MCA13

KRISHNA KANTA HANDIQUI STATE OPEN UNIVERSITY

Housefed Complex, Dispur, Guwahati - 781 006

Master of Computer Applications

ADVANCED DATABASE MANAGEMENT SYSTEM

CONTENTS

Subject Expert

Prof. Anjana Kakati Mahanta, Deptt. of Computer Science, Gauhati University
Prof. Jatindra Kr. Deka, Deptt. of Computer Science and Engineering, Indian Institute of Technology, Guwahati
Prof. Diganta Goswami, Deptt. of Computer Science and Engineering, Indian Institute of Technology, Guwahati

Course Coordinator

Tapashi Kashyap Das, Assistant Professor, Computer Science, KKHSOU
Arabinda Saikia, Assistant Professor, Computer Science, KKHSOU

SLM Preparation Team

Units	Contributors
1	**Arabinda Saikia**
2	**Jonalee Barman Kakoti,** Assistant Professor, NERIM
3,9	**Biswajit Das,** Assistant Professor, Cotton College
4,5	**Irani Hazarika**, Guest Lecturer, Gauhati University
6,7	**Pritam Medhi**, Gauhati University
8	**Ujjal Sarma**, Assistant Professor, NERIM

July 2009

Printed and published by Registrar on behalf of the Krishna Kanta Handique State Open University.

The university acknowledges with thanks the financial support provided by the **Distance Education Council, New Delhi**, for the preparation of this study material.

Housefed Complex, Dispur, Guwahati- 781006 Web: www.kkhsou.in

COURSE INTRODUCTION

In today's competitive environment, database and database management systems have become essential for managing our business, governments, banks, universities and every other kind of human endeavour. This course is on **Advanced Database Management System**. DBMS is a software package that allows data to be effectively stored, retrieve and manipulated. The course is divided into the following units :

Unit - 1 concentrates on an overview of database, database management systems(DBMS), types of DBMS and data models.

Unit - 2 deals with Entity-Relationship model and its use in database design.

Unit - 3 concentrates on relational model. Differerent types of integrity constraints are also discussed in this unit.

Unit - 4 introduces one important query language namely Structured Query Language(SQL). SQL characteristics, SQL data types and different SQL commands are discussed in this unit.

Unit - 5 deals with the basic building elements of Structured Query Language.

Unit - 6 concentrates on relational database design. Most important concept ER to Relational conversion, functional dependency and normalization is discussed in this unit.

Unit - 7 introduces the concept of transaction processing.

Unit - 8 deals with the concept of concurrency and database recovery.

Unit - 9 concentrates on the issue of database security and privacy.

Each unit of these blocks includes some along-side boxes to help you know some of the difficult, unseen terms. Some "EXERCISE" have been included to help you apply your own thoughts. You may find some boxes marked with: "LET US KNOW". These boxes will provide you with some additional interesting and relevant information. Again, you will get "CHECK YOUR PROGRESS" questions. These have been designed to self-check your progress of study. It will be helpful for you if you solve the problems put in these boxes immediately after you go through the sections of the units and then match your answers with "ANSWERS TO CHECK YOUR PROGRESS" given at the end of each unit.

MASTER OF COMPUTER APPLICATIONS

Advanced Database Management System

DETAILED SYLLABUS

Unit 1: Introduction to Database System
Database, DBMS, Characteristics of DBMS, Merits and Demerits of DBMS, Database Architecture:3-tier Architecture of DBMS (its advantages over 2-tier), Data Independence, DBMS language, Types of DBMS, Database Administrator, Data Models;

Unit 2: Database Design using ER Model
Entities, Relationships, Representation of Entities, Attributes, Relationship Attributes, Relationship Set, Generalization, Aggregation, Structure of Relational Database and different types of Keys, Expressing M: N relation

Unit 3: Relational Model
Codd's rules, Relational Data Model & Relational Algebra, Relational Model Concept, Relational Model Constraints, Relational Algebra, Relational Calculus.

Unit 4: Introduction to SQL
SQL, Characteristics of SQL, Advantages of SQL, SQL data types and literals, Types of SQL commands.

Unit 5: Elements of SQL
SQL operators and their procedure, Tables, Views and Indexes, Queries and Sub Queries, Aggregate Functions, Insert, Update and Delete Operations, Joins, Unions, Intersection, Minus.

Unit 6: Relational Database design
Database Design – ER to Relational model, Functional dependencies, Normalization, Normal forms based on Primary keys (1 NF, 2 NF, 3 NF, BCNF, 4 NF, 5 NF), Loss less joins and dependency, preserving decomposition

Unit 7: Transaction Processing Concepts
Introduction to transaction processing; transaction and system concepts; desirable properties of transaction; characteristics schedule based on recoverability; characteristics schedule based on serializability.

Unit 8: Concurrency Control and Recovery
Two phase locking techniques for concurrency control; Concurrency control based on time stamp ordering; multi-version concurrency control techniques; Validation concurrency control techniques; granularity of data items and multi granularity locking, recovery concepts and recovery techniques.

Unit 9: Security and Privacy
Database security issues, Discretionary access control based on grant & revoking privilege, Mandatory access control and role based access control for multilevel security, Encryption & public key infrastructures.

UNIT-1 : INTRODUCTION TO DATABASE SYSTEM

UNIT STRUCTURE

1.1 LEARNING OBJECTIVES

After going through this unit, you will be able to :

- define a database and a DBMS
- describe DBMS Architecture
- illustrate data independence and data dictionary
- explain DBMS language

- identify the components of database system environment
- describe the role of DBA
- describe data models

1.2 INTRODUCTION

This is the first unit of the course *Advanced Database Management System,* In this unit, we will learn some elementary concepts of database along with the concept of DBMS, database architecture, etc. Learners might have already learned these elementary concepts while studying the course "*Fundamentals of Database Management System*".

The responsibilities of a database administrator and the concept of data models will also be discussed at the end.

1.3 TRADITIONAL FILE APPROACH

In earlier days, an organization's information was stored as group of records in separate files. These file processing systems consisted with few data files and many application programs as shown in the figure below. Each file, called a *flat file,* contained and processed information for one specific function, such as accounting or inventory. At that time programmers used programming languages such as COBOL to write application programs that can directly accessed flat files to perform data management services and provide information for users. Each application files and programs were created and maintained independent of another applications.

For example, if we consider the students data in a Universitiy then

- student address may be needed for *the* applications like *registering, library management, financial office, grade reporting*...etc.
- each application separately maintains its data files and programs to manipulate those files
- possibly the same data (e.g., length of names, address etc.) are stored in different format in the above applications
- whenever some information regarding a group of students are updated

in one application that updation may not be done simultaneously in the different applications where students records are stored.

As a result, the system will provide wrong information about students. In addition, potentially different values and/or different formats for the same data are stored in different files which lead to not only wastage of space but also cause the redundancy.

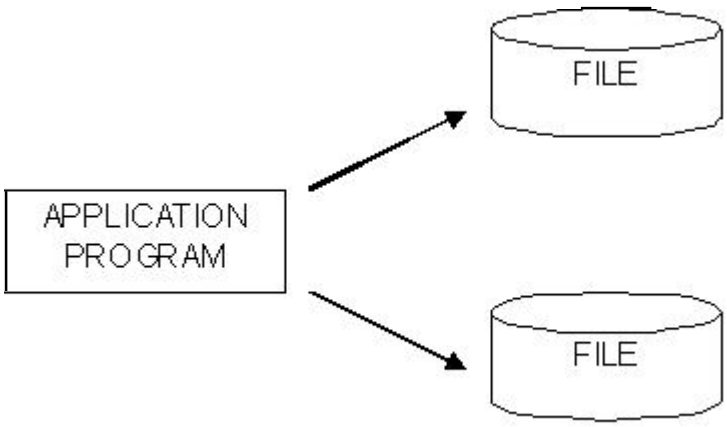


Fig. 1.1: Traditional File Approach

On creating the files and programs for the file oriented system, the developers focused on business processes, or how business was transacted, and their interaction. However, business processes are dynamic, requiring continous changes in files and applications. Moreover, programmers designed the codes in accordance with the physical storage structure of data and access procedures also depends on it. Therefore, any physical changes resulted the programmer to again rewriting the code to adjust the change.

The file-based approaches, which came into being as the first commercial applications of computers, suffered from the following significant disadvantages :

Data redundancy :

In a file system if an information is needed by two distinct applications, then it may be stored in two or more files. Repetition of same data item in more than one file is known as ***data redundancy***. This leads to increase in cost of data entry and data storage.

Data integrity problem :

Data redundancy also leads to data inconsistency or loss of data integrity. Data integrity refers to consistency of data in all files. That is, any change in a data item must be carried out in every file containing that field for consistency.

Lack of data independence :

In file processing systems, files and records were described by specific physical formats that were coded into the application program by programmer. If the format of a certain record was changed, the code in each file containing that format must be updated.

Poor data control :

A file oriented system is decentralised in nature, it means there was no centralised control at the data element level.

Incompatible file formats :

As the structure of file is embedded in the application programs, the structures are dependent on the application programming language. For example, the structure of a file generated by a COBOL program may be different from the structure of a file generated by a 'C' program. The direct incompatibility of such files makes them difficult to process jointly.

1.4 DATABASE APPROACH

An alternative approach to the traditional file processing system is the modern concept, known as the *database approach*. A *database* is an organised collection of records and files which are related to each other. In a database system, a common pool of data can be shared by a number of applications as it is data and program independant. Thus, unlike a file processing system, data redundancy and data inconsistency in the database system approach are minimised. In database approach the user is free from the detailed and complicated task of keeping up with the physical structure of data. A clear-cut distinction between traditional file system and database system is

depicted by the following diagram.

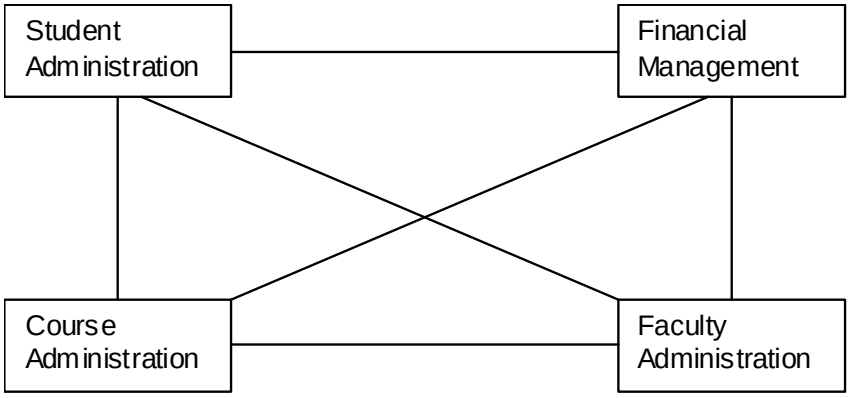


Fig.1.2: Traditional file approach

Here in the figure 1.2, the traditional record keeping system in a University is shown, where every applications are interrelated and caused repetation of same data in different files which leads to the problem of data redundancy and inconsistency.

In the following figure(Fig.1.3) it is shown that several applications share common data in a database approach.

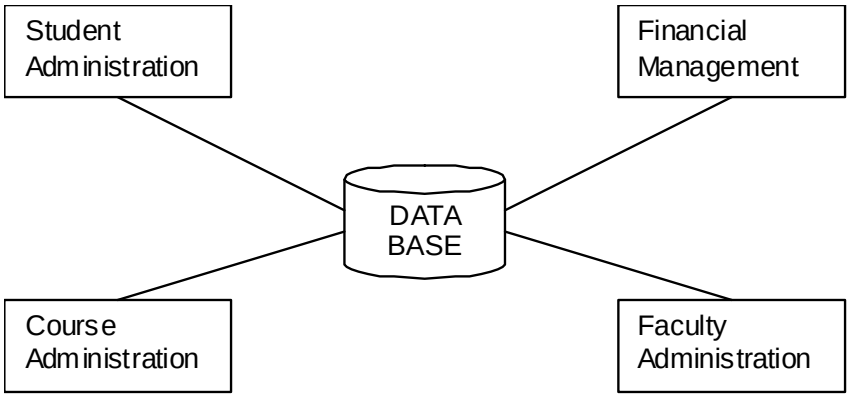


Fig.1.3: Database approach

It should be remembered that - a database is organised in such a way that a computer program can quickly select the desired piece of data. A database can further be defined as, it

- is a collection of interrelated data stored together without harmflul or unnecessary redundancy.
- stores data indepedent of programs, and any changes in data storage

structure or access strategy donot require changes in accessiing programs or queries.

- serves multiple applications in which each user has its own view of data. The data is protected from unauthorised access by security mechanism and concurrent access to data is provided with recovery mechanism.

Broadly, the objectives of the database approach are to make information access easy, fast, relatively inexpensive and flexible for the user. The specific objectives may be listed as follows :

- controlled data redundancy
- enhanced data consistency
- data independence
- application independence
- ease of use
- economical, and
- recovery from failure

1.4.1 Advantage of Database

Database approach provides the following benefits over the traditional file processing system :

Redundancy control :

In a file processing system, each application has its own data, which causes duplication of common data item in more than one file. This data duplication needs more storage space as well as multiple updation for a single transaction. This problem is overcome in database approach where data is stored only once.

Data consistency :

The problem of updating multiple files in file processing system leads to inaccurate data as different files may contain different information of the same data item at a given point of time. In database approach, this problem of inconsistent data is automatically solved wiith the

control of redundancy.

Thus, in a database, data accuracy or integrity or accessibility of data is enhanced to a great extent.

Data Independence :

This means that data and programs are independent. Most of the file processing systems are data dependent, which implies that the file structures and accessing programs are interrelated to each other.

Sharing of data and security :

Data in a database are shared among users and applications. In database approach data are protected from unauthorised access by some security mechanism.

1.5 DATABASE MANAGEMENT SYSTEM

The *database management system* (DBMS) is the interface between the users(application programmers) and the database(the data). A database management system is a program that allows user to define, manipulalte and process the data in a database, in order to produce meaningful information.

A DBMS is a set of *software programs* that controls the *organization, storage, management, and retrieval of data* in a database. It is a set of prewritten programs that are used to store, update and retrieve a database. The DBMS accepts requests for data from the application program and instructs the *operating system* to transfer the appropriate data. The followings are the examples of DBMS software :

Microsoft Visual FoxPro, MonetDB, MySQL , Oracle Database, PostgreSQL, SQL Anywhere, SQLite, FileMaker, IBM DB2, IBM UniVerse, Firebird, Microsoft Access, Microsoft SQL Server etc.

The following are examples of database applications :

- reservation systems, banking systems
- record/book keeping (corporate, university, medical), statistics
- bioinformatics, e.g., gene databases

- criminal justice
  - fingerprint matching
- multimedia systems
  - image/audio/video retrieval
- satellite imaging; require petabytes (10^{15} bytes) of storage
- the web
  - almost all data-intensive websites are database-driven;
- data mining (Knowledge Discovery in Databases) etc.

To complete our initial definitions, we will call the database and DBMS software together as a *database system*. The following figure depicts a database system.

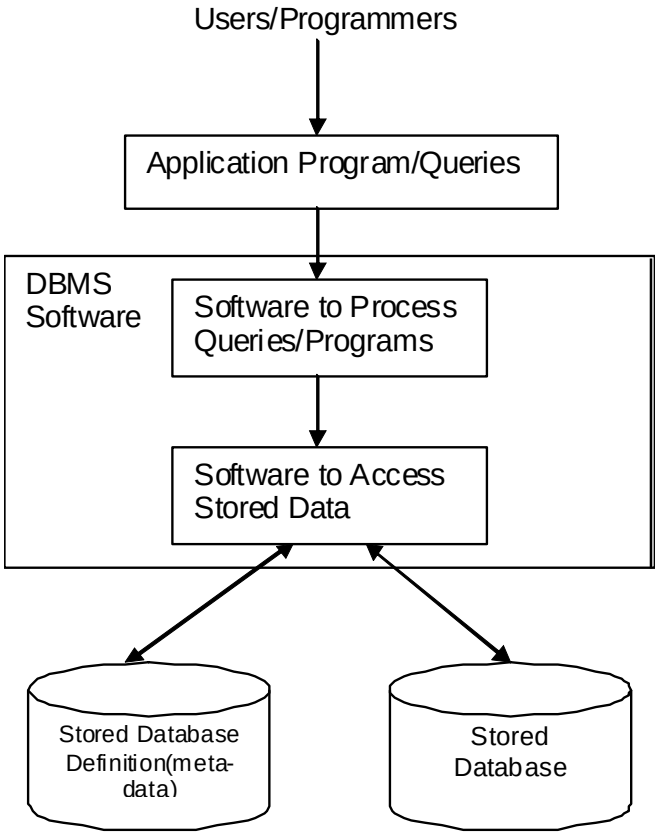


Fig. 1.4: A simplified database syste

1.5.1 Merits and Demerits of DBMS

Due to the centralised management and control, the database management system has numerous advantages, some of which are explained below :

Minimal data redundancy :

Centralized control of data avoids unnecessary duplication of data and effectively reduces the total amount of required data storage. It also eliminates the extra processing necessary to trace the required data in a large storage of data. Another advantage of avoiding duplication is the elimination of the inconsistencies that tend to be present in redundant data files.

Program-data independence :

The separation of metadata(data description) from the application programs that use the data is called ***data independence***. In the database environment, it allows for changes at one level of the database without affecting the other levels. With the database approach, metadata are stored in a central location called ***repository.*** This property of data systems allows an organizations data to change and develop without changing the application programs that process the data.

Efficient data access :

DBMS utilizes a variety of sophisticated techniques to store and retrieve data efficiently. This feature is especially important if the data is stored on external storage devices.

Improved data sharing :

Since, database system is a centralised repository of data belonging to the entire organiszation(all departments), it can be shared by all authorised users. Existing application programs as well as new application programs can share the data in the database.

Data Integrity :

Integrity of data means that data in database is always accurate, such that incorrect information cannot be stored in database. In order to maintain the integrity of data, some integrity constraints are enforced on the database.

Data security :

Database security is the protection of database from unauthorised users. The DBA(DataBase Administrator, we will discuss the responsibility of DBA in the next section) can define security rules to chueck unauthorised access to data. Some users may be given rights to only retrieve data, whereas others may be permitted to retrieve and edit the data. The DBA can formulate different rules for each type of access (retrieve, modify, delete, etc) to each piece of information in the database.

Enforcement of standards :

With the central control of the database, a DBA can defines and enforces the necessary standards. Applicable standards might include any or all of the following : departmental, organizational, industry, corporate, national or international. Standards can be defined for data formats to facilitate exchange of data between systems, naming conventions, display formats, terminology, report structure etc. The data repository provides DBAs with a powerful set of tools for developing and enforcing these standards.

Providing Backup and Recovery :

A DBMS must provide the facilities for recovering from hardware or software failures. The backup and recovery subsystem of the DBMS is responsible for recovery. For example, if the computer system fails in the middle of a complex update program, the recovery subsystem is responsible for making sure that the database is restored to the state it was in before the program started executing.

The **demerits** of the database approach are summarized below :

Complexity:

The provision of the functionality that is expected for a good DBMS makes the DBMS an extremely complex piece of software. Database designers, developers, database administrators and end-users must

understand this functionality to take full advantage of it. Failure to understand the system can lead to bad design decisions, which can have serous consequences for an organization.

Size :

The complexity and functionality makes the DBMS an extremely large piece of software, occupying many megabytes of disk space and requiring substantial amounts of memory to run efficiently.

Performance :

Typically, a file-based system is written for a specific application, such as invoicing. As a result, performance is generally very good. However, the DBMS is written to be more general, to cater for many applications rather than just one. The effect is that some applications may not run as fast as they used to.

Higher impact of a failure :

The centralization of resources increases the vulnerability of the system. Since all users and applications rely on the availability of the DBMS, the failure of any component can bring operations to a halting position.

Cost of DBMS :

The cost of DBMS varies significantly, depending on the environment and functionality provided. There is also the recurrent annual maintenance cost.

Additional Hardware cost :

The disk storage requirements for the DBMS and the database many necessitate the purchase of additional storage space. Furthermore, to achieve the required performance it may be necessary to purchase a large machine, perhaps even a machine dedicated to running the DBMS. The procurement of additional hardware results in further expenditure.

Cost of Conversion:

The cost of the DBMS and extra hardware may be significant compared with the cost of converting existing applications to run on the new DBMS and hardware. This cost also includes the cost of training staff

to use these new systems and possibly the employment of specialist staff to help with conversion and running of the system. This cost is one of the main reason why some organisations feel tired to their current systems and cannot switch to modern database technology.

CHECK YOUR PROGRESS

1. Select the correct answer :

(a) Which is not a DBMS package?

(i) Unify (ii) Ingress

(iii) IDMS (iv) All are DBMS packages

(b) Find the wrong statement

Database software

(i) provides facilities to create, use and maintain database.

(ii) supports report generation, statistical output, graphical output.

(iii) provides routine for backup and recovery.

(iv) all are correct.

(c) Which one of the following is not a valid relational database?

(i) SYBASE (ii) IMS

(iii) ORACLE (iv) UNIFY

(d) Centralized control is

(i) advantage of a DBMS (ii) disadvantage of a DBMS

(iii) Both (i) and (ii) (iv) None of the above

(e) Data are

(i) Raw facts and figures (ii) Information

(iii) Electronic representation of facts

(iv) None of these

1.6 DATABASE ARCHITECTURE

So far, we have come to know that a DBMS is a collection of interrelated files and a set of programs that allow several users to access and modify these files. A major purpose of a database system is to provide users with an abstract view of the data. That is the system hides certain details of how the data is stored and maintained. We can imagine that the whole database system is divided into levels. The generalised architecture of a database system is called the ANSI/SPARC (American National Standards Institute - Standards Planning and Requirements Committee) model. ANSI/SPARC three-tier database architecture is shown in the Fig.1.5.

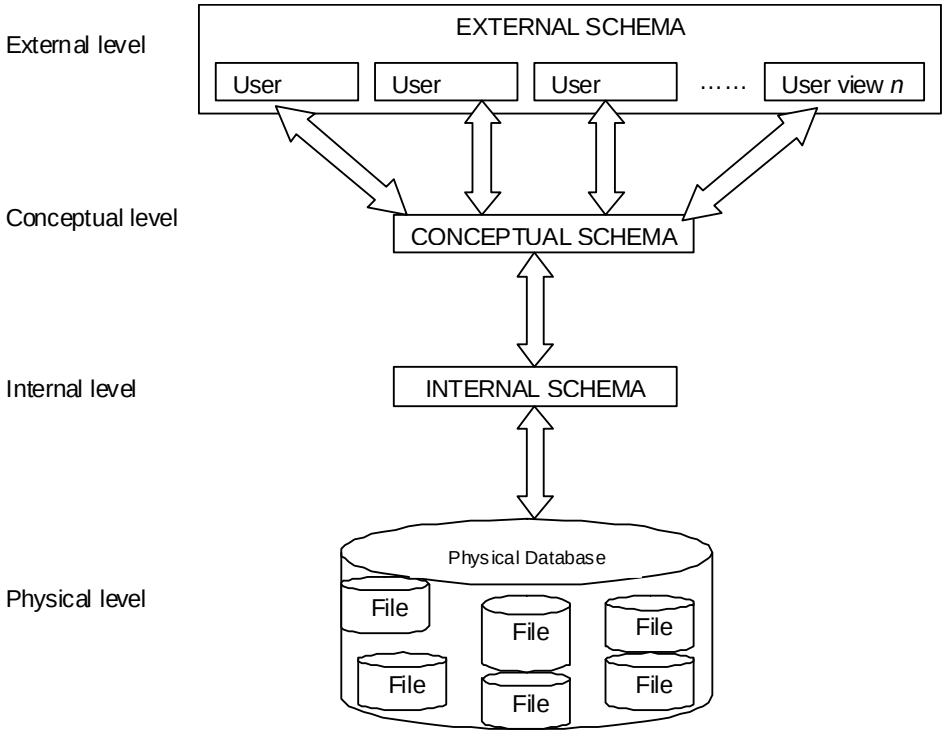


Fig.1.5: Three-tier database architecture

It consists of the following three levels :

- External level or view level,
- Conceptual level,
- Internal level or physical level.

External level :

The external level is the user's view of the database and closest to the users. This level describes that part of the database that is relevant to the user. Most of the users of database are not concerned with all the information

contained in the database. Instead, they need only a part of the database relevant to them. For example, even though the bank database stores a lot of information, an *account holder* would be interested only in the account details such as the current balance and the transactions made. They may not need the rest of the information stored in the account holders database. An e*xternal schema* describes each external view. The external schema consists of the definition of the logical records and the relationships in the external view.

In the external level, the different views may have different representations of the same data. The figure (Fig.1.6) describes the different views of the database related to different users.

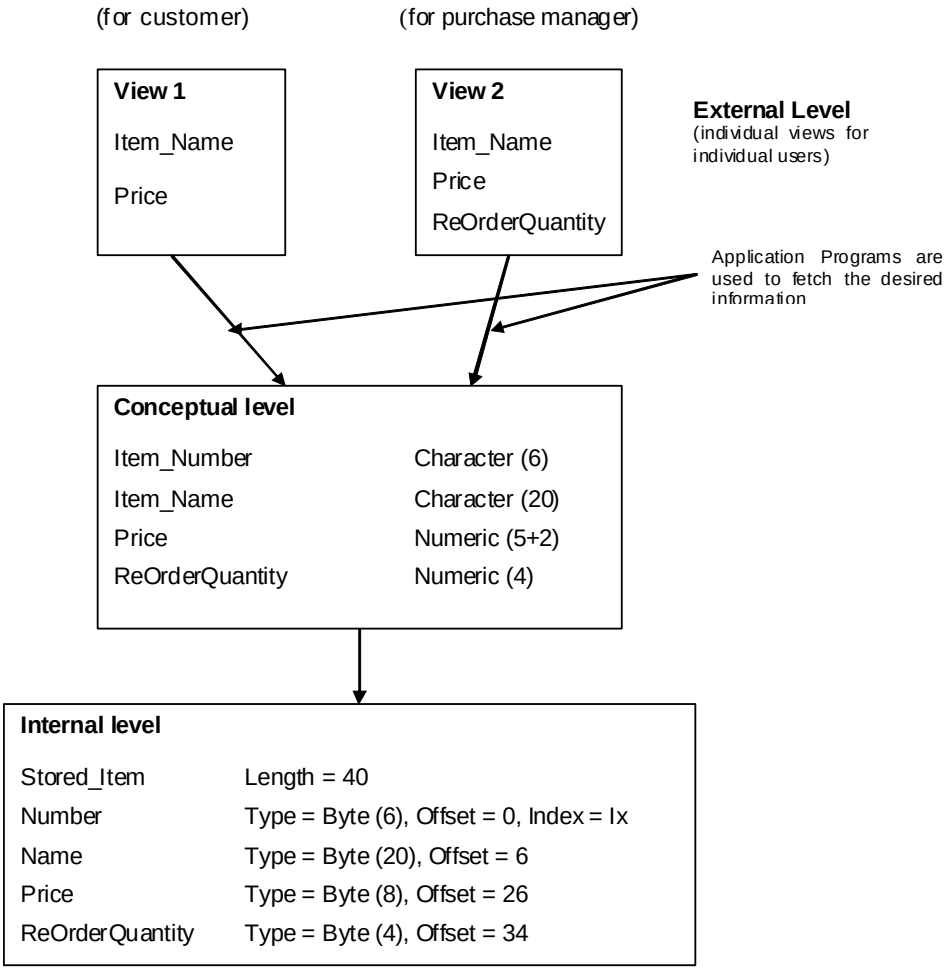


Fig.1.6: View of data at three-tier database architecture

Conceptual level :

Conceptual level is the middle level of the three-tier architecture. At this level of database abstraction, all the database entities and relationships among them are included. Conceptual level provides the community view of the database and describes what data is stored in the database and the relationships among the data. One conceptual view represents the entire database of an organization. It is a complete view of the data requirements of the organization that is independent of any storage consideration. The *conceptual schema* defines conceptual view. It is also called the *logical schema.* There is only one conceptual schema per database.

Internal level or physical level :

The lowest level of abstraction is the internal level. It is the one closest to physical storage device. This level is also termed as physical level, because it describes how data are actually stored on the storage medium such as hard disk, magnetic tape etc. This level indicates how the data will be stored in the database and describe the data structures, file structures and access methods to be used by the database. The *internal schema* defines the internal level. The internal schema contains the definition of the stored record, the methods of representing the data fields and accessed methods used.

1.7 DATA INDEPENDENCE

Data independence is the characteristics of a database system to change the schema at one level without having to change the schema at the next higher level. This characteristic of DBMS insulates the application programs from changing the data. The data independence is achieved by DBMS through the use of the three-tier architecture of data abstraction. There are two types of data independence -

(i) Logical data independence

(ii) Physical data independence

1.7.1 Logical Data Independence

Logical data independence is the ability to change the conceptual schema without having to change the external schema or application program. We may change the conceptual schema to expand the database(by adding a record type or data item) or to reduce the database(by removing a record type or data item). Only the view definition and the mapping need to be changed in a DBMS that supports logical data independence. After a logical change in the conceptual schema, the application program that refers to the external schema construct must work as before.

1.7.2 Physical Data Independence

Physical data independence implies the ability to change the internal schema without changinig the conceptual(or external) schemas. Changes to the internal schema may be required for improving the performance of the retrieval or updation operations. In other words, physical data independence indicates that the physical storage structures or devices used for storing the data could be changed without changing the conceptual view or any of the external views.

1.8 DBMS LANGUAGE

A DBMS must provide appropriate languages and interfaces for each category of users to express database queries and updates. After completing the design of a database, a DBMS is choosen to implement the database. It is important to first specify the conceptual and internal schemas for the database. Following languages are used for specifying database schemas :

- Data Definition Language (DDL)
- Storage Definition Language (SDL)
- View Definition Language (VDL)
- Data Manipulation Language (DML)

Data Definition Language (DDL)

DDL is a special language which specify the database conceptual schema using set of definitions. DDL allows the DBA or user to describe and name the entities, attributes and relationships required for the application, together with any associated integrity and security constraints. The DBMS has a DDL compiler whose function is to process DDL statements inorder to identify descriptions of the schema constructs. For example, look at the following DDL statements :

```
CREATE TABLE EMPLOYEE
(
   Fname        varchar(50    NOT NULL,
   Lastname     varchar(50)   NOT NULL,
   Eno          char(9)       NOT NULL,
   DOB          date,
   Address      varchar(60),
   PRIMARY KEY (Eno),
);
```

The execution of the above DDL statements will create a EMPLOYEE table as shown below :

EMPLOYEE				
Fname	Lastname	Eno	DOB	Address

Storage Definition Language (SDL)

Storage definition language is used to specify the internal schema in the database. In SDL, the storage structure and access methods used by the database system is specified by set of statements.

View Definition Language (VDL)

View definition language is used to specify user's views(external schema) and their mappings to the conceptual schema. There are two views of data - *logical view (*refers to the programmers view) and *physical view* (reflects the way how the data are stored on disk).

