiple processes)
Multiple-site process	File server Client/server DBMS (LAN DBMS)	Fully distributed Client/server DDBMS

Single-Site Processing, Single-Site Data (SPSD)

In the single-site processing, single-site data (SPSD) scenario, all processing is done on a single host computer (single-processor server, multiprocessor server, mainframe system) and all data are stored on the host computer's local disk system. Processing cannot be done on the

end user's side of the system. Such a scenario is typical of most mainframe and midrange server computer DBMSs. The DBMS is located on the host computer, which is accessed by dumb terminals connected to it. (See Figure 11.6.) This scenario is also typical of the first generation of single-user microcomputer databases.

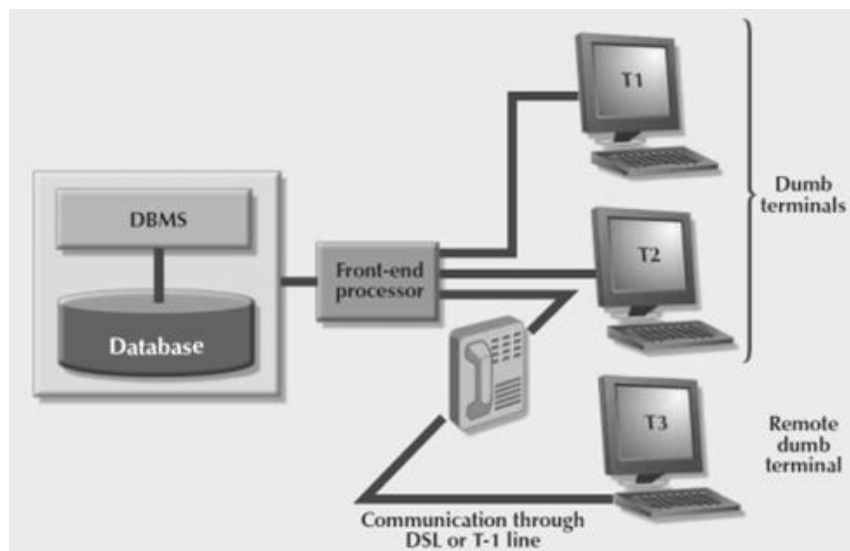


Fig. 11.6 Single-site processing, single-site data(centralized)

Using Figure 11.6 as an example, you can see that the functions of the TP and the DP are embedded within the DBMS located on a single computer. The DBMS usually runs under a time-sharing, multitasking operating system, which allows several processes to run concurrently on a host computer accessing a single DP. All data storage and data processing are handled by a single host computer.

Multiple-Site Processing, Single-Site Data (MPSD)

Under the multiple-site processing, single-site data (MPSD) scenario, multiple processes run on different computers sharing a single data repository. Typically, the MPSD scenario requires a network file server running conventional applications that are accessed through a network. Many multiuser accounting applications running under a personal computer network fit such a description. (See Figure 12.7.)

As you examine Figure 12.7, Note that:

- The TP on each workstation acts only as a redirector to route all network data requests to the file server.

- The end user sees the file server as just another hard disk. Because only the data storage input/output (I/O) is handled by the file server's computer, the MPSD offers limited capabilities for distributed processing.
- The end user must make a direct reference to the file server in order to access remote data. All record- and file-locking activities are done at the end-user location.
- All data selection, search, and update functions take place at the workstation, thus requiring that entire files travel through the network for processing at the workstation. Such a requirement increases network traffic, slows response time, and increases communication costs.

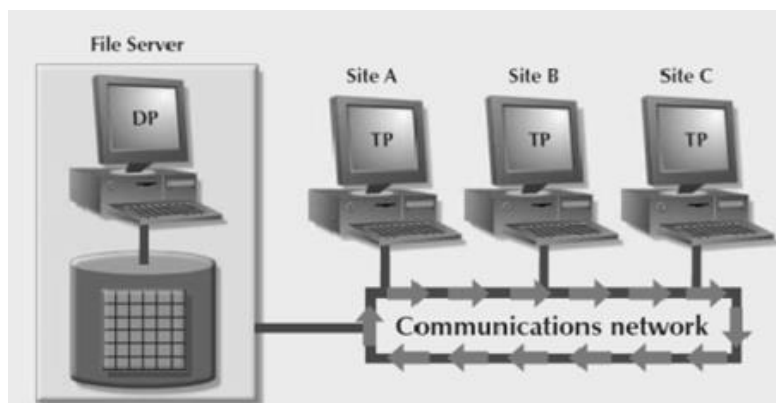


Fig 11.7. Multiple-site processing, single-site data

The inefficiency of the last condition can be illustrated easily. For example, suppose that the file server computer stores a CUSTOMER table containing 10,000 data rows, 50 of which have balances greater than Rs.1,000. Suppose that site A issues the following SQL query:

```
SELECT *
FROM CUSTOMER
WHERE CUS_BALANCE > 1000;
```

All 10,000 CUSTOMER rows must travel through the network to be evaluated at site A. A variation of the multiple-site processing, single-site data approach is known as client/server architecture. Client/server architecture is similar to that of the network file server except that all database processing is done at the server site, thus reducing network traffic. Although both the network file server and the client/server systems perform multiple-site processing, the latter's processing is distributed. Note that the network file server approach requires the database to be located at a single site. In contrast, the client/server architecture is capable of supporting data at multiple sites.

Multiple-Site Processing, Multiple-Site Data (MPMD)

The multiple-site processing, multiple-site data (MPMD) scenario describes a fully distributed DBMS with support for multiple data processors and transaction processors at multiple sites. Depending on the level of support for various types of centralized DBMSs, DDBMSs are classified as either homogeneous or heterogeneous.

Homogeneous DDBMSs integrate only one type of centralized DBMS over a network. Thus, the same DBMS will be running on different server platforms (single processor server, multiprocessor server, server farms, or server blades).

In contrast, heterogeneous DDBMSs integrate different types of centralized DBMSs over a network. Table 11.3 lists several systems that could be integrated within a single heterogeneous DDBMS over a network. A fully heterogeneous DDBMS will support different DBMSs that may even support different data models (relational, hierarchical, or network) running under different computer systems, such as mainframes and PCs.

Table 11.3. Heterogeneous distributed database scenario

PLATFORM	DBMS	OPERATING SYSTEM	NETWORK COMMUNICATIONS PROTOCOL
IBM 3090	DB2	MVS	APPC LU 6.2
DEC/VAX	VAX rdb	OpenVMS	DECnet
IBM AS/400	SQL/400	OS/400	3270
RISC Computer	Informix	UNIX	TCP/IP
Pentium CPU	Oracle	Windows Server 2008	TCP/IP

Some DDBMS implementations support several platforms, operating systems, and networks and allow remote data access to another DBMS. However, such DDBMSs are still subject to certain restrictions. For example:

- Remote access is provided on a read-only basis and does not support write privileges.
- Restrictions are placed on the number of remote tables that may be accessed in a single transaction.
- Restrictions are placed on the number of distinct databases that may be accessed.
- Restrictions are placed on the database model that may be accessed. Thus, access may be provided to relational databases but not to network or hierarchical databases.

Managing data at multiple sites leads to a number of issues that must be addressed and understood. The next section will examine several key features of distributed database management systems.

11.7 DISTRIBUTED DATABASE TRANSPARENCY FEATURES

A distributed database system requires functional characteristics that can be grouped and described as transparency features. DDBMS transparency features have the common property of allowing the end user to feel like the database's only user. In other words, the user believes that (s)he is working with a centralized DBMS; all complexities of a distributed database are hidden, or transparent, to the user.

The DDBMS transparency features are:

- Distribution transparency, which allows a distributed database to be treated as a single logical database. If a DDBMS exhibits distribution transparency, the user does not need to know
 - That the data are partitioned—meaning the table's rows and columns are split vertically or horizontally and stored among multiple sites.
 - That the data can be replicated at several sites.
 - The data location.
- Transaction transparency, which allows a transaction to update data at more than one network site. Transaction transparency ensures that the transaction will be either entirely completed or aborted, thus maintaining database integrity.
- Failure transparency, which ensures that the system will continue to operate in the event of a node failure. Functions that were lost because of the failure will be picked up by another network node.
- Performance transparency, which allows the system to perform as if it were a centralized DBMS. The system will not suffer any performance degradation due to its use on a network or due to the network's platform differences. Performance transparency also ensures that the system will find the most cost-effective path to access remote data.
- Heterogeneity transparency, which allows the integration of several different local DBMSs (relational, network, and hierarchical) under a common, or global, schema. The DDBMS is responsible for translating the data requests from the global schema to the local DBMS schema.

11.8 CHECK YOUR PROGRESS

1. Define distributed database management system
2. Define a homogeneous DDBMS and a heterogeneous DDBMS.
3. Write the main components of a DDBMS
4. What is a TP?
5. What is a DP?
6. Write three major categories of DDBMS.
7. Write any five DDBMS characteristics.
8. What is an undistributed transaction and a distributed transaction?

Answers to check your progress:

1. The term distributed database management system can describe various systems that differ from one another in many respects. The main thing that all such systems have in common is the fact that data and software are distributed over multiple sites connected by some form of communication network.
2. If all servers (or individual local DBMSs) use identical software and all users (clients) use identical software, the DDBMS is called homogeneous; otherwise, it is called heterogeneous.
3. The main components of a DDBMS are the transaction processor (TP) and the data processor (DP).
4. Which is the software component found in each computer or device that requests data. The transaction processor receives and processes the application's data requests (remote and local). The TP is also known as the application processor (AP) or the transaction manager (TM). The degree of a relationship type is the number of participating entity types.
5. The data processor (DP), which is the software component residing on each computer or device that stores and retrieves data located at the site. The DP is also known as the data manager (DM). A data processor may even be a centralized DBMS.
6. Three major categories are used to classify distributed database systems: (1) single-site processing, single-site data (SPSD); (2) multiple-site processing, single-site data (MPSD); and (3) multiple-site processing, multiple-site data (MPMD).
7. DDBMS characteristics are best described as a set of transparencies: distribution, transaction, failure, heterogeneity, and performance.

8. An undistributed transaction updates or requests data from a single site. A distributed transaction can update or request data from multiple sites

11.9 SUMMARY

- A distributed database stores logically related data in two or more physically independent sites connected via a computer network. The database is divided into fragments, which can be horizontal (a set of rows) or vertical (a set of attributes). Each fragment can be allocated to a different network node.
- Distributed processing is the division of logical database processing among two or more network nodes. Distributed databases require distributed processing. A distributed database management system (DDBMS) governs the processing and storage of logically related data through interconnected computer systems.
- The main components of a DDBMS are the transaction processor (TP) and the data processor (DP). The transaction processor component is the software that resides on each computer node that requests data. The data processor component is the software that resides on each computer that stores and retrieves data.
- Current database systems can be classified by the extent to which they support processing and data distribution. Three major categories are used to classify distributed database systems: (1) single-site processing, single-site data (SPSD); (2) multiple-site processing, single-site data (MPSD); and (3) multiple-site processing, multiple-site data (MPMD).
- A homogeneous distributed database system integrates only one particular type of DBMS over a computer network. A heterogeneous distributed database system integrates several different types of DBMSs over a computer network.
- DDBMS characteristics are best described as a set of transparencies: distribution, transaction, failure, heterogeneity, and performance. All transparencies share the common objective of making the distributed database behave as though it were a centralized database system; that is, the end user sees the data as part of a single logical centralized database and is unaware of the system's complexities.
- A transaction is formed by one or more database requests. An undistributed transaction updates or requests data from a single site. A distributed transaction can update or request data from multiple sites. Distributed concurrency control is required in a network of

distributed databases. A two-phase COMMIT protocol is used to ensure that all parts of a transaction are completed.

- The design of a distributed database must consider the fragmentation and replication of data. The designer must also decide how to allocate each fragment or replica to obtain better overall response time and to ensure data availability to the end user.
- A database can be replicated over several different sites on a computer network. The replication of the database fragments has the objective of improving data availability, thus decreasing access time. A database can be partially, fully, or not replicated. Data allocation strategies are designed to determine the location of the database fragments or replicas.
- Database vendors often label software as client/server database products. The client/server architecture label refers to the way in which two computers interact over a computer network to form a system.

11.10 KEYWORDS

- **Data manager** - which is the software component residing on each computer or device that stores and retrieves data located at the site.
- **Distributed database management system (DDBMS)** - A distributed database management system (DDBMS) governs the processing and storage of logically related data through interconnected computer systems.
- **Failure transparency** - which ensures that the system will continue to operate in the event of a node failure. Functions that were lost because of the failure will be picked up by another network node.
- **Fragmentation transparency**- Data fragmentation is transparent to the user, who sees only one logical database. The user does not need to know the name of the database fragments in order to retrieve them.
- **Subordinates** - The subordinates receive the message; write the transaction log, using the write-ahead protocol; and send an Acknowledgment

11.11 QUESTIONS FOR SELF-STUDY

1. What are the main reasons for and potential advantages of distributed databases?
2. What additional functions does a DDBMS have over a centralized DBMS? Explain.
3. What are the components of a DDBMS? Explain
4. List and explain the transparency features of a DDBMS.

5. Explain the need for the two-phase commit protocol. Then describe the two phases.

11.12 REFERENCES

1. Coronel, C., & Morris, S. (2016). Database systems: design, implementation, & management. Cengage Learning.
2. Gupta, S. B., & Mittal, A. (2009). Introduction to database management system. Laxmi Publications, Ltd..
3. Ramakrishnan, R., & Gehrke, J. (2000). Database management systems. McGraw-Hill.

UNIT -12: BUSINESS INTELLIGENCE

STRUCTURE

- 12.0 Objectives
- 12.1 The Need for Data Analysis
- 12.2 Business Intelligence
- 12.3 Decision Support Data
- 12.4 Data Analytics
- 12.5 Data Visualization
- 12.6 Check your progress
- 12.7 Summary
- 12.8 Keywords
- 12.9 Questions for self-study
- 12.10 References

12.0 OBJECTIVES

After studying this unit, you will be able to:

- ✓ Discuss how business intelligence is a comprehensive framework to support business decision making
- ✓ Define how operational data and decision support data differ
- ✓ Define what is data analytics
- ✓ Explain what data mining is and what role it plays in decision support
- ✓ Describe the need of data visualization

12.1 THE NEED FOR DATA ANALYSIS

Organizations tend to grow and prosper as they gain a better understanding of their environment. Most managers want to be able to track daily transactions to evaluate how the business is performing. By tapping into the operational database, management can develop strategies to meet organizational goals. In addition, data analysis can provide information about short-term tactical evaluations and strategies such as these: Are our sales promotions working? What market percentage are we controlling? Are we attracting new customers? Tactical and strategic decisions are also shaped by constant pressure from external and

internal forces, including globalization, the cultural and legal environment, and (perhaps most importantly) technology.

Given the many and varied competitive pressures, managers are always looking for a competitive advantage through product development and maintenance, service, market positioning, sales promotion, and so on. Managers understand that the business climate is dynamic, and thus, mandates their prompt reaction to change in order to remain competitive. In addition, the modern business climate requires managers to approach increasingly complex problems that involve a rapidly growing number of internal and external variables. It should also come as no surprise that interest is growing in creating support systems dedicated to facilitating quick decision making in a complex environment.

Different managerial levels require different decision support needs. For example, transaction-processing systems, based on operational databases, are tailored to serve the information needs of people who deal with short-term inventory, accounts payable, and purchasing. Middle-level managers, general managers, vice presidents, and presidents focus on strategic and tactical decision making. Those managers require detailed information designed to help them make decisions in a complex data and analysis environment.

Companies and software vendors addressed these multilevel decision support needs by creating independent applications to fit the needs of particular areas (finance, customer management, human resources, product support, etc.). Applications were also tailored to different industry sectors such as education, retail, health care, or financial. This approach worked well for some time, but changes in the business world (globalization, expanding markets, mergers and acquisitions, increased regulation, and more) called for new ways of integrating and managing data across levels, sectors, and geographic locations. This more comprehensive and integrated decision support framework within organizations became known as business intelligence.

12.2 BUSINESS INTELLIGENCE

Business intelligence (BI) is a term used to describe a comprehensive, cohesive, and integrated set of tools and processes used to capture, collect, integrate, store, and analyze data with the purpose of generating and presenting information used to support business decision making. As the name implies, BI is about creating intelligence about a business. This intelligence is based on learning and understanding the facts about a business environment. BI is a framework that allows a business to transform data into information, information into

knowledge, and knowledge into wisdom. BI has the potential to positively affect a company's culture by creating "business wisdom" and distributing it to all users in an organization. This business wisdom empowers users to make sound business decisions based on the accumulated knowledge of the business as reflected on recorded facts (historic operational data).

BI is a comprehensive endeavour because it encompasses all business processes within an organization. Business processes are the central units of operation in a business. Implementing BI in an organization involves capturing not only business data (internal and external) but also the metadata, or knowledge about the data. In practice, BI is a complex proposition that requires a deep understanding and alignment of the business processes, the internal and external data, and the information needs of users at all levels in an organization.

BI is not a product by itself, but a framework of concepts, practices, tools, and technologies that help a business better understand its core capabilities, provide snapshots of the company situation, and identify key opportunities to create competitive advantage. In practice, BI provides a well-orchestrated framework for the management of data that works across all levels of the organization. BI involves the following general steps:

1. Collecting and storing operational data.
2. Aggregating the operational data into decision support data.
3. Analysing decision support data to generate information.
4. Presenting such information to the end user to support business decisions.
5. Making business decisions, which in turn generate more data that is collected, stored, etc. (restarting the process).
6. Monitoring results to evaluate outcomes of the business decisions (providing more data to be collected, stored, etc.).

To implement all these steps, BI uses varied components and technologies. In the following sections, you will learn about the basic BI architecture and implementations.

12.3 DECISION SUPPORT DATA

Although BI is used at strategic and tactical managerial levels within organizations, its effectiveness depends on the quality of data gathered at the operational level. Yet operational data are seldom well suited to the decision support tasks. The differences between operational data and decision support data are examined in the next section.

Operational Data vs. Decision Support Data

Operational data and decision support data serve different purposes. Therefore, it is not surprising to learn that their formats and structures differ. Most operational data are stored in a relational database in which the structures (tables) tend to be highly normalized. Operational data storage is optimized to support transactions that represent daily operations. For example, each time an item is sold, it must be accounted for. Customer data, inventory data, and so on, are in a frequent update mode.

To provide effective update performance, operational systems store data in many tables, each with a minimum number of fields. Thus, a simple sales transaction might be represented by five or more different tables (for example, invoice, invoice line, discount, store, and department). Although such an arrangement is excellent in an operational database, it is not efficient for query processing. For example, to extract a simple invoice, you would have to join several tables. Whereas operational data are useful for capturing daily business transactions, decision support data give tactical and strategic business meaning to the operational data. From the data analyst's point of view, decision support data differ from operational data in three main areas: time span, granularity, and dimensionality.

- Time span. Operational data cover a short time frame. In contrast, decision support data tend to cover a longer time frame. Managers are seldom interested in a specific sales invoice to customer X; rather, they tend to focus on sales generated during the last month, the last year, or the last five years.
- Granularity (level of aggregation). Decision support data must be presented at different levels of aggregation, from highly summarized to near-atomic. For example, if managers must analyze sales by region, they must be able to access data showing the sales by region, by city within the region, by store within the city within the region, and so on. In that case, summarized data to compare the regions is required, and also data in a structure that enables a manager to drill down, or decompose, the data into more atomic components (that is, finer-grained data at lower levels of aggregation). In contrast, when you roll up the data, you are aggregating the data to a higher level.
- Dimensionality. Operational data focus on representing individual transactions rather than on the effects of the transactions over time. In contrast, data analysts tend to include many data dimensions and are interested in how the data relate over those dimensions. For example, an analyst might want to know how product X fared

relative to product Z during the past six months by region, state, city, store, and customer. In that case, both place and time are part of the picture.

Table 12.1 summarizes the differences between operational and decision support data from the database designer's point of view.

Table 12.1 Contrasting operational and decision support data characteristics

CHARACTERISTIC	OPERATIONAL DATA	DECISION SUPPORT DATA
Data currency	Current operations Real-time data	Historic data Snapshot of company data Time component (week/month/year)
Granularity	Atomic-detailed data	Summarized data
Summarization level	Low; some aggregate yields	High; many aggregation levels
Data model	Highly normalized Mostly relational DBMS	Non-normalized Complex structures Some relational, but mostly multidimensional DBMS
Transaction type	Mostly updates	Mostly query
Transaction volumes	High update volumes	Periodic loads and summary calculations
Transaction speed	Updates are critical	Retrievals are critical
Query activity	Low-to-medium	High
Query scope	Narrow range	Broad range
Query complexity	Simple-to-medium	Very complex
Data volumes	Hundreds of gigabytes	Terabytes to petabytes

The many differences between operational data and decision support data are good indicators of the requirements of the decision support database, described in the next section.

Decision Support Database Requirements

A decision support database is a specialized DBMS tailored to provide fast answers to complex queries. There are four main requirements for a decision support database: the database schema, data extraction and loading, the end-user analytical interface, and database size.

Database Schema

The decision support database schema must support complex (non-normalized) data representations. As noted earlier, the decision support database must contain data that are aggregated and summarized. In addition to meeting those requirements, the queries must be able to extract multidimensional time slices. If you are using an RDBMS, the conditions suggest using non-normalized and even duplicated data. To see why this must be true, take a look at the 10-year sales history for a single store containing a single department. At this point, the data are fully normalized within the single table

This structure works well when you have only one store with only one department. However, it is very unlikely that such a simple environment has much need for a decision support database. One would suppose that a decision support database becomes a factor when dealing with more than one store, each of which has more than one department. To support all of the decision support requirements, the database must contain data for all of the stores and all of their departments—and the database must be able to support multidimensional queries that track sales by stores, by departments, and over time. For simplicity, suppose that there are only two stores (A and B) and two departments (1 and 2) within each store. Let's also change the time dimension to include yearly data. You can see that the number of rows and attributes already multiplies quickly and that the table exhibits multiple redundancies.