Data manipulation language (DML)

DML provides a set of operations to support the basic data manipulation operations on data in a database. Data manipulation is applied to all the three(conceptual, internal, external)l levels of schema. The part of DML that provides data retrieval is called ***query language***. DML provides the following data manipulation operations on a database :

- retrive data or records from database
- insert (or add) records to database
- delete records from database
- retrieve records sequentially in the key sequence
- retrieve records in the physically recorded sequence
- retrieve records that have been updated
- modify data or record in the database file

In other words, we can say that DML helps in communicating with the DBMS.

1.9 TYPES OF DBMS

The modern business environment revolves around the accuracy and integrity of information. The advancements in computer technology and rapid development of graphical user interface (GUI)-based applications, networking and communications have resulted in new dimensions in database management systems. A DBMS can be classified according to the number of users, the database site locations and the type and extent of use. These classifications are as follows:

On the basis of number of users :

- Single-user DBMS
- Multi-user DBMS

Onthe basis of site locations :

- Centralised DBMS
- Parallel DBMS
- Distributed DBMS
- Client/Server DBMS

On the basis of the type and extent of use:

- Transactional DBMS
- Decision support DBMS
- Data Warehouse

1.9.1 Centralised DBMS

The centralised database system consists of a single computer system associated with its peripherals, physically located in a single location. The computer system offers data processing facilities to the users located either at the same site, or, at geographically dispersed sites, through remote terminials. The management of the system and its data are controlled centrally from any one or central site. The following figure(Fig.1.7) shows a centralised database system.

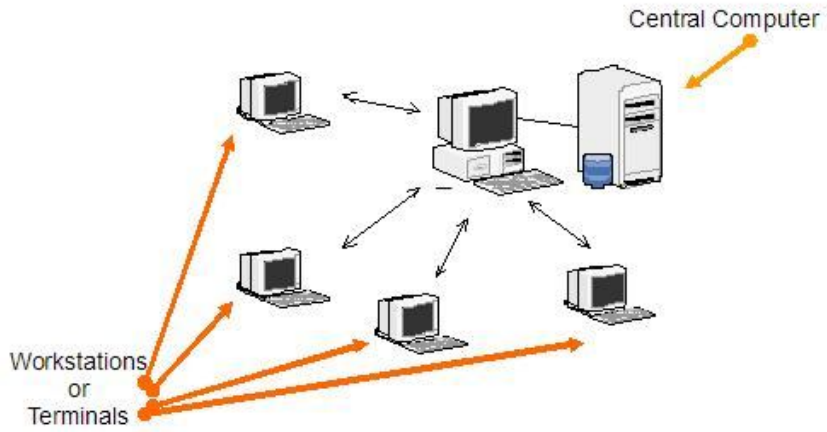


Fig.1.7: Centralised DBMS

Advantage of such a centralised system are given below :

- Most of the functions such as update, backup, query, control access etc. are easier with this system.
- Single database is shared accross the several different users.

Ofcourse, when the central site computer goes down, then every user is blocked from using the system untill it recover.

1.9.2 Parallel Database System

Parallel database system architecture consists of a multiple central processing units (CPU) and data storage disk in parallel as shown in the following figure(Fig.1.8). Hence, in such a system data processing speed is fast as well as input/output speed is also fast. The system which needs to process an extremely large number of transactions per second, in such a system parallel database system is used.

Advantage of such system is given below :

- Useful in the applications, which have to process an extremely large number of transactions per seconds (of the order of thousands of transactions per seconds)
- Performance of such database system is very high.

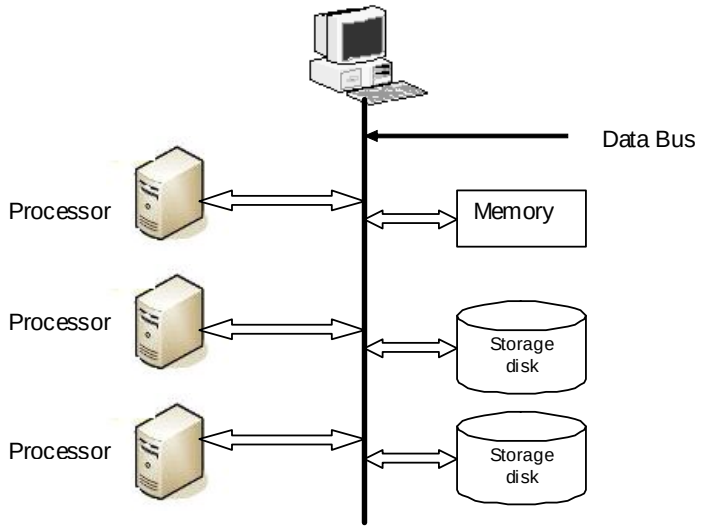


Fig.1.8: Parallel database system

1.9.3 Distributed DBMS

In a distributed database system, data are spread across a variety of different databases. These are managed by a variety of different DBMS softwares running on a variety of different computing machines having

different operating systems. These machines are actlually located on different sites and connected with some kind of communication networks as shown in the figure below (Fig.1.9). Thus, in a distributed database system, the organisation of data might be distributed on different computers in such a way that data for one portion (or department) of the organisation is stored in one computer and data for another department is stored in another computer. Each machine can have own data and applications and users of one machine can access the data of several other computers. The following figure (Fig.1.9) shows a distributed database system.

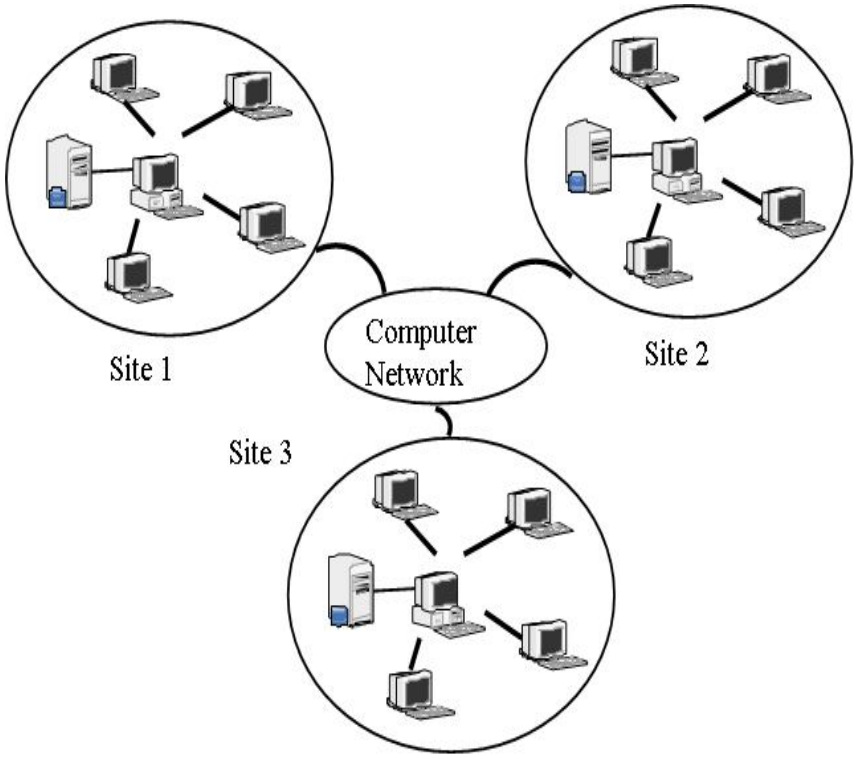


Fig.1.9: Distributed database

Advantages of distributed DBMS system is given below :

- Efficiency and performancce of this system is high.
- A single database can be shared across several distinct client systems.
- As data volume and transaction rates increase, users can grow

the system incrementally.

1.9.4 Client-Server DBMS

The client-server database system has two logical components namely - *client* and *server*. Clients are generally personal computers or workstations and the servers are the large workstations or mainframe computer system. The applications and tools of DBMS run on one or more client plateforms, while the DBMS softwares reside on the server. The server computer is called *backend* and the client computer is called *front-end*. The server and the clients are connected through networks. The clients send request to the server for performing some special tasks. The DBMS in the server side, in turn, process these requests and returns the results to the clients. The server handles parts of the job that are common to many clients, for example, database access and updates. The following figure(Fig.1.10) shows a client-server database model.

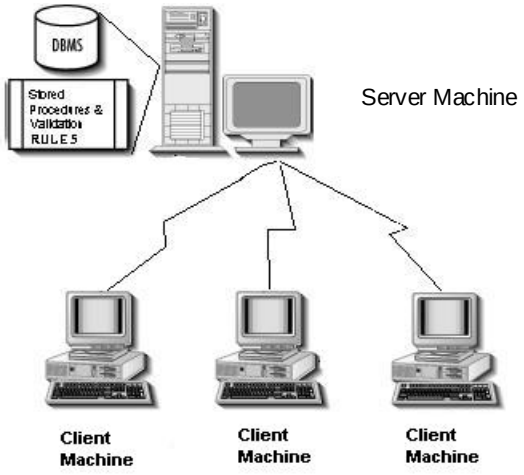


Fig. Client-Server database model

Advantages of such system is given below :

- Performance is high.
- A single database (on server) can be shared accross several distinct client system.
- More flexible as compared to the centralised system.

- Facilitates in more productive work by the users and making better use of existing data.

1.10 DATABASE ADMINISTRATOR

A database administrator (DBA) is a person or a group of person who is responsible for the environmental aspects of a database. A DBA is the central controller of the batabase system who designs database, controls and manages all the resources of database as well as provides necessary technical support for implementing policy decisions of database.

The role of a database administrator has changed according to the technology of database management systems as well as the needs of the owners of the databases.

Some of the roles of the DBA may include

- Installation of new software — It is primarily the job of the DBA to install new versions of DBMS software, application software, and other software related to DBMS administration.
- Configuration of hardware and software with the system administrator — In many cases the system software can only be accessed by the system administrator. In this case, the DBA must work closely with the system administrator to perform software installations, and to configure hardware and software so that it functions optimally with the DBMS.
- Security administration — One of the main duties of the DBA is to monitor and administer DBMS security. This involves adding and removing users, administering quotas, auditing, and checking for security problems.
- Data analysis — The DBA will frequently be called on to analyze the data stored in the database and to make recommendations relating to performance and efficiency of that data storage.

- Database design (preliminary) — The DBA is often involved at the preliminary database-design stages. Through the involvement of the DBA, many problems that might occur can be eliminated. The DBA knows the DBMS and system, can point out potential problems, and can help the development team with special performance considerations.
- Data modeling and optimization — By modeling the data, it is possible to optimize the system layouts to take the most advantage of the I/O subsystem.
- Responsible for the administration of existing enterprise databases and the analysis, design, and creation of new databases.

CHECK YOUR PROGRESS

2. Select TRUE or FALSE in the following statements:
   (i) The conceptual view is a view of the total database content.
   (ii) User's view is also called external view.
   (iii) The database schema and an instance of the database are the same thing.
   (iv) A view of a database that appears to an application program is known as schema.
   (v) Logical data independence indicates that the conceptual schema can be changed without affecting the existing external schemes.
   (vi) A database is a computer-based record keeping system whose over all purpose is to record and maintain information.

3. Multiple Choice

(a) A view of database that appear to an application program is known as –

(i) schema (ii) sub schema
(iii) virtual table (iv) none of these

(b) User's view is also called

(i) external view (ii) conceptual view
(iii) internal view (iv) none of these

(c) Which of the following schemas defines the stored data structures in terms of the database model used -

(i) external (ii) conceptual
(iii) internal (iv) none of these

(d) Data is processed by using

(i) DDL (ii) DML
(iii) DCL (iv) DPL

(e) Immunity of the conceptual (or external) schemas to changes the internal schemas is referred to as

(i) physical data independance
(ii) logical data independence
(iii) both (i) and (ii)
(iv) none of these

1.11 DATA MODELS

There are many basic structures that exist in a database system to organise the data. One of the fundamental characteristics of the database approach is that it provides some level of data abstraction by hiding details of data storage that are not needed by most database users. Data model is the main tool for providing this abstraction. **A data model is a set of concept that can be used to describe the structure of a database**. By structure of a database, we mean the data type, relationships, and constraints that should hold on the data. Most data models include a set of operation for

specifying retrievals and updates on the database. So, a data model defines :

- The logical data structure
- Data relationships
- Data consistency constraints.

1.11.1 Types of Data Models

Data models are categorised in different ways. The most general category is on the basis of the concepts they provide to describe the database structure. They are:

a) High level or Conceptual data model: It provides concepts that are close to the way many users perceive data. It uses concepts such as entities, attributes and relationships. An ***entity*** is an object that is represented in the databases. An ***attribute*** is a property that describes some aspects of an object. ***Relationship*** among objects is easily represented in high level data models which is sometimes called **Object-Based Model** because it uses objects as key data representation components that contains both data members/values and operations that are allowed on the data. The interrelationships and constraints are implemented through objects, links and message passing mechanisms. Object-Models are useful for databases where data interrelationships are complex, for example, Computer Assisted Design based components. The popular high level data model is Entity-Relationship Model.

b) Logical or implementation data model: It provides concepts understandable by end users but is not too far removed from the way data is organised within the computer. It hides some details of data storage but can be implemented on a computer system in a direct way. This model is sometimes called **Record-based data model** because it uses records as the key data representation components. It is used most frequently in current commercial DBMSs and includes the three most widely used data models – **Relational, Network and Hierarchical data model**.

c) Low-level or physical data model: It provides concepts that

describe the details of how data is stored in the computer by representing information such as record formats, record orderings and access paths. An access path is a structure that makes the search for particular database records much faster.

1.11.2 Traditional Data Models

The main classification of DBMSs on which it is based is the data model and the most often used data models in current commercial DBMSs are *Relational, Network* and *Hierarchical* models. Following are the brief discussions of these data models.

Relational data model

This is the most widely used database model that represents data as well as relationship among data in the form of tables. It looks like a file. Constraints are stored in a meta-data table. This is a very simple model and is based on a proven mathematical theory. Most relational data bases have high-level query languages and support a limited form of user views. Usually, the conceptual and internal schemas are not distinguishable, and a single DDL is used to describe all aspects of the database structure. We will be discussed more details in the subsequent unit. Fig.1.11 shows the relational representation of student database.

STUDENT	Name	Rno	Class	Major
	Smith	20	1	COSC
	John	18	2	COSC

Fig.1.11: Relational representation of student data base

Advantages of Relational data model:

a) The relational data model can be implemented with a personal computer having limited main memory and processing capability.

b) Very effective for small databases.

c) Much easier to use because it enables a computer system to accommodate a variety of enquiries in an efficient manner.

d) Very easy to represent the logical relationship among the data items since it use primary key to represent record relationships instead of pointers.

e) Relational model is very useful for representing most of the real world objects and the relationships among them.

Disadvantages of Relational data model:

In this model, as the size of the database increases, several problems may come into existence – system slowdown, performance degradation and data corruption.

Network data model

In this model, data is represented as *records* and relationship as *links*. Figure 1.12 shows the network representation for the database STUDENT with grade, course and section. In the figure(Fig.1.12), record types are shown as rectangles and set types are shown as labelled directed arcs. The network model is also known as the Computer Data System Language DataBase Task Group (CODASYL DBTG) model. It has an associated record-at-a-time language that must be embedded in a host programming language.

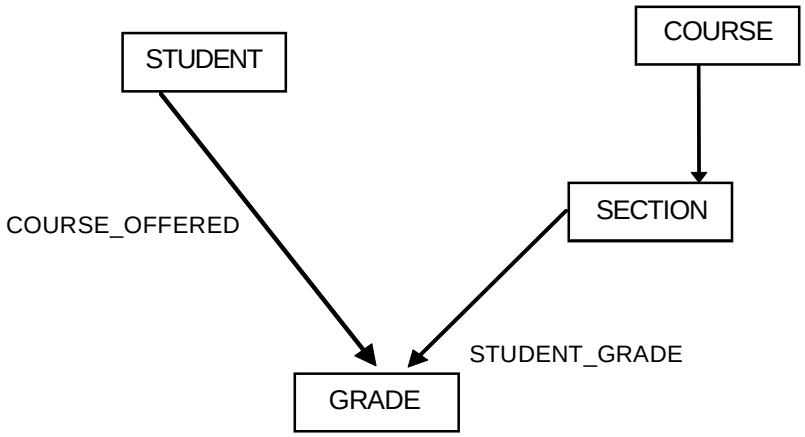


Fig1.12: A network schema

Advantages of network model:

a) Useful to represent the records having many-to-many relationships.

b) Problem of inconsistency does not exist in this model because a data element is physically located at just one place.

c) Searching a record is easy since there are multiple access paths to a data element.

Disadvantages of network model:

a) All the records are maintained using pointers and hence the whole database structure becomes very complex.

b) Insertion, modification (update) and deletion of any record require pointer adjustments.

Hierarchical data model: It represents data as hierarchical tree structures. Each hierarchy represents a number of related records. There is no standard language for the hierarchical model, although most hierarchical DBMSs have record-at-a-time languages. For example, an institute has a number of programmes to offer, each programme has a number of courses, and each course has a number of students registered in it. The following figure(Fig.1.13) depicts, the four entity types Institute, Programme, Course and Student make up the four different levels of hierarchical structure.

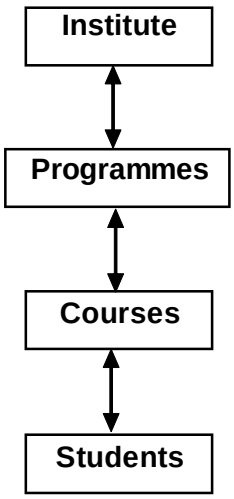


Fig.1.13: A simple hierarchy

Advantages of Hierarchical Model:

a) It is a simple, straightforward and natural method of implementing record relationships.

b) Useful when there is some hierarchical character in a database.

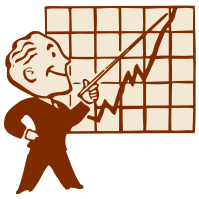
Disadvantages of Hierarchical Model:

a) It cannot represent all the relationship that occurs in the real world.

b) It cannot demonstrate the over-all data model for the enterprise because of the non availability of actual data at the time of designing the model.

c) Used only when hierarchical structure in the databases.

d) Insertion, modification (update) and deletion of any record require lots of adjustments.



CHECK YOUR PROGRESS

4. State whether the following are true or false (T/F)

i) Data model gives logical as well as physical data structure

ii) Entity relationship model is the example of conceptual data model

iii) Conceptual data model is also called object-based model.

iv) High level data model use records as the key data representation component.

v) Low level data model describes the details of how to store data in computer.

vi) Logical data model provide concept which is complicated for end users.

vii) Implementation data model is also a record-based data model.

viii) Access path can search database record much faster.

ix) Network model represents data as hierarchical tree structure.

x) Hierarchical model is the current commercial DBMS.

1.12 LET US SUM UP

1. The traditional file approach to information processing has for each application a separate master file and its own set of application programs, COBOL language used to write these application programs.
2. A database is a single organized collection of instructed data, stored with a minimum of duplication of data items so as to providea consistent and controlled pool of data.
3. A database management system (DBMS) is a collection of programs that enables users to store, modify and extract information from a database as per the requirements. DBMS is an intermediate layer between programs and the data. Programs access the DBMS, which then accesses the data.
4. According to the ANSI/SPARC architecture of a database system the whole database is divided into the following three levels :
   - External level or view level
   - Conceptual level
   - Internal level or physical level
5. Logical data independence indicates that the conceptual schema can be changed without affecting the existing external schema.
6. Physical data independence indicates that the physical storage structures or devices used for storing the data could be changed without necessitating a change in the conceptual view or any of the external views.
7. DBMS provide appropriate languages and interfaces for each category of users to express database queries and updates.
8. Following laguages are used for specifying database schemas :
   - Data definition language (DDL)
   - Storage definition language (SDL)
   - View definition language (VDL)
   - Data manipulation language (DML)
9. DBMS can be classified according to the number of users, the

database site locations and the type and extent of use.

On the basis of site locations, the followings are types of DBMS :

- Centralised DBMS
- Parallel DBMS
- Distributed DBMS
- Client/Server DBMS

On the basis of the type and extent of use DBMS are of following types :

- Transactional DBMS
- Decision support DBMS
- Data Warehouse

10. DBA is database administrator who is responsible for maintaining database. A DBA provides the necessary technical support for implementing policy decisions of database.

11. A data model is a set of concept that can be used to describe the structure of a database.

- Data models are of three types: Logical, Physical and Conceptual.
- The most often used data models in current commercial DBMSs are Relational, Network and Hierarchical models.

1.13 ANSWERS TO CHECK YOUR PROGRESS

1. a. (iv) b. (iv) c. (ii) d. (i) e. (i)
2. (i) F (ii) T (iii) F (iv) F (v) T (vi) T
3. a. (ii) b. (i) c. (ii) d. (ii) e. (i)
4. (i) F (ii) T (iii) T (iv) F (v) T
   vi) F vii) T viii) T ix) F x) T

1.14 FURTHER READINGS

1. An introduction to Database Systems, C. J. Date, Pearson Education.
2. An introduction to Database Systems, B.C. Desai.
3. Database System Concepts, S. K. Singh, Pearson Education.
4. Principles of Database systems, J.D. Ullman.

1.15 MODEL QUESTIONS

1. What is file based approach of database? Explain its limitations?
2. Explain three level database architecture. What are its objectives?
3. What do data independence and its types? How data indep-endence is achieved?
4. What are advantages of DBMS?
5. Discuss the main disadvantages of a Traditional file approach?
6. Discuss the main disadvantages of DBMS?
7. Difference between DBMS approach & traditional file approach.
8. Mention the differences between text files and database files. Why are database files preferred in a commercial organization?
9. Write short notes on :
   (i) Data independence
   (ii) Database
   (iii) DBMS
   (iv) DBMS Architecture
   (v) Client-server database model
   (vii) Distributed database system
   (vii) Physical Data Independence
   (viii) traditional File Approach
   (ix) Centralised database system

(x) DBMS language

10. What is logical data independence and why is it important?
11. Explain the difference between logical and physical data independence.
12. Describe the three levels of data abstraction?
13. What do you mean Database Language? What are the different types of data base language?
14. Explain the role of a Database Administrator.
15. Describe conceptual, logical and physical data model.

*****

UNIT- 2 DATABASE DESIGN USING ER MODEL

UNIT STRUCTURE

2.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- explain the need of Entity-Relationship modeling;
- define the terms entity, entity type, attribute, attribute value, primary key, relationship, relationship type;
- describe how entities, attributes and relationship are used to model data;
- determine the degree of a relationship;
- produce an entity-relationship diagram based on a simple description of a domain

2.2 INTRODUCTION

Previous unit is an introductory unit on database management system. We are already acquainted with databases and various introductory concepts associated with database. In this unit we will present a high-level conceptual data model, the ***Entity-Relationship*** (ER) model. It was first introduced by Peter Chen in 1976 to facilitate database design. Conceptual modeling is an important phase in designing a successful database. A conceptual data model is a set of concepts that describe the structure of a database and associated retrieval and updating transactions on the database. ER modeling should always be completed before we implement a database. We also discuss about ER diagrams which are the diagrammatic notation associated with the ER model.

2.3 BASIC CONCEPTS OF ENTITY RELATIONSHIP

The database is a collection of entities and relationship among the entities. The Entity-Relationship(ER) model is an effective as well as standard method of communication amongst different programmers, designers and end-users who tend to view data and its use in different ways. The ER model describes data as entities, their relationships and attributes.

2.3.1 Entities and Attributes

An ***entity*** is an object or a thing in the real world with an independent existence which can be uniquely identified. An entity may be an object with a physical existence such as a person, employee, car, book, or plant. Or, it may be an object with a conceptual existence such as a company, an organization, a job or an event. An entity in ER model is represented by a rectangle containing the entity type name, such as Employee.

A database may contain many different types of related entities. An entity is represented by a set of ***attributes,*** that is, descriptive properties possessed by all members of an entity set. The *Employee* entity, for example, has the attributes Employee_id, Name, Designation, Salary and Contact_no.Attributes are represented as oval in ER model. The following figure shows two entities ***Employee*** and ***Student*** with their attributes and attribute values:

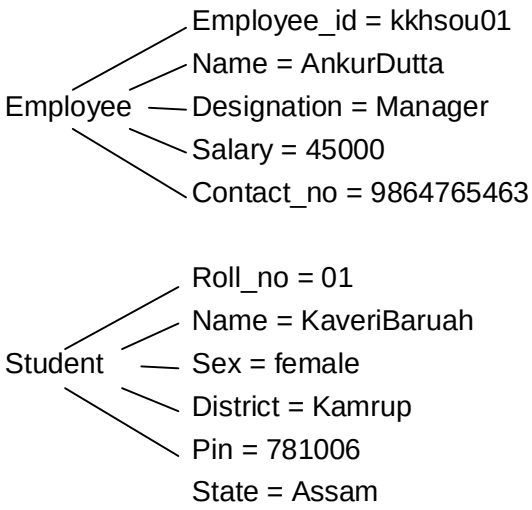


Fig.2.1: Two entities with their attributes

Several types of attributes occur in ER model which are as follows:

- Simple and Composite attribute,
- Singlevalued and Multivalued attribute,
- Stored and Derived attribute,
- Complex attribute.

Simple and Composite Attribute

Attributes that are not divisible are termed as ***simple*** or ***atomic*** attributes. The attributes *age*, *sex,* etc. are examples of simple attributes.

An attribute is considered composite if it comprises two or more other attributes. i.e., a ***composite*** attribute is an attribute that can be further subdivided. For

example, the attribute *address* can be subdivided into Street, City, State, and Pin code.

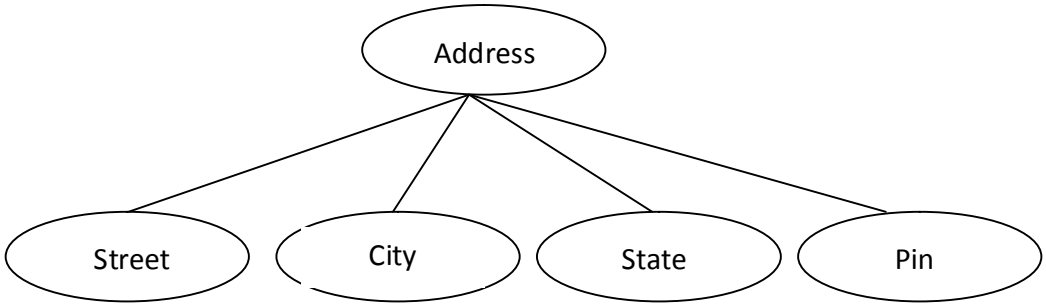


Fig. 2.2**:** Composite Attribute

Singlevalued and Multivalued Attribute

Most attribute have a single value for a particular entity; such attributes are called ***singlevalued***. For example, a person can have only one '*date of birth*', '*age*' etc. But it can be simple or composite attribute.That is, the 'date of birth' is a composite attribute , 'age' is a simple attribute. But both are singlevalued attributes.

Multivalued attributes can have multiple values. For instance, a person may have multiple degrees, phone numbers, email address etc.

Stored and Derived Attribute

An attribute that supplies a value to the related attribute is called ***stored*** attribute.For example 'Date of birth' of a person is a stored attribute. The value for the attribute 'age' can be derived by subtracting the 'Date of Birth' from the current date.

Derived attributes are the attributes which are computed from other attributes. For example, *age*, and it's value is derived from the stored attribute *Date of Birth*.

Complex Attribute

A complex attribute are attribute that is both composite and multivalued. Composite and multivalued attributes can be nested arbitrarily. Arbitrary nesting is done by grouping components of a composite attribute between parentheses () and separating the components with commas, and by displaying multivalued attributes between braces { }.

For example, for an entity ***person***, let us consider an attribute ***Person_info*** which will give the full name and address of a particular person. The attribute, *Person_info* for a person can be specified as shown in the following figure. In the figure, we can see that the *Name* and *Address* are themselves composite attributes.

{Person_info({Name(First_name, Middle_name, Last_name)},
{Address(House_no, Street_no,Street_name), City, State, Pin}}
Fig.2.3: A complex attributesPerson_info

2.3.2 Entity Type and Entity Set

The set of all entities that have the same properties or attributes is called an ***Entity Type***. The entity type is described by its name and its attributes.

For example, to store the records of the employees working in an organization, the *Employee* entity has the same attributes, like employee_id, name, address, salary and so on. Each entity has its own values for each of its attribute.

An ***Entity Set*** is the collection of all the entities of a particular entity type. Sometimes, the entity set is also known as **Extension**of the entity type.For example, EMPLOYEE refers to both an Entity type and the set of all the employee entities in the database. The figure below illustrates this.

ENTITY TYPE Name: EMPLOYEE
ATTRIBUTES: Employee id, Name, Address

ENTITY SET:

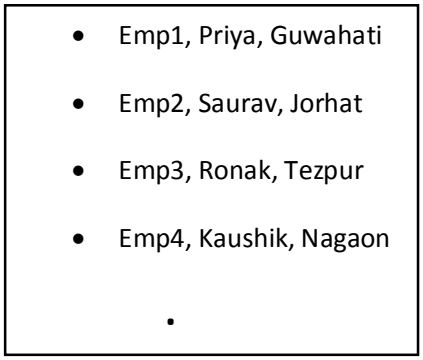


Fig.2.3

Each entity in the entity set of a particular entity type has an attribute whose value is distinct and can uniquely identify each entity in the set. Such attributes are called ***Key attributes***.

Entity types can be classified as

- Strong entity or Regular entity
- Weak entity

Those entities that have an independent existence and can be uniquely identified by key attributes are called ***Strong entities*** and those entitieswhose existence depend on the existence of some other entity and cannot be uniquely identified by key attributes are called ***weak entities***. As an example, the Employees working in the different branches of an organization may have dependents. In this case, the entities EMPLOYEE and BRANCH are Strong entities as they can exist independently. But the entity DEPENDENT is a Weak entity as it can exist only if the EMPLOYEEs have dependents.

CHECK YOUR PROGRESS

1. Fill in the blanks:

a. Any object that has existence in the real world is called _____.

b. The characteristics or a property of the entities is defined as _____.

c. The _____ attribute can be further subdivided into other attributes.

d. Multivaluedattributes can have _____ values.

e. The attribute which are computed from other attribute values is called _____.

f. The set of all entities that have the same properties or attributes is called an _____.

g. An _____ is the collection of all the entities of a particular entity type.

h. _____ is the distinct attribute of an entity, in an entity set, whose value can uniquely identify each entity in the set.

i. _____ are the entities that have an independent existence and can be uniquely identified by its key attributes.

j. Weak entity is _____ on other entities.

2.3.3 Relationship

A relationship is defined as an association among two or more entities. The entities that are involved in a relationship are called the ***participants*** of that relationship. The number of entities or participants in a relationship is called the ***degree*** of that relationship. If the relationship is between two entities then the relationship is a Binary relationship, that is, the degree of the relationship is 2. If the number of participating entities is three, then the relationship is a Ternary relationship. Relationships are denoted by a diamond shape in the ER model.

The Fig.2.4 (a) below shows an example of a binary relationship between two entities Employee and Branch.

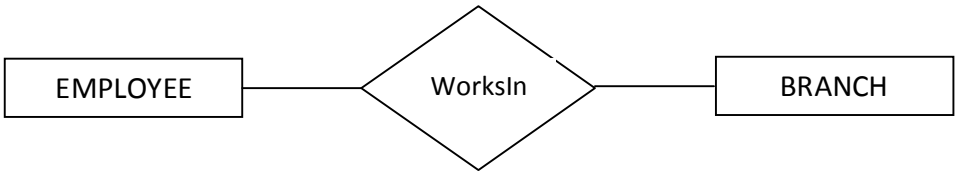


Fig.2.4(a) : Binary Relationship

The Fig.2.5(b) below shows an example of a ternary relationship "Supplies", between the entities Supplier, Project and Part. Suppose each part is supplied by a unique supplier, and is used for a given project within the organization.

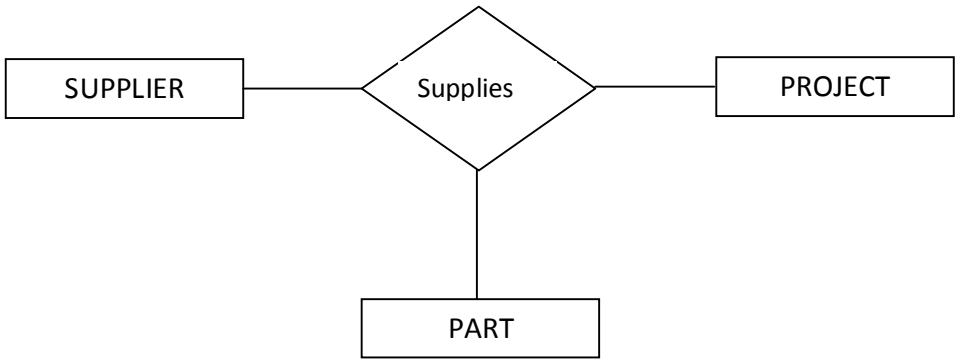


Fig.2.4(b) : Ternary Relationship

A particular occurrence of a relationship is called a ***relationship instance***. It includes one occurrence from each of the participating entity types.

2.3.4 Constraints

There are certain constraints on the relationship types that limit the possible combinations of entities participating in the corresponding relationship set. For example, suppose an employee may be appointed to only one department in the organization. This is a constraint and it needs to be described in the relational schema.

There are two main types of relationship constraints ---- cardinality ratio, and participation.

Cardinality Ratios for Binary Relationships:

The maximum number of relationship instances that an entity can participate in is called the ***cardinality ratio***. The possible cardinality ratios for binary relationships are

- One-to-one (1:1),
- One-to-many (1:N)
- Many-to-many (M:N)

The cardinality ratios on ER diagrams are shown by displaying 1, N and M on the diamonds. Suppose in an organization if an employee supervises only one branch, then the cardinality ratio of the relationship between the participating entities, Employee and Branch is 1:1 and is shown in the figure below.

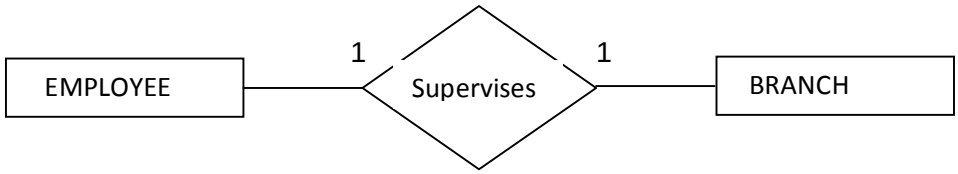


Fig.2.5(a)

A branch can have many employees working in it, or in other words many employees' works in a branch of the organization. The cardinality ratio for the Dept_Emp relationship is 1:N. This can be illustrated by the following figure.

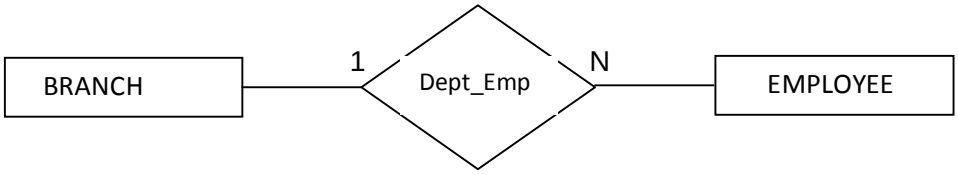


Fig.2.5(b)

If many employees can work in many projects, then the cardinality ratio for the relationship isM: N and can be shown as:

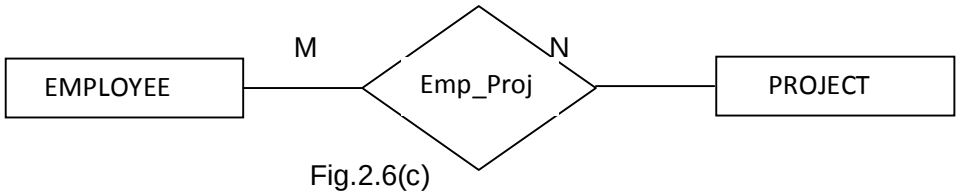


Fig.2.6(c)

Participation:

The participation constraint of a relationship determines whether the existence of an entity depends on its being related to another entity through the relationship. Participation constraints are of two types: ***total***and***partial participation***. In ER model, total participation is represented by a double line connecting the participating entity types, and the partial participation is represented by a single line.

If every entity, in the entity set E_1, must be related to another entity E_2 through a relationship, then the participation of E_1 in the relationship is said to be total. Total participation is also known as ***existence dependency***. For example,Fig.2.5(d)the employees working in an organization may have dependents on them. This means that the Dependent entity can exist only if there is aEmployeeentity, otherwise not. So in this case, the participation of the Dependent entity in the relationship is total.

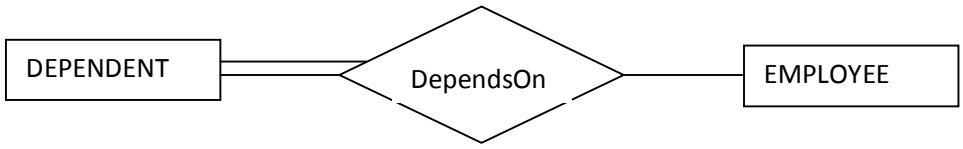


Fig.2.5(d)

Participation of an entity in a relationship can also be ***partial.*** Let us take the example of Fig.2.5(a). All the employees working in the branch are not supervising the branch. So here the participation of the Employee entity in the 'Supervises' relationship is partial, that is, only a part of the employee is given the responsibility to supervise the works of the branch.

2.4 RELATIONSHIP SET

A **relationship type** is a set of associations among entity types. For example, the *employee* entity type is related to the *branch* entity type because each employee is a member of a branch.**Relationship set** is the collection of similar relationships. An n-array relationship set R relates n entity sets E1 ... En.Each

relationship in R involves entities $e_1 \in E_1$, ….., $e_n \in E_n$. Same entity set can participate in different relationship sets.

CHECK YOUR PROGRESS

2. Say True or False:

i. Association among the attributes of an entity is relationship.

ii. Participants of a relationship are the number of entities that are involved in the relationship.

iii. If the degree of a relationship is three, then it is a Ternary relationship.

iv. A particular occurrence of a relationship is called a relationship instance.

v. Constraints do not put any limit on the combinations of entities participating in the corresponding relationship set.

3. Fill in the blanks:

a. _____ and _____ are the two types of constraints on a relationship.

b. Cardinality ratio is the _____ number of relationship instances that an entity can participate in.

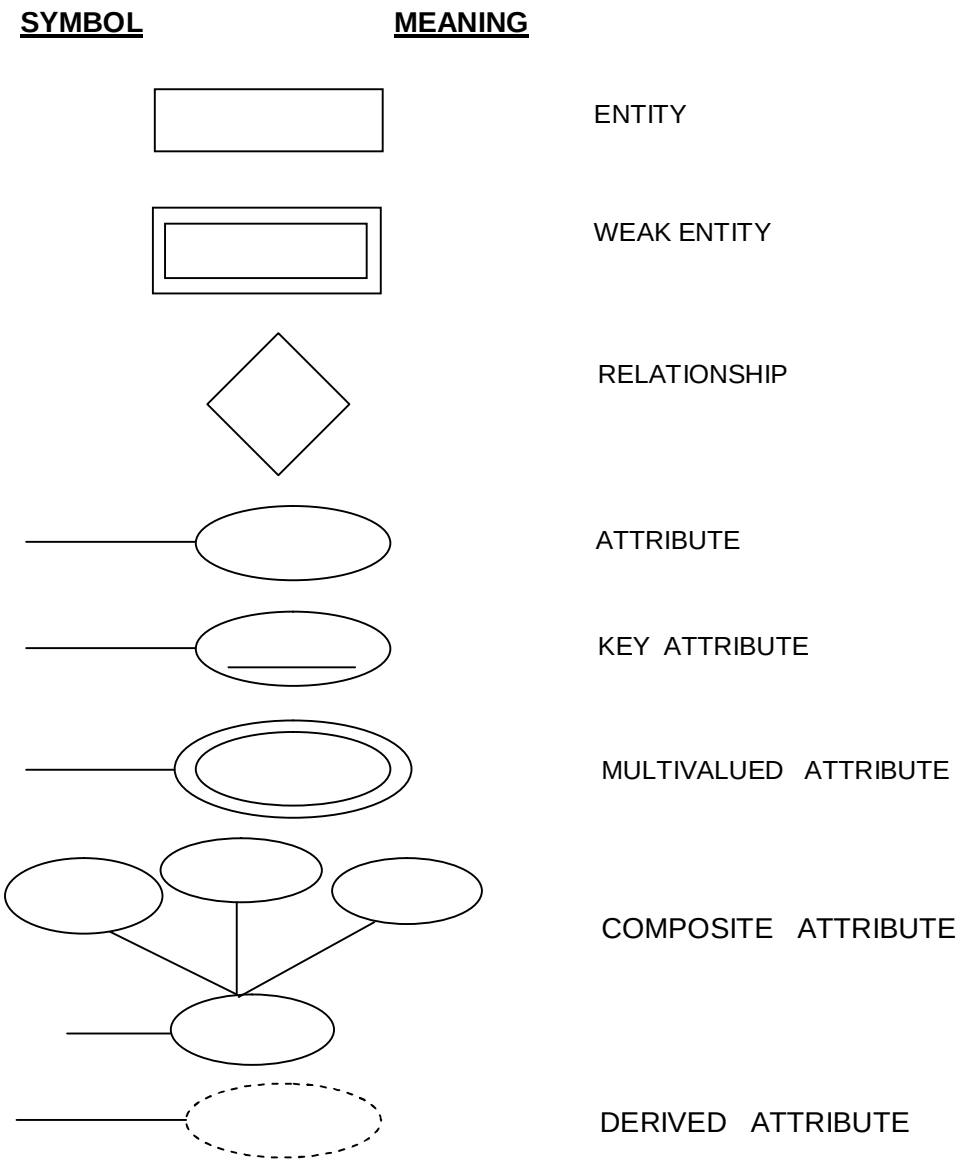
c. In an ER-diagram, the entities, attributes and relationships are represented as _____, _____ and _____ respectively.

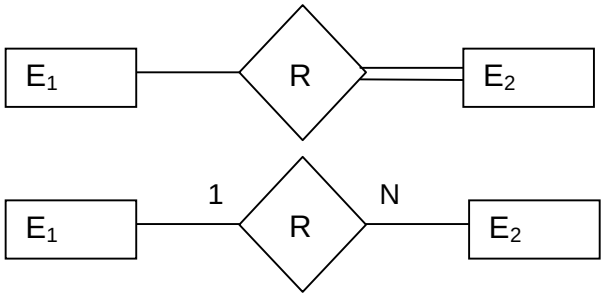
d. Participation constraints in relationships are of two types, _____ *and* _____ *participation*.

e. _____ is the collection of similar relationships.

2.5 ENTITY RELATIONSHIP DIAGRAM

The Entity-Relationship (ER) diagram illustrates the logical structure of a database. It defines the relationships between the different entities of the database application. It is a graphical representation of an ER model. The figure below shows the symbols or notations that are used for ER diagrams. They are the building blocks of ER diagram.





TOTAL PARTICIPATION OF E_2 IN R

CARDINALITY RATIO 1:N FOR E_1:E_2 IN R

An example of an ER-diagram of a conceptual model for the database of an organization is depicted in Fig.2.6.Suppose only a part of the organization is to be represented in the database.

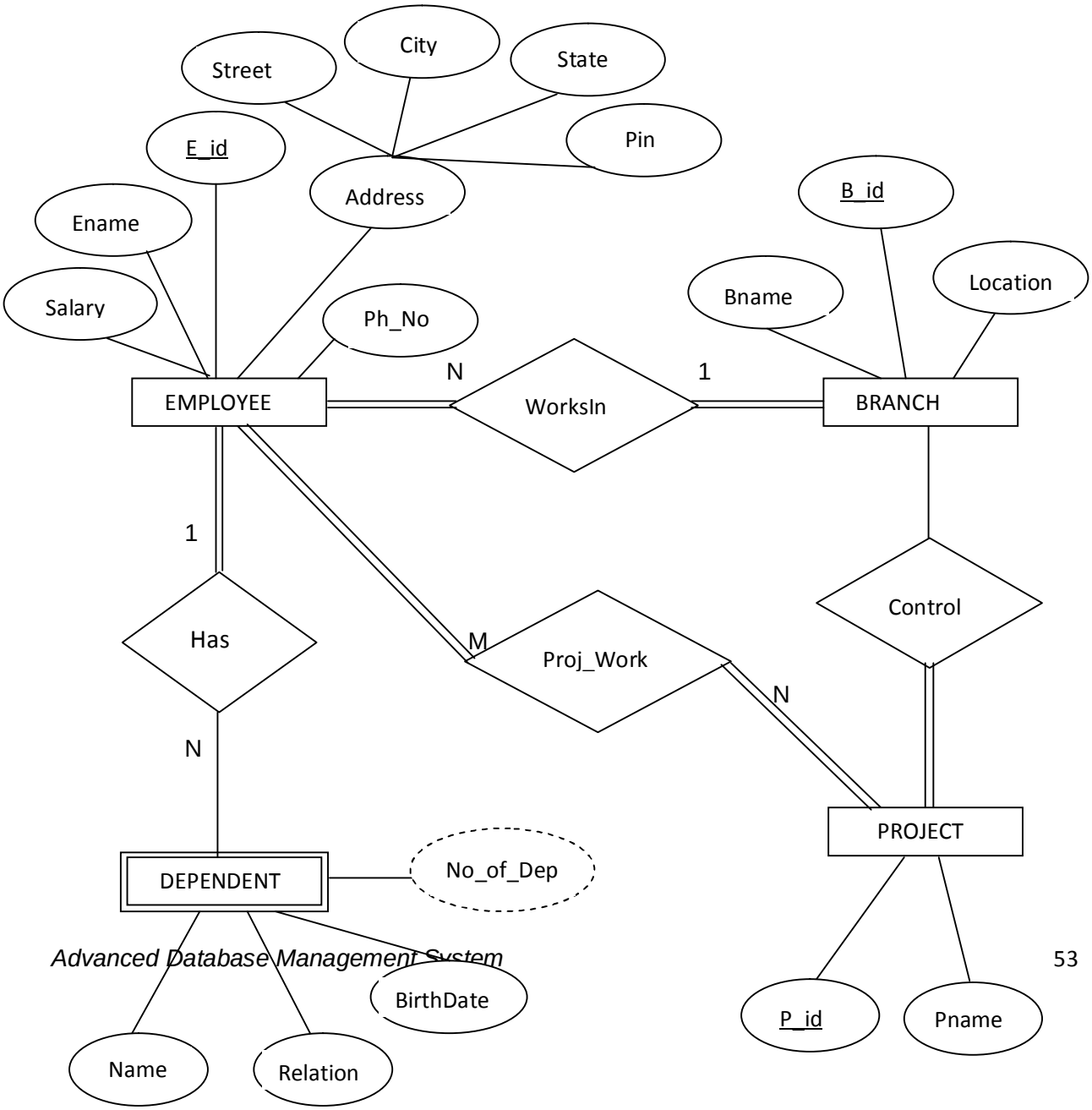


Fig.2.6

It can be seen from the above diagram that ER diagram includes entity sets, attributes, and relationship sets.

2.6 GENERALIZATION AND SPECIALIZATION

There are some extensions to the basic ER Model. These are ***generalization***, ***specialization*** and ***aggregation***. Generalization and Specialization are important concepts in database modeling. These are different approaches to the design processes of the logical database.

Generalization is a bottom-up design process of the logical database. It is a process of extracting shared characteristics from two or more entities (or classes) and combining those entities into a higher level or generalized entity set. The entity sets that are combined are called the lower-level entity sets. It is used to emphasize the similarities among the low-level entity sets and to minimize their differences. Let us consider the example of an Employee entity. The attributes of Employee may be emp_id, emp_name, contactNo. There may be different types of employees working for the organization. Then Employee is the higher level entity and Full-time employee and Part-time employee are the lower-level entity sets. It is denoted by a triangular component labeled as “IS A” as shown below(Fig.2.7)

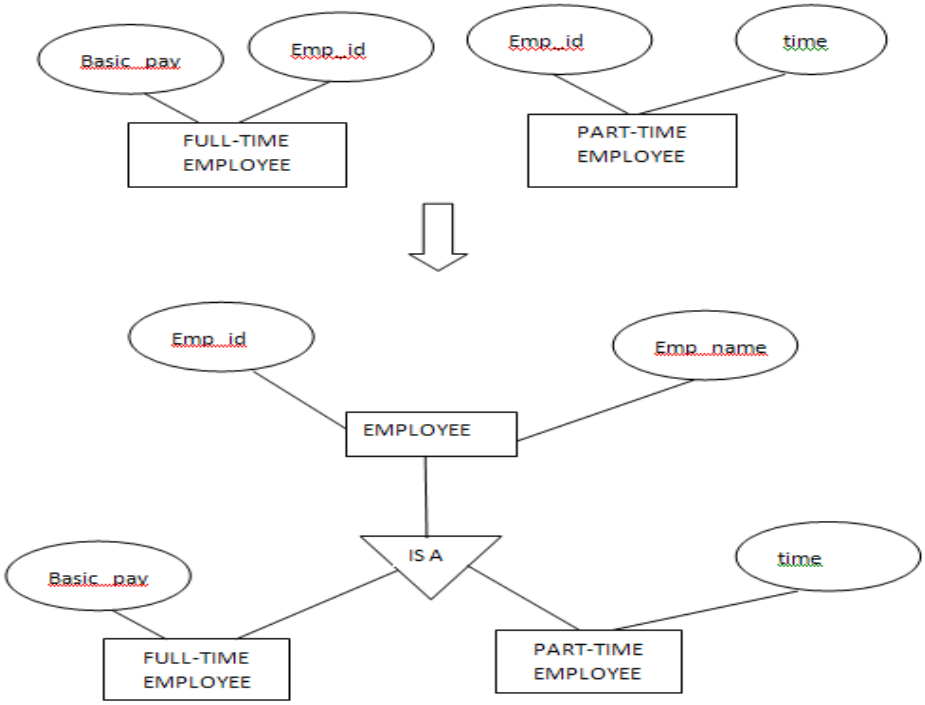


Fig. 2.7

In the above diagram, the lower-level entities, such as, Full_time Employee and Part_Time Employee are generalized to form a higher-level entity called *Employee*.

Specialization, on the other hand, is a top-down design process. It is the reverse of generalization and the process of maximizing the differences between members of an entity by identifying their distinguishing characteristics. It means creating new entities from existing ones. If certain attributes or relationships apply only to some of the entities from the entity set, then a subclass or a low-level entity can be created from that entity. In specialization, the subsets of the higher level entity sets forms the lower-level entity sets. The higher-level entity set is also called the superclass. For example, the Employee entity can be specialized into full time and part time employees or as Manager, Assistants and Executives. The subclasses are formed based on some properties of the entities in the superclass.

2.7 AGGREGATION

Aggregation treats a relationship set as an entity set so that the relationship set can participate in other relationships.Thus aggregation is an abstraction which allows relationships to be viewed as higher-level entities. Suppose in an organization, at the most one employee monitors the projects that are sponsored by the branches.

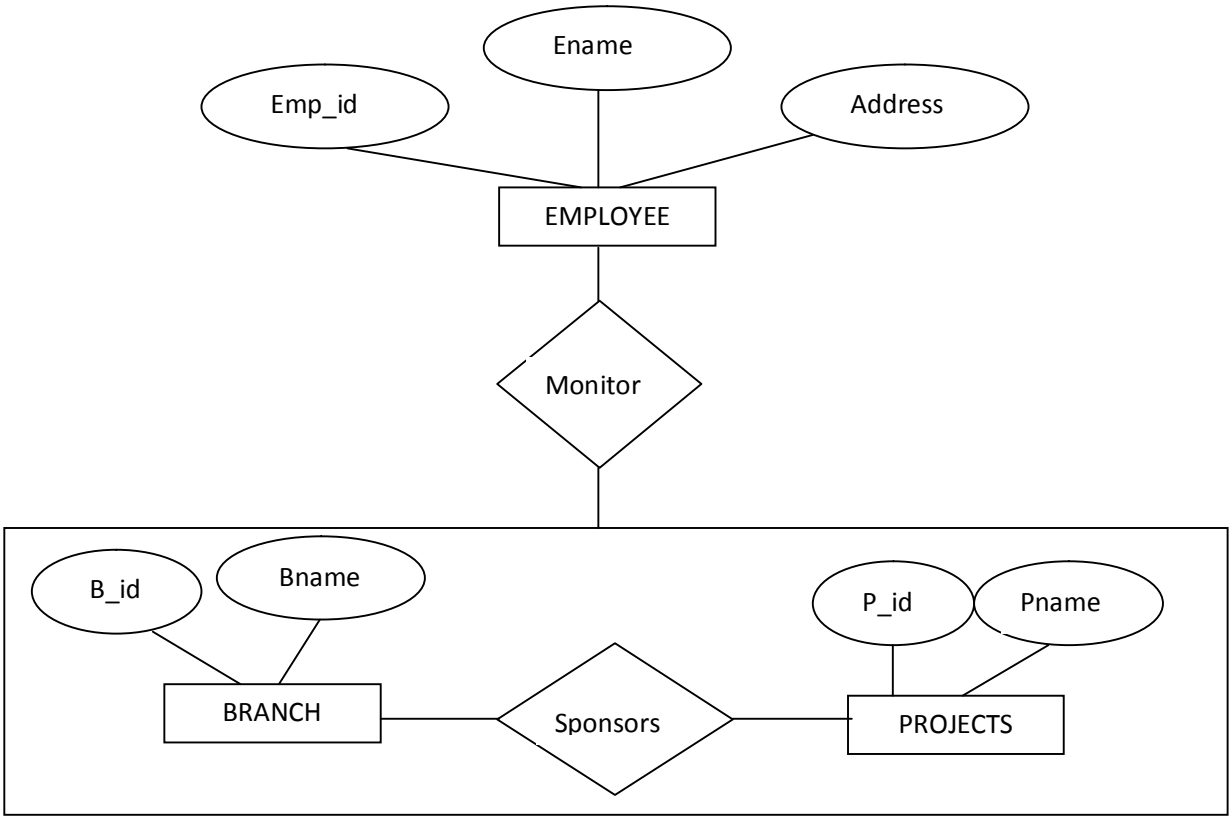


Fig.2.8

In the above example, the relationship set 'Sponsors' and the entity sets Branch and Projects are treated as a higher-level entity so as to participate in the relationship 'Monitors'.

2.8 STRUCTURE OF RELATIONAL DATABASE

A relational database consists of a collection of tables. Tables in Relational Database Systems are known as ***relations***. A relation can be defined by a set of rows and columns. Rows and columns in RDBMS are known as ***tuples*** and

fields respectively. Each relation in the database has a unique name. The entities form the relations and the attributes are the attributes or fields of that relation.

In a relational model, all data are kept in a relation. Each tuple of a relation represents an instance of the entity set. The number of attributes is called the ***degree*** and the number of tuples is called the ***cardinality***.

2.9 DIFFERENT TYPES OF KEYS

There are different types of keys in relational database model. ***Key*** is the attribute of an entitythat identifies one or more instances of the entity set. It can be a single attribute or a combination of attributes of the entity set. In other words, a key is used to uniquely identify the tuples of a relation in the relational database system.

Candidate Keys

If a relation schema has more than one key, each of the key is called a ***candidate key***. A candidate key K must have the property of being *Unique* and *Irreducible* in a relation R. Uniqueness property means no two distinct tuples in R must have the same value for the key K. If t_1 and t_2 are two tuples in any legal relation state r of R, then $t_1[K] \neq t_2[K]$. Irreducible defines that any proper subset of keycannot have the uniqueness property.

Super Key

A key K in a relation schema R is called a ***super key*** if it has the uniqueness property but not necessarily the irreducible property. Every relation has atleast one superkey – the set of all the attributes in the relation. A key is the minimal superkey. Suppose, EMP_ID is a key of the EMPLOYEE relation. Then any set of attributes of the EMPLOYEE relation that includes EMP_ID are all super keys. For example, {emp_id}, {emp_id, ename}, {emp_id, address, salary} are super keys of EMPLOYEE.

Primary Keys and Alternate Keys

Primary key is one of the candidate key that is selected to uniquely identify the tuples of a relation. For example, in the relation Employee, out of the candidate keys --- EMP_ID, PH_NO, the Emp_id attribute can be chosen to be the primary key of Employee relation. In such a case, the remaining candidate key PH_NO of the relation is called the ***Alternate key***.

Foreign Keys

A foreign key is a set of attributes of a relation R_2that matches the values of the candidate key of a relation R_1. R_1 and R_2 may not be distinct. An example of a foreign key is given below:

EMPLOYEE (R_1)

Emp_id	Ename	Salary
E1	Ishita	18000
E2	Suparna	25000

Emp_Proj (R_2)

Proj_id	E_id	No_of_Hours
P 1	E1	6
P2	E2	8
P1	E1	2

Fig.2.9

In the above example, Emp_id is the primary key in the Employee relation and in Emp_Proj it is the foreign key. The foreign key may be simple or composite according to the candidate key it matches.

2.10 EXPRESSING M:N RELATION

The Many-to-Many relationship is one of the common cardinality ratios for binary relationships in a conceptual database schema. For example, in an organization,

- An employee is associated with several projects through the Emp_Proj relationship.
- A project is associated with many employees via the Emp_Proj relationship.

In ER diagram, the M:N relationship can be expressed by putting the cardinality ratios on the relationship.

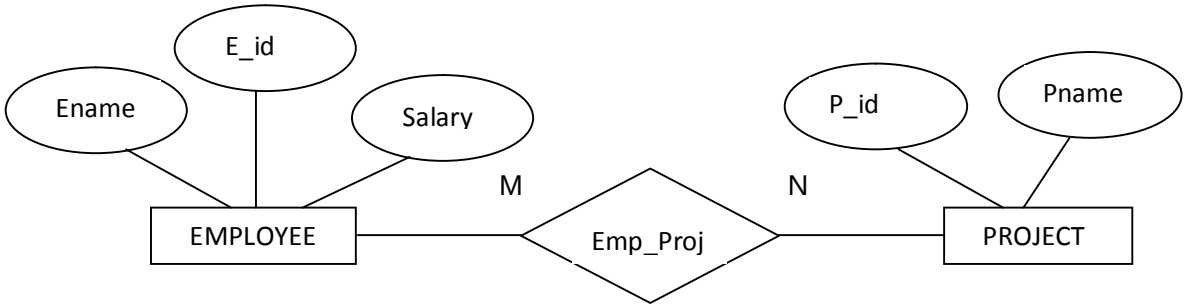


Fig.2.10

The M:N relation can also be expressed as a semantic net diagram as follows:

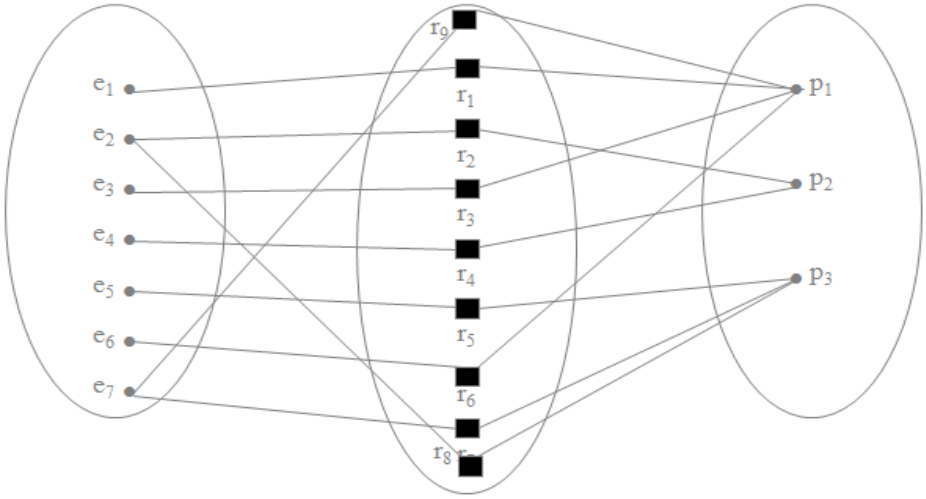
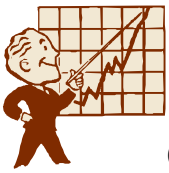


Fig.2.11



CHECK YOUR PROGRESS

4. Fill in the blanks:

a. The graphical representation of the logical database structure is the _____. ER model

b. _____ is a bottom-up design process and_____ is a top-down design process. Generalization, Specialization

c. Specialization _____ the differences between members of an entity by identifying their distinguishing characteristics. Maximizes

d. _____ treats a relationship set as an entity set. Aggregation

e. In a relational model, the tables, rows and columns are known as _____, _____ and _____. Relations, tuples, attributes

f. _____ is the attribute of an entity that identifies one or more instances of the entity set. Key

g. Once the primary key is selected from the candidate keys of a relation, the remaining candidate keys are called _____. Alternate keys

h. A _____ is a set of attributes of a relation that matches the values of the candidate key of the other relation.

2.11 LET US SUM UP

1. An ***entity*** is an object or a thing in the real world with an independent existence. It may have physical or conceptual existence.