Now suppose that the company has 10 departments per store and 20 stores nationwide. And suppose that you want to access yearly sales summaries. Now you are dealing with 200 rows and 12 monthly sales attributes per row. (Actually, there are 13 attributes per row if you add each store's sales total for each year.)

The decision support database schema must also be optimized for query (read-only) retrievals. To optimize query speed, the DBMS must support features such as bitmap indexes and data partitioning to increase search speed. In addition, the DBMS query optimizer must be enhanced to support the non-normalized and complex structures found in decision support databases.

Data Extraction and Filtering

The decision support database is created largely by extracting data from the operational database and by importing additional data from external sources. Thus, the DBMS must support advanced data extraction and data-filtering tools.

To minimize the impact on the operational database, the data extraction capabilities should allow batch and scheduled data extraction. The data extraction capabilities should also support different data sources: flat files and hierarchical, network, and relational databases, as well as multiple vendors. Data-filtering capabilities must include the ability to check for inconsistent data or data validation rules. Finally, to filter and integrate the operational data into the decision support database, the DBMS must support advanced data integration, aggregation, and classification.

Using data from multiple external sources also usually means having to solve data-formatting conflicts. For example, data such as Social Security numbers and dates can occur in different

formats; measurements can be based on different scales, and the same data elements can have different names. In short, data must be filtered and purified to ensure that only the pertinent decision support data are stored in the database and that they are stored in a standard format.

End-User Analytical Interface

The decision support DBMS must support advanced data-modelling and data presentation tools. Using those tools makes it easy for data analysts to define the nature and extent of business problems. Once the problems have been defined, the decision support DBMS must generate the necessary queries to retrieve the appropriate data from the decision support database. If necessary, the query results may then be evaluated with data analysis tools supported by the decision support DBMS. Because queries yield crucial information for decision makers, the queries must be optimized for speedy processing. The end-user analytical interface is one of the most critical DBMS components. When properly implemented, an analytical interface permits the user to navigate through the data to simplify and accelerate the decision-making process.

Database Size

Decision support databases tend to be very large; gigabyte and terabyte ranges are not unusual. For example, in 2008, Wal-Mart, the world's largest company, had more than 4 petabytes of data in its data warehouses. As mentioned earlier, the decision support database typically contains redundant and duplicated data to improve data retrieval and simplify information generation. Therefore, the DBMS must be capable of supporting very large databases (VLDBs). To support a VLDB adequately, the DBMS might be required to use advanced hardware, such as multiple disk arrays, and even more importantly, to support multiple-processor technologies, such as a symmetric multiprocessor (SMP) or a massively parallel processor (MPP).

The complex information requirements and the ever-growing demand for sophisticated data analysis sparked the creation of a new type of data repository. This repository contains data in formats that facilitate data extraction, data analysis, and decision making. This data repository is known as a data warehouse and has become the foundation for a new generation of decision support systems.

12.4 DATA ANALYTICS

The data analytics process has some components that can help a variety of initiatives. By combining these components, a successful data analytics initiative will provide a clear picture of where you are, where you have been and where you should go.

- Generally, this process begins with descriptive analytics. This is the process of describing historical trends in data. Descriptive analytics aims to answer the question “what happened?” This often involves measuring traditional indicators such as return on investment (ROI). The indicators used will be different for each industry. Descriptive analytics does not make predictions or directly inform decisions. It focuses on summarizing data in a meaningful and descriptive way.
- The next essential part of data analytics is advanced analytics. This part of data science takes advantage of advanced tools to extract data, make predictions and discover trends. These tools include classical statistics as well as machine learning. Machine learning technologies such as neural networks, natural language processing, sentiment analysis and more enable advanced analytics. This information provides new insight from data. Advanced analytics addresses “what if?” questions.
- The availability of machine learning techniques, massive data sets, and cheap computing power has enabled the use of these techniques in many industries. The collection of big data sets is instrumental in enabling these techniques. Big data analytics enables businesses to draw meaningful conclusions from complex and varied data sources, which has been made possible by advances in parallel processing and cheap computational power.

Types of Data Analytics

Data analytics is a broad field. There are four primary types of data analytics: descriptive, diagnostic, predictive and prescriptive analytics. Each type has a different goal and a different place in the data analysis process. These are also the primary data analytics applications in business.

- Descriptive analytics helps answer questions about what happened. These techniques summarize large datasets to describe outcomes to stakeholders. By developing key performance indicators (KPIs,) these strategies can help track successes or failures. Metrics such as return on investment (ROI) are used in many industries. Specialized metrics are developed to track performance in specific industries. This process

requires the collection of relevant data, processing of the data, data analysis and data visualization. This process provides essential insight into past performance.

- Diagnostic analytics helps answer questions about why things happened. These techniques supplement more basic descriptive analytics. They take the findings from descriptive analytics and dig deeper to find the cause. The performance indicators are further investigated to discover why they got better or worse. This generally occurs in three steps:
 - Identify anomalies in the data. These may be unexpected changes in a metric or a particular market.
 - Data that is related to these anomalies is collected.
 - Statistical techniques are used to find relationships and trends that explain these anomalies.
- Predictive analytics helps answer questions about what will happen in the future. These techniques use historical data to identify trends and determine if they are likely to recur. Predictive analytical tools provide valuable insight into what may happen in the future and its techniques include a variety of statistical and machine learning techniques, such as: neural networks, decision trees, and regression.
- Prescriptive analytics helps answer questions about what should be done. By using insights from predictive analytics, data-driven decisions can be made. This allows businesses to make informed decisions in the face of uncertainty. Prescriptive analytics techniques rely on machine learning strategies that can find patterns in large datasets. By analyzing past decisions and events, the likelihood of different outcomes can be estimated.

These types of data analytics provide the insight that businesses need to make effective and efficient decisions. Used in combination they provide a well-rounded understanding of a company's needs and opportunities.

What is the Role of Data Analyst?

Data analysts exist at the intersection of information technology, statistics and business. They combine these fields in order to help businesses and organizations succeed. The primary goal of a data analyst is to increase efficiency and improve performance by discovering patterns in data.

The work of a data analyst involves working with data throughout the data analysis pipeline. This means working with data in various ways. The primary steps in the data analytics process are data mining, data management, statistical analysis, and data presentation. The importance and balance of these steps depend on the data being used and the goal of the analysis.

Data mining is an essential process for many data analytics tasks. This involves extracting data from unstructured data sources. These may include written text, large complex databases, or raw sensor data. The key steps in this process are to extract, transform, and load data (often called ETL.) These steps convert raw data into a useful and manageable format. This prepares data for storage and analysis. Data mining is generally the most time-intensive step in the data analysis pipeline.

Data management or data warehousing is another key aspect of a data analyst's job. Data warehousing involves designing and implementing databases that allow easy access to the results of data mining. This step generally involves creating and managing SQL databases. Non-relational and NoSQL databases are becoming more common as well.

Statistical analysis allows analysts to create insights from data. Both statistics and machine learning techniques are used to analyze data. Big data is used to create statistical models that reveal trends in data. These models can then be applied to new data to make predictions and inform decision making. Statistical programming languages such as R or Python (with pandas) are essential to this process. In addition, open source libraries and packages such as TensorFlow enable advanced analysis.

The final step in most data analytics processes is data presentation. This step allows insights to be shared with stakeholders. Data visualization is often the most important tool in data presentation. Compelling visualizations can help tell the story in the data which may help executives and managers understand the importance of these insights.

Why Data Analytics is Important?

The applications of data analytics are broad. Analyzing big data can optimize efficiency in many different industries. Improving performance enables businesses to succeed in an increasingly competitive world.

One of the earliest adopters is the financial sector. Data analytics has an important role in the banking and finance industries, used to predict market trends and assess risk. Credit scores are an example of data analytics that affects everyone. These scores use many data points to

determine lending risk. Data analytics is also used to detect and prevent fraud to improve efficiency and reduce risk for financial institutions.

The use of data analytics goes beyond maximizing profits and ROI, however. Data analytics can provide critical information for healthcare (health informatics), crime prevention, and environmental protection. These applications of data analytics use these techniques to improve our world.

Though statistics and data analysis have always been used in scientific research, advanced analytic techniques and big data allow for many new insights. These techniques can find trends in complex systems. Researchers are currently using machine learning to protect wildlife.

The use of data analytics in healthcare is already widespread. Predicting patient outcomes, efficiently allocating funding and improving diagnostic techniques are just a few examples of how data analytics is revolutionizing healthcare. The pharmaceutical industry is also being revolutionized by machine learning. Drug discovery is a complex task with many variables. Machine learning can greatly improve drug discovery. Pharmaceutical companies also use data analytics to understand the market for drugs and predict their sales.

The internet of things (IoT) is a field that is used alongside machine learning. These devices provide a great opportunity for data analytics. IoT devices often contain many sensors that collect meaningful data points for their operation. Devices like the Nest thermostat track movement and temperature to regulate heating and cooling. Smart devices like this can use data to learn from and predict your behavior. This will provide advance home automation that can adapt to the way you live.

The applications of data analytics are seemingly endless. More and more data is being collected every day — this presents new opportunities to apply data analytics to more parts of business, science and everyday life.

12.5 DATA VISUALIZATION

Data visualization is an interdisciplinary field that deals with the graphic representation of data. It is a particularly efficient way of communicating when the data is numerous as for example a time series.

From an academic point of view, this representation can be considered as a mapping between the original data (usually numerical) and graphic elements. (for example, lines or points in a

chart). The mapping determines how the attributes of these elements vary according to the data. In this light, a bar chart is a mapping of the length of a bar to a magnitude of a variable. Since the graphic design of the mapping can adversely affect the readability of a chart, mapping is a core competency of Data visualization.

Data visualization has its roots in the field of Statistics and is therefore generally considered a branch of Descriptive Statistics. However, because both design skills and statistical and computing skills are required to visualize effectively, it is argued by some authors that it is both an Art and a Science.

Research into how people read and misread various types of visualizations is helping to determine what types and features of visualizations are most understandable and effective in conveying information.

To communicate information clearly and efficiently, data visualization uses statistical graphics, plots, information graphics and other tools. Numerical data may be encoded using dots, lines, or bars, to visually communicate a quantitative message. Effective visualization helps users analyze and reason about data and evidence. It makes complex data more accessible, understandable, and usable. Users may have particular analytical tasks, such as making comparisons or understanding causality and the design principle of the graphic (i.e., showing comparisons or showing causality) follows the task. Tables are generally used where users will look up a specific measurement, while charts of various types are used to show patterns or relationships in the data for one or more variables.

Data visualization refers to the techniques used to communicate data or information by encoding it as visual objects (e.g., points, lines, or bars) contained in graphics. The goal is to communicate information clearly and efficiently to users. It is one of the steps in data analysis or data science. According to Vitaly Friedman (2008) the "main goal of data visualization is to communicate information clearly and effectively through graphical means. It doesn't mean that data visualization needs to look boring to be functional or extremely sophisticated to look beautiful. To convey ideas effectively, both aesthetic form and functionality need to go hand in hand, providing insights into a rather sparse and complex data set by communicating its key aspects in a more intuitive way. Yet designers often fail to achieve a balance between form and function, creating gorgeous data visualizations which fail to serve their main purpose — to communicate information".

Indeed, Fernanda Viegas and Martin M. Wattenberg suggested that an ideal visualization should not only communicate clearly, but stimulate viewer engagement and attention.

Data visualization is closely related to information graphics, information visualization, scientific visualization, exploratory data analysis and statistical graphics. In the new millennium, data visualization has become an active area of research, teaching and development. According to Post et al. (2002), it has united scientific and information visualization.

In the commercial environment data visualization is often referred to as dashboards. Infographics are another very common form of data visualization.

Characteristics of effective graphical displays

Professor Edward Tufte explained that users of information displays are executing particular analytical tasks such as making comparisons. The design principle of the information graphic should support the analytical task. As William Cleveland and Robert McGill show, different graphical elements accomplish this more or less effectively. For example, dot plots and bar charts outperform pie charts.

In his 1983 book *The Visual Display of Quantitative Information*, Edward Tufte defines 'graphical displays' and principles for effective graphical display in the following passage: "Excellence in statistical graphics consists of complex ideas communicated with clarity, precision, and efficiency. Graphical displays should:

- show the data
- induce the viewer to think about the substance rather than about methodology, graphic design, the technology of graphic production, or something else
- avoid distorting what the data has to say
- present many numbers in a small space
- make large data sets coherent
- encourage the eye to compare different pieces of data
- reveal the data at several levels of detail, from a broad overview to the fine structure
- serve a reasonably clear purpose: description, exploration, tabulation, or decoration
- be closely integrated with the statistical and verbal descriptions of a data set.

Graphics reveal data. Indeed graphics can be more precise and revealing than conventional statistical computations."

Quantitative messages

Author Stephen Few described eight types of quantitative messages that users may attempt to understand or communicate from a set of data and the associated graphs used to help communicate the message:

1. Time-series: A single variable is captured over a period of time, such as the unemployment rate over a 10-year period. A line chart may be used to demonstrate the trend.
2. Ranking: Categorical subdivisions are ranked in ascending or descending order, such as a ranking of sales performance (the measure) by sales persons (the category, with each sales person a categorical subdivision) during a single period. A bar chart may be used to show the comparison across the sales persons.
3. Part-to-whole: Categorical subdivisions are measured as a ratio to the whole (i.e., a percentage out of 100%). A pie chart or bar chart can show the comparison of ratios, such as the market share represented by competitors in a market.
4. Deviation: Categorical subdivisions are compared against a reference, such as a comparison of actual vs. budget expenses for several departments of a business for a given time period. A bar chart can show comparison of the actual versus the reference amount.
5. Frequency distribution: Shows the number of observations of a particular variable for given interval, such as the number of years in which the stock market return is between intervals such as 0-10%, 11-20%, etc. A histogram, a type of bar chart, may be used for this analysis. A boxplot helps visualize key statistics about the distribution, such as median, quartiles, outliers, etc.
6. Correlation: Comparison between observations represented by two variables (X,Y) to determine if they tend to move in the same or opposite directions. For example, plotting unemployment (X) and inflation (Y) for a sample of months. A scatter plot is typically used for this message.
7. Nominal comparison: Comparing categorical subdivisions in no particular order, such as the sales volume by product code. A bar chart may be used for this comparison.
8. Geographic or geospatial: Comparison of a variable across a map or layout, such as the unemployment rate by state or the number of persons on the various floors of a building. A cartogram is a typical graphic used.

Analysts reviewing a set of data may consider whether some or all of the messages and graphic types above are applicable to their task and audience. The process of trial and error to identify meaningful relationships and messages in the data is part of exploratory data analysis.

12.6 CHECK YOUR PROGRESS

1. What is Business intelligence?
2. Define data warehouse.
3. What is data integration
4. Which are different types of data analytics?
5. Define Decision support systems (DSS)

Answers to check your progress:

1. Business intelligence (BI) is a term used to describe a comprehensive, cohesive, and integrated set of tools and processes used to capture, collect, integrate, store, and analyse data with the purpose of generating and presenting information used to support business decision making
2. Defines the term as “an integrated, subject-oriented, time-variant, non-volatile collection of data derived
3. Data integration implies that all business entities, data elements, data characteristics, and business metrics are described in the same way throughout the enterprise.
4. Descriptive, Diagnostic, prescriptive, predictive
5. Decision support systems (DSS) refers to an arrangement of computerized tools used to assist managerial decision making within a business. DSS were the original precursor of current-generation BI systems

12.7 SUMMARY

- Business intelligence (BI) is a term used to describe a comprehensive, cohesive, and integrated set of applications used to capture, collect, integrate, store, and analyze data with the purpose of generating and presenting information used to support business decision making.
- BI covers a range of technologies and applications to manage the entire data life cycle from acquisition to storage, transformation, integration, analysis, monitoring, presentation,

and archiving. BI functionality ranges from simple data gathering and extraction to very complex data analysis and presentation.

- Decision support systems (DSS) refers to an arrangement of computerized tools used to assist managerial decision making within a business. DSS were the original precursor of current-generation BI systems.
- Operational data are not well suited for decision support. From the end-user point of view, decision support data differ from operational data in three main areas: time span, granularity, and dimensionality.
- The requirements for a decision support DBMS are divided into four main categories: database schema, data extraction and filtering, end-user analytical interface, and database size requirements.
- Attributes can be ordered in well-defined attribute hierarchies. The attribute hierarchy provides a top-down organization that is used for two main purposes: to permit aggregation and to provide drill-down/roll-up data analysis.

12.8 KEYWORDS

- **Business intelligence** - BI covers a range of technologies and applications to manage the entire data life cycle from acquisition to storage, transformation, integration, analysis, monitoring, presentation, and archiving.
- **Data store** - is optimized for decision support and is generally represented by a data warehouse or a data mart.
- **Data warehouse** - is usually a read-only database optimized for data analysis and query processing.
- **Relational online analytical processing (ROLAP)** provides OLAP functionality by using relational databases and familiar relational query tools to store and analyze multidimensional data.

12.9 QUESTIONS FOR SELF-STUDY

1. Describe the BI framework.
2. What are decision support systems, and what role do they play in the business environment?
3. Explain how the main components of the BI architecture interact to form a system.

4. What is a data warehouse, and what are its main characteristics? How does it differ from a data mart?
5. Give three examples of problems likely to be encountered when operational data are integrated into the data warehouse.

12.10 REFERENCES

1. Coronel, C., & Morris, S. (2016). Database systems: design, implementation, & management. Cengage Learning.
2. Gupta, S. B., & Mittal, A. (2009). Introduction to database management system. Laxmi Publications, Ltd..
3. Ramakrishnan, R., & Gehrke, J. (2000). Database management systems. McGraw-Hill.



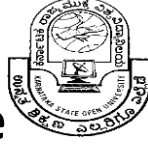
Karnataka State Open University
Mukthagangothri, Mysore – 570 006.
Dept. of Studies and Research in Management

MBA IT Specialization
III Semester

Database Management System



Block 4



Karnataka State Open University

Mukthagangothri, Mysore – 570 006.

Dept. of Studies and Research in Management

MBA. IT Specialization

III Semester

DATABASE MANAGEMENT SYSTEM

BLOCK 1

UNIT NO.	TITLE	PAGE NUMBERS
UNIT 13	DATABASE CONNECTIVITY AND WEB TECHNOLOGIES	1-21
UNIT 14	DATABASE SECURITY AND AUTHORIZATION	22-32
UNIT 15	DATABASE RECOVERY SYSTEM	33-42
UNIT 16	DATABASE ADMINISTRATION AND SECURITY	43-73

BLOCK 4 INTRODUCTION

In Unit 13 you will learn about the architectures that applications use to connect to databases and the basics of Web database technologies. Given the growing relationship between the Web and databases, database professionals must know how to create, use, and manage Web interfaces to those databases. In unit 14 various threats to database security, protection against unauthorized access and other security mechanisms have been discussed. The goal of database security is the protection of data against threats such as accidental or intentional loss, misuse or corruption by unauthorized users. The DBA is responsible for the overall security of the database system. Therefore, the DBA must identify the most serious threats to the database and accordingly develop overall policies, procedures and appropriate controls to protect the databases. In unit 15, we will discuss different failures and database recovery techniques used so that the database system is restored to the most recent consistent state that existed shortly before the time of system failure. A Computer system can be failed due to variety of reasons like disk crash, power fluctuation, software error, sabotage and fire or flood at computer site. These failures result in the loss of information. Thus, database must be capable of handling such situations to ensure that the atomicity and durability properties of transactions are preserved. An integral part of a database system is the recovery manager that is responsible for recovering the data. It ensures atomicity by undoing the actions of transactions that do not commit and durability by making sure that all actions of committed transactions survive system crashes and media failures. The recovery manager deal with a wide variety of database states because it is called during system failures. Furthermore, the database recovery is the process of restoring the database to a consistent state when a failure occurred. Unit 16, shows you the basis for a successful database administration strategy. Such a strategy requires that data be considered important and valuable resources to be treated and managed as corporate assets. The unit explores how a database fits within an organization, what the data views and requirements are at various management levels, and how the DBMS supports those views and requirements. Database administration must be fully understood and accepted within an organization before a sound data administration strategy can be implemented. In this unit, you will learn about important data management issues by looking at the managerial and technical roles of the database administrator (DBA). This unit also explores database security issues, such as the confidentiality, integrity, and availability of data.