2. An entity is represented by a set of ***attributes.*** Attributes are the properties or characteristics of an entity.
3. There are types of attributes - Simple and Composite attribute, Singlevalued and Multivalued attribute, Stored and Derived attribute, Complex attribute.
4. The set of all entities that have the same properties or attributes is called an ***Entity Type***. ***Entity Set*** is the collection of all the entities of a particular entity type.
5. Entities can be Strong or Regular, and Weak.
6. Entities that have an independent existence and can be uniquely identified by key attributes are called Strong entities.
7. Entities whose existence depend on the existence of some other entity and cannot be uniquely identified by key attributes are called weak entities.
8. A relationship is defined as an association among two or more entities.
9. The entities that are involved in a relationship are called the ***participants*** of that relationship.
10. The number of entities or participants in a relationship is called the ***degree*** of that relationship.
11. Two main types of relationship constraints are cardinality ratio, and participation.
12. The common cardinality ratios for binary relationships are One-to-one (1:1), One-to-many (1:N) and Many-to-many (M:N)
13. The participation constraint of a relationship determines whether the existence of an entity depends on its being related to another entity through the relationship. Participation constraints are of two types: *total and partial participation*.
14. A relationship type is a set of associations among entity types. Relationship set is the collection of similar relationships
15. The Entity-Relationship (ER) diagram is a graphical representation of the logical structure of a database.
16. Generalization, specialization and aggregation are the extensions to the basic ER model.

17. Generalization is a bottom-up design process of the logical database. On the other hand, specialization is a top-down design process and it is the reverse of generalization.

18. Aggregation is an abstraction which treats a relationship set as an entity set so that the relationship set can participate in other relationships.

19. Aggregation is an abstraction which allows relationships to be viewed as higher-level entities.
20. A relation can be defined by a set of tuples. The entities form the relations and the attributes are the columns or fields of that relation.
21. Key is the attribute of an entity that identifies one or more instances of the entity set.

2.13 ANSWERS TO CHECK YOUR PROGRESS

1. a) entity b) attributes c) composite d) multiple

   e) derived attribute f) Entity Type g) Entity Set h)Key attribute

   i) Strong attributes j) dependent

2. i. False ii. False iii. True iv. True v. False

3. a. Cardinality ratios, Participation constraints
   b. maximum
   c. rectangle, oval, diamond shape
   d. total, partial
   e. Relationship set

4. a. ER model

b. Generalization, Specialization

c. Maximizes

d. Aggregation

e. Relations, tuples, attributes

f. Key

g. Alternate keys

h. foreign key

2.12 FURTHER READINGS

1. "An introduction to Database Systems", C J Date, Pearson Education
2. "Database Systems – Concepts, Design and Applications", S K Singh, Pearson
3. "Fundamentals of Database Systems", Elmasri, Navathe, Somayajulu, Gupta, Pearson Education

2.14 MODEL QUESTIONS

1. Draw the ER diagram for a Library management system.
2. What is the difference between strong entity and weak entity?
3. Define entity type and entity set.
4. Briefly describe specialization and generalization in ER model with the help of a suitable example.
5. Define cardinality ratio in ER model.
6. Write short notes on composite and derived attributes in an ER model.
7. Write a short note on aggregation.

*****

UNIT-3: RELATIONAL MODEL

UNIT STRUCTURE

3.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- describe Codd's rule
- define Relational Model
- describe Relational Model Constraints

- describe Relational Algebra
- describe Relational Calculus

3.2 INTRODUCTION

In the previous unit we have studied about the Entities, their relationships and the representation of entities, attributes, relationship attributes, relationship set etc. In this unit we are going to discuss about Relational Model. The **relational model** for database management is a database model based on first order predicate logic, first formulated and proposed in 1969 by *Edgar F. Codd*. In the relational model of a database, all data is represented in terms of tuples, grouped into relations. A database organized in terms of the relational model is a relational database.

The purpose of the relational model is to provide a declarative method for specifying data and queries: users directly state what information the database contains and what information they want from it, and let the database management system software take care of describing data structures for storing the data and retrieval procedures for answering queries.

Most implementations of the relational model use the SQL data definition and query language.

3.3 CODD'S RULE

Codd's rules are a set of twelve rules proposed by Edgar F. Codd. He places the relational model's characteristics in three broad categories:

- Structural features that support the view of data. They include relations and their underlying components, views and queries.
- Integrity features such as entity and referential integrity and also application specific constraints.
- Data manipulation features for data retrieval, insertion, deletion, and update.

Following are twelve Codd's Rule:

Rule 1: Information Rule

All information in a relational database including table names and column names are represented by values in tables. This simple view of data speeds up design and learning process.

Rule 2: Guaranteed Access Rule

Every piece of data in a relational database can be accessed by using a combination of a table name, a primary key value that identifies the row and a column name, which identifies a cell.

Rule 3: Systematic Treatment of Nulls Rule

The RDBMS handles records that have unknown or inapplicable values in a predefined fashion. Also, RDBMS distinguishes between zeros, blanks, and nulls in the records and handles such values in a consistent manner that produces correct answers, comparison, and calculations.

Rule 4: Active On-line Catalog based on the Relational Model

The description of a database and its contents are database tables and therefore, can be queried on-live via the data manipulation language. The database administrator's productivity is improved since the changes and addition can be done with the same commands that are used to access any other table.

Rule 5: Comprehensive Data Sublanguage Rule

The RDBMS may support several languages. But at least one of them should allow the user to do all the following: define tables and views, query and update data, set integrity constraints, set authorizations, and define transactions.

Rule 6: View Updating Rule

Any view that can be updated theoretically can be updated using the RDBMS. Data consistency is ensured since the changes made in the view are transmitted to the base table and vice-versa.

Rule 7: High level Insert, Update, and Delete

The RDBMS supports insertion, updating, and deletion at a table level. The performance is improved since the commands act on a set of records rather than one record at a time.

Rule 8: Physical Data Independence

The execution of ad hoc requests and application programs is not affected by changes in the physical data access and storage methods. Database administrators can make changes to the physical access and storage method, which improve performance and do not require changes in the application programs or requests.

Rule 9: Logical Data Independence

Logical changes in tables and views such as adding/deleting columns or changing field lengths need not necessitate modification in the programs or in the format of ad-hoc requests. The database can change and grow to reflect changes in reality without requiring the user intervention or changes in the applications.

Rule 10: Integrity Independence

Like table and view definitions, integrity constrain are stored in the online catalog and can therefore be changed without necessitating changes in the application programs. Integrity constrains specific to a particular Relational Database must be definable in the relational data sublanguage and storable in the catalog.

Rule 11: Distribution Independence

Application programs and ad-hoc requests are not affected by the changes in the distribution of physical data.

Rule 12: Nonsubversion Rule

If the RDBMS has a language that access the information of a record at a time, this language should not be used to bypass the integrity constraints.

3.4 RELATIONAL MODEL

Relational model stores data in the form of a table. Relational database are powerful because they requires few assumptions about how data is related or how it will be extracted from the database. As a result, the same database can be viewed in the different ways. Another feature of relational system is that a single database can be spread across several tables. This differs from flat-file database, in which each database is self-contained in a single table.

The relational model uses tables to organize data elements. Each table corresponds to an application entity and each row represents an instance of that entity. In the formal relational model terminology, a row is called ***tuple***, a column header is called ***attribute***, and each table is called ***relation***.

Advantages of the relational model:

- **Structural independence –** The relational model does not depend on the navigational data access system thus freeing the database designers, programmers, and end user from learning the details of data storage. Changes in the database structure do not affect the data access. When it is possible to make change to the database structure without affecting the DBMS's capability to access data, we can say that structural independence has been achieved. So relational database model has structural independence.

- **Conceptual simplicity -** Since the relational data model frees the designer from the physical data storage details, the designer can concentrate on the logical view of the database.

- **Design, implementation, maintenance, and usage ease –** The relational database model achieves both data independence and structural independence making the database design, maintenance, administration, and usage much easier than other models.

Disadvantages of relational model:

As compared to the advantages of the relational model, the disadvantages of the relational model are very minor

- **Hardware overheads –** Relational database system hide the implementation complexities and the physical data storage details from the users. For doing this, i.e., for making things easier for the users, the relational database system need more powerful hardware- computers and data storage devices.

- **Ease of the design can lead to bad design-** The relational database is an easy to design and use system. The user need not know the complex details of the physical data storage. They need not know how the data is actually stored to access it. This ease of design and use can lead to the development and implementation of very poorly design database management systems. Since the database is efficient, these design inefficiencies will not come to light when the database is design and when there is only a small amount of data. As the database grows, the poorly designed database will slow the system down and will result in performance degradation and data corruption.

Relational Model Notation

Generally following notation are used to denote relational model:

- A relation scheme R of degree *n* is denoted by R (A_1, A2 …An).
- An *n*-tuple *t* in a relation r(R) is denoted by t=<v_1,v_2,…..v_n>, where v_i is the value corresponding to attribute A_i.
- The letter Q, R, S denote relation name.
- The letter q, r, s denote relation states.

CHECK YOUR PROGRESS

1. Choose the correct answer:

i) Who is called the father of the relational database system?

(a) E.F. Codd. (b) Donald Chamberlain

(c) C.J. Date (d) H.F. Korth

ii) What is the RDBMS terminology for a row?

(a) Tuple. (b) Relation.

(c) Attribute. (c) Domain.

iii) What is the RDBMS terminology for a table?

(a) Tuple. (b) Relation.

(c) Attribute. (c) Domain.

iv) What is the RDBMS terminology for the number of attribute in a relation?

(a) Cardinality. (b) Relation.

(c) Attribute. (d) Degree.

3.5 RELATIONAL MODEL CONSTRAINTS

There are many restrictions or **constraints** on the actual values in a database state. Constraints on database can generally be divided into three categories:

1. Constraints those are inherent in the data model. We call these inherent model based constraints.

2. Constraints that can be directly expressed in the schemas of the data model, typically by specifying them in Data Definition Language (DDL). We call these schema based constraints.

3. Constraints that cannot be directly expressed in the schemas of the data model, and hence must be expressed and enforced by the application program. We call these application based constraints.

The main constraints that can be expressed in the relational model is the schema based constraints. Theses include domain constraints, key constraints, constraints on null, entity integrity constraints, and referential integrity constraints.

3.5.1 Domain Constraints

Domain constraints specify that within each tuple, the value of each attribute A must be an attribute value from the domain *dom(A).* The data types associated with domains typically include standard numeric data types for integers (such as short integer, integer, and long integer) and real numbers (float and double-precision float). Characters, booleans, fixed length strings, and variable length strings are also available, as are date, time, time-stamp, and in some cases, money data types. Other possible domains may be described by a sub range of values from a data type or as an enumerated data type in which all possible values are explicitly listed.

3.5.2 Key Constraints and Constraints on Null Values

No two tuples can have the same combination of values for all their attributes.

Super key: No two distinct tuples in any state *r* of *R* can have the same value for SK. That is, for any distinct tuples t1 and t2 in r(R), t1[SK] ≠t2[SK]. This condition must hold in *any valid state* r(R)

Key: Key of R is a "**minimal**" super key. That is, a key is a super key K such that removal of any attribute from K results in a set of attributes that is not a super key (does not possess the super key u**niqueness** property).

Candidate key: A candidate key can be any fields or a combination of fields that uniquely identifies each record in the table. There can be multiple Candidate Keys in one table. Each candidate key can qualify as Primary Key. For example, the relation CAR has two candidate keys: Reg._No and Engine_No.

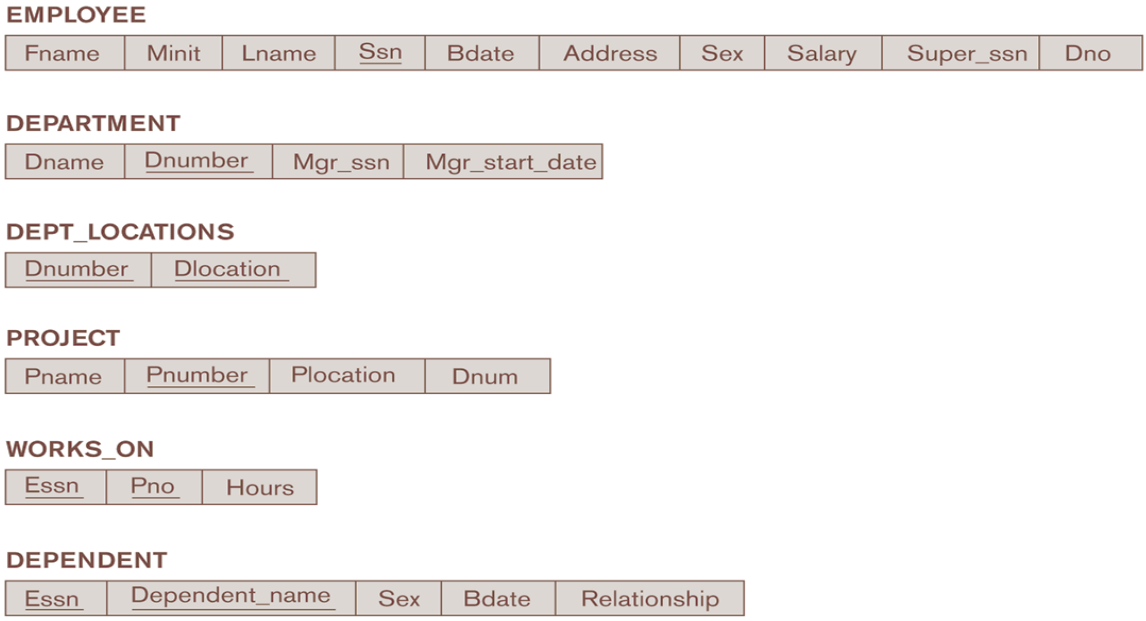
Primary Key: If a relation has several **candidate keys**, one is chosen to be the primary key. This is usually the best among the candidate keys. The primary key attributes are underlined. Example: Consider the CAR relation schema: CAR (State, Reg#, SerialNo, Make, Model, Year) .We chose SerialNo as the primary key. The primary key value is used to *uniquely identify* each tuple in a relation and provides the tuple identity. It is also used to *reference* the tuple from another tuple.

Another constraint on attribute specifies whether null values are or are not permitted. For example, if every STUDENT tuple must have a valid, nonnull value for the *Name* attribute, then *Name* of STUDENT is constrained to be **not null.**

3.5.3 Relational Databases and Relational Database Schemas

A **Relational Database Schema** S a set of relation schemas S = {R1, R2, ..., Rn} that belong to the same database. S is the name of the whole **database schema.** R1, R2, …, Rn are the names of the individual **relation schemas** within the database S

A COMPANY database schema with 6 relation schemas is shown below



3.5.4. Entity Integrity, Referential Integrity Constraints and Foreign Key

Entity Integrity: The *primary key attributes* PK of each relation schema R in S cannot have null values in any tuple of r(R). This is because primary key values are used to *identify* the individual tuples. t[PK] ≠ null for any tuple t in r(R). If PK has several attributes, null is not allowed in any of these attributes.
Note: Other attributes of R may be constrained to disallow null values, even though they are not members of the primary key.

Referential Integrity: It is a constraint that involving *two* relations. It is used to specify a *relationship* among tuples in two relations. The two relations are termed as **referencing relation** and the **referenced relation**. Tuples in the *referencing relation* R1 have attributes **FK** (called **foreign key** attributes) that reference the primary key attributes **PK** of the *referenced relation* R2. A tuple t1 in R1 is said to **reference** a tuple t2 in R2 if t1[FK] = t2[PK]. A referential integrity constraint can be displayed in a relational database schema as a directed arc from *R1.FK* to *R2.PK*

Foreign key: A set of attribute FK in a relation schema R1 is a foreign key of R1 that reference relation R2 if it satisfies the following two conditions:

1. The attributes in FK have the same domain(s) as the primary key attributes PK
2. Value of FK in a tuple *t*1 of the current state *r*1 *(R*1*)* either occurs as a value of PK for

   some tuple *t*2 in the current state *r*2*(R*2*)* or is NULL

3.5.5 Other Types of Constraints

Semantic integrity constraints may have to be specified and enforced on a relational database. Such constraints can be specified and enforced within the application programs that update the data base. Sometime **triggers** and **assertions** can be used.

Functional dependency constraint establishes a functional relationship among two sets of attributes *X* and *Y.* Value of *X* determines a unique value of *Y*

State constraints define the constraints that a valid state of the database must satisfy.

Transition constraints define to deal with state changes in the database

CHECK YOUR PROGRESS

2. Choose the correct answer:

(i) DDL stand for

(a) Data Design Language (b) Data Dictionary Language

(c) Data Define Language (d) Data Definition Language

(ii) If a relational schema has more than one key, then in this case each key is called

(a) Primary key (b) Foreign key

(c) Candidate key (d) Super Key

(iii) A key that uniquely identify different entity among entity set is called

(a) Primary key (b) Foreign key

(c) Candidate key (d) Super Key

(iv)A primary key attribute in a relation cannot have null value, this condition is

(a) Referential integrity (b) Entity integrity

(c) Foreign key (d) Domain constraint

3.6 RELATIONAL ALGEBRA AND ALGEBRAIC OPERATIONS

Relational algebra constitutes a basic set of operations for manipulating relational data. These operations enable the users to perform basic retrieval operations. The result of a retrieval operation on a table (or relation) is another relation. Thus, the relational algebraic operations produce new relations, which can be further manipulated using the same relational algebraic operation.

Relational Algebraic Operation

The relational algebraic operations are divided into two groups-unary and binary. Unary operations are those operations which operate on single relation and binary operations are those which operate on two relations.

Unary operation include

SELECT

PROJECT

RENAME

Binary operation include

UNION

INTERSECTION

SET DIFFERENCE

CARTESIAN PRODUCT

DIVISION

JOIN

3.6.1 SELECT

The SELECT operation is used to select a subset of the tuples from a relation that satisfy a selection condition. So we can consider the SELECT operation to be a filter that keeps only those tuples that satisfy a qualifying condition. It is represented as follows:

$$\sigma_{\text{<select condition>}}(R)$$

The symbol **σ** (sigma) is used to denote the SELECT operator and the **<select condition>** is a Boolean expression specified on the attributes of the relation **R.** The Boolean expression specified in **select condition** is made up of a number of **clauses** of the form

<attribute name><comparison operator><constant value>

where **<attribute name>** is the name of attribute (column) of relation R; **<comparison operator>** is one of the following comparison operator; =,≠,<,≤,> or ≥ ; **<constant value>** is constant value from the attribute domain.

Following is the example of SELECT operation

$\sigma_{DNO=4}$ **(EMPLOYEE)** -Select the tuples for all employee whose department is 4.

3.6.2 PROJECT

The PROJECT operation is used to select certain columns (attribute) from a table, while discarding others. So if we interested in certain columns (attribute) of a table, we can use PROJECT operation to project the relation over these attribute. The PROJECT operation is represented as follows:

$$\pi_{\text{<attribute list>}}(R)$$

The symbol **π** (pi) is used to denote the POJECT operation and the **<attribute list>** is a list of attributes from the attribute of the relation **R**. The result of the PROJECT operation has only the attributes specified in the

attribute list and in the same order as they appear in the list. Hence the degree (number of columns) of the result is equal to the number of attribute specified in the attribute list..

Following is the example of PROJECT operation

> $\pi_{EANME,ADDRESS}$ **(EMPLOYEE)** **-**Get the column ENAME and ADDRESS from the EMPLOYEE table.

3.6.3 RENAME

The RENAME operation is used to rename a relation or attribute. The general RENAME operation when applied to a relation can take any of the following forms

> $\rho_{S(\text{new attribute names})}(R)$
>
> $\rho_S(R)$
>
> $\rho_{(\text{new attribute names})}(R)$

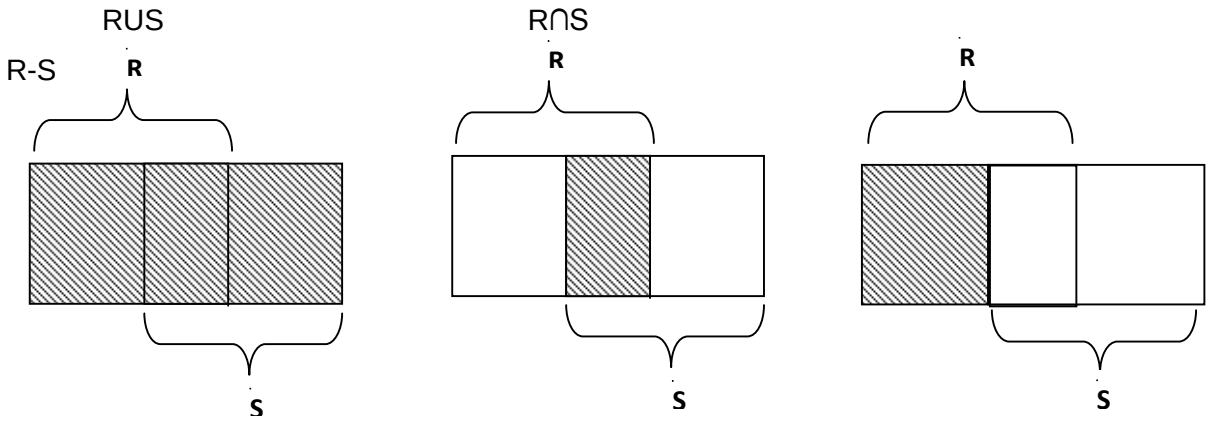
The symbol **ρ**(rho) is used to denote the RENAME operator. **S** is the new relation name and **R** original relation. The first expression renames both the relation and its attribute, the second expression renames the relation only and the third one renames only the attributes.

Following are the example of RENAME operation

> $\rho_{TEMP(SNAME,SADDRESS)}$**(EMPLOYEE)** **-** Here both the relation name and attribute names are rename.
>
> ρ_{TEMP}**(EMPLOYEE)** **-** Here only the relation name is renamed
>
> $\rho_{(SNAME,\ SADDRESS)}$**(EMPLOYEE)** -In this case only the attribute names are renamed.

3.6.4 UNION, INTERSECTION, and DIFFRERENCE

These three operations are represented by the following pictures



UNION INTERSECTION DIFFERENCE

The operations UNION, INTERSECTION, and DIFFEENCE –requires that the tables (relation) involved be union compatible. Two relations are said to be union compatible if the following conditions are satisfied.

- ☞ The two relations/tables must contain the same number of columns (have the same degree).
- ☞ Each column of the first relation/table must be either same data type as the corresponding column of the second relation/table or convertible to the same data type as the corresponding column of the second.

Suppose there are two relations R and S

R

ROHIT	KOLKATA
HARISH	DELHI
JEEVAN	MUMBAI
KRISH	GHY

S

KRISH	GHY
RAM	DELHI
ROHIT	KOLKATA
RITESH	MUMBAI
HARSH	CHENNAI

UNION- The result of this operation denoted by **RUS** is the relation that includes all tuples that are either in the relation **R** or in the relation **S**, or in both R and S. Duplicates are eliminated.

INTERSECTION- The result of this operation is a relation that includes all tuples that are in both R and S. The intersection operation is denoted by **R∩S.**

DIFFERENCE- The difference operation is denoted by **R-S.** The result of this operation is the relation that includes all tuples that are in **R** but not in **S**

ROHIT	KOLKATA
HARISH	DELHI
JEEVAN	MUMBAI
KRISH	GHY

R-S

ROHIT	KOLKATA
KRISH	GHY

RUS

HARISH	DELHI
JEEVAN	MUMBAI

R∩S

3.6.5 CARTESIAN PRODUCT

The **CARTESIAN PRODUCT** is also known as **CROSS PRODUCT** or **CROSS JOIN**. It is denoted by 'X'. The **CARTESIAN PRODUCT** between relations R and S is denoted by **R X S**. The relations on which we are performing this operation need not to be union compatible. This operation is used to combine tuples from two relations in a combinational fashion.

The result of **CARTESIAN PRODUCT** of two relations, which have **m** and **n** columns, is a relation that has **m+n** columns. The resulting relation will have one tuple for each combination of tuples one from each participating relation. Hence, if the relations have $\mathbf{m_R}$ and $\mathbf{n_S}$ tuples, respectively, then the CARTESIAN PRODUCT will have $\mathbf{m_R * n_S}$ tuples. Consider the following two tables **BORROWER** and **LOAN,** one containing the borrower details and the other containing loan details.

BORROWER

BName	BIno
JHON	L-102
ROHIT	L-201
KRISH	L-315

LOAN

Loanno	Branch	Amount
L102	GUWAHATI	30000
L-201	PANBAZAR	40000

The **BORROWER** has 2 columns and 3 tuples and **LOAN** has 3 columns and 2 tuples. So the **CARTESIAN PRODUCT** has 5 columns and 6 tuples.

BORROWER X LOAN				
BName	**Blno**	**Loanno**	**Branch**	**Amount**
JHON	L-102	L-102	GUWAHATI	30000
JHON	L-102	L-201	PANBAZAR	40000
ROHIT	L-201	L-102	GUWAHATI	30000
ROHIT	L-201	L-201	PANBAZAR	40000
KRISH	L-315	L-102	GUWAHATI	30000
KRISH	L-315	L-201	PANBAZAR	40000

3.6.6 DIVISION (÷)

The division is a binary operation that is written as $R \div S$. The result consists of the restrictions of tuples in *R* to the attribute names unique to *R*, i.e., in the header of *R* but not in the header of *S*, for which it holds that all their combinations with tuples in *S* are present in *R*. For an example see the tables *Completed*, *DBProject* and their division:

Completed	
Student	**Task**
Fred	Database1
Fred	Database2
Fred	Compiler1
Eugene	Database1
Eugene	Compiler1
Sarah	Database1
Sarah	Database2

DBProject
Task
Database1
Database2

Completed ÷ DBProject
Student
Fred
Sarah

If *DBProject* contains all the tasks of the Database project, then the result of the division above contains exactly the students who have completed both of the tasks in the Database project.

More formally the semantics of the division is defined as follows:

$$R \div S = \{\, t[a_1,...,a_n] : t \in R \wedge \forall s \in S\ (\, (t[a_1,...,a_n] \cup s) \in R \,) \,\}$$

where $\{a_1,...,a_n\}$ is the set of attribute names unique to R and $t[a_1,...,a_n]$ is the restriction of t to this set. It is usually required that the attribute names in the header of S are a subset of those of R because otherwise the result of the operation will always be empty.

The simulation of the division with the basic operations is as follows. We assume that $a_1,...,a_n$ are the attribute names unique to R and $b_1,...,b_m$ are the attribute names of S. In the first step we project R on its unique attribute names and construct all combinations with tuples in S:

$$T := \pi_{a1,...,an}(R) \times S$$

In the prior example, T would represent a table such that every Student (because Student is the unique key / attribute of the Completed table) is combined with every given Task. So Eugene, for instance, would have two rows, Eugene -> Database1 and Eugene -> Database2 in T.

In the next step we subtract R from this relation:

$$U := T - R$$

Note that in U we have the possible combinations that "could have" been in R, but weren't. So if we now take the projection on the attribute names unique to R then we have the restrictions of the tuples in R for which not all combinations with tuples in S were present in R:

$$V := \pi_{a1,...,an}(U)$$

So what remains to be done is take the projection of R on its unique attribute names and subtract those in V:

$$W := \pi_{a1,...,an}(R) - V$$

3.6.7 JOIN OPERATION

Join operation is denoted by '⋈' is used to combine related tuples from two relations in to a single tuple. The general form of join operation on two relations **R**(A_1, A_2,…..A_m) and **S**(B_1,B_2,…..B_n) is **R** ⋈ $_{<join\ condition>}$**S**. The result of the join is a relation **Q** with **m+n** attributes. The order of the relation **Q** will be **Q**(A_1, A_2,…..A_m, B_1,B_2,…..B_n). **Q** has one tuple for each combination of tuple, one from **R** and one from **S** whenever the join condition satisfied. Let us take two relations

BORROWER

BName	Blno
JHON	L-102
ROHIT	L-201
KRISH	L-315

LOAN

Loanno	Branch	Amount
L102	GUWAHATI	30000
L-201	PANBAZAR	40000
L-301	MALIGAON	50000

BORROWE ⋈ $_{<Loanno=Blno>}$**LOAN**

BName	Blno	Loanno	Branch	Amount
JHON	L-102	L-102	GUWAHATI	30000
ROHIT	L-201	L-201	PANBAZAR	40000

***Theta join*:** A general join condition is of the $A_i\theta B_j$, where A_i is an attribute of the relation **R** and B_j is an attribute of the relation **S,** A_i and B_j have common domain and θ is one of the comparison operator {=,≠,>,<,>=,<=}. A join operation with such general join condition is called **theta join**.

Natural Join: Natural join is an equijoin of two relations R and S over all common attributes. One occurrence of each common attribute is eliminated from the result. The Natural Join operation performs an Equijoin over all the attributes in the two relations that have the same name. The degree of the Natural Join is the sum of the degree of the two relations R and S less the number of common attributes

Outer join: Often in the joining two relations, a tuple in one relation does not have a matching tuple in other relation: in other words, there is no matching value in the join attributes. We may want a tuple from one of the relations to appear in the result even when there is no matching value in the other relation. This may be accomplished by using **Outer join**. The *left outer join* is a join in which the tuple from R that do not have matching value in the common attributes of S are also included in the result relation. Missing value

in the second relation are set to null. A left outer join keeps every tuple in the left-hand relation in the result. Similarly, there is a *right outer join* that keeps every tuple in the right-hand relation in the result. There is also a *full outer join* that keeps all tuples in both relations, padding tuples with null values when no matching tuples are found.

CHECK YOUR PROGRESS

3. Choose the correct answer

(i) Which operation is used to select subset of tuples from a relation?

(a) Project (b) Select

(c) Rename (d) Join

(ii) Which operation is used to select attributes from a relation?

(a) Project (b) Select

(c) Rename (d) Join

(iii)If R and S are two relations, then intersection between two relations is denoted by

(a) R∩S (b) R-S

(c) RUS (d) S-R

(iv) JOIN operation is denoted by the symbol

(a) σ (b) **π**

(c) ρ (d) ⋈

3.7 REALATIONAL CALCULUS

Relational calculus is a formal query language. It can be categories into **Procedural Query Language** and **Declarative Query Language**. In Procedural Query language, query specification involves giving a step by step process of obtaining the query result. *E.g.* Relational Algebra. It is difficult for the use of non-experts. In Declarative Query language, query specification

involves giving the logical conditions the results are required to satisfy. It is easy for the use of non-experts. Relational calculus is of two types 1) Tuple Relation Calculus and 2) Domain Relation Calculus.

Tuple Relation Calculus: The Tuple Relation Calculus is based on specifying a number of tuple variables. Each tuple variable usually ranges over a particular database relation, meaning that the variable may take as its value any individual tuple from that relation. A simple tuple relational calculus query is of the form {t | Cond(t)}where t is the tuple variable and Cond(t) is a conditional expression involving t. { T | R(T) }: returns all tuples T such that T is a tuple in relation R.

{ T.name | FACULTY (T) AND T.DeptId =" CS"}.returns the values of name field of all faculty tuples with the value 'CS' in their department id field.

The variable T is said to be free since it is not bound by a quantifier (for all, exists). The result of this statement is a relation (or a set of tuples) that correspond to all possible ways to satisfy this statement. It will find all possible instances of T that make this statement true.

QUERY:

Retrieve the birth date and address of the employee whose name is 'Jhon B. Smith'

{t.bdate,t.address | EMPLOYEE(t) AND t.Fname='Jhon' AND t.Mname= 'B' AND t.Lname='Smith'}

In the Tuple Relation Calculus, we first specify the requested attributes t.bdate and t.address for each selected tuple t. Then we specify the condition for selecting a tuple following the bar (|)- namely, that t be a tuple of the EMPLOYEE relation whose Fname, Mname and Lname attribute values are 'Jhon', 'B' and ''Smith' respectively.