This block consists of 4 units and is organized as follows:

Unit 13: Database Connectivity and Web Technologies Database Connectivity, Database Internet Connectivity, Extensible Markup Language (XML)

Unit 14. Database Security and Authorization Introduction, Security Violations, Authorization (Access Rights), Views, Granting of Privileges, Notion of Roles (Grouping of Users), Audit Trails

Unit 15. Database Recovery System Introduction, Classification of Failures, Recovery Concept, Log based recovery, Shadow Paging

Unit 16: Database Administration and Security Data as a Corporate Asset , The Need for a Database and Its Role in an Organization , Introduction of a Database: Special Considerations, The Evolution of Database Administration, The Database Environment's Human Component, Security, Database Administration Tools, Developing a Data Administration Strategy, The DBA's Role in the Cloud , The DBA at Work: Using Oracle for Database Administration

UNIT-13: DATABASE CONNECTIVITY AND WEB TECHNOLOGIES

STRUCTURE

- 13.0 Objectives
- 13.1 Database Connectivity
- 13.2 Database Internet Connectivity
- 13.3 Extensible Markup Language (XML)
- 13.4 Check your progress
- 13.5 Summary
- 13.6 Keywords
- 13.7 Questions for self-study
- 13.8 References

13.0 OBJECTIVES

After studying this unit, you will be able to:

- ✓ Discuss about various database connectivity technologies
- ✓ Explain how Web-to-database middleware is used to integrate databases with the Internet
- ✓ Use Web browser plug-ins and extensions
- ✓ Describe what services are provided by Web application servers
- ✓ Understand Extensible Markup Language (XML) is and why it is important for Web database development
- ✓ Discuss about SQL data services and how they can reduce the cost of data management

13.1 DATABASE CONNECTIVITY

Database connectivity refers to the mechanisms through which application programs connect and communicate with data repositories. The data repository, also known as the data source, represents the data management application, such as Oracle RDBMS, SQL Server DBMS, or IBM DBMS, that will be used to store the data generated by the application

program. Ideally, a data source or data repository could be located anywhere and hold any type of data. For example, the data source could be a relational database, a hierarchical database, a spreadsheet, or a text data file.

The need for standard database connectivity interfaces cannot be overstated. Just as SQL has become the de facto data manipulation language, there is a need for a standard database connectivity interface that will enable applications to connect to data repositories. There are many different ways to achieve database connectivity. This section will cover only the following interfaces:

- Native SQL connectivity (vendor provided).
- Microsoft's Open Database Connectivity (ODBC), Data Access Objects (DAO), and Remote Data Objects (RDO).
- Microsoft's Object Linking and Embedding for Database (OLE-DB).
- Microsoft's ActiveX Data Objects (ADO.NET).
- Sun's Java Database Connectivity (JDBC).

You should not be surprised to learn that most interfaces you are likely to encounter are Microsoft offerings. After all, client applications connect to databases, and the majority of those applications run on computers that are powered by some version of Microsoft Windows. The data connectivity interfaces illustrated here are dominant players in the market and more importantly, they enjoy the support of the majority of database vendors. In fact, ODBC, OLE-DB, and ADO.NET form the backbone of Microsoft's Universal Data Access (UDA) architecture, a collection of technologies used to access any type of data source and manage the data through a common interface. As you will see, Microsoft's database connectivity interfaces have evolved over time: each interface builds on top of the other, thus providing enhanced functionality, features, flexibility, and support.

Native SQL Connectivity

Most DBMS vendors provide their own methods for connecting to their databases. Native SQL connectivity refers to the connection interface that is provided by the database vendor and that is unique to that vendor. The best example of that type of native interface is the Oracle RDBMS. To connect a client application to an Oracle database, you must install and configure the Oracle's SQL*Net interface in the client computer.

Native database connectivity interfaces are optimized for "their" DBMS, and those interfaces support access to most, if not all, of the database features. However, maintaining multiple

native interfaces for different databases can become a burden for the programmer. Therefore, the need for “universal” database connectivity arises. Usually, the native database connectivity interface provided by the vendor is not the only way to connect to a database; most current DBMS products support other database connectivity standards, the most common being ODBC.

ODBC, DAO, and RDO

Developed in early 1990s, Open Database Connectivity (ODBC) is Microsoft’s implementation of a superset of the SQL Access Group Call Level Interface (CLI) standard for database access. ODBC is probably the most widely supported database connectivity interface. ODBC allows any Windows application to access relational data sources, using SQL via a standard application programming interface (API). Most operating environments, such as Microsoft Windows, provide an API so that programmers can write applications consistent with the operating environment. Although APIs are designed for programmers, they are ultimately good for users because they guarantee that all programs using a common API will have similar interfaces. That makes it easy for users to learn new programs.

ODBC was the first widely adopted database middleware standard, and it enjoyed rapid adoption in Windows applications. As programming languages evolved, ODBC did not provide significant functionality beyond the ability to execute SQL to manipulate relational style data. Therefore, programmers needed a better way to access data. To answer that need, Microsoft developed two other data access interfaces:

- Data Access Objects (DAO) is an object-oriented API used to access MS Access, MS FoxPro, and dBase databases (using the Jet data engine) from Visual Basic programs. DAO provided an optimized interface that exposed to programmers the functionality of the Jet data engine (on which the MS Access database is based). The DAO interface can also be used to access other relational-style data sources.
- Remote Data Objects (RDO) is a higher-level object-oriented application interface used to access remote database servers. RDO uses the lower-level DAO and ODBC for direct access to databases. RDO was optimized to deal with server-based databases, such as MS SQL Server, Oracle, and DB2. Figure 14.2 illustrates how Windows applications can use ODBC, DAO, and RDO to access local and remote relational data sources. As you can tell by examining Figure 13.2, client applications can use ODBC to access relational data sources. However, the DAO

and RDO object interfaces provide more functionality. DAO and RDO make use of the underlying ODBC data services. ODBC, DAO, and RDO are implemented as shared code that is dynamically linked to the Windows operating environment through dynamic-link libraries (DLLs), which are stored as files with the .dll extension. Running as a DLL, the code speeds up load and run times.

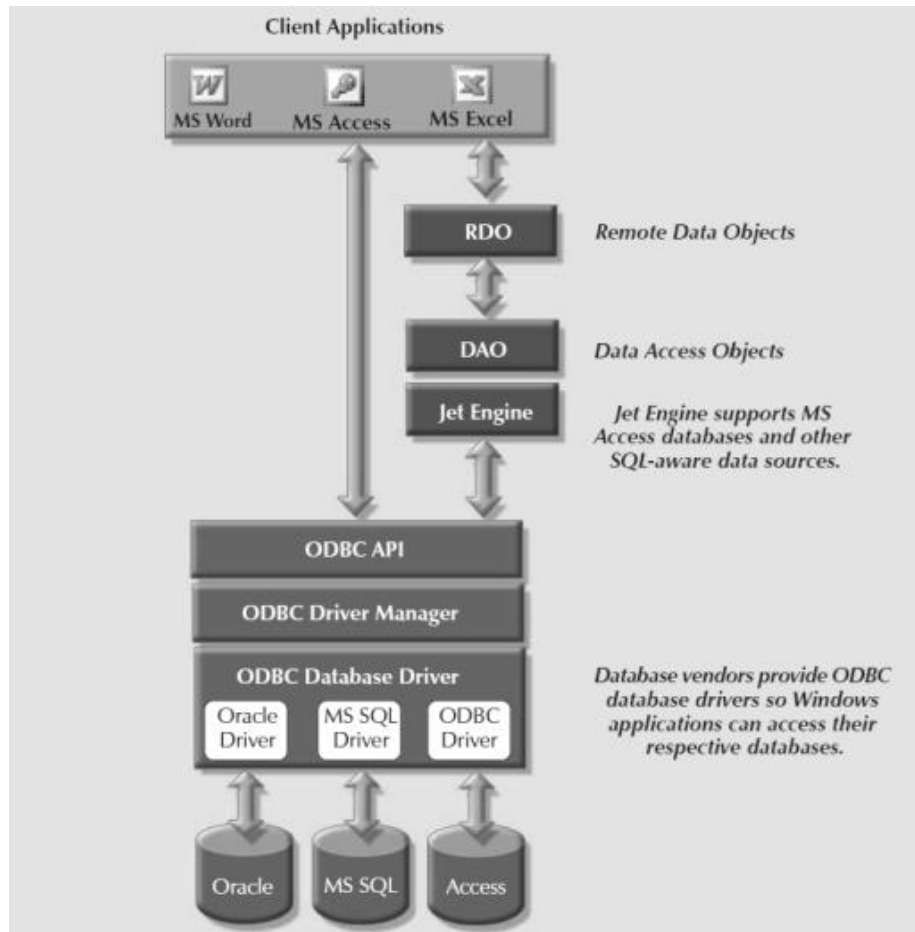


Fig 13.1 Using ODBC, DAO and RDO to access databases

The basic ODBC architecture has three main components:

- A high-level ODBC API through which application programs access ODBC functionality.
- A driver manager that is in charge of managing all database connections.
- An ODBC driver that communicates directly to the DBMS.

Defining a data source is the first step in using ODBC. To define a data source, you must create a data source name (DSN) for the data source. To create a DSN you need to provide:

- An ODBC driver. You must identify the driver to use to connect to the data source. The ODBC driver is normally provided by the database vendor, although Microsoft provides

several drivers that connect to most common databases. For example, if you are using an Oracle DBMS, you will select the Oracle ODBC driver provided by Oracle, or if desired, the Microsoft-provided ODBC driver for Oracle.

- A DSN name. This is a unique name by which the data source will be known to ODBC, and therefore, to applications. ODBC offers two types of data sources: user and system. User data sources are available only to the user. System data sources are available to all users, including operating system services.
- ODBC driver parameters. Most ODBC drivers require specific parameters in order to establish a connection to the database. For example, if you are using an MS Access database, you must point to the location of the MS Access file, and if necessary, provide a username and password. If you are using a DBMS server, you must provide the server name, the database name, the username, and the password needed to connect to the database. Figure 13.2 shows the ODBC screens required to create a System ODBC data source for an Oracle DBMS. Note that some ODBC drivers use the native driver provided by the DBMS vendor.

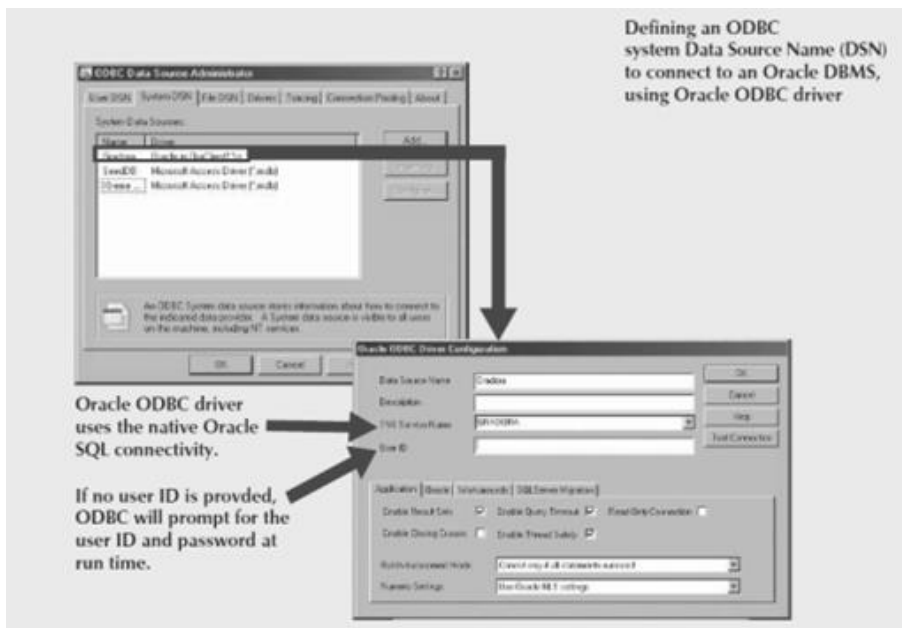


Fig.13.2 Configuring an oracle ODBC data source

Once the ODBC data source is defined, application programmers can write to the ODBC API by issuing specific commands and providing the required parameters. The ODBC Driver Manager will properly route the calls to the appropriate data source. The ODBC API standard defines three levels of compliance: Core, Level-1, and Level-2, which provide increasing levels of functionality. For example, Level-1 might provide support for most SQL DDL and

DML statements, including subqueries and aggregate functions, but no support for procedural SQL or cursors. The database vendors can choose which level to support. However, to interact with ODBC, the database vendor must implement all of the features indicated in that ODBC API support level.

OLE-DB

Although ODBC, DAO, and RDO were widely used, they did not provide support for non-relational data. To answer that need and to simplify data connectivity, Microsoft developed Object Linking and Embedding for Database (OLE-DB). Based on Microsoft's Component Object Model (COM), OLE-DB is database middleware that adds object-oriented functionality for access to relational and non-relational data. OLE-DB was the first part of Microsoft's strategy to provide a unified object-oriented framework for the development of next-generation applications.

OLE-DB is composed of a series of COM objects that provide low-level database connectivity for applications. Because OLE-DB is based on COM, the objects contain data and methods, also known as the interface. The OLE-DB model is better understood when you divide its functionality into two types of objects:

- Consumers are objects (applications or processes) that request and use data. The data consumers request data by invoking the methods exposed by the data provider objects (public interface) and passing the required parameters.
 - Providers are objects that manage the connection with a data source and provide data to the consumers. Providers are divided into two categories: data providers and service providers.
 - Data providers provide data to other processes. Database vendors create data provider objects that expose the functionality of the underlying data source (relational, object-oriented, text, and so on).
- Service providers provide additional functionality to consumers. The service provider is located between the data provider and the consumer. The service provider requests data from the data provider, transforms the data, and then provides the transformed data to the data consumer. In other words, the service provider acts like a data consumer of the data provider and as a data provider for the data consumer (end-user application). For example, a service provider could offer cursor management services, transaction management services, query processing services, and indexing services.

As a common practice, many vendors provide OLE-DB objects to augment their ODBC support, effectively creating a shared object layer on top of their existing database connectivity (ODBC or native) through which applications can interact. The OLE-DB objects expose functionality about the database; for example, there are objects that deal with relational data, hierarchical data, and flat-file text data. Additionally, the objects implement specific tasks, such as establishing a connection, executing a query, invoking a stored procedure, defining a transaction, or invoking an OLAP function. By using OLE-DB objects, the database vendor can choose what functionality to implement in a modular way, instead of being forced to include all of the functionality all of the time.

Although the ADO model is a tremendous improvement over the OLE-DB model, Microsoft is actively encouraging programmers to use its newer data access framework, ADO.NET.

ADO.NET

Based on ADO, ADO.NET is the data access component of Microsoft's .NET application development framework. The Microsoft .NET framework is a component-based platform for developing distributed, heterogeneous, interoperable applications aimed at manipulating any type of data over any network under any operating system and any programming language.

Therefore, this section will only introduce the basic data access component of the .NET architecture, ADO.NET. It's important to understand that the .NET framework extends and enhances the functionality provided by the ADO/OLE-DB duo. ADO.NET introduced two new features critical for the development of distributed applications:

DataSets and XML support

The ADO.NET framework consolidates all data access functionality under one integrated object model. In this object model, several objects interact with one another to perform specific data manipulation functions. Those objects can be grouped as data providers and consumers.

Data provider objects are provided by the database vendors. However, ADO.NET comes with two standard data providers: a data provider for OLE-DB data sources and a data provider for SQL Server. That way ADO.NET can work with any previously supported database, including an ODBC database with an OLE-DB data provider. At the same time, ADO.NET includes a highly optimized data provider for SQL Server.

Java Database Connectivity (JDBC)

Java is an object-oriented programming language developed by Sun Microsystems that runs on top of Web browser software. Java is one of the most common programming languages for Web development. Sun Microsystems created Java as a “write once, run anywhere” environment. That means that a programmer can write a Java application once and then without any modification, run the application in multiple environments (Microsoft Windows, Apple OS X, IBM AIX, etc.).

When Java applications want to access data outside the Java runtime environment, they use predefined application programming interfaces. Java Database Connectivity (JDBC) is an application programming interface that allows a Java program to interact with a wide range of data sources (relational databases, tabular data sources, spreadsheets, and text files). JDBC allows a Java program to establish a connection with a data source, prepare and send the SQL code to the database server, and process the result set.

As a matter of fact, JDBC allows direct access to a database server or access via database middleware. Furthermore, JDBC provides a way to connect to databases through an ODBC driver. Figure 13.3 illustrates the basic JDBC architecture and the various database access styles.

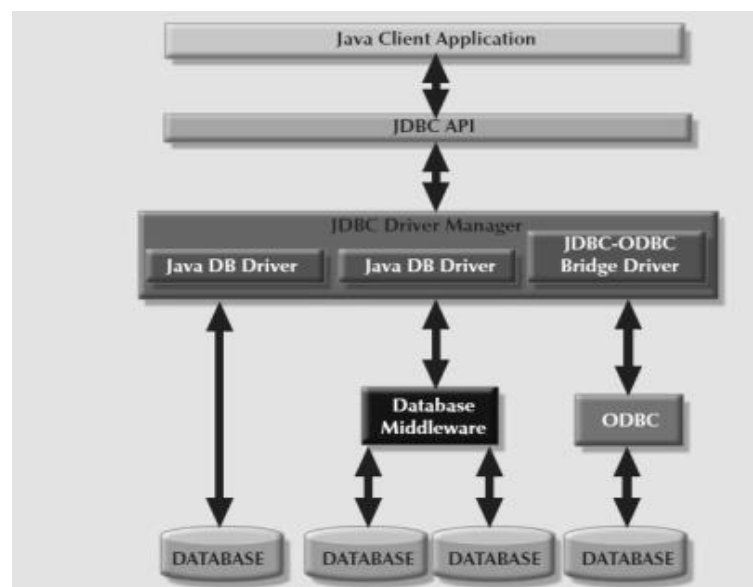


Fig. 13.3 JDBC architecture

Every day, more and more companies are investing resources in developing and expanding their Web presence and finding ways to do more business on the Internet. Such business will

generate increasing amounts of data that will be stored in databases. Java and the .NET framework are part of the trend toward increasing reliance on the Internet as a critical business resource. In fact, the Internet is likely to become the development platform of the future. In the next section you will learn more about Internet databases and how they are used.

13.2 DATABASE INTERNET CONNECTIVITY

Millions of people all over the world access the Internet, connecting to databases via Web browsers or data services (i.e., using a smart phone applet to get weather information). Internet database connectivity opens the door to new innovative services that:

- Permit rapid responses to competitive pressures by bringing new services and products to market quickly.
- Increase customer satisfaction through the creation of Web-based support services.
- Allow anywhere/anytime data access using mobile smart devices via the Internet
- Yield fast and effective information dissemination through universal access from across the street or across the globe.

Given those advantages, many organizations rely on their IS departments to create universal data access architectures based on Internet standards. Table 13.3 shows a sample of Internet technology characteristics and the benefits they provide.

The Internet is rapidly changing the way information is generated, accessed, and distributed. At the core of this change is the Web's ability to access data in databases (local and remote), the simplicity of the interface, and cross-platform (heterogeneous) functionality. The Web has helped create a new information dissemination standard.

INTERNET CHARACTERISTIC	BENEFIT
Hardware and software independence	Savings in equipment/software acquisition Ability to run on most existing equipment Platform independence and portability No need for multiple platform development
Common and simple user interface	Reduced training time and cost Reduced end-user support cost No need for multiple platform development
Location independence	Global access through Internet infrastructure and mobile smart devices Reduced requirements (and costs!) for dedicated connections
Rapid development at manageable costs	Availability of multiple development tools Plug-and-play development tools (open standards) More interactive development Reduced development times Relatively inexpensive tools Free client access tools (Web browsers) Low entry costs. Frequent availability of free Web servers Reduced costs of maintaining private networks Distributed processing and scalability, using multiple servers

Table 13.3 Characteristics and Benefits of Internet Technologies

The following sections examine how Web-to-database middleware enables end users to interact with databases over the Web.

Web-to-Database Middleware: Server-Side Extensions

In general, the Web server is the main hub through which all Internet services are accessed. For example, when an end user uses a Web browser to dynamically query a database, the client browser requests a Web page. When the Web server receives the page request, it looks for the page on the hard disk; when it finds the page (for example, a stock quote, product catalog information, or an airfare listing), the server sends it back to the client. Dynamic Web pages are at the heart of current Web sites. In this database-query scenario, the Web server generates the Web page contents before it sends the page to the client Web browser. The only problem with the preceding query scenario is that the Web server must include the database query result on the page before it sends that page back to the client. Unfortunately, neither the Web browser nor the Web server knows how to connect to and read data from the database. Therefore, to support this type of request (database query), the Web server's capability must be extended so that it can understand and process database requests. This job is done through a server-side extension.

A server-side extension is a program that interacts directly with the Web server to handle specific types of requests. In the preceding database query example, the server-side extension

program retrieves the data from databases and passes the retrieved data to the Web server, which, in turn, sends the data to the client's browser for display purposes.

The server-side extension makes it possible to retrieve and present the query results, but what's more important is that it provides its services to the Web server in a way that is totally transparent to the client browser. In short, the server-side extension adds significant functionality to the Web server, and therefore, to the Internet.

A database server-side extension program is also known as Web-to-database middleware. Figure 13.8 shows the interaction between the browser, the Web server, and the Web-to-database middleware.

Trace the Web-to-database middleware actions in Figure 13.8:

1. The client browser sends a page request to the Web server.
2. The Web server receives and validates the request. In this case, the server will pass the request to the Web-to-database middleware for processing. Generally, the requested page contains some type of scripting language to enable the database interaction.
3. The Web-to-database middleware reads, validates, and executes the script. In this case, it connects to the database and passes the query using the database connectivity layer.
4. The database server executes the query and passes the result back to the Web-to-database middleware.
5. The Web-to-database middleware compiles the result set, dynamically generates an HTML-formatted page that includes the data retrieved from the database, and sends it to the Web server.
6. The Web server returns the just-created HTML page, which now includes the query result, to the client browser.
7. The client browser displays the page on the local computer.

The interaction between the Web server and the Web-to-database middleware is crucial to the development of a successful Internet database implementation. Therefore, the middleware must be well integrated with the other Internet services and the components that are involved in its use.