Expressions and Formulas in Tuple Relation Calculus

A general expression of the tuple relation calculus is of the from

$\{t_1.A_j, t_2.A_k, \ldots\ldots, t_n.A_m \mid COND(t_1,t_2, \ldots., t_n, t_{n+1}, t_{n+2}, \ldots., t_{n+m})\}$

Where $t_1, t_2, \ldots, t_n, t_{n+1}, t_{n+2}, \ldots, t_{n+m}$ are tuple variables, each A_i is an attribute of the relation on which t_i ranges, and COND is a condition or formula of the relational calculus. A formula is made up of the predicate calculus **atoms,** which can be one of the following

1. An atom of the form $R(t_i)$, where R is a relation name and t_i is a tuple variables. This atom identifies the range of the tuple variable t_i as the relation whose name is R.

2. An atom of the form $t_i.A$ op t_jB, where **op** is one of the comparison operators in the set {=, ≠, >, >=, <, <=}, t_i and t_j are the tuple variables. A is an attribute of the relation on which t_i ranges, B is an attribute of the relation on which t_j ranges.

3. An atom of the form $t_i.A$ **op** c or c **op** t_jB, where **op** is one of the comparison operators in the set {=, ≠, >, >=, <, <=}, t_i and t_j are the tuple variables. A is an attribute of the relation on which t_i ranges, B is an attribute of the relation on which t_j ranges and c is a constant value.

A formula (condition) is made up of one or more atoms connected via the logical operator **AND, OR,** and **NOT** and define recursively as follows:

1. Every atom is a formula.

2. If F_1 and F_2 are formulas, then so are (F_1 **AND** F_2), (F_1 **OR** F_2), **NOT** (F_1), and **NOT** (F_2). The truth values of these formulas are derived from their component formula F_1 and F_2 as follows:

    a. (F_1 **AND** F_2) is TRUE if both F_1 and F_2 are TRUE; otherwise, it is FALSE.

    b. (F_1 **OR** F_2) is FALSE if both F_1 and F_2 are FALSE; otherwise, it is TRUE.

    c. **NOT** (F_1) is TRUE if both F_1 is FALSE; otherwise, it is TRUE.

    d. **NOT (**F_2) is TRUE if both F_2 is FALSE; otherwise, it is TRUE.

The Existential and Universal Quantifiers

Variables can be constrained by quantified statements to tuples in a single relation:

Existential Quantifier- ∃T ∈ R(Cond) will succeed if Cond succeeds for at least one tuple in T.
Universal Quantifier -∀ T ∈ R(Cond) will succeed if Cond succeeds for at all tuples in T.

Any variable that is not bounded by any quantifier is termed as **free tuple variable**; otherwise it is a **bound variable.**

Example Queries using the existential Quantifiers
Query: Retrieve the name and address of all employees who works for the 'Research' department.

Q: {t.FNAME, t.LNAME, T.ADDRESS | EMPLOYEE (t) **AND** (∃d) (DEPARTMENT (d) **AND** d.DNAME ='Research' **AND** d.DNUMBER=t.DNO)}

The only free tuple variables in a relational calculus expression should be those that appear to the left of the bar (|). In the above query, t is the only free variable; it is then successively bound to each tuple. If a tuple satisfies the conditions specified in the query, the attributes FNAME, LNAME, and ADDRESS are retrieved for each such tuple. The conditions EMPLOYEE (t) and DEPATMENT (d) specify the range relations for t and d. the condition d.DNAME ='Research' is a selection condition and corresponds to a SELECT operation to relation algebra, whereas the join condition d.DNUMBER=t.DNO is a join condition.

Example Queries using the Universal Quantifiers
Query: Find the name of employees who works on all the project controlled by the department number 5.

Q: {e.LNAME, e.FNAME |EMPLOYEE (e) **AND** ((∀ x)(**NOT**(PROJECT(x)) **OR** NOT(x.DNUM=5)

Safe Expressions

Whenever we use universal quantifiers, existential quantifies, or negation of predicates in a calculus expression, we must make sure that the resulting expression makes sense. A safe expression in a relational calculus is one that is guaranteed to yield a finite number of tuples as its result; otherwise, the expression is called **unsafe**. For example, the expression {t | **NOT**

(EMPLOYEE (t))} is unsafe because it yields all tuples in the universe that are not EMPLOYEE tuples, which are infinitely numerous.

Domain Relation Calculus

Domain Relation Calculus differs from tuple relation calculus in the type of variables used in formulas: rather than having variables range over tuples, the variables range over the single values from the domains attributes. To form a relation of degree n for a query result, we must have n of these domain variables- one for each attribute. An expression of the domain calculus is of the form

$\{x_1,x_2,\ldots x_n \mid \text{COND}(x_1,x_2,\ldots x_n,x_{n+1},x_{n+2},\ldots\ldots x_{n+m})\}$ where $x_1,x_2,\ldots x_n,x_{n+1},x_{n+2},\ldots\ldots x_{n+m}$ are domain variable that range over domains (of attribute), and COND is **condition** or **formula** of the domain relation calculus.

A formula is made up of atoms. The atoms of a formula are slightly different from those for the tuple calculus and can be one of the following

1. An atom of the form $R(x_1,x_2,\ldots x_j)$, where R is the relation name and j is the degree and each x_i ,$1 \leq i \leq j$, is a domain variable. This atom states that a list of values of $< x_1,x_2,\ldots x_j>$ must be a tuple in the relation whose name is R, and x_i is the value of the ith attribute value of the tuple. To make domain relation calculus more concise, we can drop the comas in a list of variables; thus we can write

   $\{ x_1,x_2,\ldots x_n, \mid R(x_1x_2x_3)$ **AND …**$\}$

   Instead of

   $\{ x_1,x_2,\ldots x_n, \mid R(x_1,x_2,x_3)$ **AND …**$\}$

2. An atom of the form x_i op x_j, where **op** is one of the comparison operators in the set {=, ≠, >, >=, <, <=}, x_i and x_j are the domain variables.

3. An atom of the form x_i. op x_j, where **op** is one of the comparison operators in the set {=, ≠, >, >=, <, <=}, x_i and x_j are the domain variables and c is a constant value.

Example Query

Query: Retrieve the name and address of all employees who works for the 'Research' department.

Q: {qsv | (∃.z) (∃.l) (∃.m) (EMPLOYEE (qrstuvwxyz) **AND** DEPARTMENT (lmno) **AND** l='Reasearch' **AND** m=z)}

A condition relating two domain variables that range attributes from two relations, such as m=z in the Query, is a join condition; whereas a condition that relates a domain variable to a constant, such as l='Research', is a selection condition.

CHECK YOUR PORGRESS 4

4. Choose the correct option:

(i) Tuple relational calculus is based on

(a) Domain (b)Tuple variable

(c) Column (d) Entity

(ii) Every atom is a

(a) Expression (b) Formula

(c) Solution (d) Operation.

(iii) Existential quantifier is denoted by

(a) ρ (b) σ

(c) ∃ (d) ∀

(iv)Universal quantifier is denoted by

(a) ρ (b) σ

(c) ∃ (d) ∀

3.8 LET US SUM UP

- Relational model for database management is a database model based on first order predicate logic, first formulated and proposed in 1969 by Edgar F. Codd.
- Codd's rules are a set of twelve rules proposed by Edgar F. Codd.
- Application programs are not affected by changes in the physical data access and storage methods are the Physical Data Independence.
- Logical Data Independence is the logical changes in tables and views such as adding/deleting columns or changing field lengths need not necessitate modification in the programs or in the format of ad-hoc requests.
- In relational model terminology, a row is called ***tuple***, a column header is called ***attribute***, and each table is called ***relation***.
- Domain constraints specify that within each tuple, the value of each attribute A must be an attribute value from the domain *dom(A).*
- A relation schema may have more than one key. In this case, each of the key is called candidate key.
- The relational algebraic operations are divided into two groups-unary and binary.
- Unary operations are those operations which operate on single relation.
- Binary operations are those which operate on two relations.
- The operations UNION, INTERSECTION, and DIFFEENCE –requires that the tables (relation) involved be union compatible.
- Join operation is denoted by '⋈' is used to combine related tuples from two relations in to a single tuple.

- Relational Calculus can be categories into **Procedural Query Language** and **Declarative Query Language**.
- Tuple Relation Calculus is based on specifying a number of tuple variables.
- Any variable that is not bounded by any quantifier is termed as free tuple variable; otherwise it is a bound variable**.**
- A safe expression in a relational calculus is one that is guaranteed to yield a finite number of tuples as its result

3.9 ANSWERS TO CHECK YOUR PROGRESS

1.(i) (a)	(ii)(a)	(iii)(b)	(iv)(d)
2.(i) (d)	(ii) (c)	(iii) (a)	(iv)(d)
3.(i)(b)	(ii)(a)	(iii) (a)	(iv)(d)
4.(i)(b)	(ii)(b)	(iii)(c)	(iv)(d)

3.10 FURTHER READINGS

1. R. Elmasri, S.B. Navathe, Fundamentals of Database System, Pearson

2. A. Leon, M. Leon, Fundamentals of Database Management System, Tata McGraw Hill.

3.11 MODEL QUESTIONS

1. What are Codd's rules?

2. What is logical data independence?

3. What is physical data independence?

4. What is primary, candidate, foreign key?

5. What are the constraints include in relational model?

6. What is entity and referential integrity constraint?

7. What are the advantages and disadvantages of relational model?

8. What are the notations used to denote Relational model?

9. What is relational algebra and what are its uses?

10. What is SELECT, PROJECT and RENAME operation? How is it represented? Explain with example.

11. What is a JOIN operation? How is it represented? Explain with example.

12. How does a tuple relational calculus differ from domain relational calculus?

13. Discuss the meaning of the existential quantifier (∃) and universal quantifier (∀).

14. Define the following terms with respect to the domain relational calculus: tuple variable, range relation, atom, formula, and expression.

15. What is mean by safe expression in relational calculus?

UNIT - 4 INTRODUCTION TO SQL

UNIT STRUCTURE

4.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- learn about the Structured Query Language
- know the data types and literals in SQL
- learn the different SQL commands

4.2 INTRODUCTION

The language used to communicate with a database is called Structured Query Language (SQL). SQL has clearly established itself as the standard relational database language. It is a very easy language and looks as simple as normal English. The Structured Query Language (SQL) is the standard relational database language. Originally it was the interface of an experimental relational system SYSTEM R, which was implemented by IBM research. Earlier SQL was called SEQEUEL (Structured English QUEry Language). Today, SQL is used by all modern relational databases, (Microsoft Access, Microsoft SQL

Server, Oracle etc) as the basic building block. It is used to access and manipulate database operations. All data entry operations performed by graphical user interface use SQL translator to convert them to SQL commands understood by the database.

- SQL is a standard language for accessing and manipulating databases.
- SQL stands for Structured Query Language.
- SQL lets you access and manipulate databases.
- SQL is an ANSI (American National Standards Institute) standard language.

The main parts of SQL are –

- **Data definition language (DDL) –** It provides commands to define relation schemes, delete relations, create relation, modify relation etc.
- **Data manipulation language (DML) –** It is a query language based on both relational algebra and tuple relational calculus. It has commands to insert, delete and modify tuples.
- **Transaction control –** It specifies beginning and end of transactions.

4.3 CHARACTERISTICS OF SQL

- SQL is a very powerful ANSI and ISO standard high level computer language for creating and manipulating databases.
- Using SQL, end users and system administrators can execute queries like CREATE, UPDATE, DELETE, and retrieve data from a database.
- SQL is a non procedural language used for table-oriented and record-oriented operations.
- SQL is an abstract language consisting of statements for data definition, query and updates.
- SQL is very simple and easy to learn.
- Databases like DB2, Oracle, MS Access, Sybase, MS SQL Server etc use SQL.

4.4 ADVANTAGES OF SQL

- **High Speed –**

SQL Queries can retrieve records from a database quickly and efficiently.

- **Well Defined Standards Exist –**
  SQL databases use an established standard adopted by ANSI & ISO.

- **No Coding Required –**
  Standard SQL uses substantial amount of code for different database operations. Hence, it is easier to manage database systems.

- **Emergence of ORDBMS –**
  In early days SQL databases were identical to relational databases. In presence of Object Oriented DBMS, object storage capabilities are extended to relational databases.

4.5 DISADVANTAGES OF SQL

- **Difficulty in Interfacing –**
  Interfacing an SQL database is more complex than adding a few lines of code.

- **More Features Implemented in Proprietary way –**
  Although SQL databases conform to ANSI & ISO standards, some databases go for proprietary extensions to standard SQL to ensure vendor look-in.

CHECK YOUR PROGRESS

1. Fill in the blanks.

(i) SQL means _____.

(ii) DDL means _____

(iii) DML means _____.

(iv) SQL is an _____ and _____ standard.

4.6 SQL DATA TYPE AND LITERAL

SQL includes the following data types.

1. Character strings –

Character string data types are of either fixed length or varying length. CHAR (n), CHARACTER (n) are data types for fixed length string field and VARCHAR (), CHAR VARYING (), CHARACTER VARYING () are data types for variable length string fields; where n is the maximum number of characters. For fixed length string, a shorter string is added with blank characters to the right. Specifying a string value is case sensitive and it is placed between single quotation marks.

Some character string data types are.

- CHARACTER(n) or CHAR(n) — fixed-n-character string field and added blank character as needed.
- CHARACTER VARYING(n) or VARCHAR(n) — variable-string field of a maximum size n characters.
- NATIONAL CHARACTER(n) or NCHAR(n) — fixed string field size, supporting an international character set.
- NATIONAL CHARACTER- VARYING(n) or NVARCHAR(n) — variable-string size NCHAR string.

2. Bit strings –

Bit string data types are either of fixed length BIT (n) or varying length BIT VARYING(n), where n is the maximum number of bits. The default length of character string or bit string is 1. Litarel bit strings are preceded by a B and placed between single quotes.

- BIT(n) — an array of n bits.
- BIT VARYING(n) — an array of up to n bits.

3. Numbers –

Numeric data types include integer numbers of various sizes and floating point numbers of various precessions. For floating point numbers data types are FLOAT, REAL, DOUBLE PRECESION and for integer number data types are INTEGER, INT, and SMALLINT. Formatted numbers can be declared by using DECIMAL(), NUMERIC(), DEC(i,j), where i the precision, is the total number of decimal digits and j the scale is the number of digits after the decimal point. Precision is a positive integer that determines the number of significant digits in a particular radix (binary or decimal). Scale is a non-negative integer. A scale of 0 indicates that the number is an integer.

The default for scale is zero, and the default for precision is implementation defined.

A Boolean data type in SQL has value TRUE, FALSE or UNKNOWN. This UNKNOWN is used for NULL value presents in SQL.

4. Date and time –

In SQL, there are two data type DATE and TIME for date and time. The DATE data type has ten width and format is YYYY-MM-DD where YYYY for YEAR, MM for MONTH and DD for DAY. The TIME data type has at least eight position, and format is HH:MM:SS where HH for HOUR, MM for MINUTE and SS for SECOND. SQL allows only valid date and time.

Some date and time data types are.

- DATE — for date values (e.g., 2010-03-03)
- TIME — for time values (e.g., 17:49:36).
- TIME WITH TIME ZONE or TIMETZ — the same as TIME data type and also includes details about the time zone.

5. Additional data type –

(a) TIMESTAMP data type put together the DATE and TIME fields. It also includes minimum of six positions for decimal fractions of seconds and an optional WITH TIME ZONE qualifier. Literal values are represented by single quoted strings preceded by keyword TIMEZONE with a blank space between data and time.

   i. TIMESTAMP — this data type is used to put together a DATE and a TIME in one variable (e.g., 2010-03-03 17:49:36).
   ii. TIMESTAMP WITH TIME ZONE or TIMESTAMPTZ — the same as TIMESTAMP data type and also includes details about the time zone.

(b) Another data type is the interval data type. This is related to DATE, TIME and TIMESTAMP data types. This specifies a relative value that can be used to change an absolute value of a date, time or timestamp. This relative value is called an interval that are either YEAR/MONTH interval or DAY/TIME interval.

CHECK YOUR PROGRESS

2. Fill in the blanks.

(i) Data types for date and time are _____ and _____.

(ii) Character string data types are either _____ or _____

(iii) TIMESTAMP data type put together _____ and _____ field.

(iv) A Boolean data type in SQL has values TRUE FALSE or _____.

4.7 SQL COMMANDS

Data in a database has to be operated on in order to obtain the desired result. For this purpose we need to insert the data in the tables, and later update the data as needed. Data can also b deleted from the tables. To do that we need various SQL commands that operate on data in the tables. Apart from data manipulation in table SQL commands also allow for some more advanced operations to be performed that make it easier for users to maintain and use the data in a desired organized way. This section discusses the different types of SQL commands that make possible the task of performing such operations on database tables.

4.7.1 DATA DEFINITION LANGUAGE (DDL) COMMANDS

Data definition language (DDL) commands enable us to perform the following tasks-

- Create, alter, and drop schema objects
- Grant and revoke privileges and roles
- Add comments to the data dictionary

CREATE, ALTER, and DROP commands are used to create, alter, and drop a schema, while GRANT and REVOKE command are used for privatization tasks.

ALTER COMMAND –

ALTER is the command used to add, modify data from tables, databases, and views. ALTER TABLE is the command responsible for making changes to an SQL table by renaming, adding or dropping columns etc.

The ALTER TABLE syntax,

(a) To add a column in a table is:

```
ALTER TABLE table_name
ADD column_name datatype;
```

(b) To delete a column in a table is:

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

For example-

To delete the column named "Date_Of_Birth" in the "Persons" table, the ALTER TABLE statement shall be –

```
ALTER TABLE Persons
DROP COLUMN Date_Of_Birth;
```

(c) To change the data type of a column in a table is:

```
ALTER TABLE table_name
ALTER COLUMN column_name datatype;
```

CREATE COMMAND –

The main SQL command for data definition is the CREATE statement, which can be used to create schema, tables, and domains.

- **CREATE SCHEMA**

To identify SQL schema, each schema has a schema name, descriptor for each element and an authorization identifier to indicate the user and account that owns the schema. Schema element includes tables, constraints, views, domains and other constraints.

A schema is created using the CREATE SCHEMA statement as –

The following schema creates a schema called PERSON, owned by the user with authorization identifier ‘SMITH’.

CREATE SCHEMA PERSON AUTHORIZATION SMITH;

In general, not all users are authorization to create schemas and schema elements. The privilege to create schema, tables and other constructs must be explicitly granted to the relevant user accounts by the system administrator or DBA.

- **CREATE TABLE**

The CREATE TABLE command is used to create a new relation by giving it a name and specifying its attributes and initial constraints. The attributes are specified first and each attribute is given a

name, a data type to specify its domain of values and any attribute constraints. The key, entity integrity and referential integrity constraints can be specified within the CREATE TABLE statement after the attributes are declared or they can be added later using the ALTER TABLE command.

The syntax of this command is,

CREATE TABLE tablename (col1name datatype [constraint], col2name datatype [constraint], …….colnname datatype [constraint]);

Here,

tablename specifies the name of the table to be created.

datatype specifies-

- char(size) - Fixed length string of characters of the set size. The size of the string is limited to 255 characters.
- Date
- Number (maxsize) - Number with a maximum number of digits specified by "maxsize".
- Number (maxdigits, maxright) - A decimal number with a maximum number of "maxdigits" with "a maximum number of digits to the right of the decimal, "maxright".
- varchar(maxsize) - A character string with variable lingth limited to "maxsize".

constraints are rules for the column. Possible values are:

- not null - The column values cannot be null.
- primary key - Each record is uniquely identified by this value.
- unique - No two values may be the same in the column

For Example

The statement to create a table EMPLOYEE_INFO with column NAME, DEPENDENT_NO, STATE, PIN is as follows-

CREATE TABLE EMPLOYEE_INFO (NAME varchar(20), DEPENDENT_NO number(5), STATE varchar(8), PIN number(5) unique);

DROP COMMAND –

The DROP command can be used to drop table, domain, constraints etc.

- **DROP SCHEMA**

The DROP SCHEMA command can be used to remove the whole schema.

There are two drop options

1. CASCADE

2. RESTRICT

CASCADE is used to remove the schema and all its tables, domains and other elements.

RESTRICT option is used to remove the schema only if it has no elements in it; otherwise DROP command will not be executed.

For example-

```
DROP SCHEMA EMPLOYEE CASCADE;
```

This will remove the EMPLOYEE database schema and its entire table, domain and other element.

- **DROP TABLE**

To delete a table within a schema the DROP TABLE command is used. This command deletes all records in the table and also removes the table definition from the schema.

For example: If we no longer needed to keep track of dependent of EMPLOYEE database then DROP TABLE command can be used to delete the DEPENDENT table as follows-

DROP TABLE DEPENDENT CASCADE;

If RESTRICT is applied then the table is dropped only if it is not referenced in any constraints or views.

GRANT COMMAND –

SQL GRANT is a command used to provide access or privileges on the database objects to the users.

The Syntax for the GRANT command is:

```
GRANT privilegename
ON     tablename
TO      username
[WITH GRANT OPTION];
```

- privilegename is the access right or privilege granted to the user. Some of the access rights are ALL, EXECUTE, and SELECT.
- tablename is the name of an database object like TABLE, VIEW etc.
- username is the name of the user or users to whom an access right is being granted. This can be also declared as PUBLIC to grant access rights to all users.
- WITH GRANT OPTION - allows a user to grant access rights to other users.

For Example:

```
GRANT EXECUTE
ON EMPLOYEE_INFO
TO USER_JOY;
```

This command grants execute permission on EMPLOYEE_INFO table to USER_JOY.

REVOKE COMMAND –

The REVOKE command is used to remove user access rights and privileges to the database objects.

The Syntax for the REVOKE command is:

```
REVOKE  privilegename
ON      tablename
FROM    username
```

- privilegename is the access right or privilege remove from the user. Here commands are ALL, EXECUTE, and SELECT.
- tablename is the name of an database object like TABLE, VIEW etc.
- username is the name of the user or users from whom an access right is being removed. This can be also declared as PUBLIC to remove access rights from all users.
- WITH GRANT OPTION - allows a user to grant access rights to other users.

For Example:

```
REVOKE SELECT
ON EMPLOYEE_INFO
FROM USER_JOY;
```

This command will remove a SELECT privilege on employee table from user USER_JOY. If REVOKE SELECT privilege is used on a table from a user, then the user will not be able to SELECT data from that table anymore.

CHECK YOUR PROGRESS

4. Fill in the blanks.

(i) _____ command is used to remove user access rights and privileges to the database objects

(ii) We use the _____ command To delete a table within a schema

(iii) _____ command is used to create a new relation

(iv) _____ command is responsible for making changes to a SQL table.

4.7.2 DATA MANIPULATION LANGUAGE (DML) COMMANDS

Data manipulation language (DML) commands are used to query and manipulate data in existing schema objects.

DELETE command –

The DELETE Statement is used to delete rows from a table. The Syntax of DELETE statement is:

DELETE
FROM <tablename>
WHERE <condition>;

Here <tablename> specifies name of the table. The optional WHERE condition identifies the rows in the column that get deleted. If it is not included then all the rows in the table are deleted.

For Example:

- To delete an employee record with EMP_id 100 from the EMPLOYEE_INFO table, the DELETE command is,

  DELETE FROM EMPLOYEE_INFO WHERE EMP_id = 100;

- To delete all the rows from the EMPLOYEE_INFO table

  DELETE FROM EMPLOYEE_INFO;

INSERT COMMAND –

The INSERT command is used to insert data as row into a table.

Syntax for INSERT is-

INSERT
INTO <tablename> [(col1, col2, col3,...,coln)]
VALUES (value1, value2, value3,...,valuen);

Here, col1, col2,..,.coln are the names of the columns in the table into which data are inserted.

To insert data into a row i.e to all the columns of the table, the [(col1, col2,..,.coln)] need not be specify. The command is as follows-

INSERT
INTO <tablename>
VALUES (value1, value2, value3,...,valuen);

For Example:

To insert a row to the EMPLOYEE_INFO table, the query is-

INSERT INTO EMPLOYEE_INFO (ID, NAME, DEPEDENT_NO, STATE, PIN) VALUES (105, 'Srinath', 3, ‘Assam’, 784164);

If we insert data to all columns in a row then the above INSERT statement can also be written as,

INSERT INTO EMPLOYEE_INFO VALUES (105, 'Srinath', 3, 'Assam', 784164);

Now, the table shall be –

Table: EMPLOYEE_INFO

ID	NAME	DEPENDENT_NO	STATE	PIN
105	Srinath	3	Assam	784164

Inserting data to a table through a SELECT statement:

Syntax –

INSERT
INTO <tablename> [(col1, col2 , ... ,coln)]
SELECT col1, col2, ...,coln
FROM < tablename> [**WHERE** condition];

Here, [(col1, col2, ...,coln)] and [WHERE condition] is optional.

For Example,

To insert a row into the EMPLOYEE_INFO table from a temporary table TEMP_INFO,

TEMP_INFO

ID	NAME	DEPENDENT_NO	STATE	PIN	AGE
15	koo	2	Assam	7841	20

The INSERT command is-

INSERT INTO EMPLOYEE_INFO (ID, NAME, DEPEDENT_NO, STATE, PIN) **SELECT** (ID, NAME, DEPEDENT_NO, STATE, PIN) **FROM** TEMP_INFO;

Now, the table is-

EMPLOYEE_INFO

ID	NAME	DEPENDENT_NO	STATE	PIN
105	Srinath	3	Assam	784164
15	koo	2	Assam	7841

If data is inserted in all the columns, the above insert statement can also be written as-

INSERT INTO EMPLOYEE_INFO
SELECT * **FROM** TEMP_INFO;

SELECT COMMAND –

The most commonly used SQL command SELECT is used to query or retrieve data from a table in the database. A query may

retrieve information from specified columns or from all of the columns in the table.

Syntax of SQL **SELECT** Statement is:

```
SELECT <col1,col2,..,coln>
 FROM <tablename>
 [WHERE <condition>]
```

Here,

- <tablename> is the name of the table from which the information is retrieved.
- <col1,col2,..,coln> include columns from which data is retrieved.
- Where clause is optional.

For example

To select the NAME from EMPLOYEE_INFO table the query is as follows-

SELECT NAME, PIN FROM EMPLOYEE_INFO;

This query will select the NAME and PIN columns from EMPLOYEE_INFO table.

NAME	PIN
Srinath	74164
Koo	7841

UPDATE COMMAND –

The UPDATE Statement is used to modify the existing rows in a table.

The Syntax for UPDATE Command is:

UPDATE <tablename>
SET <col1 = value1, col2=value2, ...>
[**WHERE** <condition>]

- tablename - the table name which has to be updated.
- col1, col2.. - the columns that gets changed.
- value1, value2... - are the new values.
- WHERE clause is optional. It identifies the rows that get affected. If it do not include then column values for all the rows get affected.

For Example:

To update the name of an employee in the table EMPLOYEE_INFO the UPDATE query is as follows,

```
UPDATE EMPLOYEE_INFO
SET NAME ='JOY'
WHERE id = 105;
```

Now the table is-

EMPLOYEE_INFO

ID	NAME	DEPENDENT_NO	STATE	PIN
105	JOY	3	Assam	784164
15	Koo	2	Assam	7841

4.7.3 TRANSACTION CONTROL COMMANDS

Transaction control commands manage changes made by DML commands. When a transaction successfully completes, transactional control commands finalize the transaction, either saving the changes made by the transaction to the database or reversing the changes made by the transaction. Transactional control commands are only used with the *DML* commands INSERT, UPDATE, and DELETE.

COMMIT COMMAND –

The COMMIT command is the transactional command used to save changes made by a transaction to the database since the last COMMIT or ROLLBACK command.

The syntax for COMMIT command is -

```
COMMIT;
```

ROLLBACK COMMAND –

The ROLLBACK command is used to abort transactions that have not already been saved to the database since the last COMMIT or ROLLBACK command was issued.

The syntax for ROLLBACK command is as follows:

```
ROLLBACK;
```

SAVEPOINT COMMAND –

A SAVEPOINT is a point in a transaction to rollback the transaction to a certain point without rolling back the entire transaction.

The syntax for SAVEPOINT command is -

```
SAVEPOINT <SAVEPOINTNAME>;
```

Where <SAVEPOINTNAME> is the name of the point to which rollback is needed.

SET TRANSACTION –

The SET TRANSACTION command can be used to initiate a database transaction and specify characteristics for the transaction.

The syntax for SET TRANSACTION is -

```
SET TRANSACTION [READ WRITE | READ ONLY];
```

CHECK YOUR PROGRESS

5. Fill in the blanks.

(i). _____ statement is used to delete rows from a table.

(ii). The most commonly used SQL command is _____

(iii). _____ command is used to abort transactions

(iv). _____ command can be used to initiate a database transaction.

4.8 LET US SUM UP

- The Structured Query Language (SQL) is the standard relational database language.
- Earlier SQL is called as SEQEUEL(Structured English QUEry Language)
- SQL is a standard language for accessing and manipulating databases
- Main parts of SQL are- Data definition language (DDL), Data manipulation language (DML).
- SQL databases use a n established standard adopted by ANSI & ISO.
- Character string data types are either fixed length or varying length.
- For fixed length string, a shorter string is filled with blank character to the right.
- Bit string data types are either of fixed length BIT(n) or varying length BIT VARYING(n), where n is the maximum number of bits.
- Numeric data types include integer numbers of various sizes and floating point numbers of various precessions
- A Boolean data type in SQL has value TRUE, FALSE or UNKNOWN.
- The DATE data type has ten width and format is YYYY-MM-DD

- The TIME data type has at least eight position, and format is HH:MM:SS
- TIMESTAMP data type put together the DATE and TIME field
- ALTER is the command used to add, modify data from tables, databases, and views**.**
- CREATE statement can be used to create schema, tables, and domains.
- The DROP command can be used to drop table, domain, constraints etc.
- There are two drop options CASCADE and RESTRICT.
- To delete a table within a schema the DROP TABLE command is used.
- DELETE Statement is used to delete rows from a table.
- INSERT command is used to insert data as row into a table.
- UPDATE Statement is used to modify the existing rows in a table.
- Transactional control commands are only used with the DML commands INSERT, UPDATE, and DELETE.
- COMMIT command is the transactional command used to save changes made by a transaction.
- ROLLBACK command is used to abort transactions.

4.9 ANSWERS TO CHECK YOUR PROGRESS

1. (i) Structured Query Language
   (ii) Data Definition Language
   (iii) Data Manipulation Language
   (iv) ANSI, ISO

2. (i) DATE, TIME
   (ii) fixed length, variable length
   (iii) DATE, TIME
   (iv) UNKNOWN

3. (i) REVOKE
   (ii) DROP
   (iii) CREATE

(iv) ALTER TABLE

4. (i) DELETE
   (ii) SELECT
   (iii) ROLLBACK
   (iv) SET TRANSACTION

4.10 FURTHER READINGS

Fundamentals of Database Systems

- R. Elmasri, S. B. Navathe. Pearson Education

Database System Concept

- A. Silberschatz, H. F. Korth, S. Sudarshan. MC Graw Hill, 5th edition

4.11 MODEL QUESTIONS

1. What is SQL?
2. What are the characteristics of SQL?
3. What are the advantage and disadvantage of SQL?
4. What are the data types and literals in SQL?
5. What is Boolean data type?
6. What is timestamp data type?
7. Create a table PROJECT (PRO_ID, PRO_NAME, LOCATION, STATE).
8. Add a new column COUNTRY to PROJECT. Insert data into 5 rows.
9. Delete the first row from the table PROJECT.
10. Select project name with project number 100 from the table PROJECT.

11. Update the project name with project number 100 to ‘water’.

12. Consider Hotel schema consisting of three tables Hotel, Booking and Guest having the following Schema. Create all the tables in SQL.

    Hotel (HotelID, HotelName, HotelCity). The primary key of the table is the HotelID.

    Guest (GuestID, GuestName, GuestAddress, GuestPhone). The primary key of the table is the GuestID.

    Booking (HotelID, GuestID, DateFrom, DateTo, RoomNo). Identify the Primary key to this table and all the Foreign keys.

    Room (RoomNo, HotelID, RoomType, RoomRent) Identify the Primary key to this table and all the Foreign keys.

13. Insert some sample meaningful data into the tables created above making sure that the Integrity rules are not violated.

*****

UNIT - 5: ELEMENTS OF SQL

UNIT STRUCTURE

5.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- know about SQL operators and procedures
- know the different SQL operations
- learn about triggers in SQL
- know what are SQL Views

5.2 INTRODUCTION

Data can be designed, manipulated and implemented in a variety of ways to obtain the required and desired results. We know about the design of data, and the storing of data in tables. In this unit we will learn how to insert, modify, and delete data in tables, and also how to view the data in tables. In other words, in this unit we will learn how to communicate with a database. We will learn about

the different operator in SQL that can be used to operate on data. The concept of a View, which in the SQL terminology is a single table that is derived from other tables, shall also be discussed in this unit. The various forms of JOINs that can be performed in a table shall be discussed in this unit. After that some of the SQL operators that are used to perform certain operations that are used to retrieve a specific set of desired result will be dealt with. Aggregate functions which are used to perform mathematical calculation shall also be dealt with. Another important concept in data manipulation is cursors which can be considered as a pointer that points to a single tuple (row) from the result of a query that retrieves multiple tuples. Later we deal with this important concept of cursors. Finally we deal with two important concepts related to databases which are Triggers and Procedures which can be used to perform some important tasks with the data in the database tables.

5.3 SQL OPERATORS

An operator is a reserved word or a character used in an SQL statement's (WHERE clause) to perform different operations. It is mainly used to specify conditions in an SQL statement. Some SQL operators are-

- Arithmetic operators
- Comparison operators
- Logical operators

Arithmetic Operators

Operator	Description
+	Addition - Adds two values
-	Subtraction - Subtracts two values
*	Multiplication - Multiplies two values
/	Division – divide two values
%	Modulus - Divides left hand operand by right hand operand and returns remainder

Comparison Operators

Operator	Description
=	Check if the value of two operands are equal then condition becomes true, Otherwise false.
!=	Check if the value of two operands are not equal then condition becomes true, Otherwise false.

>	Check if the value of left operand is greater than right operand then condition becomes true, Otherwise false.
<	Check if the value of left operand is less than right operand then condition becomes true, Otherwise false.
>=	Check if the value of left operand is greater than or equal to right operand then condition becomes true, Otherwise false.
<=	Check if the value of left operand is less than or equal to right operand then condition becomes true, Otherwise false.
!<	Check if the value of left operand is not less than right operand then condition becomes true, Otherwise false.
!>	Checks if the value of left operand is not greater than right operand then condition becomes true, Otherwise false.

Logical Operators

Operator	Description
ALL	Compare a value to all values in another set.
AND	Used to give multiple conditions in WHERE clause.
ANY	Compare a value to any value in the list according to the condition.
BETWEEN	Search for values that are within a range of values.
EXISTS	Search for an existing of a tuple in a specified table that meets certain condition.
IN	Compare a value to a list of literal values that have been specified.
LIKE	Compare a value to similar values using wildcard operators.
NOT	Complementing the meaning of the logical operator.
OR	Combine multiple conditions in WHERE clause.
IS NULL	Check the value is NULL or not.
UNIQUE	Searches for no duplicate row of a specified table.

OR Operator Example

```
SELECT * FROM EMPLOYEE_INFO
WHERE NAME='JOY' OR NAME='KOO';
```

This query will select all the information of employee named JOY and KOO from EMPLOYEE_INFO.

CHECK YOUR PROGRESS

1. Fill in the blanks

(i). != is a _____ operator in SQL

(ii). _____ operator is used to give multiple conditions in WHERE clause.

(iii)._____ operator searches for values that are within a range of values

5.4 TABLE AND VIEW

Table

SQL uses the term table, row and column for the formal relational term relation, tuple and attribute respectively. Tables are created by CREATE TABLE command. Already existing table can be altered and deleted using ALTER TABLE and DELETE table command respectively.

Views

A VIEW is a virtual table, through which a selective portion of the data from one or more tables can be seen. Views do not contain data of their own. They are used to restrict access to the database or to hide data complexity. A view is stored as a SELECT statement in the database. DML operations on a view like INSERT, UPDATE, DELETE affects the data in the original table upon which the view is based.

The Syntax to create a VIEW is

```
CREATE VIEW <viewname>
AS
SELECT <col1,col2….coln>
FROM tablename
 [WHERE condition];
```

viewname is the name of the VIEW.

For Example:

To create a view on the EMPLOYEE_INFO table the query is-

```
CREATE VIEW EMPLOYEE_VIEW
AS
SELECT NAME, PIN
FROM EMPLOYEE_INFO;
```

5.5 SQL JOINS

SQL Joins are used to combine related tuples from different tables. A Join condition is a part of the SQL query that retrieves rows from two or more tables. A SQL Join condition is used in the SQL WHERE Clause of select, update, delete statements.

The Syntax for joining two tables is:

```
SELECT <col1, col2, col3..., coln >
FROM <tablename1, tablename2>
WHERE <tablename1.col2 = tablename2.col1>;
```

If a SQL join condition is omitted or if it is invalid the joint operation will result in a Cartesian product. The Cartesian product returns a number of rows equal to the product of all rows in all the tables being joined. For example, if the first table has 20 rows and the second table has 10 rows, the result will be 20 * 10, or 200 rows.

SQL Joins can be classified into-

- Equi join and
- Non-Equi join.

Equijoins –

In equi join, join condition uses the only one comparison operator i.e equal operator (=) as the comparison operator.

There are two types of equijoins-

1. **Inner join –**

This inner join returns all the rows that satisfy the join condition which is specified within the query.

For example:

Given two tables DEPERTMENT and EMPLOYEE, display the project information for each employee the query will be as given below.

DEPERTMENT

DEPT_NO	DNAME	PRO_NAME
1	PHY	XY
2	CHE	AB

EMPLOYEE

EMP_ID	DEPT_NO	NAME
100	1	JOY
101	1	RAM
102	3	JONY

The query is as follows-

```
SELECT PRO_NAME
FROM DEPARTMENT, EMPLOYEE
```

```
WHERE  DEPARTMENT.DEPT_NO = EMPLOYEE.DEPT_NO;
```

Here attribute DEPT_NO is called join attribute. This attribute value is common to both the relation DEPARTMENT and EMPLOYEE.

Now, the result of the join query is-

EMP_ID	DEPT_NO	DNAME	PRO_NAME	NAME
100	1	PHY	XY	JOY
101	1	PHY	XY	RAM

If the join attributes have the same name in both relations then the join is called natural join. It is denoted by '*'.
The number of join conditions is (n-1), if there are more than two tables joined in a query where 'n' is the number of tables involved. The rule must be true to avoid Cartesian product.

2. Outer Join –

This outer join returns all rows from both tables which satisfy the join condition and also keeps rows which do not satisfy the join condition in other tables.

Two types of outer join-

- **Left outer join –**

Left outer join keeps every rows from left relation. If no matching rows found in right relation then that attributes are filled with NULL.
For example- If we apply LEFT OUTER JOIN to the above query then it will be,

```
SELECT PRO_NAME
FROM DEPARTMENT LEFT OUTER JOIN EMPLOYEE
WHERE EPARTMENT.DEPT_NO = EMPLOYEE.DEPT_NO;
```

- **Right outer join –**

Right outer join keeps every rows from right relation. If no matching rows found in left relation then that attributes are filled with NULL.

If we apply RIGHT OUTER JOIN to the above query then it will be,

```
SELECT PRO_NAME
FROM DEPARTMENT RIGHT OUTER JOIN EMPLOYEE
WHERE DEPARTMENT.DEPT_NO = EMPLOYEE.DEPT_NO;
```

Non-equijoin –

A non-equi join is a join whose condition can use all comparison operators like >=, <=, <, > except the equal (=) operator.

For example:

If you want to find the names of students from STUDENT_DETAIL table, who are not studying either Economics, the SQL query would be,

```
SELECT first_name, last_name, subject
FROM   STUDENT_DETAIL
WHERE subject != 'Economics'
```

CHECK YOUR PROGRESS

1. Fill in the blanks.

(i) A VIEW is a _____ table.

(ii) In equijoin, join condition uses the only _____ operator

(iii) Inner join returns all the rows that satisfy the _____.

(iv) Right outer join keeps every row from _____ relation.

5.6 UNION OPERATOR

The UNION operator is used to combine the result-set of two or more SELECT statements. For union operation the relations must be union compatible i.e each SELECT statement within the UNION must have the same number of columns and the columns must have similar data types. Also, the columns in each SELECT statement must be in the same order. The UNION operator selects only distinct values. Column names in the resultant table after a UNION are always from the column names in the first SELECT statement in the UNION.

- Syntax of **UNION** operator is-

  SELECT <col1,col2,…,coln> FROM tablename1
  UNION
  SELECT <col1,col2…,coln> FROM tablename2

  UNION ALL operator selects duplicate values-

- Syntax of **UNION ALL** is-

  SELECT <col1, col2,…,coln> FROM tablename1
  UNION ALL
  SELECT <col1, col2…,coln> FROM tablename2

The order of rows may not maintain after UNION operation i.e rows from the second table may appear before, after, or mixed with rows from the first table. To find a specific order ORDER BY is used.

5.7 INTERSECTION OPERATOR

The SQL INTERSECT operator returns only rows that appear in both relations. The INTERSECT operator removes duplicate rows from the output relation. To keeps duplicate roes INTERSECT ALL operator is used.

```
SELECT EMP_ID FROM EMP_INFO
INTERSECT
SELECT EMP_ID FROM DEPERTMENT;
```

5.8 EXCEPT/MINUS

The EXCEPT operator takes the distinct rows of one relation and returns the rows that do not appear in a second relation. The EXCEPT ALL operator does not remove duplicates. EXCEPT operator does not distinguish between NULLs.

```
SELECT EMP_ID FROM EMP_INFO
MINUS
SELECT EMP_ID FROM DEPERTMENT;
```

5.9 SQL QUERY AND SUBQUERY

A SQL query is an operation defined on one or more tables or views to retrieves data from it. Usually a SELECT statement is called a query. A query nested within another SQL query is called a sub-query. For example- The SELECT sub-query is a query that is nested in the main SELECT statement. The sub-query can be nested inside a SELECT, INSERT, UPDATE, or DELETE statement or inside another sub-query.

The places where a sub-query can be used in the place of a query are:

- Within the list of columns in the SELECT statement
- With the FROM clause
- With the WHERE clause
- With the HAVING clause
- With the GROUP BY clause.

A sub-query is an inner query, which is executed first before its outer query i.e the main query. The result is passed to outer query from inner query.

5.10 AGGREGATE FUNCTION

Aggregate functions are used to perform a calculation on a set of values. It returns a single value. Except COUNT, any other aggregate functions do not take null values. Aggregate functions are frequently used with the GROUP BY clause of the SELECT statement.

Aggregate functions are as follows-

Aggregate Function	Meaning
min(x)	Find the smallest value in a column.
max(x)	Find the largest value in a column.
avg(x)	Find the average value in a column.
stdev(x)	Find the standard deviation of the values in a column.
count(x)	Find the number of values in a column.
count(*)	Find the number of records in the table being searched.

CHECK YOUR PROGRESS

2. Fill in the blanks.

(i) Aggregate returns a single value except _____ function.

(ii) For union operator the relation must be _____

(iii) INTERSECT operator returns only rows that appear in _____

5.11 CURSORS

A cursor is a temporary work area created in the system memory when a SQL statement is executed. A cursor contains information on a select statement and the rows of data accessed by it. This temporary work area is used to store the data retrieved from the database, and manipulate this data. A cursor can hold more than one row, but can process only one row at a time. The set of rows the cursor holds is called the active set.

There are two types of cursors in PL/SQL:

1. Implicit cursors –

To process DML statements like, INSERT, UPDATE, and DELETE, implicit cursors are created by default. If a SELECT statement returns just one row then also implicit cursors are created. To check whether any row has been returned by the SELECT statement or not, implicit cursor is used.

2. Explicit cursors –

To execute a SELECT statement that returns more than one row an explicit cursor must be created. This cursor can store multiple rows records, but at time only the current row record can proceed. Both implicit and explicit cursors have the same functionality, but they differ in the way they are accessed. The status of the cursor for each of these attributes are defined in the below table.

Attributes	Return Value	Example
%FOUND	The return value is TRUE, if the DML statements like INSERT, DELETE and UPDATE affect at least one row and if SELECT ….INTO statement return at least one row.	SQL%FOUND
	The return value is FALSE, if DML statements like INSERT, DELETE and UPDATE do not affect row and if SELECT….INTO statement do not return a row.	
%NOTFOUND	The return value is FALSE, if DML statements like INSERT, DELETE and UPDATE at least one row and if SELECT ….INTO statement return at least one row.	SQL%NOTFOUND
	The return value is TRUE, if a DML statement like INSERT, DELETE and UPDATE does not affect even one row and if SELECT ….INTO statement does not return a row.	
%ROWCOUNT	Return the number of rows affected by the DML operations INSERT, DELETE, UPDATE, SELECT	SQL%ROWCOUNT

For Example:

Implicit cursor attribute is as follows-

```
DECLARE  rows_no number(5);
BEGIN
  UPDATE EMP_INFO
  SET age = age + 5;

  IF SQL%NOTFOUND THEN
    dbms_output.put_line(' none of the age is not updated');

   ELSIF SQL%FOUND THEN
     Rows_no := SQL%ROWCOUNT;
     dbms_output.put_line('age for ' || rows_no || 'Employee
     are updated');
    END IF;
END;
```

This PL/SQL block, updates the age of all the employees in the 'EMP_INFO' table. If ages do not get updated then a message saying 'None of the age is updated' is received. Else a message for age updates is received.

5.12 TRIGGERS

A trigger is triggered automatically when an associated DML statement like Insert, Delete, Update is executed.

There are two types of triggers –

- **Row level trigger** – An event is triggered for each row updated, inserted or deleted.
- **Statement level trigger** – An event is triggered for each SQL statement executed.

Syntax of Triggers-

```
CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF column_name]
ON  table_name
[REFERENCING OLD AS m NEW AS n]
[FOR EACH ROW]
WHEN (condition)
BEGIN
   SQL statements
END;
```

Here,

- CREATE [OR REPLACE] TRIGGER trigger_name – Creates a trigger named as trigger_name.

- {BEFORE | AFTER | INSTEAD OF} – Indicates the time for the trigger.
- {INSERT [OR] | UPDATE [OR] | DELETE} – Determines the triggering event. For more than one triggering events OR keyword can be used.
- [OF column_name] – Used only with update triggers to update a specific column.
- [ON table_name] – Identifies the name of the table to which the trigger is associated.
- [REFERENCING OLD AS m NEW AS n] – Reference the old and new values of the data being changed.
- [FOR EACH ROW] – Determine whether a trigger is a Row Level Trigger or level Trigger.
- WHEN (condition) – The trigger is triggered only for rows that satisfy the condition specified.

For Example:

The price of a product changes constantly. It is important to maintain the history of the prices of the products. We can create a trigger to update the 'product_price_history' table when the price of the product is updated in the 'product' table.

1. Create the 'product' table and 'product_price_history' table

```
CREATE TABLE product
(product_id number(5),
product_name varchar2(32),
supplier_name varchar2(32),
unit_price number(7,2) );

CREATE TABLE product_price_history
(product_id number(5),
product_name varchar2(32),
supplier_name varchar2(32),
unit_price number(7,2) );
```

2. Create the price_history_trigger and execute it.

```
CREATE or REPLACE TRIGGER price_history_trigger
BEFORE UPDATE OF unit_price
ON product
FOR EACH ROW
BEGIN
INSERT INTO product_price_history
VALUES
(:old.product_id,
```

```
 :old.product_name,
 :old.supplier_name,
 :old.unit_price);
END;
```

3. Lets update the price of a product.

UPDATE PRODUCT SET unit_price = 800 WHERE product_id = 100

5.13 PROCEDURE

A stored procedure or in simple a proc is a named PL/SQL block which performs one or more specific task. A procedure has a header and a body. The header consists of the name of the procedure and the parameters or variables passed to the procedure. The body consists or declaration section, execution section and exception section similar to a general PL/SQL Block. A procedure is similar to an anonymous PL/SQL Block but it is named for repeated usage. A procedure may or may not return any value.

General Syntax to create a procedure is:

```
CREATE [OR REPLACE] PROCEDURE
proc_name [list of parameters]
IS
   Declaration section
BEGIN
   Execution section
EXCEPTION
  Exception section
END;
```

CHECK YOUR PROGRESS

4. Fill in the blanks.

(i) The set of rows the cursor holds is called _____

(ii) _____ cursor can store multiple rows record, but at time only the current row record can procede

5.14 LET US SUM UP

- An operator is a reserved word or a character used in an SQL statement's (WHERE clause) to perform different operations
- SQL operators are- arithmetic operators, comparison operators, logical operators
- SQL uses the term table, row and column for the formal relational term relation, tuple and attribute respectively.
- A VIEW is a virtual table, through which a selective portion of the data from one or more tables can be seen
- SQL Joins are used to combine related tuples from different tables.
- A Join condition is a part of the SQL query that retrieves rows from two or more tables
- In equi join, join condition uses the only one comparison operator(=).
- Two types of equi join are- Outer join and Inner join.
- Inner join returns all the rows that satisfy the join condition which is specified within the query.
- This outer join returns all rows from both tables which satisfy the join condition and also keeps rows which do not satisfy the join condition in other tables.
- The UNION operator is used to combine the result-set of two or more SELECT statements.
- The order of rows may not maintain after UNION operation.
- The SQL INTERSECT operator returns only rows that appear in both relation.
- The EXCEPT operator takes the distinct rows of one relation and returns the rows that do not appear in a second relation.
- A query nested within another SQL query is called a sub-query
- Aggregate functions are used to perform a calculation on a set of values.
- A cursor is a temporary work area created in the system memory when a SQL statement is executed.

- A trigger is triggered automatically when an associated DML statement like Insert, Delete, Update is executed

5.15 ANSWERS TO CHECK YOUR PROGRESS

1.
(i). Comparsion operator
(ii). AND
(iii). BETWEEN

2.
(i). Virtual
(ii). Equal
(iii). join condition
(iv). right

3.
(i). COUNT
(ii). union compatible
(iii). both relation

4.
(i). active set
(ii). Explicit

5.16 FURTHER READINGS

Fundamentals of Database Systems

- R. Elmasri, S. B. Navathe. Pearson Education

Database System Concept

- A. Silberschatz, H. F. Korth, S. Sudarshan. MC Graw Hill, 5th edition

5.17 MODEL QUESTIONS

1. What are procedures in SQL?
2. What are the workings of the various SQL operators?
3. What is a JOIN in SQL?
4. What are INNER JOIN and OUTER JOIN in SQL?
5. Explain LEFT OUTER JOIN and RIGHT OUTER JOIN.
6. What are TRIGGERS and CURSORS in SQL?
7. What is an AGGREGATE function? Illustrate the workings of the different AGGREGATE functions
8. What is EQUIJOIN and NATURAL JOIN?
9. Create two tables EMPLOYEE (EMP_ID, NAME, AGE, STATE, PIN, WORK_ID) and WORK (WORK_ID, WNAME, LOCATION). Find out the all employees information who works in location 'Assam'.
10. Find out all the employee names who do not work as a "teacher".
11. Consider the following table

SNO (Supplier Number)	SNAME (Supplier Name)	STATUS	CITY	NOPUB (No of publications)
S1	Wiley	20	Kolkata	4000
S2	Pearson	20	Delhi	5500
S3	Prentice Hall	10	Chennai	4500
S4	McGraw Hill	30	Mumbai	5000
S5	Thompson Learning	10	Delhi	4200

(a) List the supplier number of the suppliers who have a status of 20 or more.

(b) List the suppliers whose NOPUB is more than 4300.

(c) List the number of Publishers whose NOPUB is more than 4500.

(d) Find the total and average number of publications.

(e) Find the details of publisher with maximum number of publication.

12. Consider the following relational database.

Employees (eno, ename, address, basic_salary)
Projects (Pno, Pname, enos-of-staff-alotted)
Workin (pno, eno, pjob)

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

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

(b) SELECT Pname
FROM projects
WHERE Pno IN (SELECT Pno
FROM projects
MINUS
GROUP BY eno
(SELECT DISTINCT Pno FROM workin));

13. CONSIDER the following relational schema.

EMPLOYEE

FNAME	INITIAL	LNAME	ENO	DOB	ADDRESS	SEX	SALARY	SUPERENO	DNO

DEPARTMENT

DNAME	DNUMBER	MGRENO	MGRSTARTDATE

DEPT_LOCATIONS

DNUMBER	DLOCATION

PROJECT

PNAME	PNUMBER	PLOCATION	DNUM

WORKS_ON

EENO	PNO	HOURS

DEPENDENT

EENO	DEPENDENT_NAME	SEX	DOB	RELATIONSHIP

Specify the following views in SQL on the database schema given.

a. A view that has the department name, manager name and manager salary for every department.
b. A view that has the employee name, supervisor name, and employee salary for each employee who works in the ‘Research’ department.

c. A view that has the project name, controlling department name, number of employees, and total hours worked per week on the project for each project.
d. A view that has the project name, controlling department name, number of employees, and total hours worked per week on the project for each project with more than one employee working on it.

14. Consider the following relational schema and specify the given queries below.

STUDENT

Name	StudentNumber	Class	Major
Uday	17	1	CS
Nitin	8	2	CS

COURSE

CourseName	CourseNumber	CreditHours	Department
Intro to Computer Science	CS1310	4	CS
Data Structures	CS3320	4	CS
Discrete Mathematics	MATH2410	3	MATH
Databases	CS3380	3	CS

SECTION

SectionIdentifier	CourseNumber	Semester	Year	Instructor
85	MATH2410	First	98	Jain
92	CS1310	First	98	Rao
102	CS3320	Second	99	Ramesh
112	MATH2410	First	99	Ravinder
119	CS1310	First	99	Rao
135	CS3380	First	99	Srinivas

GRADE_REPORT

StudentNumber	SectionIdentifier	Grade
17	112	B
17	119	C
8	85	A
8	92	A
8	102	B
8	135	A

PREREQUISITE

CourseNumber	PrerequisiteNumber
CS3380	CS3320
CS3380	MATH2410
CS3320	CS1310

a. Retrieve the names of all senior students majoring in 'CS'.
b. Retrieve the names of all courses taught by Professor Jain in 1998 and 1999.
c. For each section taught by Professor Jain, retrieve the course number, semester, year, and number of students who took the section.
d. Retrieve the name and transcript of each senior student (Class 5) majoring in CS. A transcript includes course name, course number, credit hours, semester, year, and grade for each course completed by the student.
e. Retrieve the names and major department of all straight-A students (students having a grade A in all their courses).
f. Retrieve the names and major departments of all students who do not have a grade of A in any of their courses.

*****

UNIT - 6: RELATIONAL DATABASE DESIGN

UNIT STRUCTURE

6.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- learn about relational database design
- know what is ER-to-Relational mapping
- know what are functional dependencies in a relation
- know about the concept of normalization
- know the different normal forms of a relation
- learn about some properties of a decomposition

6.2 INTRODUCTION

In the previous units we were introduced to some of the few basic concepts related to relational database design. We got to know what a relational model is and the different elementary components of any standard relational model. In this unit we focus further on some more advanced issues that relate to relational database design concepts.

We start this unit with a discussion on how to design a relational database schema based on a conceptual schema design, by presenting a 7-step algorithm to create a relational schema from

an entity-relationship (ER) schema which includes converting the basic ER-model constructs-entity types (strong and weak), binary relations (with various structural constraints), n-ary relationships, and attributes (simple, composite and multivalued into relations).

Then we define the concept of functional dependency, a formal constraint among attributes that is the main tool for formally measuring the appropriateness of attribute groupings into relational schemas. A relational schema is in a normal form when it satisfies certain desirable properties. The process of normalization consists of analyzing relations to meet increasingly more stringent normal forms leading to progressively better grouping of attributes. Normal forms are specified in terms of functional dependencies – which are identified by the database designer – and key attributes of relation schemas. We also describe the two desirable properties of decompositions, namely, the dependency preservation property and the lossless join property, which are both used by the design algorithms to achieve desirable decompositions.

6.3 RELATIONAL DATABASE DESIGN USING ER-TO-RELATIONAL MAPPING

In this section we describe an algorithm for ER-to-Relational mapping. We will use a COMPANY database example to illustrate the mapping procedure. The COMPANY ER schema is shown in **Fig 6.1** below, and the corresponding COMPANY relational database schema is shown in **Fig 6.2** to illustrate the mapping steps.

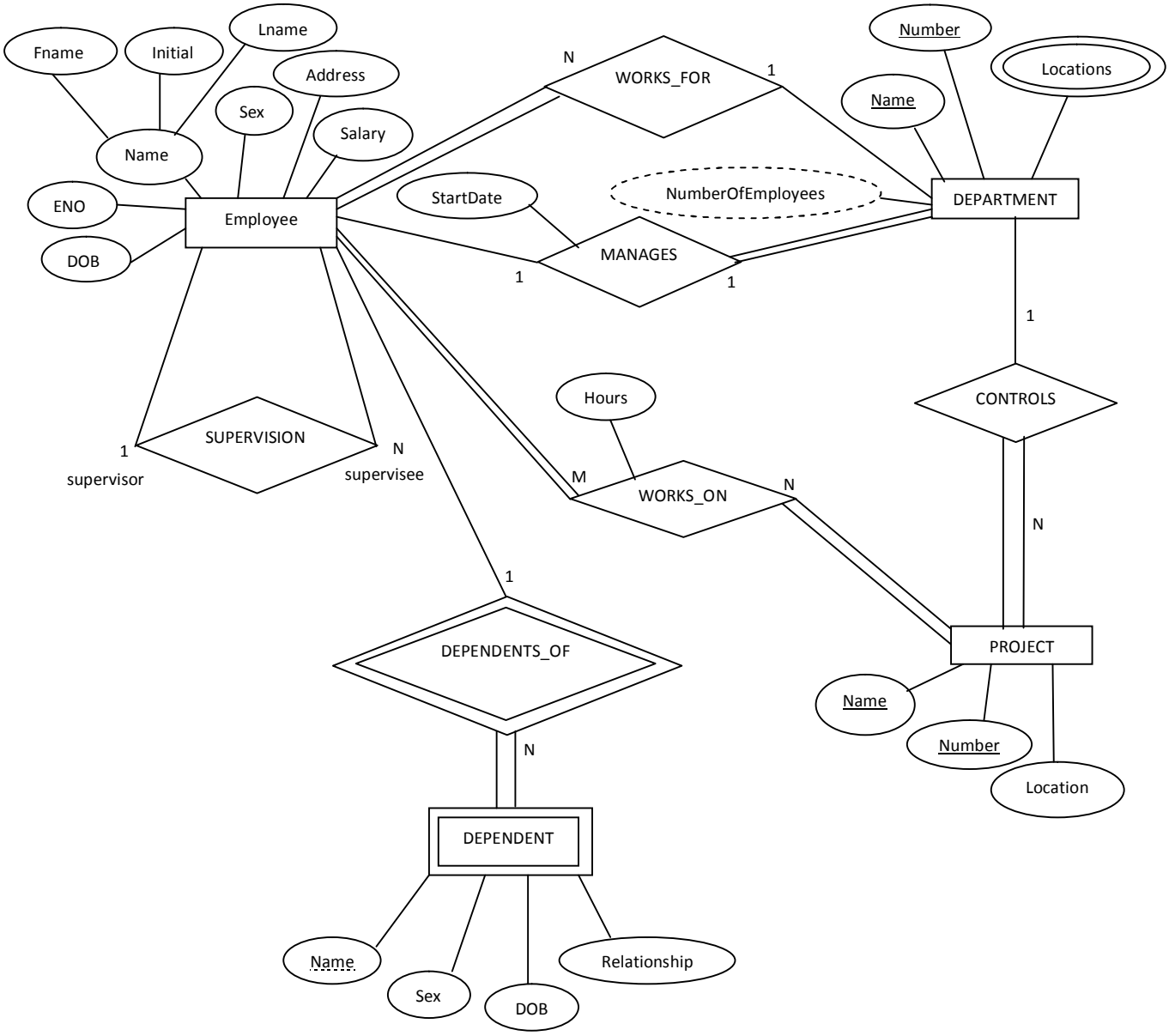


Fig 6.1: The ER conceptual schema diagram for a COMPANY database

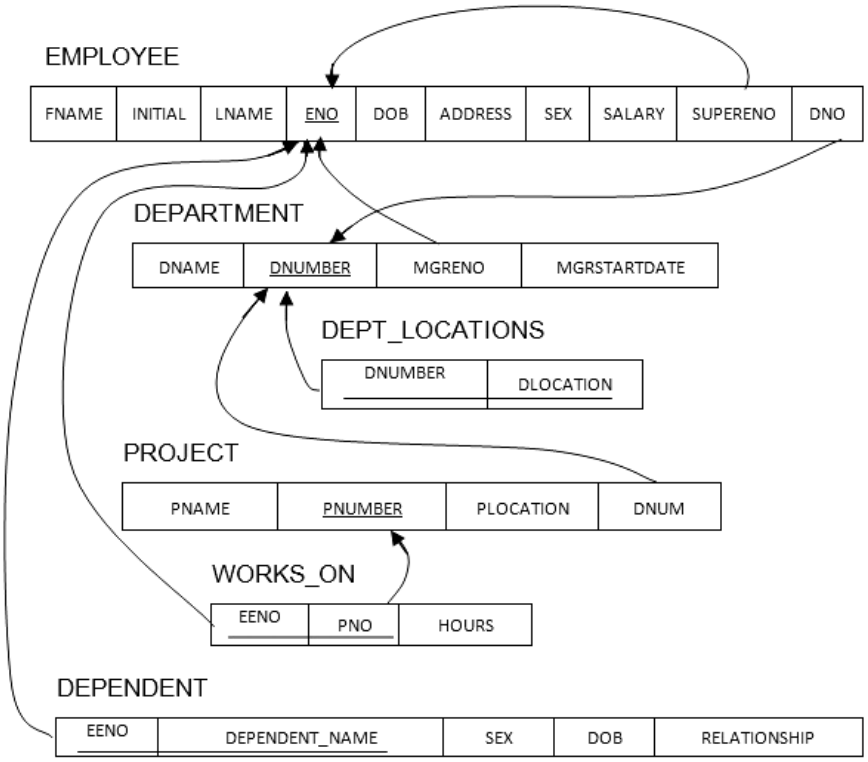


Fig 6.2: Result of mapping the COMPANY ER schema into a relational database schema

ER-to-Relational Mapping Algorithm

Step 1: Mapping of Regular Entity Types –

For each regular entity type E in the ER schema, create a relation R that includes all the simple attributes of E. include only the simple component attributes of a composite attribute. Choose one of the key attributes of E as primary key for R. if the chosen key of E is composite, the set of simple attributes that form it will together form the primary key of R. if multiple keys were identified for E during the conceptual design, the information describing the attributes that form each additional key is kept in order to specify secondary keys of relation R. knowledge about keys is also kept for indexing purposes and other types of analysis.