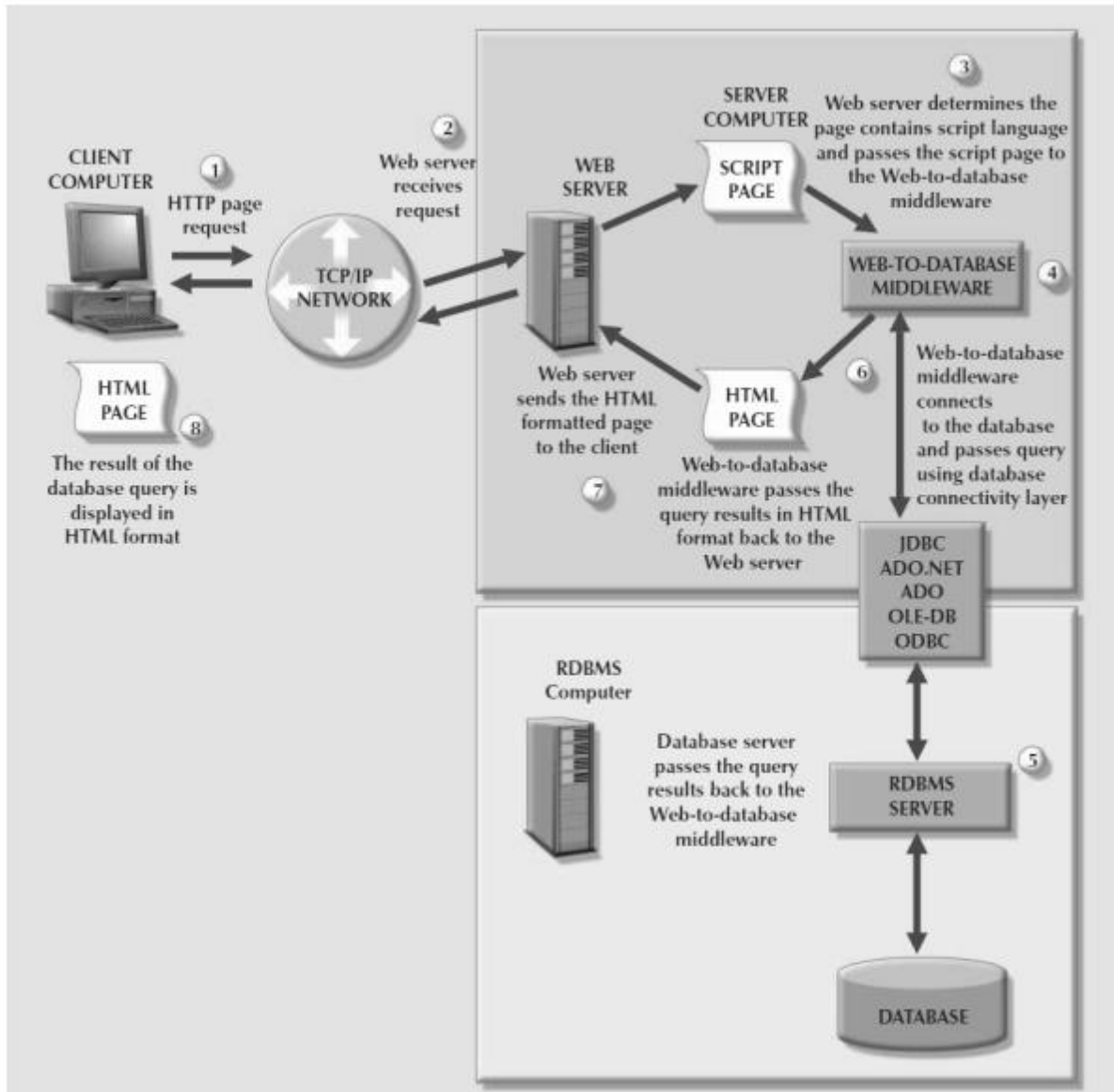


Fig 13,8 Web to Database middleware

Web Server Interfaces

Extending Web server functionality implies that the Web server and the Web-to-database middleware will properly communicate with each other. (Database professionals often use the word interoperate to indicate that each party can respond to the communications of the other. This book's use of communicate assumes interoperation.) If a Web server is to communicate successfully with an external program, both programs must use a standard way to exchange messages and to respond to requests. A Web server interface defines how a Web server communicates with external programs. Currently, there are two well-defined Web server interfaces:

- Common Gateway Interface (CGI).
- Application programming interface (API).

The Common Gateway Interface (CGI) uses script files that perform specific functions based on the client's parameters that are passed to the Web server. The script file is a small program containing commands written in a programming language—usually Perl, C#, or Visual Basic. The script file's contents can be used to connect to the database and to retrieve data from it, using the parameters passed by the Web server. Next, the script converts the retrieved data to HTML format and passes the data to the Web server, which sends the HTML-formatted page to the client.

The main disadvantage of using CGI scripts is that the script file is an external program that is individually executed for each user request. That scenario decreases system performance.

An application programming interface (API) is a newer Web server interface standard that is more efficient and faster than a CGI script. APIs are more efficient because they are implemented as shared code or as dynamic-link libraries (DLLs). That means the API is treated as part of the Web server program that is dynamically invoked when needed. APIs are faster than CGI scripts because the code resides in memory, so there is no need to run an external program for each request. Instead, the same API serves all requests. Another advantage is that an API can use a shared connection to the database instead of creating a new one every time, as is the case with CGI scripts.

Although APIs are more efficient in handling requests, they have some disadvantages. Because the APIs share the same memory space as the Web server, an API error can bring down the server. The other disadvantage is that APIs are specific to the Web server and to the operating system.

The Web Browser

The Web browser is the application software in the client computer, such as Microsoft Internet Explorer, Apple Safari, or Mozilla Firefox, that lets end users navigate (browse) the Web. Each time the end user clicks a hyperlink, the browser generates an HTTP GET page request that is sent to the designated Web server, using the TCP/IP Internet protocol.

The Web browser's job is to interpret the HTML code that it receives from the Web server and to present the various page components in a standard formatted way.

Client-Side Extensions

Client-side extensions add functionality to the Web browser. Although client-side extensions are available in various forms, the most commonly encountered extensions are:

- Plug-ins.
- Java and JavaScript.
- ActiveX and VBScript.

A plug-in is an external application that is automatically invoked by the browser when needed. Because it is an external application, the plug-in is operating-system-specific. The plug-in is associated with a data object—generally using the file extension—to allow the Web server to properly handle data that are not originally supported.

As noted earlier, Java runs on top of the Web browser software. Java applications are compiled and stored in the Web server. (In many respects, Java resembles C++.) Calls to Java routines are embedded inside the HTML page. When the browser finds this call, it downloads the Java classes (code) from the Web server and runs that code in the client computer. Java's main advantage is that it enables application developers to develop their applications once and run them in many environments.

JavaScript is a scripting language (one that enables the running of a series of commands or macros) that allows Web authors to design interactive sites. Because JavaScript is simpler to generate than Java, it is easier to learn. JavaScript code is embedded in the Web pages. It is downloaded with the Web page and is activated when a specific event takes place—such as a mouse click on an object or a page being loaded from the server into memory.

ActiveX is Microsoft's alternative to Java. ActiveX is a specification for writing programs that will run inside the Microsoft client browser (Internet Explorer). Because ActiveX is oriented mainly toward Windows applications, it has low portability.

VBScript is another Microsoft product that is used to extend browser functionality. VBScript is derived from Microsoft Visual Basic. Like JavaScript, VBScript code is embedded inside an HTML page and is activated by triggering events such as clicking a link.

A Web application server is a middleware application that expands the functionality of Web servers by linking them to a wide range of services, such as databases, directory systems, and search engines. The Web application server also provides a consistent run-time environment for Web applications.

13.3 Extensible Markup Language (XML)

The Internet has brought about new technologies that facilitate the exchange of business data among business partners and consumers. Companies are using the Internet to create new types of systems that integrate their data to increase efficiency and reduce costs. Electronic commerce (e-commerce) enables all types of organizations to market and sell products and services to a global market of millions of users. E-commerce transactions—the sale of products or services—can take place between businesses (business-to-business, or B2B) or between a business and a consumer (business-to-consumer, or B2C).

Most e-commerce transactions take place between businesses. Because B2B e-commerce integrates business processes among companies, it requires the transfer of business information among different business entities. But the way in which businesses represent, identify, and use data tends to differ substantially from company to company (“product code” vs. “item ID”).

Until recently, the expectation was that a purchase order traveling over the Web would be in the form of an HTML document. The HTML Web page displayed on the Web browser would include formatting as well as the order details.

If an application wants to get the order data from the Web page, there is no easy way to extract the order details (such as the order number, the date, the customer number, the item, the quantity, the price, or payment details) from an HTML document. The HTML document can only describe how to display the order in a Web browser; it does not permit the manipulation of the order’s data elements, that is, date, shipping information, payment details, product information, and so on. To solve that problem, a new markup language, known as Extensible Markup Language, or XML, was developed.

Extensible Markup Language (XML) is a meta-language used to represent and manipulate data elements. XML is designed to facilitate the exchange of structured documents, such as orders and invoices, over the Internet. The World Wide Web Consortium (W3C)¹ published the first XML 1.0 standard definition in 1998. That standard set the stage for giving XML the real-world appeal of being a true vendor-independent platform. Therefore, it is not surprising that XML has rapidly become the data exchange standard for e-commerce applications.

The XML metalanguage allows the definition of news, such as <ProdPrice>, to describe the data elements used in an XML document. This ability to extend the language explains the X in XML; the language is said to be extensible.

Just like HTML, an XML document is a text file. However, it has a few very important additional characteristics, as follows:

- XML allows the definition of new tags to describe data elements, such as <ProductId>.
- XML is case sensitive: <ProductID> is not the same as <Productid>.
- XMLs must be well formed; that is, tags must be properly formatted. Most openings also have a corresponding closing. For example, the product identification would require the format <ProductId>2345-AA</ProductId>.
- XMLs must be properly nested. For example, a properly nested XML might look like this:

```
<Product><ProductId>2345-AA</ProductId></Product>.
```

- You can use the <-- and --> symbols to enter comments in the XML document.
- The XML and xml prefixes are reserved for XMLs only.

XML is not a new version or replacement for HTML. XML is concerned with the description and representation of the data, rather than the way the data are displayed. XML provides the semantics that facilitate the sharing, exchange, and manipulation of structured documents over organizational boundaries. XML and HTML perform complementary, rather than overlapping, functions. Extensible Hypertext Markup Language (XHTML) is the next generation of HTML based on the XML framework. The XHTML specification expands the HTML standard to include XML features.

Although more powerful than HTML, XHTML requires very strict adherence to syntax requirements. The XML schema is an advanced data definition language that is used to describe the structure (elements, data types, relationship types, ranges, and default values) of XML data documents. One of the main advantages of an XML schema is that it more closely maps to database terminology and features. For example, an XML schema will be able to define common database types such as date, integer, or decimal; minimum and maximum values; a list of valid values; and required elements. Using the XML schema, a company would be able to validate the data for values that may be out of range, incorrect dates, valid values, and so on.

XML Presentation

One of the main benefits of XML is that it separates data structure from its presentation and processing. By separating data and presentation, you are able to present the same data in different ways—which is similar to having views in SQL.

This section will list some of the uses of XML. Keep in mind that the future use of XML is limited only by the imagination and creativity of the developers, designers, and programmers.

- B2B exchanges. As noted earlier, XML enables the exchange of B2B data, providing the standard for all organizations that need to exchange data with partners, competitors, the government, or customers.
- Legacy systems integration. XML provides the “glue” to integrate legacy system data with modern e-commerce Web systems. Web and XML technologies could be used to inject some new life in “old but trusted” legacy applications. Another example is the use of XML to import transaction data from multiple operational databases to a data warehouse database.
- Web page development. XML provides several features that make it a good fit for certain Web development scenarios.
- Database support. Databases are at the heart of e-commerce applications. A DBMS that supports XML exchanges will be able to integrate with external systems (Web, mobile data, legacy systems, and so on) and thus enable the creation of new types of systems. These databases can import or export data in XML format or generate XML documents from SQL queries while still storing the data, using their native data model format.
- Database meta-dictionaries. XML can also be used to create meta-dictionaries, or vocabularies, for databases. These meta-dictionaries can be used by applications that need to access other external data sources. (Until now, each time an application wanted to exchange data with another application, a new interface had to be built for that purpose.) DBMS vendors can publish meta-dictionaries to facilitate data exchanges and the creation of data views from multiple applications—hierarchical, relational, object-oriented, object-relational, or extended relational.
- XML databases. Given the huge number of expected XML-based data exchanges, businesses are already looking for ways to better manage and utilize the data. Currently, many different products are on the market to address this problem. The approaches range from simple middleware XML software, to object databases with

XML interfaces, to full XML database engines and servers. The current generation of relational databases is tuned for the storage of normalized rows—that is, manipulating one row of data at a time. Because business data do not always conform to such a requirement, XML databases provide for the storage of data in complex relationships. For example, an XML database would be well suited to store the contents of a book. (The book’s structure would dictate its database structure: a book typically consists of chapters, sections, paragraphs, figures, charts, footnotes, endnotes, and so on.) Examples of XML databases are Oracle, IBM DB2, MS SQL Server, Ipedo XML Database (www.ipedo.com), Tamino from Software AG (www.softwareag.com), and the open source dbXML from <http://sourceforge.net/projects/dbxml-core>.

- XML services. Many companies are already working on the development of a new breed of services based on XML and Web technologies. These services promise to break down the interoperability barriers among systems and companies alike. XML provides the infrastructure that facilitates heterogeneous systems to work together across the desk, the street, and the world. Services would use XML and other Internet technologies to publish their interfaces. Other services, wanting to interact with existing services, would locate them and learn their vocabulary (service request and replies) to establish a “conversation.”

13.4 CHECK YOUR PROGRESS

1. What is Universal Data Access (UDA) architecture?
2. What is ODBC?
3. What is DAO?
4. What is Java Database Connectivity?
5. What is JavaScript ?
6. What is a Web application server?
7. What is Extensible Markup Language (XML)?
8. What is XML schema?
9. What is mapping constraint?
10. What is an integrity constraint?

Answers to check your progress:

1. Microsoft's Universal Data Access (UDA) architecture, a collection of technologies used to access any type of data source and manage the data through a common interface.
2. Open Database Connectivity (ODBC) is Microsoft's implementation of a superset of the SQL Access Group Call Level Interface (CLI) standard for database access.
3. Data Access Objects (DAO) is an object-oriented API used to access MS Access, MS FoxPro, and dBase databases (using the Jet data engine) from Visual Basic programs
4. Java Database Connectivity (JDBC) is an application programming interface that allows a Java program to interact with a wide range of data sources (relational databases, tabular data sources, spreadsheets, and text files). JDBC allows a Java program to establish a connection with a data source, prepare and send the SQL code to the database server, and process the result set.
5. JavaScript is a scripting language (one that enables the running of a series of commands or macros) that allows Web authors to design interactive sites. Because JavaScript is simpler to generate than Java, it is easier to learn.
6. A Web application server is a middleware application that expands the functionality of Web servers by linking them to a wide range of services, such as databases, directory systems, and search engines.
7. Extensible Markup Language (XML) is a meta-language used to represent and manipulate data elements. XML is designed to facilitate the exchange of structured documents, such as orders and invoices, over the Internet.
8. The XML schema is an advanced data definition language that is used to describe the structure (elements, data types, relationship types, ranges, and default values) of XML data documents

13.5 SUMMARY

• Database connectivity refers to the mechanisms through which application programs connect and communicate with data repositories. Database connectivity software is also known as database middleware. The data repository is also known as the data source because it represents the data management application (that is, an Oracle RDBMS, SQL Server

DBMS, or IBM DBMS) that will be used to store the data generated by the application program.

- Microsoft database connectivity interfaces are dominant players in the market and enjoy the support of most database vendors. In fact, ODBC, OLE-DB, and ADO.NET form the backbone of Microsoft's Universal Data Access (UDA) architecture. UDA is a collection of technologies used to access any type of data source and manage any type of data, using a common interface.
- Based on Microsoft's Component Object Model (COM), Object Linking and Embedding for Database (OLE-DB) is database middleware developed with the goal of adding object-oriented functionality for access to relational and nonrelational data. ActiveX Data Objects (ADO) provides a high-level application-oriented interface to interact with OLE-DB, DAO, and RDO. Based on ADO, ADO.NET is the data access component of Microsoft's .NET application development framework, a component-based platform for developing distributed, heterogeneous, interoperable applications aimed at manipulating any type of data over any network under any operating system and any programming language. Java Database Connectivity (JDBC) is the standard way to interface Java applications with data sources (relational, tabular, and text files).
- Database access through the Web is achieved through middleware. To improve the capabilities on the client side of the Web browser, you must use plug-ins and other client-side extensions such as Java and JavaScript, or ActiveX and VBScript. On the server side, Web application servers are middleware that expands the functionality of Web servers by linking them to a wide range of services, such as databases, directory systems, and search engines.
- Extensible Markup Language (XML) facilitates the exchange of B2B and other data over the Internet. XML provides the semantics that facilitates the exchange, sharing, and manipulation of structured documents across organizational boundaries. XML produces the description and the representation of data, thus setting the stage for data manipulation in ways that were not possible before XML. XML documents can be validated through the use of Document Type Definition (DTD) documents and XML schema definition (XSD) documents. The use of DTD, XML schemas, and XML documents permits greater level of integration among diverse systems than was possible before this technology was made available.

13.6 KEYWORDS

- **API** - An application programming interface (API) is a newer Web server interface standard that is more efficient and faster than a CGI script
- **CGI** - The Common Gateway Interface (CGI) uses script files that perform specific functions based on the client's parameters that are passed to the Web server. relationship.
- **Database middleware** - Database connectivity software is also known as database middleware.
- **Extensible Markup Language (XML)** - is a meta-language used to represent and manipulate data elements.
- **Java Database Connectivity (JDBC)** - is the standard way to interface Java applications with data sources (relational, tabular, and text files).
- **Open Database Connectivity (ODBC)** - is Microsoft's implementation of a superset of the SQL Access Group Call Level Interface (CLI) standard for database access.

13.7 QUESTIONS FOR SELF STUDY

1. Give some examples of database connectivity options and what they are used for.
2. What are the three basic components of the ODBC architecture? Explain.
3. What are Web server interfaces used for? Give some examples.
4. What is XML, and why is it important?
5. Explain two webserver interfaces.

13.8 REFERENCES

1. Coronel, C., & Morris, S. (2016). Database systems: design, implementation, & management. Cengage Learning.
2. Gupta, S. B., & Mittal, A. (2009). Introduction to database management system. Laxmi Publications, Ltd..
3. Ramakrishnan, R., & Gehrke, J. (2000). Database management systems. McGraw-Hill.

UNIT-14: DATABASE SECURITY AND AUTHORIZATION

STRUCTURE

- 14.0 Objectives
- 14.1 Introduction
- 14.2 Security Violations
- 14.3 Authorization (Access Rights)
- 14.4 Views
- 14.5 Complement Granting of Privileges
- 14.6 Notion of Roles (Grouping of Users)
- 14.7 Audit Trails
- 14.8 Check your progress
- 14.9 Summary
- 14.10 Keywords
- 14.11 Questions for self-study
- 14.12 References

14.0 OBJECTIVES

After studying this unit, you will be able to:

- ✓ Discuss about various threats to database security
- ✓ Provide protection against unauthorized access
- ✓ Provide a personalized model of database to a particular user
- ✓ Describe granting of privileges and grouping of users
- ✓ Explain about audit trails

14.1 INTRODUCTION

Database security is an important issue in Database Management System (DBMS) because of the importance and sensitivity of data and information of an enterprise. Therefore, the data in the database system need to be protected from unauthorized access and corruption. Database security allows or disallows users from performing actions on the objects contained within the database. Almost all DBMS's provide discretionary access control to regulate all user access to the objects through privileges. A privilege is permission to access the objects in a well-defined manner.

The goal of database security is the protection of data against threats such as accidental or intentional loss, misuse or corruption by unauthorized users. The DBA is responsible for the overall security of the database system. Therefore, the DBA must identify the most serious threats to the database and accordingly develop overall policies, procedures and appropriate controls to protect the databases.

In this unit, various threats to database security, protection against unauthorized access and other security mechanisms have been discussed.

14.2 SECURITY VIOLATIONS

Three types of security violations are:

- Unauthorized modification of data in database.
- Unauthorized deletion of data in database.
- Unauthorized reading of data in database.

To protect the database from these unauthorized access or security violations, the security measures at following levels must be taken:

- Database system: Different access rights to different users can be given so that they can use data which they really want. Now, the database is responsible to maintain these restrictions and security.
- Operating system: Operating System must be secured to unauthorized access. Only a secured database system cannot maintain security.
- Network: Systems or Computers are connected through LAN's, internet etc. and data can be shared through these networks. So, security within the network software is important.
- Human factors: Authorization must be given carefully to reduce human errors.
- Physical security: Servers and computer systems must be secured physically from looting, fire, failure of systems like disasters.
- Security measures at these different levels must be taken to make database secure.

14.3 AUTHORIZATION (ACCESS RIGHTS)

Security can be maintained by giving different authorities to different users according to their work. Different authorizations are:

- Read access: It allows only reading of data. User cannot modify, delete, alter or update the data.
- Update access: It allows only updation of data. User cannot delete data.
- Insert access: It allows addition of new data. User cannot modify and delete the existing data.
- Delete access: It allows deletion of data. User cannot modify the existing data.
- Index access: It allows the creation and deletion of indices.
- Alteration access: It allows the addition or deletion of attributes in a relation.
- Resource access: It allows the creation of new relations.
- Drop access: It allows the deletion of relations.

Difference between Delete Access and Drop Access

A user having delete access can only delete data but not relations. Even if user delete all the data, relation exists. While if relation will be deleted nothing remains.

Ex. Consider the Student Information System.