In our example, we create the relations EMPLOYEE, DEPARTMENT and PROJECT in **Fig 6.2** to correspond to the regular entity types EMPLOYEE, DEPARTMENT, and PROJECT from **Fig 6.1**. The foreign key and relationship attributes, if any, are not included yet; they will be added during subsequent steps. These include the attributes SUPERENO and DNO of EMPLOYEE, MGRENO and MGRSTARTDATE of DEPARTMENT, and DNUM of PROJECT. In our example, we choose ENO, DNUMBER, and PNUMBER as primary keys for the relations EMPLOYEE, DEPARTMENT, and PROJECT respectively. Knowledge that DNAME of DEPARTMENT and PNAME of PROJECT are secondary keys is kept for possible use later in the design.

The relations that are created from the mapping of entity types are sometimes called entity relations because each tuple represents an entity instance.

Step 2: Mapping of Weak Entity Types –

For each weak entity type W in the ER schema with owner entity type E, create a relation R and include all simple attributes of W as attributes of R. in addition, include as foreign key attributes of R the primary key attributes(s) of the relation(s) that correspond to the owner entity type(s); this takes care of the identifying relationship type of W. The primary key of R is the combination of the primary key(s) of the owner(s) and the partial key of the weak entity type W, if any. If there is a weak entity type E2 whose owner is also a weak entity type E1, then E1 should be mapped before E2 to determine its primary key first.

In our example, we create the relation DEPENDENT in this step to correspond to the weak entity type DEPENDENT. We include the primary key ENO of the EMPLOYEE relation – which corresponds to the owner entity type – as a foreign key attribute of DEPENDENT; we renamed it EENO, although this is not necessary. The primary key of the DEPENDENT relation is the combination {EENO, DEPENDENT_NAME} because DEPENDENT_NAME is the partial key of DEPENDENT.

Step 3: Mapping of Binary 1:1 Relationship Types –

For each binary 1:1 relationship type R in the ER schema, identify the relations S and T that correspond to entity types participating in R.

There are three possible approaches:

1. **Foreign key approach** – Choose one of the relations S, and include as a foreign key in S the primary key of T. it is better to choose an entity type with total participation in R in the role of S. include all the simple attributes of the 1:1 relationship type R as attributes of S.

   In our example, we map the 1:1 relationship type MANAGES from **Fig 6.1** by choosing the participating entity type DEPARTMENT to serve in the role of S, because its participation in the MANAGES relationship type is total (every department has a manager). We include the primary key of the EMPLOYEE relation as foreign key in the DEPARTMENT relation and rename it MGRENO. We also include the simple attribute STARTDATE of the MANAGES relation type in the DEPARTMENT relation and rename it MGRSTARTDATE.

2. **Merged relationship approach** – An alternative mapping of a 1:1 relationship type is possible by merging the two entity types and the relationship into a single relation. This may be appropriate when both participations are total.

3. **Cross-reference or relationship relation approach** – The third alternative is to set up a third relation R for the

purpose of cross-referencing the primary keys of the two relations S and T representing the entity types. The relation R is called a relationship relation, because each tuple in R represents a relationship instance that relates one tuple from S with one tuple of T.

Step 4: Mapping of Binary 1:N Relationship Types –

For each regular binary 1:N relationship type R, Identify the relation S that represents the participating entity type at the N-side of the relationship type. Include as foreign key in S the primary key of the relation T that represents the other entity type participating in R; this is done because each entity instance on the N-side is related to at most one entity instance on the 1-side of the relationship type. Include any simple attributes of the 1:N relationship type as attributes of S.

In our example, we now map the 1:N relationship types WORKS_FOR, CONTROLS and SUPERVISION from **Fig 6.1**. For WORKS_FOR we include the primary key DNUMBER of the DEPARTMENT relation as foreign key in the EMPLOYEE relation and call it DNO. For SUPERVISION we include the primary key of the EMPLOYEE relation as foreign key in the EMPLOYEE relation itself – because the relationship is recursive – and call it SUPERENO. The CONTROLS relationship is mapped to the foreign key attribute DNUM of PROJECT, which references the primary key DNUMBER of the DEPARTMENT relation

Another alternative approach that can be used here is again the relationship relation option as in the case of binary 1:1 relationships. We create a separate relation R whose attributes are the keys of S and T, and whose primary key is the same as the key of S. this option can be used if few tuples in S participate in the relationship to avoid excessive null values in the foreign key.

Step 5: Mapping of Binary M:N Relationship Types –

For each binary M:N relationship type R, create a new relation S to represent R. Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types; their combination will form the primary key of S. Also include any simple attributes of the M:N relationship type as attributes of S. Notice that we cannot represent an M:N relationship type by a single foreign key attribute in one of the participating relations because of the M:N cardinality ratio; we must create a separate relationship relation S.

In our example we map the M:N relationship type WORKS_ON from **Fig 6.1** by creating the relation WORKS_ON in **Fig 6.2**. We include the primary keys of the PROJECT and EMPLOYEE relations as foreign keys in WORKS_ON and rename them PNO and EENO respectively. We also include an attribute HOURS in WORKS_ON to represent the HOURS attribute of the relationship type. The primary key of the WORKS_ON relation is the combination of the foreign key attributes {EENO, PNO}.

Step 6: Mapping of Multi-valued attributes –

For each multi-valued attribute A, create a new relation R. this relation R will include an attribute corresponding to A, plus the primary key attribute K – as a foreign key in R – of the relation that represents the entity type or relationship type that has A as an attribute. The primary key of R is the combination of A and K. If the multi-valued attribute is composite, we include its simple components.

In our example, we create a relation DEPT_LOCATIONS. The attribute DLOCATION represents the multi-valued attribute LOCATIONS of DEPARTMENT, while DNUMBER – as foreign key – represents the primary key of the DEPARTMENT relation. The primary key of DEPT_LOCATIONS is the combination of {DNUMBER, DLOCATION}. A separate tuple will exist in DEPT_LOCATIONS for each location that a department has.

Step 7: Mapping of N–ary relationship types –

For each n–ary relationship type R, where n>2, create a new relation S to represent R. include as foreign key attributes in S the primary keys of the relations that represent the participating entity types. Also include any simple attributes of the n–ary relationship type as attributes of S. the primary key of S is usually a combination of all the foreign keys that reference the relations representing the participating entity types. However, if the cardinality constraints on any of the entity types E participating in R is 1, then the primary key of S should not include the foreign key attribute that references the relation E corresponding to E.

5.4 FUNCTIONAL DEPENDENCIES

The single most important concept in relational schema design theory is that of a functional dependency. 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,….A_n; let us think of the whole database as being described by a single universal relation schema R = {A_1, A_2, …. 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.

Definition – A functional dependency denoted by X→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 determine the values of the Y component. We also say that there is a functional dependency X to Y, or that Y is functionally dependent on X. the abbreviation for functional

dependency is FD or fd. The set of attributes X is called the left hand side of the FD, and Y is called the right hand side. Thus, X functionally determines Y in a relation schema R if, and only if, whenever two tuples of r(R) agree on their X–value, they must necessarily agree on their Y–value.

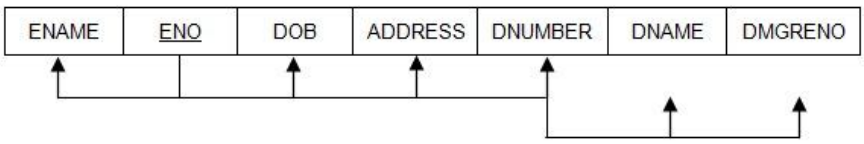
Consider the relation schema EMP_PROJ in **Fig 6.3 (b)**; from the semantics of the attributes, we know that the following functional dependencies should hold:

a. ENO → ENAME
b. PNUMBER → {PNAME, PLOCATION}
c. {ENO, PNUMBER} → HOURS

These functional dependencies specify that (a) the value of an employee's employee number (ENO) uniquely determines the employee name (ENAME), (b) the value of a project's number (PNUMBER) uniquely determines the project name (PNAME) and location (PLOCATION), and (c) a combination of ENO and PNUMER values uniquely determines the number of hours the employee currently works on the project per week (HOURS). Alternatively, we say that ENAME is functionally determined by ENO, or given a value of ENO, we know the value of ENAME.

A functional dependency is a property of the relation schema R not of a particular legal relation state r of R. hence an FD cannot be inferred automatically from a given relation extension r but must be defined explicitly by someone who knows the semantics of the attributes of R.

(a) EMP_DEPT



(b) EMP_PROJ

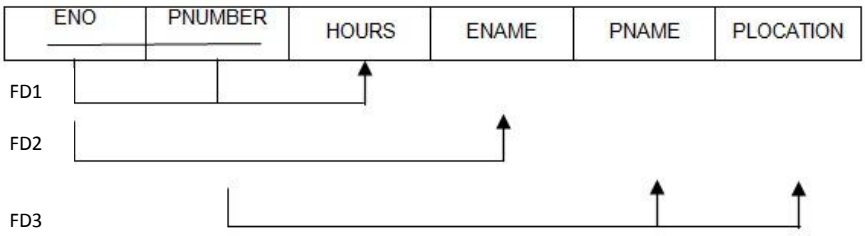


Fig 5.3: Two relation schemas

5.5 NORMALIZATION

Database normalization is the process of organizing the fields and tables of a relational database to minimize redundancy and dependency. Normalization usually involves dividing large tables into smaller (and less redundant) tables and defining relationships between them. The objective is to isolate data so that additions, deletions, and modifications of a field can be made in just one table and then propagated through the rest of the database via the defined relationships.

Edgar F. Codd, the inventor of the relational model, introduced the concept of normalization and initially he proposed three normal forms, which he called first, second and third normal form. A stronger definition of 3NF – called Boyce-Codd Normal Form (BCNF) was proposed later by Codd and Raymond F. Boyce. Informally, a relational database table is often described as "normalized" if it is in the Third Normal Form. Most 3NF tables are free of insertion, update, and deletion anomalies.

Objectives of normalization:

A basic objective of the first normal form defined by Codd was to permit data to be queried and manipulated using a "universal data sub-language" grounded in first-order logic (SQL is an example of such a data sub-language).

The objectives of normalization beyond 1NF (First Normal Form) were stated as follows by Codd:

1. **To free the database of modification anomalies** –

   When an attempt is made to modify (update, insert into, or delete from) a table, undesired side-effects may follow. Not all tables can suffer from these side-effects; rather, the side-effects can only arise in tables that have not been sufficiently normalized. An insufficiently normalized table might have one or more of the following characteristics:

   (a) The same information can be expressed on multiple rows; therefore updates to the table may result in logical inconsistencies. For example, each record in an "**Employees' Skills**" table of **Fig 6.4** might contain an Employee ID, Employee Address, and Skill; thus a change of address for a particular employee will potentially need to be applied to multiple records (one for each of his skills). If the update is not carried through successfully — if, that is, the employee's address is updated on some records but not others — then the table is left in an inconsistent state. Specifically, the table provides conflicting answers to the question of what this particular employee's address is. This phenomenon is known as an update anomaly.

Employee ID	Employee address	Skill
426	87 Syncamore	Typist
426	87 Syncamore	Shorthand
519	94 Park Street	Carpentry
519	96 Hall Avenue	Acting

Fig 6.4: Employees' Skills – An update anomaly, Employee 519 has different addresses on different records

(b) There are circumstances in which certain facts cannot be recorded at all. For example, each record in a "**Faculty and Courses**" table of **Fig 6.5** might contain a Faculty ID, Faculty Name, Faculty Hire Date, and Course Code—thus we can record the details of any faculty member who teaches at least one course, but we cannot record the details of a newly hired faculty member who has not yet been assigned to teach any courses except by setting the Course Code to null. This phenomenon is known as an insertion anomaly.

Faculty ID	Faculty name	Hire date	Course code
389	Dr. Giddins	12-04-1987	CS-765
407	Dr. Saperson	17-12-1999	CS-381
407	Dr. Saperson	17-12-1999	CS-592
332	Dr. Nilson	18-09-98	null

Fig 6.5: Faculty and Courses table – An insertion anomaly, until the new faculty member with ID 332 is assigned to teach at least one course, his details cannot be inserted

(c) Under certain circumstances, deletion of data representing certain facts necessitates deletion of data representing completely different facts. The "**Faculty and Courses**" table in **Fig 6.5** suffers from this type of anomaly, for if a faculty member temporarily ceases to be assigned to any courses, we must delete the last of the records on which that faculty member appears, effectively also deleting the faculty member. This phenomenon is known as a deletion anomaly.

2. **To minimize redesign in extending the database structure –**

When a fully normalized database structure is extended to allow it to accommodate new types of data, the pre-existing aspects of the database structure can remain largely or entirely unchanged. As a result, applications interacting with the database are minimally affected.

3. **To make the relational model more informative to users** –

Normalized tables, and the relationship between one normalized table and another, mirror real-world concepts and their interrelationships.

4. **To avoid bias towards any particular pattern of querying** –

Normalized tables are suitable for general-purpose querying. This means any queries against these tables, including future queries whose details cannot be anticipated, are supported. In contrast, tables that are not normalized lend themselves to some types of queries, but not others.

Background to normalization: definitions

1. Trivial functional dependency –

A trivial functional dependency is a functional dependency of an attribute on a superset of itself. {Employee ID, Employee Address} → {Employee Address} is trivial, as is {Employee Address} → {Employee Address}.

2. Full functional dependency –

An attribute is fully functionally dependent on a set of attributes X if it is:

- functionally dependent on X, and
- not functionally dependent on any proper subset of X. {Employee Address} has a functional dependency on {Employee ID, Skill}, but not a full functional dependency, because it is also dependent on {Skill}.Even by the removal of {Employee ID} functional dependency still holds between {Employee Address} and {Skill}.

3. Transitive dependency –

A transitive dependency is an indirect functional dependency, one in which X→Z only by virtue of X→Y and Y→Z.

4. Multi-valued dependency –

A multi-valued dependency is a constraint according to which the presence of certain rows in a table implies the presence of certain other rows.

5. Join dependency –

A table T is subject to a join dependency if T can always be recreated by joining multiple tables each having a subset of the attributes of T.

6. Superkey –

A superkey is a combination of attributes that can be used to uniquely identify a database record. A table might have many superkeys.

7. Candidate key –

A candidate key is a special subset of superkeys that do not have any extraneous information in them: it is a minimal superkey.

Example:

A table with the fields <Name>, <Age>, <SSN> and <Phone Extension> has many possible superkeys. Three of these are <SSN>, <Phone Extension, Name> and <SSN, Name>. Of those, only <SSN> is a candidate key as the others contain information not necessary to uniquely identify records ('SSN' here refers to Social Security Number, which is unique to each person).

8. Non-prime attribute –

A non-prime attribute is an attribute that does not occur in any candidate key. Employee Address would be a non-prime attribute in the "Employees' Skills" table.

9. Prime attribute –

A prime attribute, conversely, is an attribute that does occur in some candidate key.

10. Primary key –

One candidate key in a relation may be designated the primary key. While that may be a common practice (or even a required one in some environments), it is strictly notational and has no bearing on normalization. With respect to normalization, all candidate keys have equal standing and are treated the same.

CHECK YOUR PROGRESS

1. Fill in the blanks

(a) A _____ is a formal constraint among attributes.

(b) _____ consists of analyzing relations to meet increasingly more stringent normal forms.

(c) Normalization is the process of organizing tables to minimize _____ and _____.

(d) A _____ dependency is a functional dependency of an attribute on a superset of itself.

(e) A _____ is an indirect functional dependency.

(f) A _____ is a constraint due to which the presence of certain rows in a table implies the presence of certain other rows.

(g) A _____ is a _____ of attributes that can be used to uniquely identify a database record.

(h) A candidate key is a minimal _____.

(i) A _____ attribute is an attribute that does not occur in any _____.

6.6 NORMAL FORMS BASED ON PRIMARY KEYS

The normal forms of relational database theory provide criteria for determining a table's degree of vulnerability to logical inconsistencies and anomalies. The higher the normal form applicable to a table, the less vulnerable it is. Each table has a "**highest normal form**" (HNF): by definition, a table always meets the requirements of its HNF and of all normal forms lower than its HNF; also by definition, a table fails to meet the requirements of any normal form higher than its HNF.

The normal forms are applicable to individual tables; to say that an entire database is in normal form n is to say that all of its tables are in normal form n. Newcomers to database design sometimes suppose that normalization proceeds in an iterative fashion, i.e. a 1NF design is first normalized to 2NF, then to 3NF,

and so on. This is not an accurate description of how normalization typically works. A sensibly designed table is likely to be in 3NF on the first attempt; furthermore, if it is 3NF, it is overwhelmingly likely to have an HNF of 5NF. Achieving the "higher" normal forms (above 3NF) does not usually require an extra expenditure of effort on the part of the designer, because 3NF tables usually need no modification to meet the requirements of these higher normal forms.

The main normal forms are summarized below.

	Normal Form	Brief definition
1NF	First Normal Form	Table faithfully represents a relation, primarily meaning it has at least one candidate key
2NF	Second Normal Form	No non-prime attribute in the table is functionally dependent on a proper subset of any candidate key
3NF	Third Normal Form	Every non-prime attribute is non-transitively dependent on every candidate key in the table. The attributes that do not contribute to the description of the primary key are removed from the table. In other words, no transitive dependency is allowed.
BCNF	Boyce-Codd Normal Form	Every non-trivial functional dependency in the table is a dependency on a superkey
4NF	Fourth Normal Form	Every non-trivial multivalued dependency in the table is a dependency on a superkey
5NF	Fifth Normal Form	Every non-trivial join dependency in the table is implied by the superkeys of the table

First Normal Form –

First normal form (1NF) is a property of a relation in a relational database. A relation is in first normal form if the domain of each attribute contains only atomic values, and the value of each attribute contains only a single value from that domain. Edgar Codd defined a relation in first normal form to be one such that none of the domains of that relation should have elements which are themselves sets.

First normal form is an essential property of a relation in a relational database. Database normalization is the process of representing a database in terms of relations in standard normal forms, where first normal is a minimal requirement

The following scenario illustrates how a database design might violate first normal form.

Suppose a designer wishes to record the names and telephone numbers of customers. He defines a customer table which looks like this:

Customer

CustID	First Name	Surname	Telephone Number
123	Robert	Ingram	555-861-2025
456	Jane	Wright	555-403-1659
789	Maria	Fernandez	555-808-9633

The designer then becomes aware of a requirement to record multiple telephone numbers for some customers. He reasons that the simplest way of doing this is to allow the "Telephone Number" field in any given record to contain more than one value:

Customer

CustID	First Name	Surname	Telephone Number
123	Robert	Ingram	555-861-2025
456	Jane	Wright	555-403-1659 555-776-4100
789	Maria	Fernandez	555-808-9633

Assuming, however, that the Telephone Number column is defined on some telephone number-like domain, such as the domain of 12-character strings, the representation above is not in first normal form. It is in violation of first normal form as a single field has been allowed to contain multiple values. A typical relational database management system will not allow fields in a table to contain multiple values in this way.

A design that complies with 1NF

A design that is unambiguously in first normal form makes use of two tables: a Customer Name table and a Customer Telephone Number table.

Customer Name

Customer_ID	First Name	Surname
123	Robert	Ingram
456	Jane	Wright
789	Maria	Fernandez

Customer Telephone Number

Customer ID	Telephone Number
123	555-861-2025
456	555-403-1659
456	555-776-4100
789	555-808-9633

Repeating groups of telephone numbers do not occur in this design. Instead, each Customer-to-Telephone Number link appears on its own record. With Customer ID as key fields, a "parent-child" or one-to-many relationship exists between the two tables. A record in the "parent" table, Customer Name, can have many telephone number records in the "child" table, Customer Telephone Number, but each telephone number belongs to one, and only one customer. It is worth noting that this design meets the additional requirements for second and third normal form.

Second Normal Form –

A table that is in first normal form (1NF) must meet additional criteria if it is to qualify for second normal form. Specifically: a table is in 2NF if and only if it is in 1NF and no non-prime attribute is dependent on any proper subset of any candidate key of the table. A non-prime attribute of a table is an attribute that is not a part of any candidate key of the table.

Put simply, a table is in 2NF if and only if it is in 1NF and every non-prime attribute of the table is either dependent on the whole of a candidate key, or on another non-prime attribute.

Note that when a 1NF table has no composite candidate keys (candidate keys consisting of more than one attribute), the table is automatically in 2NF.

Consider a table describing employees' skills

Employees' Skills

Employee	**Skill**	**Current Work Location**
Jones	Typing	114 Main Street
Jones	Shorthand	114 Main Street
Jones	Whittling	114 Main Street
Bravo	Light Cleaning	73 Industrial Way
Ellis	Alchemy	73 Industrial Way
Ellis	Flying	73 Industrial Way
Harrison	Light Cleaning	73 Industrial Way

Neither {Employee} nor {Skill} is a candidate key for the table. This is because a given Employee might need to appear more than once (he might have multiple Skills), and a given Skill might need to appear more than once (it might be possessed by multiple Employees). Only the composite key {Employee, Skill} qualifies as a candidate key for the table.

The remaining attribute, Current Work Location, is dependent on only part of the candidate key, namely Employee. Therefore the table is not in 2NF. Note the redundancy in the way Current Work Locations are represented: we are told three times that Jones works at 114 Main Street, and twice that Ellis works at 73 Industrial Way. This redundancy makes the table vulnerable to update anomalies: it is, for example, possible to update Jones' work location on his "Typing" and "Shorthand" records and not update his "Whittling" record. The resulting data would imply

contradictory answers to the question "What is Jones' current work location?"

A 2NF alternative to this design would represent the same information in two tables: an "Employees" table with candidate key {Employee}, and an "Employees' Skills" table with candidate key {Employee, Skill}:

Employees

Employee	**Current Work Location**
Jones	114 Main Street
Bravo	73 Industrial Way
Ellis	73 Industrial Way
Harrison	73 Industrial Way

Employees' Skills

Employee	**Skill**
Jones	Typing
Jones	Shorthand
Jones	Whittling
Bravo	Light Cleaning
Ellis	Alchemy
Ellis	Flying
Harrison	Light Cleaning

Neither of these tables can suffer from update anomalies.

Not all 2NF tables are free from update anomalies, however. An example of a 2NF table which suffers from update anomalies is:

Tournament Winners

Tournament	**Year**	**Winner**	**Winner Date of Birth**
Des Moines Masters	1998	Chip Masterson	14 March 1977
Indiana Invitational	1998	Al Fredrickson	21 July 1975
Cleveland Open	1999	Bob Albertson	28 September 1968
Des Moines Masters	1999	Al Fredrickson	21 July 1975
Indiana Invitational	1999	Chip Masterson	14 March 1977

Even though Winner and Winner Date of Birth are determined by the whole key {Tournament, Year} and not part of it, particular Winner / Winner Date of Birth combinations are shown redundantly on multiple records. This leads to an update anomaly: if updates are not carried out consistently, a particular winner could be shown as having two different dates of birth. The underlying problem is the transitive dependency to which the Winner Date of Birth attribute is subject. Winner Date of Birth actually depends on Winner, which in turn depends on the key Tournament / Year.

This problem is addressed by third normal form (3NF).

Third normal form –

In computer science, the **third normal form** (3NF) is a normal form used in database normalization. 3NF was originally defined by E.F. Codd. Codd's definition states that a table is in 3NF if and only if both of the following conditions hold:

- The relation R (table) is in second normal form (2NF)
- Every non-prime attribute of R is non-transitively dependent (i.e. directly dependent) on every superkey of R.

A **non-prime attribute** of R is an attribute that does not belong to any candidate key of R. A transitive dependency is a functional dependency in which $X \rightarrow Z$ (*X* determines *Z*) indirectly, by virtue of $X \rightarrow Y$ and $Y \rightarrow Z$ (where it is not the case that $Y \rightarrow X$).

A 3NF definition that is equivalent to Codd's, but expressed differently, was given by Carlo Zaniolo. This definition states that a table is in 3NF if and only if, for each of its functional dependencies $X \rightarrow A$, **at least one** of the following conditions holds:

- *X* contains *A* (that is, $X \rightarrow A$ is trivial functional dependency), or
- *X* is a superkey, or
- Every element of *A*-*X*, the set difference between A and X, is a **prime attribute** (i.e., each column in *A*-*X* is contained in some candidate key)

Zaniolo's definition gives a clear sense of the difference between 3NF and the more stringent Boyce–Codd normal form (BCNF). BCNF simply eliminates the third alternative ("*A* is a prime attribute").

An example of a 2NF table that fails to meet the requirements of 3NF is:

Tournament Winners

Tournament	Year	Winner	Winner Date of Birth
Indiana Invitational	1998	Al Fredrickson	21 July 1975
Cleveland Open	1999	Bob Albertson	28 September 1968
Des Moines Masters	1999	Al Fredrickson	21 July 1975
Indiana Invitational	1999	Chip Masterson	14 March 1977

Because each row in the table needs to tell us who won a particular Tournament in a particular Year, the composite key {Tournament, Year} is a minimal set of attributes guaranteed to uniquely identify a row. That is, {Tournament, Year} is a candidate key for the table.

The breach of 3NF occurs because the non-prime attribute Winner Date of Birth is transitively dependent on the candidate key

{Tournament, Year} via the non-prime attribute Winner. The fact that Winner Date of Birth is functionally dependent on Winner makes the table vulnerable to logical inconsistencies, as there is nothing to stop the same person from being shown with different dates of birth on different records.

In order to express the same facts without violating 3NF, it is necessary to split the table into two.

Tournament Winners

Tournament	Year	Winner
Indiana Invitational	1998	Al Fredrickson
Cleveland Open	1999	Bob Albertson
Des Moines Masters	1999	Al Fredrickson
Indiana Invitational	1999	Chip Masterson

Player Dates of Birth

Player	Date of Birth
Chip Masterson	14 March 1977
Al Fredrickson	21 July 1975
Bob Albertson	28 September 1968

Update anomalies cannot occur in these tables, which are both in 3NF.

Boyce–Codd normal form –

Boyce–Codd normal form (or **BCNF** or **3.5NF**) is a normal form used in database normalization. It is a slightly stronger version of the third normal form (3NF). BCNF was developed by Raymond F. Boyce and Edgar F. Codd to address certain types of anomaly not dealt with by 3NF as originally defined.

If a relational scheme is in BCNF then all redundancy based on functional dependency has been removed, although other types of redundancy may still exist. A relational schema *R* is in Boyce–Codd normal form if and only if for every one of its dependencies $X \rightarrow Y$, at least one of the following conditions hold:

- $X \rightarrow Y$ is a trivial functional dependency ($Y \subseteq X$)
- *X* is a superkey for schema *R*

3NF tables not meeting BCNF

Only in rare cases does a 3NF table not meet the requirements of BCNF. A 3NF table which does not have multiple overlapping candidate keys is guaranteed to be in BCNF. Depending on what its functional dependencies are, a 3NF table with two or more overlapping candidate keys may or may not be in BCNF

An example of a 3NF table that does not meet BCNF is:

Today's Court Bookings

Court	Start Time	End Time	Rate Type
1	09:30	10:30	SAVER
1	11:00	12:00	SAVER
1	14:00	15:30	STANDARD
2	10:00	11:30	PREMIUM-B
2	11:30	13:30	PREMIUM-B
2	15:00	16:30	PREMIUM-A

- Each row in the table represents a court booking at a tennis club that has one hard court (Court 1) and one grass court (Court 2)
- A booking is defined by its Court and the period for which the Court is reserved
- Additionally, each booking has a Rate Type associated with it. There are four distinct rate types:
  - SAVER, for Court 1 bookings made by members
  - STANDARD, for Court 1 bookings made by non-members
  - PREMIUM-A, for Court 2 bookings made by members
  - PREMIUM-B, for Court 2 bookings made by non-members

The table's superkeys are:

- S_1 = {Court, Start Time}
- S_2 = {Court, End Time}
- S_3 = {Rate Type, Start Time}
- S_4 = {Rate Type, End Time}
- S_5 = {Court, Start Time, End Time}
- S_6 = {Rate Type, Start Time, End Time}
- S_7 = {Court, Rate Type, Start Time}
- S_8 = {Court, Rate Type, End Time}
- S_T = {Court, Rate Type, Start Time, End Time}, the trivial superkey

Note that even though in the above table *Start Time* and *End Time* attributes have no duplicate values for each of them, we still have to admit that in some other days two different bookings on court 1 and court 2 could *start at the same time* or *end at the same time*. This is the reason why {Start Time} and {End Time} cannot be considered as the table's superkeys.

However, only S_1, S_2, S_3 and S_4 are candidate keys (that is, minimal superkeys for that relation) because e.g. $S_1 \subset S_5$, so S_5 cannot be a candidate key.

Recall that 2NF prohibits partial functional dependencies of non-prime attributes (i.e. an attribute that does not occur in ANY candidate key) on candidate keys, and that 3NF prohibits transitive functional dependencies of non-prime attributes on candidate keys.

In **Today's Court Bookings** table, there are no non-prime attributes: that is, all attributes belong to some candidate key. Therefore the table adheres to both 2NF and 3NF. The table does not adhere to BCNF. This is because of the dependency Rate Type → Court, in which the determining attribute (Rate Type) is neither a candidate key nor a superset of a candidate key. Dependency Rate Type → Court is respected as a Rate Type should only ever apply to a single Court.

The design can be amended so that it meets BCNF.