Student Information System

Reg. No.	Roll No.	Name	Class
001 A	1	Vikas	Commerce
002 A	2	Amit	Computers
003 A	3	Lalit	Mechanical

[Table Name is
Student]

Fig 14.1 Student information system

After deletion of all the data, relation or table exists as shown in fig 14.1

Reg. No.	Roll No.	Name	Class
----------	----------	------	-------

But drop command deletes the table completely with its data.

Ex. Drop table student.

This command deletes the whole table named Student. By giving above authorization, DBA maintains security in database and avoid unauthorized accesses.

14.4 VIEWS

This facility provides a personalized model of database to a particular user. By using this facility, DBA can hide data from user that is not required by the user.

The main advantages of views are:

1. Simplify the system usage : User can only see the data of its own interest on which he really wants to do work. This is also helpful in optimization of resources.
2. Limits user access of data : User can see only a particular part of database. If a user cannot see other data then how he can violate the security rules. A good mixture of authorization and view helps in maintaining a secure server.

14.5 GRANTING OF PRIVILEGES

A user can grant his access rights (authorizations) to any other user only when DBA gives granting right to that user.

A user has an authorization if there exists a path from root (DBA) to the node, that represent the user.

Consider an example of delete authorization.



Fig. 14.2 Delete authorization for user 5

User 5 has two paths by which he can get delete authorization

1. DBA → User 2 → User 4 → User 5
2. DBA → User 1 → User 5

After sometime if DBA revoked that authority from user 1, then user 5 can use 1st path

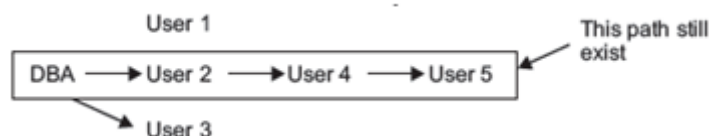


Fig. 14.3 Delete authorization for user 5 after DBA revoked delete authority from user 1

Advantage: Helps in maintaining large database.

Disadvantage: Users can attempt to defeat authorization revocation.

Ex. Consider the authorization's as given in Figure 14.4.

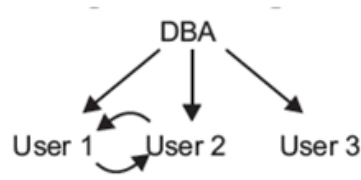


Fig. 14.4 User authorizations

Here DBA gives authority to user1,2 and 3. User 2 grants authority to user 1 and vice versa. At any time if DBA revokes authority from user 2 then even it has authority by user 1

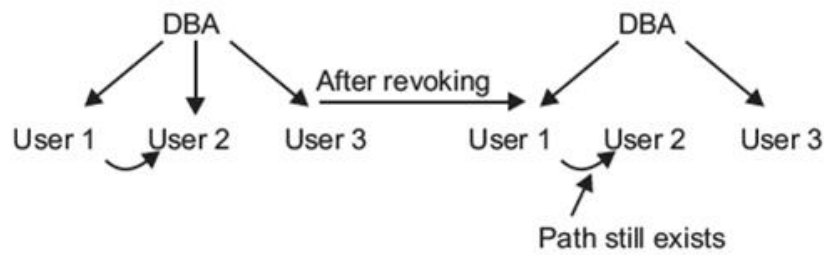


Fig. 14.5 Modified user authorization

14.6 NOTION OF ROLES (GROUPING OF USERS)

DBA gives different rights to different users. Suppose there are many users having same access rights. Then, it is beneficial to group these users and give authorization (rights) to this particular group. If some new user joins the company, then it will be added to any group.

Ex. Consider an ABC software company as shown in Figure 14.6.

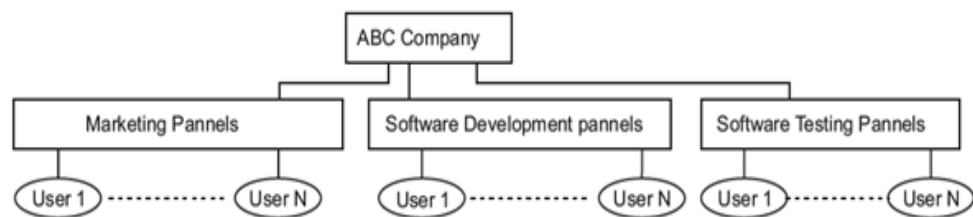


Fig 14.6. Notion of roles in software company

The DBA makes three different groups (roles) of users :

1. Marketing
2. Software Development
3. Software Testing.

The DBA gives rights or authorization to these groups but not to individual users. After that users are added to these different groups. When a user is added to any group, then automatically authorizations given to that group will be granted to that user.

The advantages of notion of roles are:

1. Ease the work of DBA.
2. Resources can be optimized.

The disadvantages of notion of roles are:

1. It would not be possible to identify which transaction is carried out by which user.

This will lead to security risks.

14.7 AUDIT TRAILS

To overcome the disadvantages of notion of roles, Audit trails can be maintained. An audit trail is a time-stamped record of significant activities on a system. Recorded events can include user logins and logouts to the system, as well as what commands were issued by the user to the system while logged in. Audit trails helps in detecting security violations, performance problems, and flaws in applications.

Audit trails keep a record of data accesses i.e. logging file creation, reading, updating and deleting for each user. Uses of system resources may also be logged, such as printing of files or copying data from one storage location to another. Unsuccessful access attempts may also be tracked.

An audit trail keeps track of who did what, to what, and when they did it, as well as who tried to do something but was unsuccessful. A computer system may have several audit trails, each devoted to a particular type of activity.

Audit trails may be used as either a support for regular system operations or a kind of insurance policy or as both of these. As insurance, audit trails are maintained but are not used unless needed, such as after a system outage. As a support for operations, audit trails are used to help system administrators ensure that the system or resources have not been harmed by hackers, insiders, or technical problems.

Uses of Audit Trails: Audit trails are a fundamental part of computer security, particularly useful for tracing unauthorized users and uses. They can also be used to assist with information recovery in the event of a system failure.

14.7.1 Advantages of Audit Trails

Audit trails help to accomplish many security-related issues such as individual accountability, reconstruction of events, intrusion detection, and problem analysis.

1. **Individual Accountability:** Audit trails are a technical mechanism that helps managers maintain individual accountability. By advising users that they are personally accountable for their actions, which are tracked by an audit trail that logs user activities, managers can help promote proper user behavior. Users are less likely to attempt to evade security policy if they know that their actions will be recorded in an audit log.
2. **Reconstruction of Events:** Audit trails can also be used to reconstruct events after a problem has occurred. Damage can be more easily assessed by reviewing audit trails of system activity to pinpoint how, when, and why normal operations ceased. Audit trail analysis can often distinguish between operator-induced errors or system-created errors.
3. **Intrusion Detection:** Intrusion detection refers to the process of identifying attempts to penetrate a system and gain unauthorized access. If audit trails have been designed and implemented to record appropriate information, they can assist in intrusion detection. Intrusions can be detected in real time, by examining audit records as they are created or after the fact.
4. **Problem Analysis:** Audit trails may also be used as on-line tools to help identify problems other than intrusions as they occur. This is often referred to as real-time auditing or monitoring. If a system or application is deemed to be critical to an organization's business or mission, real-time auditing may be implemented to monitor the status of these processes.

14.7.2 Audit Trails and Logs

A system can maintain several different audit trails concurrently. There are typically two kinds of audit records

- a) **An event-oriented log.** An audit trail should include sufficient information to establish what events occurred and who or what caused them. In general, an event record should specify when the event occurred, the user ID associated with the event, the program or command used to initiate the event, and the result. Date and time can help determine if the user was a masquerader or the actual person specified. Event based

logs usually contain records describing system events, application events, or user events.

- b) **Keystroke Monitoring.** It is also called record of every keystroke. Keystroke monitoring is the process used to view or record both the keystrokes entered by a computer user and the computer's response during an interactive session. Keystroke monitoring is usually considered a special case of audit trails. Examples of keystroke monitoring would include viewing characters as they are typed by users, reading users' electronic mail, and viewing other recorded information typed by users.

Keystroke monitoring is conducted in an effort to protect systems and data from intruders who access the systems without authority or in excess of their assigned authority. Monitoring keystrokes typed by intruders can help administrators assess and repair damage caused by intruders.

14.7.3 Review of Audit Trails

Audit trails can be used to review what occurred after an event, for periodic reviews, and for real-time analysis. Reviewers should know what to look for to be effective in spotting unusual activity. They need to understand what normal activity looks like. Audit trail review can be easier if the audit trail function can be queried by user ID, terminal ID, application name, date and time, or some other set of parameters to run reports of selected information.

There are many types of reviews. Some of them are as follows:

- a) **Audit Trail Review after an Event:** Following a known system or application software problem, a known violation of existing requirements by a user, or some unexplained system or user problem, the appropriate system-level or application-level administrator should review the audit trails. Review by the application/data owner would normally involve a separate report, based upon audit trail data, to determine if their resources are being misused.
- b) **Periodic Review of Audit Trail Data:** The persons associated with the security of data such as system administrators, function managers, and computer security managers should determine how much review of audit trail records is necessary, based on the importance of identifying unauthorized activities. This determination should have a direct correlation to the frequency of periodic reviews of audit trail data.

- c) Real-Time Audit Analysis: Traditionally, audit trails are analyzed in a batch mode at regular intervals e.g., daily. Audit records are archived during that interval for later analysis. Audit analysis tools can also be used.

14.8 CHECK YOUR PROGRESS

1. The DBA is responsible for the overall security of the database system.
2. Database security refers to protection from malicious access.
3. Which of the following is the process by which a user's privileges ascertained?
(a) Authorization (b) Authentication (c) Access control (d) None of these
4. Data security threats include
(a) Privacy invasion (b) Hardware failure (c) Fraudulent manipulation of data (d) Encryption and decryption
5. _____ security mechanisms are used to grant privileges to users.
6. The security of authorization information for each relation will be kept by _____
7. What is database Security?
8. Name various security violations in databases.
7. What is the degree of relationship set?
8. What are three types of data relationships?

Answers to check your progress:

1. T
2. T
3. a
4. a
5. Discretionary
6. Data Dictionary
7. Database security refers to protection from malicious access.
8. The types of data relationships are:
 - Unauthorized modification of data in database.
 - Unauthorized deletion of data in database.
 - Unauthorized reading of data in database.

14.9 SUMMARY

Database security is an important issue in Database Management System (DBMS) because of the importance and sensitivity of data and information of an enterprise. To protect the database from these unauthorized access or security violations, the security measures at following levels must be taken: *Database system, Operating system, Network, Human factors, Physical security*. Security can be maintained by giving different authorities to different users according to their work. Different authorizations are *Read access, Update access, Insert access, Delete access, Index access, Alteration access, Resource access, Drop access*.

View facility provides a personalized model of database to a particular user. By using this facility, DBA can hide data from user that is not required by the user. A user can grant his access rights (authorizations) to any other user only when DBA gives granting right to that user. A user has an authorization if there exists a path from root (DBA) to the node that represents the user.

DBA gives different rights to different users. Suppose there are many users having same access rights. Then, it is beneficial to group these users and give authorization (rights) to this particular group. To overcome the disadvantages of notion of roles, Audit trails can be maintained. An audit trail is a time-stamped record of significant activities on a system. Recorded events can include user logins and logouts to the system, as well as what commands were issued by the user to the system while logged in. Audit trails helps in detecting security violations, performance problems, and flaws in applications.

14.10 KEYWORDS

- **Audit trail** - audit trail is a time-stamped record of significant activities on a system
- **DBA** – Data Base Administrator.
- **Views** - This facility provides a personalized model of database to a particular user.
- **Database security** : It refers to the various measures organizations take to ensure their databases are protected from internal and external threats.
- **Database protection**: Database security includes protecting the database itself, the data it contains, its database management system, and the various applications that access it.

14.11 QUESTIONS FOR SELF STUDY

1. Discuss concepts of security.
2. What do you mean by security of database? Explain the various ways of database security and recovery procedures in brief.
3. Explain the method for controlling the user access to database for security
4. Explain various database security methods.
5. Assuming that you are the data security administrator of a public sector bank, what are the different security and privacy measures that you will propose for its customer's data?

14.12 REFERENCES

1. Coronel, C., & Morris, S. (2016). *Database systems: design, implementation, & management*. Cengage Learning.
2. Gupta, S. B., & Mittal, A. (2009). *Introduction to database management system*. Laxmi Publications, Ltd..
3. Ramakrishnan, R., & Gehrke, J. (2000). *Database management systems*. McGraw-Hill.

UNIT-15: DATABASE RECOVERY SYSTEM

STRUCTURE

- 15.0 Objectives
- 15.1 Introduction
- 15.2 Classification of Failures
- 15.3 Recovery Concept - Log based recovery
- 15.4 Shadow Paging
- 15.5 Check your progress
- 15.7 Summary
- 15.7 Keywords
- 15.8 Questions for self-study
- 15.9 References

15 OBJECTIVES

After studying this unit, you will be able to:

- ✓ Explain about classification of database failures
- ✓ Describe how to recover from failures
- ✓ Discuss the concept shadow paging
- ✓ Differentiate redo and undo operations
- ✓ Explain how exactly log based recovery works

15.1 INTRODUCTION

A Computer system can be failed due to variety of reasons like disk crash, power fluctuation, software error, sabotage and fire or flood at computer site. These failures result in the loss of information. Thus, database must be capable of handling such situations to ensure that the atomicity and durability properties of transactions are preserved. An integral part of a database system is the recovery manager that is responsible for recovering the data. It ensures atomicity by undoing the actions of transactions that do not commit and durability by making sure that all actions of committed transactions survive system crashes and media failures.

The recovery manager deal with a wide variety of database states because it is called during system failures. Furthermore, the database recovery is the process of restoring the database to a consistent state when a failure occurred. In this unit, we will discuss different failures and

database recovery techniques used so that the database system is restored to the most recent consistent state that existed shortly before the time of system failure.

15.2 CLASSIFICATION OF FAILURES

Failure refers to the state when system can no longer continue with its normal execution and that results in loss of information. Different types of failure are as follows:

1. **System crash:** This failure happens due to the bugs in software or by hardware failure etc.
2. **Transaction failure:** This failure happens due to any logical error such as overflow of stack, bad input, data not found, less free space available etc., or by system error such as deadlocks etc.
3. **Disk failure:** This failure happens due to head crash, tearing of tapes, failure during transfer of data etc.

15.3 RECOVERY CONCEPT

Recovery from failure state refers to the method by which system restore its most recent consistent state just before the time of failure. There are several methods by which you can recover database from failure state. These are defined as follows:

15.3.1 Log Based Recovery

In log based recovery system, a log is maintained, in which all the modifications of the database are kept. A log consists of log records. For each activity of database, separate log record is made. Log records are maintained in a serial manner in which different activities have happened. There are various log records. A typical update log record must contain following fields:

- i. Transaction identifier: A unique number given to each transaction.
- ii. Data-item identifier: A unique number given to data item written.
- iii. Date and time of updation.
- iv. Old value: Value of data item before write.
- v. New value: Value of data item after write.

Logs must be written on the non-volatile (stable) storage. In log-based recovery, the following two operations for recovery are required:

- i. Redo: It means, the work of the transactions that completed successfully before crash is to be performed again.

- ii. Undo: It means, all the work done by the transactions that did not complete due to crash is to be undone.

The redo and undo operations must be idempotent. An idempotent operation is that which gives same result, when executed one or more times.

For any transaction T_i , various log records are:

[T_i start] : It records to log when T_i starts execution.

[T_i, A_j]: It records to log when T_i reads data item A_j

[T_i, A_j, V_1, V_2]: It records to log when T_i updates data item A_j , where V_1 refers to old value and V_2 refers to new value of A_j

[T_i Commit]: It records to log when T_i successfully commits

[T_i aborts] : It records to log if T_i aborts

There are two types of log based recovery techniques and are discussed below

15.3.1.1 Recovery Based on Deferred Database Modification

In deferred database modification technique, deferred (stops) all the write operations of any Transaction T_i until it partially commits. It means modify real database after T_i partially commits. All the activities are recorded in log. Log records are used to modify actual database. Suppose a transaction T_i wants to write on data item A_j , then a log record [T_i, A_j, V_1, V_2] is saved in log and it is used to modify database. After actual modification T_i enters in committed state. In this technique, the old value field is not needed.

Consider the example of Banking system. Suppose you want to transfer Rs. 200 from Account A to B in Transaction T_1 and deposit Rs. 200 to Account C in T_2 . The transactions T_1 and T_2 are shown in Figure 15.1.

T_1	T_2
read(A); A = A - 200; write(A); read(B); B = B + 200; write(B);	read(C); C = C + 200; write(C);

Fig. 15.1 Transactions T_1 and T_2

Suppose, the initial values of A, B and C Accounts are Rs. 500, Rs.1,000 and Rs 600 respectively, Various log records for T_1 and T_2 are as shown in Figure 15.2.

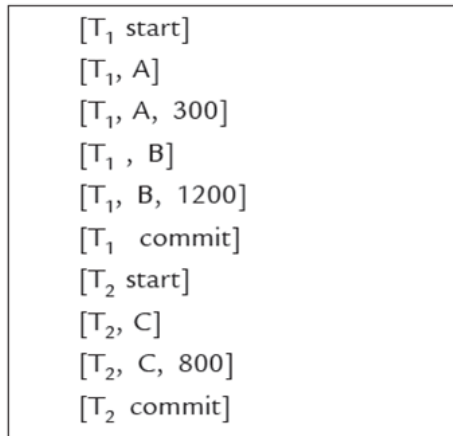


Fig. 15.2 Log records for transaction T1 and T2

For a redo operation, log must contain [Ti start] and [Ti commit] log records.

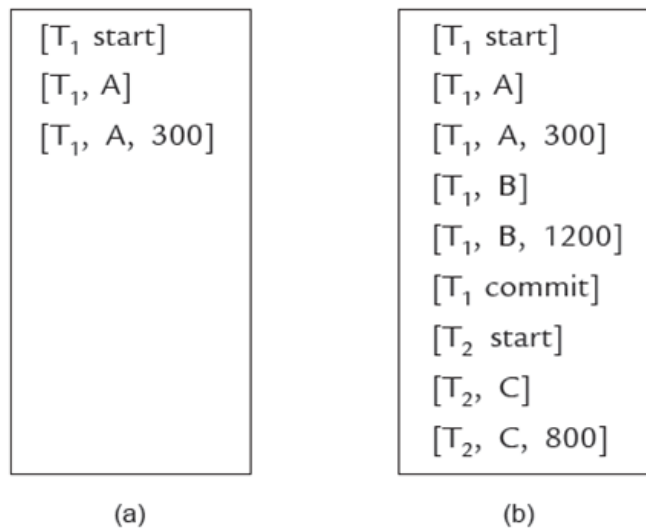


Fig. 15.3 Log of transactions T1 and T2 in case of crash

Crash will happen at any time of execution of transactions. Suppose crash happened

(i) After write (A) of T1: At that time log records in log are shown in Figure 15.3(a). There is no need to redo operation because no commit record appears in the log.

Log records of T1 can be deleted.

(ii) After write (C) of T2 : At that time log records in log are shown in Figure 15.3(b).

In this situation, you have to redo T1 because both [T1 start] and [T1 commit] appears in log. After redo operation, value of A and B are 300 and 1200 respectively. Values remain same because redo is idempotent.

(iii) During recovery: If system is crashed at the time of recovery, simply starts the recovery again.

15.3.1.2 Recovery Based on Immediate Database Modification

In immediate database modification technique, database is modified by any transaction T_i during its active state. It means, real database is modified just after the write operation but after log record is written to stable storage. This is because log records are used during recovery. Use both Undo and Redo operations in this method. Old value field is also needed (for undo operation). Consider again the banking transaction of Figure 15.1. Corresponding log records after successful completion of T_1 and T_2 are shown in Figure 15.4.

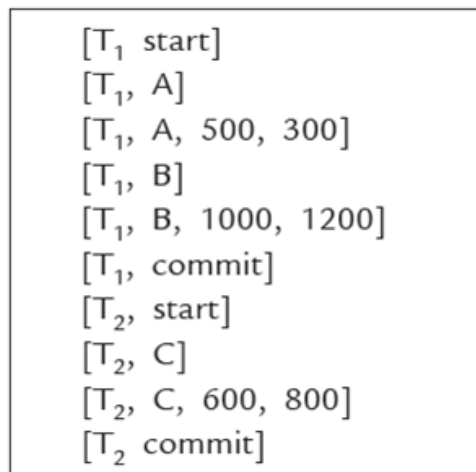


Fig. 15.4 Log records for transactions T_1 and T_2

- For a transaction T_i to be redone, log must contain both $[T_i \text{ start}]$ and $[T_i \text{ commit}]$ records.
- For a transaction T_i to be undone, log must contain only $[T_i \text{ start}]$ record.

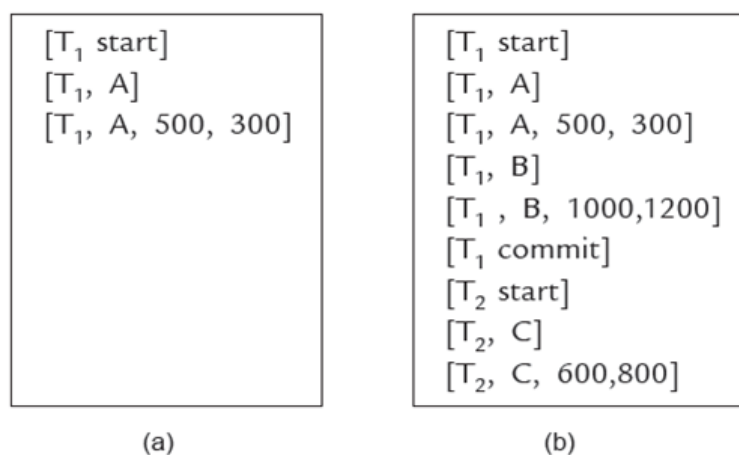


Fig. 15.5 Log of transactions T_1 and T_2 in case of crash

Crash will happen at any time of execution of transaction. Suppose crash happened.

(i) After write (A) of T_1 : At that time log records in log are shown in Figure 15.5(a). Here

only [T1 start] exists so undo transaction T1. As a result, Account A restores its old value 500.

(ii) After write (C) of T2 : At that time log records in log are shown in Figure 15.5(b). During back-up record [T2 start] appears but there is no [T2 commit], so undo transaction T2. As a result, Account C restores its old value 600. When you found both [T1 start] and [T1 commit] records in log, redo transaction T1 and account A and B both keep their new value.

(iii) During recovery: If system is crashed at the time of recovery simply starts recovery again.

15.3.1.3 Checkpoints

Both the techniques discussed earlier ensures recovery from failure state, but they have some disadvantages such as:

- (i) They are time consuming because successfully completed transactions have to be redone.
- (ii) Searching procedure is also time consuming because the whole log has to be searched. So, use checkpoints to reduce overhead. Any of previous recovery techniques can be used with checkpoints. All the transactions completed successfully or having [Ti commit] record before [checkpoint] record need not to be redone. During the time of failure, search the most recent checkpoint. All the transactions completed successfully after checkpoint need to be redone. After searching checkpoint, search the most recent transaction Ti that started execution before that checkpoint but not completed. After searching that transaction, redo/undo transaction as required in applied method.

The major advantages of check points are

1. No need to redo successfully completed transactions before most recent checkpoints.
2. Less searching required.
3. Old records can be deleted.

15.4 SHADOW PAGING

Shadow Paging is an alternative technique for recovery to overcome the disadvantages of log-based recovery techniques. The main idea behind the shadow paging is to keep two page tables in database, one is used for current operations and other is used in case of recovery.

Database is partitioned into fixed length blocks called pages. For memory management, adopt any paging technique.

Page Table

To keep record of each page in database, maintain a page table. Total number of entries in page table is equal to the number of pages in database. Each entry in page table contains a pointer to the physical location of pages.

The two page tables in Shadow Paging are:

- (i) Shadow page table: This table cannot be changed during any transaction.
- (ii) Current page table: This table may be changed during transaction.

Both page tables are identical at the start of transaction. Suppose a transaction T_i performs a write operation on data item V , that resides in page j . The write operation procedure is as follows:

1. If j th page is in main memory then ok otherwise first transfer it from secondary memory to main memory by instruction input (V).
2. Suppose page is used first time then :
 - (a) System first finds a free page on disk and delete its entry from free page list.
 - (b) Then modify the current page table so that it points to the page found in step 2(a).
3. Modify the contents of page or assign new value to V .

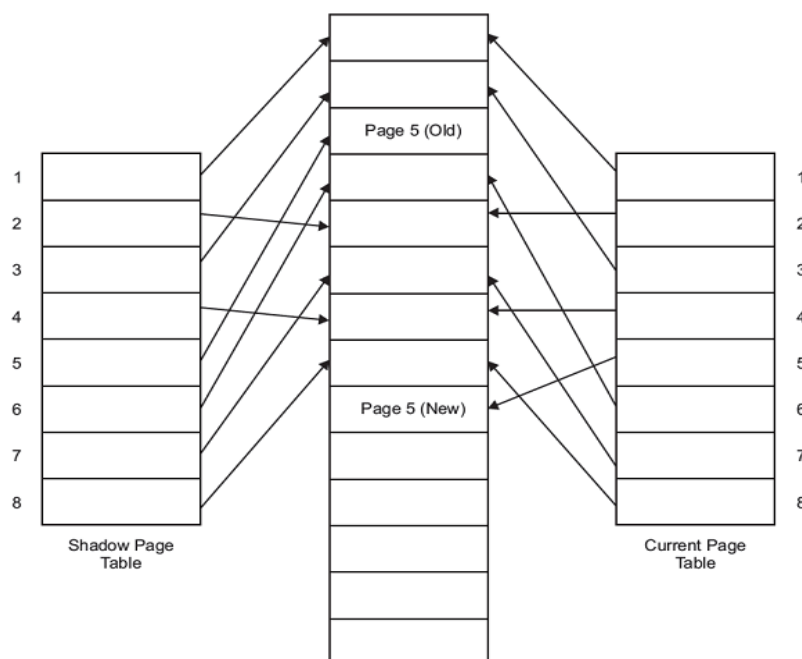


Fig.15.6 Shadow paging

After performing all the above operations, the following steps are needed to commit T_i .

1. Modified pages are transferred from main memory to disk.

2. Save current page table on disk.
3. Change current page table into shadow page table (by changing the physical address).

Shadow page table must be stored in non-volatile or stable storage because it is used at the time of recovery. Suppose the system crashed and you have to recover it. After every successful completion of transaction, current page table is converted into shadow page table.

So, shadow page table has to be searched. For making the search simple, store shadow page table on fixed location of stable storage. No redo operations need to be invoked. Shadow page table and Current page table are shown in Figure 15.6 after earlier write operation.

The major disadvantages of shadow paging are

1. Data fragmentation due to fixed sized pages.
2. Wastage of memory space.
3. Extra overhead at the time of commit (by transferring of page tables)
4. It provides no concurrency.

15.5 CHECK YOUR PROGRESS

1. To optimize a query whose selection clause mentions a view, you'll get the best result if you optimize the view definition and the query separately, and then paste the two of them together..(T/F)
2. Query optimization is particularly important for queries with very long selection clauses (*e.g.*, involving a lot of view definitions)..(T/F)
3. ____ and ____ of the ACID properties are ensured by the recovery system.
4. In shadow paging, ____ table and ____ table are used.
5. In DBMS, deferred update means:
 - (a) All the updates are done first but the entries are made in the log file later.
 - (b) All the log files entries are made first but the actual updates are done later.
 - (c) Every update is done first followed by a writing on the log file.
 - (d) Changes in the views are deferred till a query asks for a view.
6. Immediate updates as a recovery protocol is preferable, when;
 - (a) Database reads more than writes
 - (b) Writes as more than reads
 - (c) It does not matter as it is good in both the situations
 - (d) There are only writes

7. What are the two different approaches for log-based recovery?
8. For log-based recovery with deferred DB modifications: What actions are performed if a transaction is rolled back?
9. In log-based recovery with deferred DB modifications. What actions are required after a rolled back transaction?
10. For log-based recovery with immediate DB modifications. What actions are performed after a crash?

Answers to check your progress

1. F
2. T
3. Atomicity, durability
4. Shadow page, current page
5. b
6. a
7. Deferred DB modifications and immediate DB modifications.
8. No actions need to be done
9. Nothing; the log is ignored.
10. Transaction T needs to be undone if the log contains a (T, start) record but not a (T, commit) record; T needs to be redone if the log contains both a (T, start) record and a (T, commit) record.

15.6 SUMMARY

A Computer system can be failed due to variety of reasons like disk crash, power fluctuation, software error, sabotage and fire or flood at computer site. Failure refers to the state when system can no longer continue with its normal execution and that results in loss of information. Different types of failure are as follows System crash, Transaction failure, Disk failure. Recovery from failure state refers to the method by which system restore its most recent consistent state just before the time of failure. There are several methods by which you can recover database from failure state. Log based recovery and Checkpoints are two popular techniques of database recovery. Shadow Paging is an alternative technique for recovery to overcome the disadvantages of log-based recovery techniques. The main idea behind the shadow paging is to keep two page tables in database, one is used for *current operations* and other is used *in case of recovery*.

15.7 KEYWORDS

- **Disk failure** -This failure happens due to head crash, tearing of tapes, failure during transfer of data etc.
- **Transaction identifier**- A unique number given to each transaction.
- **Data-item identifier**- A unique number given to data item written.
- **Checkpoint** - is a mechanism where all the previous logs are removed from the system and stored permanently in a storage disk.

15.8 QUESTIONS FOR SELF-STUDY

1. Discuss recovery. What is the need of performing recovery ? Discuss the various methods of performing recovery.
2. Compare the deferred and immediate-modification versions of the log-based recovery in terms of overhead and implementation.
3. How can you recover from media failure on which your database was stored ? Describe the mechanism of such recovery.
- 4 Describe the shadow paging recovery scheme along with a diagram. Compare this scheme with the log-based recovery scheme in terms of implementation.
5. What is checkpoint? Why is it needed? What are the actions which are taken by DBMS at a checkpoint? Explain with the help of an example.

15.9 REFERENCES

1. Coronel, C., & Morris, S. (2016). Database systems: design, implementation, & management. Cengage Learning.
2. Gupta, S. B., & Mittal, A. (2009). Introduction to database management system. Laxmi Publications, Ltd..
3. Ramakrishnan, R., &Gehrke, J. (2000). Database management systems. McGraw-Hill.

UNIT -16: DATABASE ADMINISTRATION AND SECURITY

Structure

- 16.0 Objectives
- 16.1 Data as a Corporate Asset
- 16.2 The Need for a Database and Its Role in an Organization
- 16.3 Introduction of a Database: Special Considerations
- 16.4 The Evolution of Database Administration
- 16.5 The Database Environment's Human Component
- 16.6. Security
- 16.7 Database Administration Tools
- 16.8 Developing a Data Administration Strategy
- 16.9 The DBA's Role in the Cloud
- 16.10 The DBA at Work: Using Oracle for Database Administration
- 16.11 Check your progress
- 16.12 Summary
- 16.13 Keywords
- 16.14 Questions for self-study
- 16.15 References

16.0 OBJECTIVES

After studying this unit, you will learn:

- ✓ That data is a valuable business asset requiring careful management
- ✓ How a database plays a critical role in an organization?
- ✓ That the introduction of a DBMS has important technological, managerial, and cultural consequences for an organization
- ✓ What the database administrator's managerial and technical roles are
- ✓ About data security, database security, and the information security framework
- ✓ About several database administration tools and strategies
- ✓ How various database administration technical tasks are performed with Oracle

16.1 DATA AS A CORPORATE ASSET

In unit 1, Database Systems, you learned that data are the raw material from which information is produced. Therefore, it is not surprising that in today's information-driven environment, data are a valuable asset that requires careful management.

To assess data's monetary value, take a look at what's stored in a company database: data about customers, suppliers, inventory, operations, and so on. How many opportunities are lost if the data are lost? What is the actual cost of data loss?

For example, an accounting firm whose entire database is lost would incur significant direct and indirect costs. The accounting firm's problems would be magnified if the data loss occurred during tax season. Data loss puts any company in a difficult position. The company might be unable to handle daily operations effectively, it might be faced with the loss of customers who require quick and efficient service, and it might lose the opportunity to gain new customers. Data are a valuable resource that can translate into information. If the information is accurate and timely, it is likely to trigger actions that enhance the company's competitive position and generate wealth. In effect, an organization is subject to a data-information-decision cycle; that is, the data user applies intelligence to data to produce information that is the basis of knowledge used in decision making by the user. This cycle is illustrated in Figure 16.1.

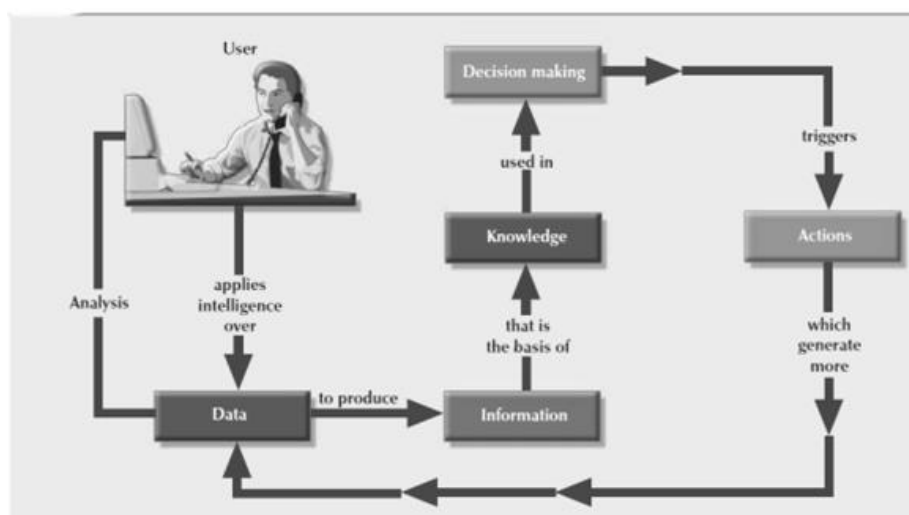


Fig. 16.1 The data-information-decision-making cycle

Note in Figure 16.1 that the decisions made by high-level managers trigger actions within the organization's lower levels. Such actions produce additional data to be used for monitoring company performance. In turn, the additional data must be recycled within the data-

information-decision framework. Thus, data form the basis for decision making, strategic planning, control, and operations monitoring.

A critical success factor of an organization is efficient asset management. To manage data as a corporate asset, managers must understand the value of information—that is, processed data. In fact, there are companies (for example, those that provide credit reports) whose only product is information and whose success is solely a function of information management.

Most organizations continually seek new ways to leverage their data resources to get greater returns. This leverage can take many forms, from data warehouses that support improved customer relationship management to tighter integration with customers and suppliers in support of electronic supply chain management. As organizations become more dependent on information, the accuracy of that information becomes ever more critical. Dirty data, or data that suffer from inaccuracies and inconsistencies, becomes an even greater threat to these organizations. Data can become dirty for many reasons, such as:

- Lack of enforcement of integrity constraints (not null, uniqueness, referential integrity, etc.).
- Data entry typographical errors.
- Use of synonyms and/or homonyms across systems.
- Non standardized use of abbreviations in character data.
- Different decompositions of composite attributes into simple attributes across systems.

Some causes of dirty data can be addressed at the individual database level, such as the proper implementation of constraints. However, addressing other causes of dirty data is more complicated. Some sources of dirty data come from the movement of data across systems, as in the creation of a data warehouse. Efforts to control dirty data are generally referred to as data quality initiatives.

Data quality is a comprehensive approach to ensuring the accuracy, validity, and timeliness of the data. The idea that data quality is comprehensive is important. Data quality is concerned with more than just cleaning dirty data; it also focuses on the prevention of future inaccuracies in the data, and building user confidence in the data. Large-scale data quality initiatives tend to be complex and expensive projects. As such, the alignment of these initiatives with business goals is a must, as is buy-in from top management. While data quality efforts vary greatly from one organization to another, most involve an interaction of:

- A data governance structure that is responsible for data quality.
- Measurements of current data quality.
- Definition of data quality standards in alignment with business goals.
- Implementation of tools and processes to ensure future data quality.

There are a number of tools that can assist in the implementation of data quality initiatives. In particular, data profiling and master data management software is available from many vendors to assist in ensuring data quality. Data profiling software consists of programs that gather statistics and analyze existing data sources. These programs analyze existing data and the metadata to determine data patterns, and can compare the existing data patterns against standards that the organization has defined. This analysis can help the organization to understand the quality of the data that is currently in place and identify sources of dirty data. Master data management (MDM) software helps to prevent dirty data by coordinating common data across multiple systems. MDM provides a `_master_` copy of entities, such as customers, that appear in numerous systems throughout the organization.

While these technological approaches provide an important piece of data quality, the overall solution to high-quality data within an organization still relies heavily on the administration and management of the data.

16.2 THE NEED FOR A DATABASE AND ITS ROLE IN AN ORGANIZATION

Data are used by different people in different departments for different reasons. Therefore, data management must address the concept of shared data. unit 1 showed how the need for data sharing made the DBMS almost inevitable. Used properly, the DBMS facilitates:

- Interpretation and presentation of data in useful formats by transforming raw data into information.
- Distribution of data and information to the right people at the right time.
- Data preservation and monitoring the data usage for adequate periods of time.
- Control over data duplication and use, both internally and externally.

Whatever the type of organization, the database's predominant role is to support managerial decision making at all levels in the organization while preserving data privacy and security. An organization's managerial structure might be divided into three levels: top, middle, and operational. Top-level management makes strategic decisions, middle management makes

tactical decisions, and operational management makes daily operational decisions. Operational decisions are short term and affect only daily operations; for example, deciding to change the price of a product to clear it from inventory. Tactical decisions involve a longer time frame and affect larger-scale operations; for example, changing the price of a product in response to competitive pressures. Strategic decisions are those that affect the long-term well-being of the company or even its survival; for example, changing pricing strategy across product lines to capture market share.

The DBMS must provide tools that give each level of management a useful view of the data and that support the required level of decision making. The following activities are typical of each management level.

At the top management level, the database must be able to:

- Provide the information necessary for strategic decision making, strategic planning, policy formulation, and goals definition.
- Provide access to external and internal data to identify growth opportunities and to chart the direction of such growth. (Direction refers to the nature of the operations: Will a company become a service organization, a manufacturing organization, or some combination of the two?)
- Provide a framework for defining and enforcing organizational policies. (Remember that such policies are translated into business rules at lower levels in the organization.)
- Improve the likelihood of a positive return on investment for the company by searching for new ways to reduce costs and/or by boosting productivity.
- Provide feedback to monitor whether the company is achieving its goals.

At the middle management level, the database must be able to:

- Deliver the data necessary for tactical decisions and planning.
- Monitor and control the allocation and use of company resources and evaluate the performance of the various departments.
- Provide a framework for enforcing and ensuring the security and privacy of the data in the database. Security means protecting the data against accidental or intentional use by unauthorized users. Privacy deals with the rights of individuals and the organization to determine the “who, what, when, where, and how” of data usage.

At the operational management level, the database must be able to:

- Represent and support the company operations as closely as possible. The data model must be flexible enough to incorporate all required present and expected data.
- Produce query results within specified performance levels. Keep in mind that the performance requirements increase for lower levels of management and operations. Thus, the database must support fast responses to a greater number of transactions at the operational management level.
- Enhance the company's short-term operational ability by providing timely information for customer support and for application development and computer operations.

A general objective for any database is to provide a seamless flow of information throughout the company.

The company's database is also known as the corporate or enterprise database. The enterprise database might be defined as "the company's data representation that provides support for all present and expected future operations."

Most of today's successful organizations depend on the enterprise database to provide support for all of their operations—from design to implementation, from sales to services, and from daily decision making to strategic planning.

16.3 INTRODUCTION OF A DATABASE: SPECIAL CONSIDERATIONS

Having a computerized database management system does not guarantee that the data will be properly used to provide the best solutions required by managers. A DBMS is a tool for managing data; like any tool, it must be used effectively to produce the desired results. Consider this analogy: in the hands of a carpenter, a hammer can help produce furniture; in the hands of a child, it might do damage. The solution to company problems is not the mere existence of a computer system or its database, but, rather, its effective management and use.

The introduction of a DBMS represents a big change and challenge; throughout the organization, the DBMS is likely to have a profound impact, which might be positive or negative depending on how it is administered. For example, one key consideration is adapting the DBMS to the organization rather than forcing the organization to adapt to the DBMS. The main issue should be the organization's needs rather than the DBMS's technical capabilities. However, the introduction of a DBMS cannot be accomplished without affecting the

organization. The flood of new DBMS-generated information has a profound effect on the way the organization functions and, therefore, on its corporate culture.

The introduction of a DBMS into an organization has been described as a process that includes three important aspects:

- Technological: DBMS software and hardware.
- Managerial: Administrative functions.
- Cultural: Corporate resistance to change.

The technological aspect includes selecting, installing, configuring, and monitoring the DBMS to make sure that it efficiently handles data storage, access, and security. The person or people in charge of addressing the technological aspect of the DBMS installation must have the technical skills necessary to provide or secure adequate support for the various users of the DBMS: programmers, managers, and end users. Therefore, database administration staffing is a key technological consideration in the DBMS introduction. The selected personnel must exhibit the right mix of technical and managerial skills to provide a smooth transition to the new shared-data environment.

The managerial aspect of the DBMS introduction should not be taken lightly. A high-quality DBMS does not guarantee a high-quality information system, just as having the best race car does not guarantee winning a race.

The introduction of a DBMS into an organization requires careful planning to create an appropriate organizational structure to accommodate the person or people responsible for administering the DBMS. The organizational structure must also be subject to well-developed monitoring and controlling functions. The administrative personnel must have excellent interpersonal and communications skills combined with broad organizational and business understanding.

Top management must be committed to the new system and must define and support the data administration functions, goals, and roles within the organization.

The cultural impact of the introduction of a database system must be assessed carefully. The DBMS's existence is likely to have an effect on people, functions, and interactions. For example, additional personnel might be added, new roles might be allocated to existing personnel, and employee performance might be evaluated using new standards. A cultural impact is likely because the database approach creates a more controlled and structured information flow. Department managers who are used to handling their own data must

surrender their subjective ownership to the data administration function and must share their data with the rest of the company. Application programmers must learn and follow new design and development standards. Managers might be faced with what they consider to be an information overload and might require some time to adjust to the new environment. When the new database comes online, people might be reluctant to use the information provided by the system and might question its value or accuracy. (Many will be surprised and possibly chagrined to discover that the information does not fit their preconceived notions and strongly held beliefs.) The database administration department must be prepared to open its doors to end users, listen to their concerns, act on those concerns when possible, and educate end users about the system's uses and benefits.

16.4 THE EVOLUTION OF DATABASE ADMINISTRATION

Data administration has its roots in the old, decentralized world of the file system. The cost of data and managerial duplication in such file systems gave rise to a centralized data administration function known as the electronic data processing (EDP) or data processing (DP) department. The DP department's task was to pool all computer resources to support all departments at the operational level. The DP administration function was given the authority to manage all existing company file systems as well as resolve data and managerial conflicts created by the duplication and/or misuse of data.

The advent of the DBMS and its shared view of data produced a new level of data management sophistication and led the DP department to evolve into an information systems (IS) department. The responsibilities of the IS department were broadened to include:

- A service function to provide end users with active data management support.
- A production function to provide end users with specific solutions for their information needs through integrated application or management information systems.

The functional orientation of the IS department was reflected in its internal organizational structure. IS departments typically were structured as shown in Figure 16.2. As the demand for application development grew, the IS application development segment was subdivided by the type of supported system: accounting, inventory, marketing, and so on. However, this development meant that the database administration responsibilities were divided. The application development segment was in charge of gathering database requirements and

logical database design, whereas the database operations segment took charge of implementing, monitoring, and controlling the DBMS operations.

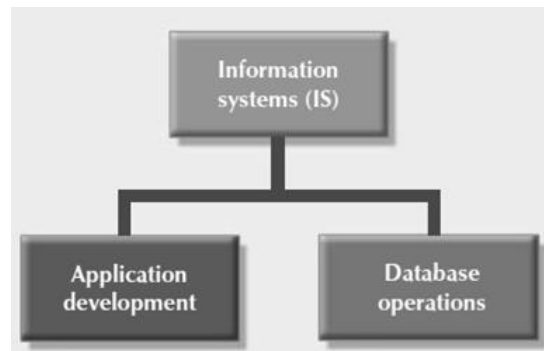


Fig. 16.2 The IS department internal organization

As the number of database applications grew, data management became an increasingly complex job, thus leading to the development of the database administration function. The person responsible for the control of the centralized and shared database became known as the database administrator (DBA).

The size and role of the DBA function varies from company to company, as does its placement within a company's organizational structure. On the organization chart, the DBA function might be defined as either a staff or line position. Placing the DBA function in a staff position often creates a consulting environment in which the DBA is able to devise the data administration strategy but does not have the authority to enforce it or to resolve possible conflicts. The DBA function in a line position has both the responsibility

and the authority to plan, define, implement, and enforce the policies, standards, and procedures used in the data administration activity. The two possible DBA function placements are illustrated in Figure 16.3.

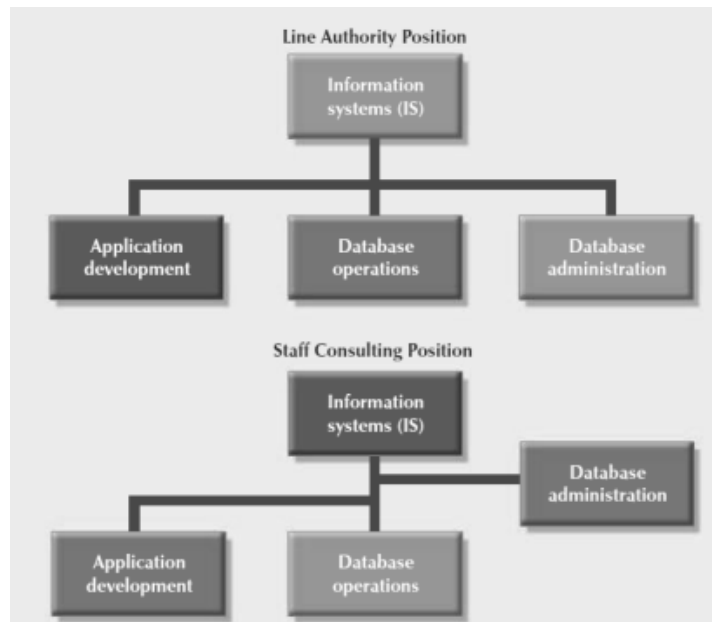


Fig16.3 The placement of the DBA function

There is no standard for how the DBA function fits in an organization's structure. In part, that is because the DBA function itself is probably the most dynamic of any organization's functions. In fact, the fast-paced changes in DBMS technology dictate changing organizational styles. For example:

- The development of distributed databases can force an organization to decentralize the data administration function further. The distributed database requires the system DBA to define and delegate the responsibilities of each local DBA, thus imposing new and more complex coordinating activities on the system DBA.
- The growing use of Internet-accessible data and the growing number of data-warehousing applications are likely to add to the DBA's data modeling and design activities, thus expanding and diversifying the DBA's job.
- The increasing sophistication and power of microcomputer-based DBMS packages provide an easy platform for the development of user-friendly, cost-effective, and efficient solutions to the needs of specific departments. But such an environment also invites data duplication, not to mention the problems created by people who lack the technical qualifications to produce good database designs. In short, the new microcomputer environment requires the DBA to develop a new set of technical and managerial skills. It is common practice to define the DBA function by dividing the DBA operations according to the Database Life Cycle (DBLC) phases. If that approach is used, the DBA function requires personnel to cover the following activities:

- Database planning, including the definition of standards, procedures, and enforcement.
- Database requirements gathering and conceptual design.
- Database logical and transaction design.
- Database physical design and implementation.
- Database testing and debugging.
- Database operations and maintenance, including installation, conversion, and migration.
- Database training and support.
- Data quality monitoring and management.

Figure 16.4 represents an appropriate DBA functional organization according to that model.

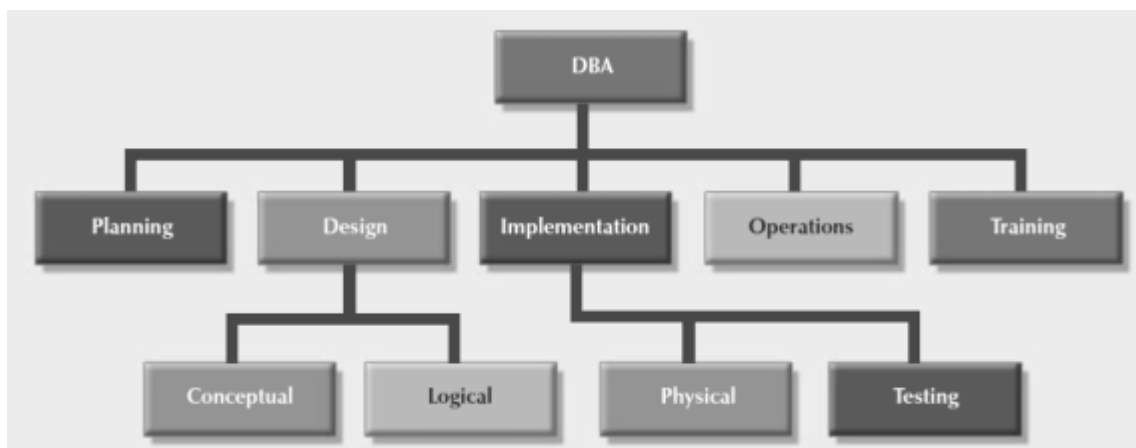


Fig 16.4 A DBA functional organization

Keep in mind that a company might have several different and incompatible DBMSs installed to support different operations. For example, it is not uncommon to find corporations with a hierarchical DBMS to support the daily transactions at the operational level and a relational database to support middle and top management's ad hoc information needs. There may also be a variety of microcomputer DBMSs installed in the different departments. In such an environment, the company might have one DBA assigned for each DBMS. The general coordinator of all DBAs is sometimes known as the systems administrator; that position is illustrated in Figure 16.5.

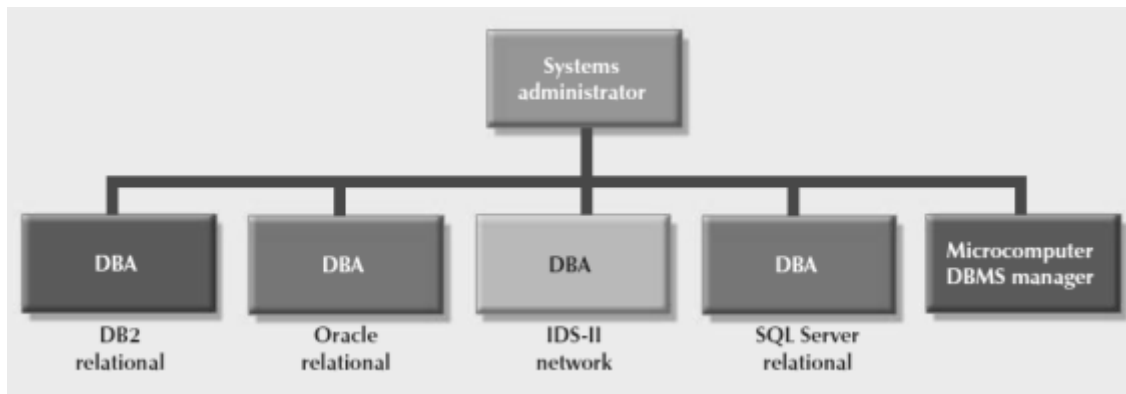


Fig16.5 Multiple database administrators in an organization

There is a growing trend toward specialization in the data management function. For example, the organization charts used by some of the larger corporations make a distinction between a DBA and the data administrator (DA). The DA, also known as the information resource manager (IRM), usually reports directly to top management and is given a higher degree of responsibility and authority than the DBA, although the two roles overlap some. The DA is responsible for controlling the overall corporate data resources, both computerized and manual. Thus, the DA's job description covers a larger area of operations than that of the DBA because the DA is in charge of controlling

not only the computerized data but also the data outside the scope of the DBMS. The placement of the DBA within the expanded organizational structure may vary from company to company. Depending on the structure's components, the DBA might report to the DA, the IRM, the IS manager, or directly to the company's CEO.

16.5 THE DATABASE ENVIRONMENT'S HUMAN COMPONENT

In this section, you will explore how people perform the data administration activities that make a good database design useful. Effective data administration requires both technical and managerial skills. For example, the DA's job typically has a strong managerial orientation with company-wide scope. In contrast, the DBA's job tends to be more technically oriented and has a narrower DBMS-specific scope. However, the DBA, too, must have a considerable store of people skills. After all, both the DA and the DBA perform "people" functions common to all departments in an organization. For example, both the DA and DBA direct and control personnel staffing and training within their respective departments.

Table 16.1 contrasts the general characteristics of both positions by summarizing the typical DA and DBA activities.

All activities flowing from the characteristics shown in Table 16.1 are invested in the DBA if the organization does not employ both a DA and a DBA.

Table 16.1 Contrasting DA and DBA activities and characteristics

DATA ADMINISTRATOR (DA)	DATABASE ADMINISTRATOR (DBA)
Does strategic planning	Controls and supervises
Sets long-term goals	Executes plans to reach goals
Sets policies and standards	Enforces policies and procedures Enforces programming standards
Is broad in scope	Is narrow in scope
Focuses on the long term	Focuses on the short term (daily operations)
Has a managerial orientation	Has a technical orientation
Is DBMS-independent	Is DBMS-specific

Let's now see DBA's role as an arbitrator between data and users.

The arbitration of interactions between the two most important assets of any organization, people and data, places the DBA in the dynamic environment portrayed in Figure 16.6.

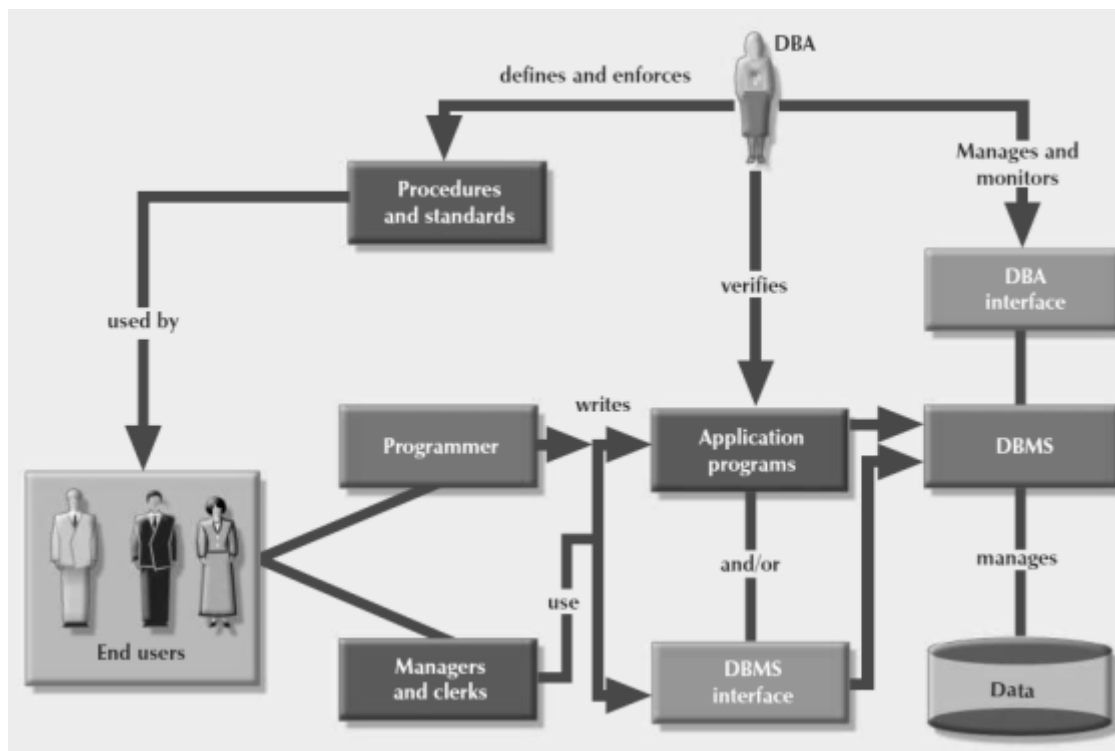


Fig. 16.6 A summary of DBA activities

The DBA activities portrayed in Figure 16.6 suggest the need for a diverse mix of skills. In large companies, such skills are likely to be distributed among several people who work within the DBA function. In small companies, the skills might be the domain of just one individual. The skills can be divided into two categories—managerial and technical—as summarized in Table 16.2.

Table 16.2 Desired DBA skills

MANAGERIAL	TECHNICAL
Broad business understanding	Broad data-processing background
Coordination skills	Systems Development Life Cycle knowledge
Analytical skills	Structured methodologies: Data flow diagrams Structure charts Programming languages
Conflict resolution skills	Database Life Cycle knowledge
Communications skills (oral and written)	Database modeling and design skills Conceptual Logical Physical
Negotiation skills	Operational skills: Database implementation, data dictionary management, security, and so on
Experience: 10 years in a large DP department	

The DBA’s responsibilities are shown in Table 16.3.

Table 16.3 DBA activities and services

DBA ACTIVITY	DBA SERVICE
Planning	End-user support
Organizing	Policies, procedures, and standards
Testing	Data security, privacy, and integrity
Monitoring	Data backup and recovery
Delivering	Data distribution and use

← of →

Table 16.3 illustrates that the DBA is generally responsible for planning, organizing, testing, monitoring, and delivering quite a few services. Those services might be performed by the DBA or, more likely, by the DBA’s personnel. Let’s examine the services in greater detail.

End-User Support

The DBA interacts with the end user by providing data and information support services to the organization’s departments. Because end users usually have dissimilar computer backgrounds, end-user support services include:

Gathering user requirements, building end-user confidence, resolving conflicts and problems, finding solutions to information needs, ensuring quality and integrity of data and applications, Managing the training and support of DBMS users. Policies, Procedures, and Standards

A prime component of a successful data administration strategy is the continuous enforcement of the policies, procedures, and standards for correct data creation, usage, distribution, and deletion within the database. The DBA must define, document, and communicate the policies, procedures, and standards before they can be enforced.

Basically:

Policies are general statements of direction or action that communicate and support DBA goals.

Standards describe the minimum requirements of a given DBA activity; they are more detailed and specific than policies. In effect, standards are rules that are used to evaluate the quality of the activity.

Procedures are written instructions that describe a series of steps to be followed during the performance of a given activity. Procedures must be developed within existing working conditions, and they must support and enhance that environment.

Data Security, Privacy, and Integrity

The security, privacy, and integrity of the data in the database are of great concern to DBAs who manage current DBMS installations. Technology has pointed the way to greater productivity through information management. Technology has also resulted in the distribution of data across multiple sites, thus making it more difficult to maintain

data control, security, and integrity. The multiple-site data configuration has made it imperative that the DBA use the security and integrity mechanisms provided by the DBMS to enforce the database administration policies defined. In addition, DBAs must team up with Internet security experts to build security mechanisms to safeguard data from possible attacks or unauthorized access.

Data Backup and Recovery

When data are not readily available, companies face potentially ruinous losses. Therefore, data backup and recovery procedures are critical in all database installations. The DBA must also ensure that the data in the database can be fully recovered in case of physical data loss or loss of database integrity.

The management of database security, integrity, backup, and recovery is so critical that many DBA departments have created a position called the database security officer (DSO). The

DSO's sole job is to ensure database security and integrity. In large organizations, the DSO's activities are often classified as disaster management.

Disaster management includes all of the DBA activities designed to secure data availability following a physical disaster or a database integrity failure. Disaster management includes all planning, organizing, and testing of database contingency plans and recovery procedures. A full backup, also known as a database dump, produces a complete copy of the entire database. An incremental backup produces a backup of all data since the last backup date; a concurrent backup takes place while the user is working on the database.

Data Distribution and Use

Data are useful only when they reach the right users in a timely fashion. The DBA is responsible for ensuring that the data are distributed to the right people, at the right time, and in the right format. The DBA's data distribution and use tasks can become very time-consuming, especially when the data delivery capacity is based on a typical applications programming environment, where users depend on programmers to deliver the programs to access the data in the database. Although the Internet and its intranet and extranet extensions have opened databases to corporate users, their use has also created a new set of challenges for the DBA. Current data distribution philosophy makes it easy for authorized end users to access the database. One way to accomplish that task is to facilitate the use of a new generation of more sophisticated query tools and the new Internet Web front ends. They enable the DBA to educate end users to produce the required information without being dependent on applications programmers. Naturally, the DBA must ensure that all users adhere to appropriate standards and procedures. This data-sharing philosophy is common today, and it is likely that it will become more common as database technology marches on. Such an environment is more flexible for the end user. Clearly, enabling end users to become relatively self-sufficient in the acquisition and use of data can lead to more efficient use of data in the decision process.

16.6 SECURITY

Security refers to activities and measures to ensure the confidentiality, integrity, and availability of an information system and its main asset, data. It is important to understand that securing data requires a comprehensive, company-wide approach. That is, you cannot secure data if you do not secure all the processes and systems around it. Indeed, securing data

entails securing the overall information system architecture, including hardware systems, software applications, the network and its devices, people (internal and external users), procedures, and the data itself.

To understand the scope of data security, let's discuss each of these security goals in more detail:

- Confidentiality deals with ensuring that data is protected against unauthorized access, and if the data are accessed by an authorized user, that the data are used only for an authorized purpose.
- Compliance refers to activities undertaken to meet data privacy and security reporting guidelines. These reporting guidelines are either part of internal procedures or are imposed by external regulatory agencies such as the government.
- Integrity, within the data security framework, is concerned with keeping data consistent, free of errors, or anomalies. Integrity focuses on maintaining the data free of inconsistencies and anomalies.
- Availability refers to the accessibility of data whenever required by authorized users and for authorized purposes. To ensure data availability, the entire system (not only the data component) must be protected from service degradation or interruption caused by any source (internal or external). Service interruptions could be very costly for companies and users alike. System availability is an important goal of security.

16.6.1 Security Policies

Normally, the tasks of securing the system and its main asset, the data, are performed by the database security officer and the database administrator(s), who work together to establish a cohesive data security strategy. Such security strategy begins with defining a sound and comprehensive security policy. A security policy is a collection of standards, policies, and procedures created to guarantee the security of a system and ensure auditing and compliance.

The security audit process starts by identifying the security vulnerabilities in the organization's information system infrastructure and identifying measures to protect the system and data against those vulnerabilities.

16.6.2 Security Vulnerabilities

A security vulnerability is a weakness in a system component that could be exploited to allow unauthorized access or cause service disruptions. The nature of such vulnerabilities could be of multiple types: technical (such as a flaw in the operating system or Web browser),

managerial (for example, not educating users about critical security issues), cultural (hiding passwords under the keyboard or not shredding confidential reports), procedural (not requiring complex passwords or not checking user IDs), and so on. Whatever the case, when a security vulnerability is left unchecked, it could become a security threat. A security threat is an imminent security violation that could occur at any time due to unchecked security vulnerability.

A security breach occurs when a security threat is exploited to negatively affect the integrity, confidentiality, or availability of the system. Security breaches can yield a database whose integrity is either preserved or corrupted:

16.6.3 Database Security

Database security refers to the use of the DBMS features and other related measures to comply with the security requirements of the organization. From the DBA's point of view, security measures should be implemented to protect the DBMS against service degradation and the database against loss, corruption, or mishandling. In short, the DBA should secure the DBMS from the point of installation through operation and maintenance.

To protect the DBMS against service degradation there are certain minimum recommended security safeguards such as:

- Change default system passwords.
- Change default installation paths.
- Apply the latest patches.
- Secure installation folders with proper access rights.
- Make sure only required services are running.
- Set up auditing logs.
- Set up session logging.
- Require session encryption.

Furthermore, the DBA should work closely with the network administrator to implement network security to protect the DBMS and all services running on the network. In current organizations, one of the most critical components in the information architecture is the network.

Protecting the data in the database is a function of authorization management. Authorization management defines procedures to protect and guarantee database security and integrity.

Those procedures include, but are not limited to, user access management, view definition, DBMS access control, and DBMS usage monitoring.

- User access management. This function is designed to limit access to the database and likely includes at least the following procedures:
- Define each user to the database. This is achieved at the operating system level and at the DBMS level. At the operating system level, the DBA can request the creation of a logon user ID that allows the end user to log on to the computer system. At the DBMS level, the DBA can either create a different user ID or employ the same user ID to authorize the end user to access the DBMS.
- Assign passwords to each user. This, too, can be done at both operating system and DBMS levels. The database passwords can be assigned with predetermined expiration dates. The use of expiration dates enables the DBA to screen end users periodically and to remind users to change their passwords periodically, thus making unauthorized access less probable.
- Define user groups. Classifying users into user groups according to common access needs facilitates the DBA's job of controlling and managing the access privileges of individual users. Also, the DBA can use database roles and resource limits to minimize the impact of rogue users in the system
- Assign access privileges. The DBA assigns access privileges or access rights to specific users to access specified databases. An access privilege describes the type of authorized access. For example, access rights may be limited to read-only, or the authorized access might include READ, WRITE, and DELETE privileges. Access privileges in relational databases are assigned through SQL GRANT and REVOKE commands.
- Control physical access. Physical security can prevent unauthorized users from directly accessing the DBMS installation and facilities. Some common physical security practices found in large database installations include secured entrances, password-protected workstations, electronic personnel badges, closed-circuit video, voice recognition, and biometric technology.
- View definition. The DBA must define data views to protect and control the scope of the data that are accessible to an authorized user. The DBMS must provide the tools that allow the definition of views that are composed of one or more tables and the assignment of access rights to a user or a group of users. The SQL command CREATE

VIEW is used in relational databases to define views. Oracle DBMS offers Virtual Private Database (VPD), which allows the DBA to create customized views of the data for multiple different users. With this feature, the DBA could restrict a regular user querying a payroll database to see only the rows and columns necessary, while the department manager would see only the rows and columns pertinent to that department.

- DBMS access control. Database access can be controlled by placing limits on the use of DBMS query and reporting tools. The DBA must make sure that those tools are used properly and only by authorized personnel.
- DBMS usage monitoring. The DBA must also audit the use of the data in the database. Several DBMS packages contain features that allow the creation of an audit log, which automatically records a brief description of the database operations performed by all users. Such audit trails enable the DBA to pinpoint access violations. The audit trails can be tailored to record all database accesses or just failed database accesses.

16.7 Database Administration Tools

This section will examine the data dictionary as a data administration tool, as well as the DBA's use of computer-aided software engineering (CASE) tools to support database analysis and design.

16.7.1 The Data Dictionary

a data dictionary was defined as “a DBMS component that stores the definition of data characteristics and relationships.” “data about data” are called metadata. The DBMS data dictionary provides the DBMS with its self-describing characteristic. In effect, the data dictionary resembles an X-ray of the company's entire data set, and it is a crucial element in data administration.

Two main types of data dictionaries exist: integrated and standalone. An integrated data dictionary is included with the DBMS. For example, all relational DBMSs include a built-in data dictionary or system catalog that is frequently accessed and updated by the RDBMS. Other DBMSs, especially older types, do not have a built-in data dictionary; instead, the DBA may use third-party standalone data dictionary systems.

Data dictionaries can also be classified as active or passive. An active data dictionary is automatically updated by the DBMS with every database access, thereby keeping its access

information up to date. A passive data dictionary is not updated automatically and usually requires running a batch process. Data dictionary access information is normally used by the DBMS for query optimization purposes.

The data dictionary's main function is to store the description of all objects that interact with the database. Integrated data dictionaries tend to limit their metadata to the data managed by the DBMS. Standalone data dictionary systems are usually more flexible and allow the DBA to describe and manage all of the organization's data, whether or not they are computerized. Whatever the data dictionary's format, its existence provides database designers and end users with a much-improved ability to communicate. In addition, the data dictionary is the tool that helps the DBA resolve data conflicts.

Although there is no standard format for the information stored in the data dictionary, several features are common.

If the data dictionary can be organized to include data external to the DBMS itself, it becomes an especially flexible tool for more general corporate resource management. The management of such an extensive data dictionary thus makes it possible to manage the use and allocation of all of the organization's information, regardless of whether the information has its roots in the database data. That is why some managers consider the data dictionary to be a key element of information resource management. And that is also why the data dictionary might be described as the information resource dictionary.

The metadata stored in the data dictionary are often the basis for monitoring database use and for assigning access rights to the database users. The information stored in the data dictionary is usually based on a relational table format, thus enabling the DBA to query the database with SQL commands. For example, SQL commands can be used to extract information about the users of a specific table or about the access rights of a particular user. In the following example, the IBM DB2 system catalog tables will be used as the basis for several examples of how a data dictionary is used to derive information:

- SYSTABLES stores one row for each table or view.
- SYSCOLUMNS stores one row for each column of each table or view.
- SYSTABAUTH stores one row for each authorization given to a user for a table or view in a database.

Example

List the names and creation dates of all tables created by the user RAMA in the current database.

```
SELECT NAME, CTIME
FROM SYSTABLES
WHERE CREATOR = 'RAMA';
```

The DBA can use the data dictionary to support data analysis and design. For example, the DBA can create a report that lists all data elements to be used in a particular application; a list of all users who access a particular program; a report that checks for data redundancies, duplications, and the use of homonyms and synonyms; and a number of other reports that describe data users, data access, and data structure. The data dictionary can also be used to ensure that applications programmers have met all of the naming standards for the data elements in the database and that the data validation rules are correct. Thus, the data dictionary can be used to support a wide range of data administration activities and to facilitate the design and implementation of information systems. Integrated data dictionaries are also essential to the use of computer-aided software engineering tools.

16.7.2. CASE Tools

CASE is the acronym for computer-aided systems engineering. A CASE tool provides an automated framework for the Systems Development Life Cycle (SDLC). CASE uses structured methodologies and powerful graphical interfaces. Because they automate many tedious system design and implementation activities, CASE tools play an increasingly important role in information systems development.

CASE tools are usually classified according to the extent of support they provide for the SDLC. For example, front-end CASE tools provide support for the planning, analysis, and design phases; back-end CASE tools provide support for the coding and implementation phases. The benefits associated with CASE tools include:

- A reduction in development time and costs.
- Automation of the SDLC.
- Standardization of systems development methodologies.
- Easier maintenance of application systems developed with CASE tools.

One of the CASE tools' most important components is an extensive data dictionary, which keeps track of all objects created by the systems designer. For example, the CASE data dictionary stores data flow diagrams, structure charts, descriptions of all external and internal entities, data stores, data items, report formats, and screen formats. A CASE data dictionary also describes the relationships among the components of the system.

A typical CASE tool provides five components:

- Graphics designed to produce structured diagrams such as data flow diagrams, ER diagrams, class diagrams, and object diagrams.
- Screen painters and report generators to produce the information system's input/output formats (for example, the end-user interface).
- An integrated repository for storing and cross-referencing the system design data. This repository includes a comprehensive data dictionary.
- An analysis segment to provide a fully automated check on system consistency, syntax, and completeness.
- A program documentation generator.

16.8 DEVELOPING A DATA ADMINISTRATION STRATEGY

For a company to succeed, its activities must be committed to its main objectives or mission. Therefore, regardless of a company's size, a critical step for any organization is to ensure that its information system supports its strategic plans for each of its business areas.

The database administration strategy must not conflict with the information systems plans. After all, the information systems plans are derived from a detailed analysis of the company's goals, its condition or situation, and its business needs. Several methodologies are available to ensure the compatibility of data administration and information systems plans and to guide the strategic plan development. The most commonly used methodology is known as information engineering.

Information engineering (IE) allows for the translation of the company's strategic goals into the data and applications that will help the company achieve those goals. IE focuses on the description of the corporate data instead of the processes. The IE rationale is simple: business data types tend to remain fairly stable. In contrast, processes change often and thus require the frequent modification of existing systems. By placing the emphasis on data, IE helps decrease the impact on systems when processes change.

The output of the IE process is information systems architecture (ISA) that serves as the basis for planning, development, and control of future information systems. Figure 16.8 shows the forces that affect ISA development.

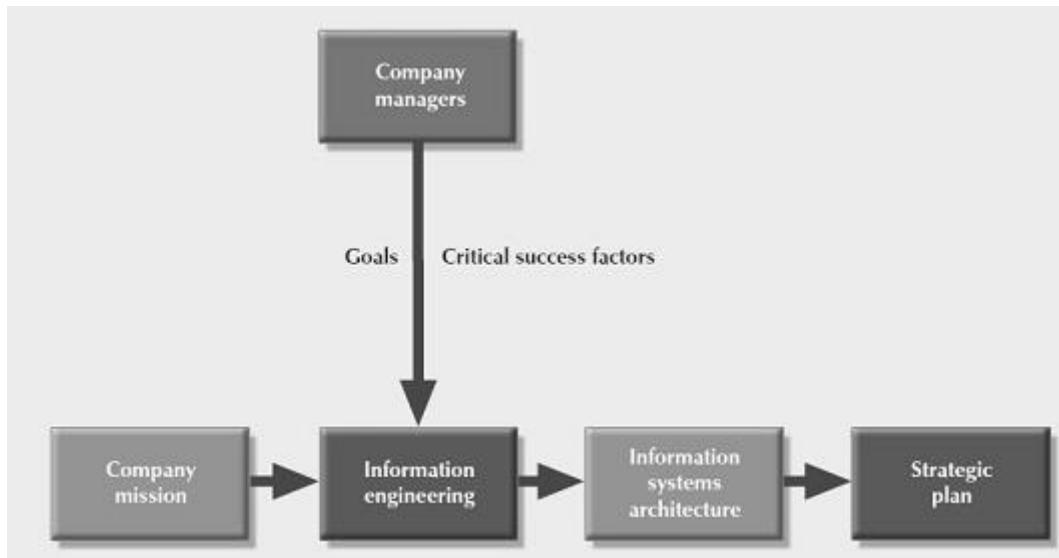


Fig. 16.8 Forces affecting the development of ISA

Implementing IE methodologies in an organization is a costly process that involves planning, a commitment of resources, management liability, well-defined objectives, identification of critical factors, and control. An ISA provides a framework that includes the use of computerized, automated, and integrated tools such as a DBMS and CASE tools.

16.9 THE DBA'S ROLE IN THE CLOUD

The core tasks of DBAs do not change with cloud. What changes is how DBAs perform these tasks.

Capacity Planning

Capacity planning has been a core DBA task since the release of the first database engines. Performing this planning in IaaS is very similar to doing it on premise for a virtualized environment. The key difference is the added flexibility due to the fact that cloud providers will not run out of physical VM hosts. As a result, DBAs should provision for only what the business needs today instead of old-style over-provisioning for a physical machine that is expected to last three to four years.

DBAs need to make use of the bursting capability of the cloud to quickly scale up or scale down to meet demand. Proper capacity planning also involves responding to changes in

offerings by cloud service providers to get the best deals. For example, a provider may offer a new type of storage or a new type of VM that would be a better fit for one of your workloads.

Monitoring

Monitoring is another core task that DBAs perform daily. All of the major cloud IaaS providers offer monitoring. Amazon Web Services (AWS) has a built-in service called Amazon CloudWatch that monitors any type of AWS asset. Google offers Stackdriver Monitoring and logging that integrates with all services provided by Google Cloud Platform, including Cloud SQL. Microsoft Azure provides the Azure Monitor service, which enables DBAs to put different thresholds on different metrics tied to a service, provides alerts dependent on those values and can be used for analysis over time

High Availability and Disaster Recovery

Mission-critical databases inherently have very high up-time requirements that entail the need for high availability (HA) and disaster recovery (DR) architectures and strategies. This is still true for IaaS(Infrastructure as a Service) in the cloud.

Configuration and Performance

In cloud IaaS, configuration optimization and performance tuning also change. On-premises, a business usually has just one hardware provider. The DBA changes the hardware settings as required to optimize configuration and improve performance. By contrast, each IaaS provider has a different family of VM types for different database workloads.

Performance Tuning

A DBA needs to know the mechanisms that the service provider makes available to increase or decrease the compute power, and have the knowledge to choose the most effective and cost-efficient solution. To increase or decrease compute power in Amazon Relational Database Service (RDS), a DBA can change the instance type of an RDS database server. In Google Cloud Platform, the DBA can change the instance type for Google Cloud SQL deployment. And for Microsoft Azure, the DBA can increase or decrease the database transaction units or use a corebased sizing model.

Automation Through Scripting

Automation is another core task that must be adopted by DBAs in this cloud age. The DBaaS(DB as a Service) providers offer functionality to automate deployment of objects. AWS has CloudFormation templates. Google has Google Cloud Deployment Manager.

Microsoft Azure has Azure Resource Manager templates. These are all the same concept, but with different implementations. Automation templates are just a starting point. There are also popular third-party options like Ansible and Terraform.

Security

Security is a major concern in the cloud, and it will continue to be a DBA responsibility. DBAs need to be familiar with which parts of their databases hold sensitive data, including any type of personal identifiable information. DBAs also need to know the change control and procedure measures that must be followed to provide access to sensitive database resources. Finally, they need to be aware of new security features released by DBaaS providers, to determine which features can be used to improve the organization's security, including encryption and protection of its databases.

16.10 THE DBA AT WORK: USING ORACLE FOR DATABASE ADMINISTRATION

Thus far, you've learned about the DBA's work environment and responsibilities in general terms. In this section, you will get a more detailed look at how a DBA might handle the following technical tasks in a specific DBMS:

- _ Creating and expanding database storage structures.
- _ Managing database objects such as tables, indexes, triggers, and procedures.
- _ Managing the end-user database environment, including the type and extent of database access.
- _ Customizing database initialization parameters.

Many of those tasks require the DBA to use software tools and utilities that are commonly provided by the database vendor. In fact, all DBMS vendors provide a set of programs to interface with the database and to perform a wide range of database administrative tasks.

We chose Oracle 11g for Windows to illustrate the selected DBA tasks because it is typically found in organizations that are sufficiently large and have a sufficiently complex database environment to require (and afford) the use of a DBA, it has good market presence, and it is also often found in small colleges and universities.

Oracle Database Administration Tools

All database vendors supply a set of database administration tools. In Oracle, you perform most DBA tasks via the Oracle Enterprise Manager interface.

The Default Login

To perform any administrative task, you must connect to the database, using a username with administrative (DBA) privileges. By default, Oracle automatically creates SYSTEM and SYS user IDs that have administrative privileges with every new database you create. You can define the preferred credentials for each database by clicking on the Preferences link at the top of the page, then click on Preferred Credentials. Finally, choose your target username under Set Credentials. Keep in mind that usernames and passwords are database-specific. Therefore, each database can have different usernames and passwords. One of the first things you must do is change the password for the SYSTEM and SYS users.

Immediately after doing that, you can start defining your users and assigning them database privileges.

Ensuring an Automatic RDBMS Start

One of the basic DBA tasks is to ensure that your database access is automatically started when you turn on the computer. Startup procedures will be different for each operating system. Because Oracle is used for this section's examples, you would need to identify the required services to ensure automatic database startup. (A service is the Windows system name for a special program that runs automatically as part of the operating system. This program ensures the availability of required services to the system and to end users on the local computer or over the network.)

Creating Tablespaces and Datafiles

Each DBMS manages data storage differently. In this example, the Oracle RDBMS will be used to illustrate how the database manages data storage at the logical and the physical levels. In Oracle A database is logically composed of one or more tablespaces. A tablespace is a logical storage space. Tablespaces are used primarily to group related data logically. The tablespace data are physically stored in one or more datafiles. A datafile physically stores the database's data. Each datafile is associated with one and only one tablespace, but each datafile can reside in a different directory on the hard disk or even on one or more different hard disks. Given the preceding description of tablespaces and datafiles, you can conclude that a database has a one-to-many relationship with tablespaces and that a tablespace has a one-to-many relationship with datafiles. This set of 1:M hierarchical relationships isolates the end user from any physical details of the data storage. However, the DBA must be aware of these details in order to properly manage the database.

To perform database storage management tasks such as creating and managing tablespaces and datafiles, the DBA uses the Enterprise Manager, Administration, Storage option.

Managing the Database Objects: Tables, Views, Triggers, and Procedures

Another important aspect of managing a database is monitoring the database objects that were created in the database. The Oracle Enterprise Manager gives the DBA a graphical user interface to create, edit, view, and delete database objects in the database. A database object is basically any object created by end users; for example, tables, views, indexes, stored procedures, and triggers. Oracle Schema Manager is the tool used to manage different types of objects.

16.11 CHECK YOUR PROGRESS

1. Define dirty data
2. What is data quality?
3. Define security and privacy.
4. What are the DBA's managerial roles?
5. What DBA activities are used to support the end-user community?

Answers to check your progress:

1. Data that contain inaccuracies and/or inconsistencies. T
2. Data quality is concerned with cleaning dirty data, preventing future inaccuracies in the data, and building user confidence in the data. dotted
3. Security refers to activities and measures to ensure the confidentiality, integrity, and availability of an information system and its main asset, data. Privacy deals with the rights of individuals and the organization to determine the "who, what, when, where, and how" of data usage
4. Coordinating, monitoring, and allocating database administration resources: people and data. Defining goals and formulating strategic plans for the database administration function.
5. Gathering user requirements, Building end-user confidence, Resolving conflicts and problems, Finding solutions to information needs, Ensuring quality and integrity of data and applications, Managing the training and support of DBMS users.

16.12 SUMMARY

- Data management is a critical activity for any organization. Data must be treated as a corporate asset. The value of a data set is measured by the utility of the information derived from it. Good data management is likely to produce good information, which is the basis for better decision making.
- Data quality is a comprehensive approach to ensuring the accuracy, validity, and timeliness of the data. Data quality is concerned with cleaning dirty data, preventing future inaccuracies in the data, and building user confidence in the data.
- The DBMS is the most commonly used electronic tool for corporate data management. The DBMS supports strategic, tactical, and operational decision making at all levels of the organization. The company data that are managed by the DBMS are stored in the corporate or enterprise database.
- The introduction of a DBMS into an organization is a very delicate job. In addition to managing the technical details of DBMS introduction, the impact of the DBMS on the organization's managerial and cultural framework must be carefully examined.
- Development of the data administration function is based on the evolution from departmental data processing to the more centralized electronic data processing (EDP) department to the more formal "data as a corporate asset" information systems (IS) department. Typical file systems were characterized by applications that tended to behave as distinct "islands of information." As applications began to share a common data repository, the need for centralized data management to control such data became clear.
- The database administrator (DBA) is responsible for managing the corporate database. The internal organization of the database administration function varies from company to company. Although no standard exists, it is common practice to divide DBA operations according to the Database Life Cycle phases. Some companies have created a position with a broader data management mandate to manage computerized and other data within the organization. This broader data management activity is handled by the data administrator (DA).
- The DA and the DBA functions tend to overlap. Generally speaking, the DA is more managerially oriented than the more technically oriented DBA. Compared to the DBA function, the DA function is DBMS-independent, with a broader and longer-term focus.

However, when the organization chart does not include a DA position, the DBA executes all of the DA's functions. Because the DBA has both technical and managerial responsibilities, the DBA must have a diverse mix of skills.

- The managerial services of the DBA function include at least: supporting the end-user community; defining and enforcing policies, procedures, and standards for the database function; ensuring data security, privacy, and integrity; providing data backup and recovery services; and monitoring the distribution and use of the data in the database.
- The technical role requires the DBA to be involved in at least these activities: evaluating, selecting, and installing the DBMS; designing and implementing databases and applications; testing and evaluating databases and applications; operating the DBMS, utilities, and applications; training and supporting users; and maintaining the DBMS, utilities, and applications.
- Security refers to activities and measures to ensure the confidentiality, integrity, and availability of an information system and its main asset, data. A security policy is a collection of standards, policies, and practices created to guarantee the security of a system and ensure auditing and compliance.
- A security vulnerability is weakness in a system component that could be exploited to allow unauthorized access or service disruption. A security threat is an imminent security violation caused by an unchecked security vulnerability. Security vulnerabilities exist in all components of an information system: people, hardware, software, network, procedures, and data. Therefore, it is critical to have robust database security. Database security refers to the use of DBMS features and related measures to comply with the security requirements of the organization.
- The development of the data administration strategy is closely related to the company's mission and objectives. Therefore, the development of an organization's strategic plan corresponds to that of data administration, requiring a detailed analysis of company goals, situation, and business needs. To guide the development of this overall plan, an integrating methodology is required. The most commonly used integrating methodology is known as information engineering (IE).
- To help translate strategic plans into operational plans, the DBA has access to an arsenal of database administration tools. These tools include the data dictionary and computer-aided software engineering (CASE) tools.

16.13 KEYWORDS

- **Access plan** - An access plan is a set of instructions generated at application compilation time that predetermines how the application will access the database at run time.
- **Active data dictionary** – An active data dictionary is automatically updated by the DBMS with every database access, thereby keeping its access information up to date.
- **Audit log** - audit log, which automatically records a brief description of the database operations performed by all users..
- **CASE** - computer-aided systems engineering
- **DA** -broader data management activity is handled by the data administrator (DA)
- **Incremental backup** - produces a backup of all data since the last backup date
- **Tablespace** - is a logical storage space. Tablespaces are used primarily to group related data logically.

16.14 QUESTIONS FOR SELF-STUDY

1. Explain the difference between data and information. Give some examples of raw data and information.
2. Suppose that you are a DBA staff member. What data dimensions would you describe to top-level managers to obtain their support for the data administration function?
3. Describe and contrast the information needs at the strategic, tactical, and operational levels in an organization. Use examples to explain your answer.
4. Describe the DBA's responsibilities.
5. Discuss the importance and characteristics of database backup and recovery procedures. Then describe the actions that must be detailed in backup and recovery plans.

16.15 REFERENCES

1. Coronel, C., & Morris, S. (2016). Database systems: design, implementation, & management. Cengage Learning.
2. Gupta, S. B., & Mittal, A. (2009). Introduction to database management system.Laxmi Publications, Ltd..
3. Ramakrishnan, R., &Gehrke, J. (2000). Database management systems.McGraw-Hill.