Rate Types

Rate Type	Court	Member Flag
SAVER	1	Yes
STANDARD	1	No
PREMIUM-A	2	Yes
PREMIUM-B	2	No

Today's Bookings

Rate Type	Start Time	End Time
SAVER	09:30	10:30
SAVER	11:00	12:00
STANDARD	14:00	15:30
PREMIUM-B	10:00	11:30
PREMIUM-B	11:30	13:30
PREMIUM-A	15:00	16:30

The candidate keys for the Rate Types table are {Rate Type} and {Court, Member Flag}; the candidate keys for the Today's Bookings table are {Rate Type, Start Time} and {Rate Type, End Time}. Both tables are in BCNF. Having one Rate Type associated with two different Courts is now impossible, so the anomaly affecting the original table has been eliminated.

Fourth normal form –

Fourth normal form (**4NF**) is a normal form used in database normalization. Introduced by Ronald Fagin, 4NF is the next level of normalization after Boyce–Codd normal form (BCNF). Whereas the second, third, and Boyce–Codd normal forms are concerned with functional dependencies, 4NF is concerned with a more general type of dependency known as a multivalued dependency. A Table is in 4NF if and only if, for every one of its non-trivial multivalued dependencies X ⇒ Y, X is a superkey—that is, X is either a candidate key or a superset thereof.

Multivalued dependencies

If the column headings in a relational database table are divided into three disjoint groupings X, Y, and Z, then, in the context of a particular row, we can refer to the data beneath each group of headings as x, y, and z respectively. A multivalued dependency X ⇒ Y signifies that if we choose any x actually occurring in the table

(call this choice x_c), and compile a list of all the $x_c$yz combinations that occur in the table, we will find that x_c is associated with the same *y* entries regardless of z.

A **trivial multivalued dependency** X ⇒Y is one where either Y is a subset of X, or X and Y together form the whole set of attributes of the relation.

A functional dependency is a special case of multivalued dependency. In a functional dependency X → Y, every x determines exactly one y, never more than one.

Example

Pizza Delivery Permutations

Restaurant	Pizza Variety	Delivery Area
A1 Pizza	Thick Crust	Springfield
A1 Pizza	Thick Crust	Shelbyville
A1 Pizza	Thick Crust	Capital City
A1 Pizza	Stuffed Crust	Springfield
A1 Pizza	Stuffed Crust	Shelbyville
A1 Pizza	Stuffed Crust	Capital City
Elite Pizza	Thin Crust	Capital City
Elite Pizza	Stuffed Crust	Capital City
Vincenzo's Pizza	Thick Crust	Springfield
Vincenzo's Pizza	Thick Crust	Shelbyville
Vincenzo's Pizza	Thin Crust	Springfield
Vincenzo's Pizza	Thin Crust	Shelbyville

Each row indicates that a given restaurant can deliver a given variety of pizza to a given area.

The table has no non-key attributes because its only key is {Restaurant, Pizza Variety, Delivery Area}. Therefore it meets all normal forms up to BCNF. If we assume, however, that pizza varieties offered by a restaurant are not affected by delivery area, then it does not meet 4NF. The problem is that the table features two non-trivial multivalued dependencies on the {Restaurant} attribute (which is not a superkey). The dependencies are:

- {Restaurant} ⇒{Pizza Variety}
- {Restaurant} ⇒{Delivery Area}

These non-trivial multivalued dependencies on a non-superkey reflect the fact that the varieties of pizza a restaurant offers are independent from the areas to which the restaurant delivers. This state of affairs leads to redundancy in the table: for example, we are told three times that A1 Pizza offers Stuffed Crust, and if A1 Pizza starts producing Cheese Crust pizzas then we will need to add multiple rows, one for each of A1 Pizza's delivery areas. There is, moreover, nothing to prevent us from doing this incorrectly: we might add Cheese Crust rows for all but one of A1 Pizza's delivery areas, thereby failing to respect the multivalued dependency {Restaurant} ⇒{Pizza Variety}.

To eliminate the possibility of these anomalies, we must place the facts about varieties offered into a different table from the facts about delivery areas, yielding two tables that are both in 4NF:

Varieties By Restaurant

Restaurant	Pizza Variety
A1 Pizza	Thick Crust
A1 Pizza	Stuffed Crust
Elite Pizza	Thin Crust
Elite Pizza	Stuffed Crust
Vincenzo's Pizza	Thick Crust
Vincenzo's Pizza	Thin Crust

Delivery Areas By Restaurant

Restaurant	Delivery Area
A1 Pizza	Springfield
A1 Pizza	Shelbyville
A1 Pizza	Capital City
Elite Pizza	Capital City
Vincenzo's Pizza	Springfield
Vincenzo's Pizza	Shelbyville

Fifth normal form –

Fifth normal form (**5NF**), also known as **project-join normal form** (**PJ/NF**) is a level of database normalization designed to reduce redundancy in relational databases recording multi-valued facts by isolating semantically related multiple relationships. A table is said to be in the 5NF if and only if every join dependency in it is implied by the candidate keys.

A join dependency *{A, B, … Z} on R is implied by the candidate key(s) of R if and only if each of A, B, …, Z is a superkey for R.

Example

Traveling Salesman Product Availability By Brand

Traveling Salesman	Brand	Product Type
Jack Schneider	Acme	Vacuum Cleaner
Jack Schneider	Acme	Breadbox
Willy Loman	Robusto	Pruning Shears
Willy Loman	Robusto	Vacuum Cleaner
Willy Loman	Robusto	Breadbox
Willy Loman	Robusto	Umbrella Stand
Louis Ferguson	Robusto	Vacuum Cleaner
Louis Ferguson	Robusto	Telescope
Louis Ferguson	Acme	Vacuum Cleaner
Louis Ferguson	Acme	Lava Lamp
Louis Ferguson	Nimbus	Tie Rack

The table's predicate is: Products of the type designated by Product Type, made by the brand designated by Brand, are available from the traveling salesman designated by Traveling Salesman.

In the absence of any rules restricting the valid possible combinations of Traveling Salesman, Brand, and Product Type, the three-attribute table above is necessary in order to model the situation correctly.

Suppose, however, that the following rule applies: A Traveling Salesman has certain Brands and certain Product Types in his repertoire. If Brand B is in his repertoire, and Product Type P is in his repertoire, then (assuming Brand B makes Product Type P), the Traveling Salesman must offer only the products of Product Type P made by Brand B.

In that case, it is possible to split the table into three:

Product Types By Traveling Salesman

Traveling Salesman	Product Type
Jack Schneider	Vacuum Cleaner
Jack Schneider	Breadbox
Willy Loman	Pruning Shears
Willy Loman	Vacuum Cleaner
Willy Loman	Breadbox
Willy Loman	Umbrella Stand
Louis Ferguson	Telescope
Louis Ferguson	Vacuum Cleaner
Louis Ferguson	Lava Lamp
Louis Ferguson	Tie Rack

Brands By Traveling Salesman

Traveling Salesman	Brand
Jack Schneider	Acme
Willy Loman	Robusto
Louis Ferguson	Robusto
Louis Ferguson	Acme
Louis Ferguson	Nimbus

Product Types By Brand

Brand	Product Type
Acme	Vacuum Cleaner
Acme	Breadbox
Acme	Lava Lamp
Robusto	Pruning Shears
Robusto	Vacuum Cleaner
Robusto	Breadbox
Robusto	Umbrella Stand
Robusto	Telescope
Nimbus	Tie Rack

5.7DEPENDENCY PRESERVING DECOMPOSITION

It would be useful if each functional dependency $X \rightarrow Y$ specified in F either appeared directly in one of the relation schemas R, in the decomposition D or could be inferred from the dependencies that appear in some R. informally, this is the dependency preservation condition. We want to preserve the dependencies because each dependency in F represents a constraint on the database. If one of the dependencies is not represented in some individual relation R of the decomposition, we cannot enforce this constraint by dealing with an individual relation: instead, we have to join two or more of the relations in the decomposition and then check that the functional dependency holds in the result of the JOIN operation. This is clearly an inefficient and impractical procedure.

It is not necessary that the exact dependencies specified in F appear themselves in individual relations of the decomposition D. it is sufficient that the union of the dependencies that hold on the individual relations in D be equivalent to F. we now define these concepts more formally.

Definition: Given a set of dependencies F on R, the projection of F on R, denoted by $\pi R_i(F)$ where R, is a subset of R, is the set of dependencies $X \rightarrow Y$ in F such that the attributes in $X \rightarrow Y$ are all contained in R. hence, the projection of F on each relation schema R, in the decomposition D is the set of functional dependencies in F^+, the closure of F, such that all their left-and right-hand-side attributes are in R. We say that a decomposition $D=\{R_1, R_2,\ldots, R_m\}$ of R is dependency-preserving with respect to F if the union of the projections of F on each R, in D is equivalent to F; that is,

$$((\pi R_1(F)) \cup \ldots \cup (\pi_{Rm}(F)))^+ = F^4$$

If a decomposition is not dependency-preserving, some dependency is lost in the decomposition. To check that a lost dependency holds we must take the join of two or more relations in the decomposition to get a relation that includes all left-and right-hand-side attributes of the lost dependency, and then check that the dependency holds on the result of the JOIN – an option that is not practical.

An example of a decomposition that does not preserve dependencies is show in **Fig 6.6**, in which the functional dependency FD2 is lost when LOTS1A is decomposed into {LOTS1AX, LOTS1AY}. The decomposition in **Fig 6.7**, however are dependency preserving.

LOTS1A

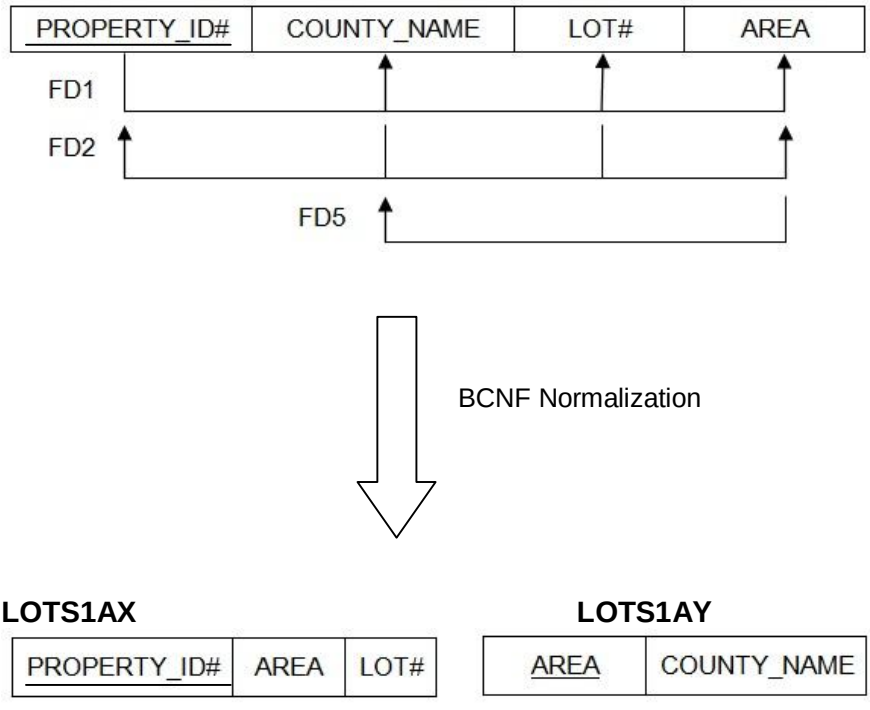
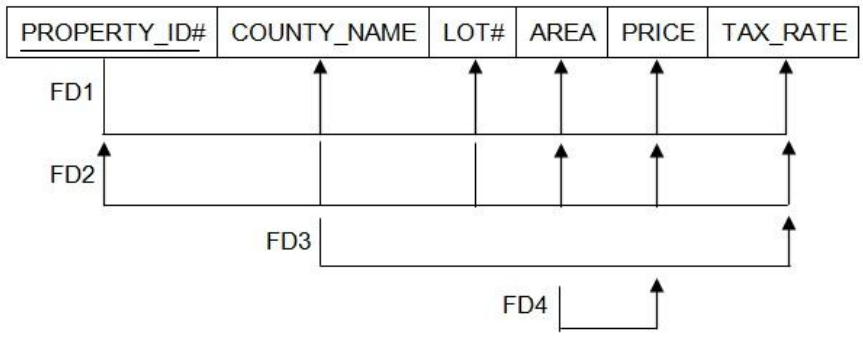
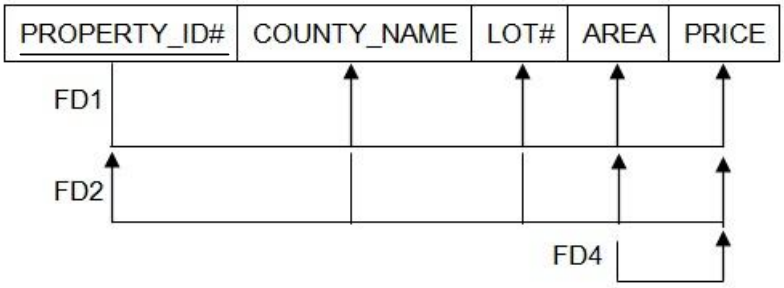


Fig 6.6: Boyce-Codd Normal form. BCNF Normalization of LOTS1A with the Functional Dependency FD2 being lost in the decomposition

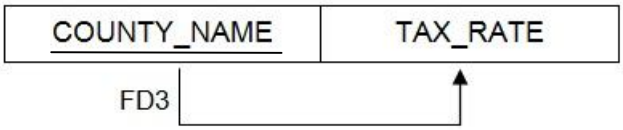
(a) LOTS



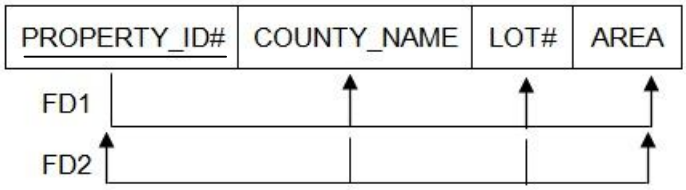
(b) LOTS1



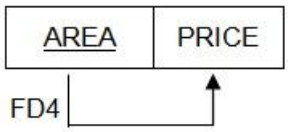
LOTS2



(c) LOTS1A



LOTS1B



(d)

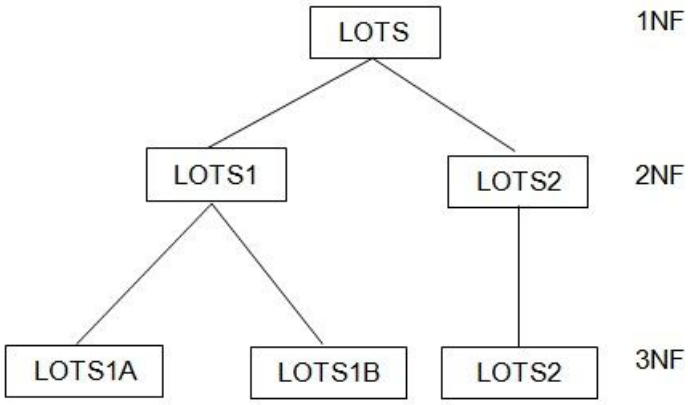


Fig 6.7: Normalization into 2NF and 3NF. (a) The LOTS relation with its functional dependencies FD1 through FD4 (b) Decomposing into the 2NF relations LOTS1 and LOTS2. (c) Decomposing LOTS1 into the 3NF relations LOTS1A and LOTS1B. (d) Summary of the progressive normalization of LOTS

6.8 LOSSLESS JOIN PROPERTY OF A DECOMPOSITION

Another property that a decomposition D should possess is the lossless join or non-additive join property, which ensures that no spurious tuples are generated when a NATURAL JOIN operation is applied to the relations in the decomposition.

Definition:

Formally a decomposition D = {R_1, R_2, … , R} of R has the lossless (non-additive) join property with respect to the set of dependencies F on R if, for every relation state r of R that satisfies F, the following holds where * is the NATURAL JOIN of all the relations in D.

$$*(\pi R_1(r), \ldots, \pi_{Rm}(r)) = r$$

The word loss in lossless refers to loss of information, not to loss of tuples. If a decomposition does not have the lossless join property, we may get additional spurious tuples after the PROJECT (π) and NATURAL JOIN (*) operations are applied; these additional tuples represent erroneous information. We prefer the term non-additive join because it describes the situation more accurately. If the property holds on a decomposition we are guaranteed that no spurious tuples bearing wrong information are added to the result after the project and natural join operations are applied.

Algorithm to test for lossless join property:

Input: A universal relation R, a decomposition D={R_1, R_2, … , R_m} of R, and a set F of functional dependencies.

1. Create an initial matrix S with one row i for each relation R_i in D, and one column j for each attribute A_j in R.
2. Set S(I , j) := b_{ij} for all matrix entries. (* each b_{ij} is a distinct symbol associated with indices (i, j) *)
3. For each row i representing relation schema R,

{ for each column j representing attribute A,
{ if (relation R includes attribute A,) then set S (I, j).. = a_j; } ; };
(* each a is a distinct symbol associated with index (j) *)

4. Repeat the following loop until a complete loop execution results in no changes to S

   { for each functional dependency X → Y in F
   { for all rows in S that have the same symbols in the columns corresponding to attributes in X
   { make the symbols in each column that correspond to an attribute in Y be the same in all these rows as follows: if any of the rows has an “a” symbol for the column, set the other rows to the same “a” symbol in the column. If no “a” symbol exists for the attribute in any of the rows, choose one of the “b” symbols that appears in one of the rows for

the attribute and set the other rows to that same "b" symbol in the column; } ; } ; } ;

5. If a row is made up entirely of "a" symbols, then the decomposition has the lossless join property; otherwise, it does not.

Given a relation R that is decomposed into a number of relations R_1, R_2, …, R_m, The above Algorithm to test for lossless join property begins the matrix S that we consider to be some relation state r of R. row i in S represents a tuple t_i (corresponding to relation R_i) that has "a" symbols in the columns that correspond to the attributes of R, and "b" symbols in the remaining columns. The algorithm then transforms the rows of this matrix (during the loop of step 4) so that they represent tuples that satisfy all the functional dependencies in F. at the end of step 4, any two rows in S – which represent two tuples in r – that agree in their values for the left-hand-side attributes X of a functional dependency X → Y in F will also agree in their values for the right-hand-side attributes Y. it can be shown that after applying the loop of step 4, if any row in S ends up with all "a" symbols then the decomposition D has the lossless join property with the respect to F.

CHECK YOUR PROGRESS

2. Fill in the blanks:

(a) In a ____ table no non-prime attribute in the table is _____ on a proper subset of any _____ key.

(b) In a ____ table every non-trivial functional dependency in the table is a dependency on a _____.

(c) In a ___ table every non-trivial join dependency in the table is implied by the _____ of the table.

(d) If a decomposition is not _____, some dependency is lost in the decomposition.

(e) The _____ property of a decomposition which ensures that no spurious tuples are generated when a _____ operation is applied to the relations in the decomposition

6.9 LET US SUM UP

- A relational schema is in a ***normal form*** when it satisfies certain desirable properties.
- Normal forms are specified in terms of ***functional dependencies***.
- A ***functional dependency*** is a constraint between two sets of attributes from the database.
- ***Normalization*** usually involves dividing large tables into smaller tables and defining relationships between them.
- The normal forms of relational database theory provide criteria for determining a table's degree of vulnerability to ***logical inconsistencies*** and ***anomalies***.
- ***First normal form (1NF)*** is a property of a relation in a relational database.
- A table is in 2NF if and only if it is in 1NF and no ***non-prime attribute*** is dependent on any proper subset of any candidate key of the table.
- If a relational scheme is in ***BCNF*** then all redundancy based on functional dependency has been removed.

6.10 ANSWERS TO CHECK YOUR PROGRESS

1.
   (a) functional dependency.
   (b) Normalization.
   (c) redundancy, dependency.
   (d) trivial functional.
   (e) transitive dependency.
   (f) multi-valued dependency.
   (g) superkey, combination.
   (h) superkey.
   (i) non-prime, candidate key.
2.
   (a) 2NF, functionally dependent, candidate.
   (b) BCNF, superkey
   (c) 5NF, superkeys
   (d) dependency-preserving
   (e) lossless join, NATURAL JOIN

6.11 FURTHER READINGS

- Prof. Sushant S. Sundikar: Introduction to Database Management System
- Elmasri, Navathe, Somayajulu, Gupta: Fundamentals of Database Systems

6.12 MODEL QUESTIONS

1. What do you mean by relational database schema? Illustrate an example.

2. Discuss insert, delete and update anomalies.

3. Discuss the relation between the ER model constructs and the relational model constructs. Show how each ER model construct can be mapped to the relational model.

4. What is Functional Dependency? Explain.

5. Define First, Second and Third normal forms with examples.

6. Define BCNF? How does it differ from 3NF, and why is it considered a stronger form of BCNF?

7. Consider the relational schema R(ABC) with FDs AB → C, C → A. show that the schema R is in 3NF but not in BCNF.

8. What is the dependency preserving property of decomposition and what is its importance?

9. What is lossless join property of decomposition?

UNIT - 7: TRANSACTION PROCESSING CONCEPTS

UNIT STRUCTURE

7.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- learn the basics of transaction processing
- differentiate between single-user and multiuser systems
- know some basic concepts of DBMS transactions
- know the concepts of concurrency and recovery
- learn the different elements and operations of a transaction
- learn about characterizing schedules

7.2 INTRODUCTION

In the previous units we came to know about several basic concepts of a Database Management System. We got to know the different database models, their different architectures, the relation between elements etc. We were also introduced to the relational model that uses the advantage of the Structured Query Language to perform several database operations. Normalization, which is a primary operation to be performed in a database, was also discussed along with its different forms.

In this unit we discuss the concept of transaction processing systems. We also define the concept of a transaction, which is used to represent a logical unit of database processing that must be completed in its entirety to ensure correctness. We also discuss the concurrency control problem, which occurs when multiple transactions submitted by various users interfere with one another in a way that produces incorrect results.

7.3 INTRODUCTION TO TRANSACTION PROCESSING

Transaction Processing Systems are the systems with large databases and hundreds of concurrent users executing database transactions. The concept of transaction provides a mechanism for describing logical units of database processing. Examples of Transaction Processing Systems include systems for reservations, banking, credit card processing, stock markets, supermarket checkouts etc. Such systems require high availability and fast response time for hundreds of concurrent users.

7.3.1 SINGLE USER V/S MULTIUSER SYSTEMS

In this section we compare single-user and multiuser database systems and demonstrate how concurrent execution of transactions can take place in multiuser systems.

One criterion for classifying a database system is according to the number of users who can use the system concurrently-that is, at the same time. A DBMS is single-user if at most one user at a time can use the system, and it is multiuser if many users can use the system-and hence access the database-concurrently. Single-user DBMSs are mostly restricted to personal computer systems; most other DBMSs are multiuser. For example, an airline reservations system is used by hundreds of travel agents and reservation clerks concurrently.

Multiple users can access databases-and use computer systems-simultaneously because of the concept of multiprogramming, which allows the computer to execute multiple programs-or processes-at the same time. If only a single central processing unit (CPU) exists, it can actually execute at most one process at a time. However, multiprogramming operating systems execute some commands from one process, then suspend that

process and execute some commands from the next process, and so on. A process is resumed at the point where it was suspended whenever it gets its turn to use the CPU again. Hence, concurrent execution of processes is actually interleaved. **Figure 7.1** shows two processes A and B executing concurrently in an interleaved fashion. Interleaving keeps the CPU busy when a process requires an input or output (I/O) operation, such as reading a block from disk. The CPU is switched to execute another process rather than remaining idle during I/O time. Interleaving also prevents a long process from delaying other processes.

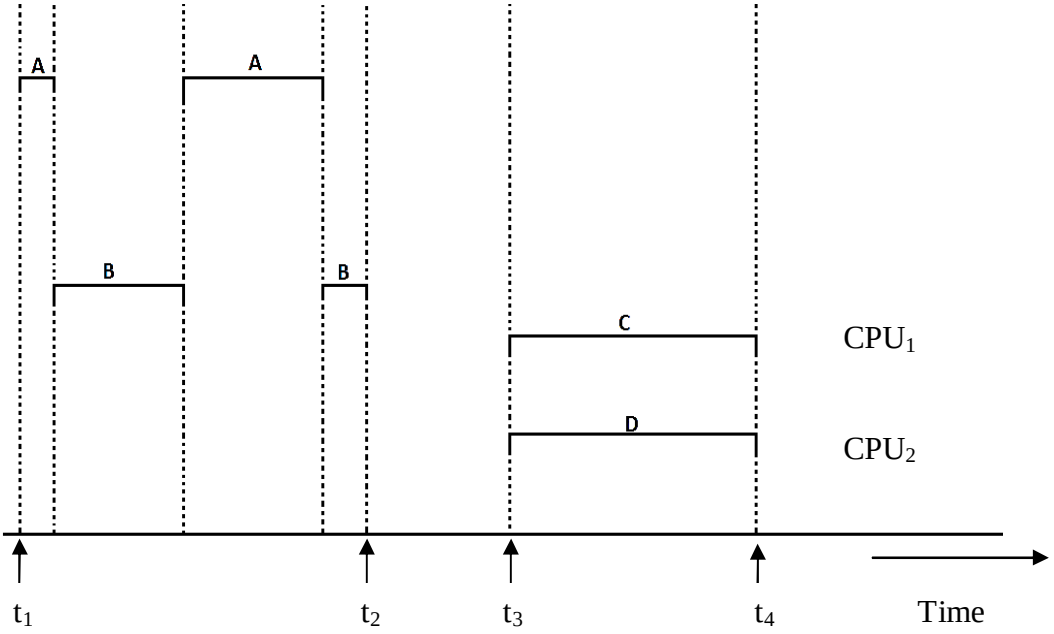


Fig 5.1: Interleaved processing versus parallel processing of concurrent transactions

If the computer system has multiple hardware processors (CPUs), parallel processing of multiple processes is possible, as illustrated by processes C and D in **Figure 7.1**

7.3.2 TRANSACTIONS, R/W OPERATIONS AND DBMS BUFFERS

A transaction is an executing program that forms a logical unit of database processing. A transaction includes one or more database access operations-these can include insertion, deletion, modification, or retrieval operations. The database operations that form a transaction can either be embedded within an application program or they can be specified interactively via a high-level query language such as SQL.

One way of specifying the transaction boundaries is by specifying explicit begin transaction and end transaction statements in an application program; in this case, all database

access operations between the two are considered as forming one transaction. A single application program may contain more than one transaction if it contains several transaction boundaries. If the database operations in a transaction do not update the database but only retrieve data, the transaction is called a read-only transaction.

The basic database access operations that a transaction can include are as follows:

- read_item(X): Reads a database item named X into a program variable.
- write_item(X): Writes the value of program variable X into the database item named X.

Executing a read_item(X) command includes the following steps:

1. Find the address of the disk block that contains item X.
2. Copy that disk block into a buffer in main memory (if that disk block is not already in some main memory buffer).
3. Copy item X from the buffer to the program variable named X.

Executing a write_item(X) command includes the following steps:

1. Find the address of the disk block that contains item X.
2. Copy that disk block into a buffer in main memory (if that disk block is not already in some main memory buffer).
3. Copy item X from the program variable named X into its correct location in the buffer.
4. Store the updated block from the buffer back to disk (either immediately or at some later point in time).

The DBMS will generally maintain a number of buffers in main memory that hold database disk blocks containing the database items being processed. When these buffers are all occupied, and additional database blocks must be copied into memory, some buffer replacement policy is used to choose which of the current buffers is to be replaced. If the chosen buffer has been modified, it must be written back to disk before it is reused.

A transaction includes **read_item** and **write_item** operations to access and update the database. **Figure 7.2** shows examples of two very simple transactions. The **read-set** of a transaction is the set of all items that the transaction reads, and the **write-set** is the set of all items that the transaction writes. For example, the read-set of T1 in **Figure 7.2** is {X, Y} and its write-set is also {X, Y}.

7.3.3 NEED OF CONCURRENCY CONTROL

Several problems can occur when concurrent transactions execute in an uncontrolled manner. **Figure 7.2(a)** shows a transaction T_1 that transfers N reservations from one flight whose number of reserved seats is stored in the database item named X to another flight whose number of reserved seats is stored in the database item named Y. **Figure 7.2(b)** shows a simpler transaction T_2 that just reserves M seats on the first flight (X) referenced in transaction T_1.

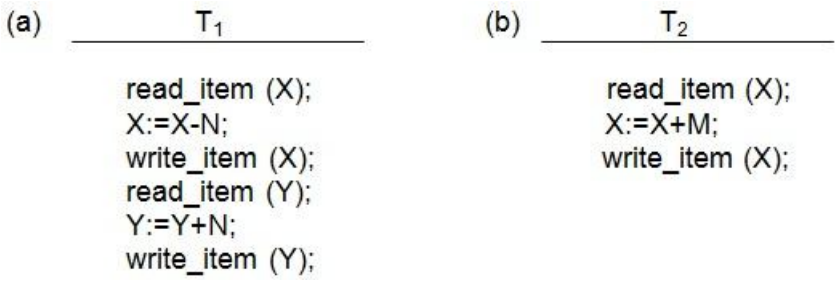
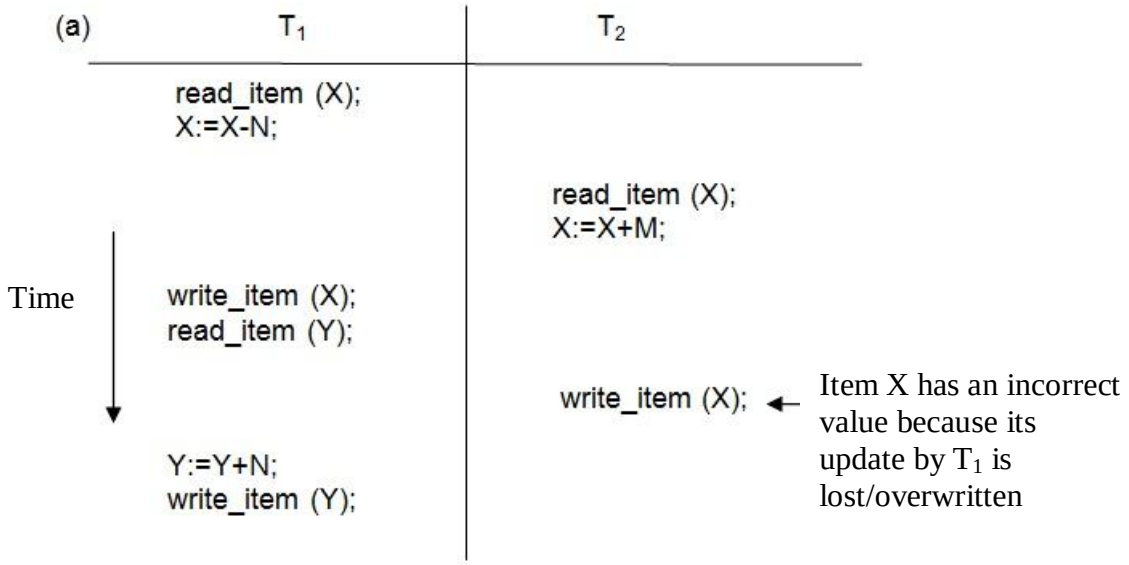


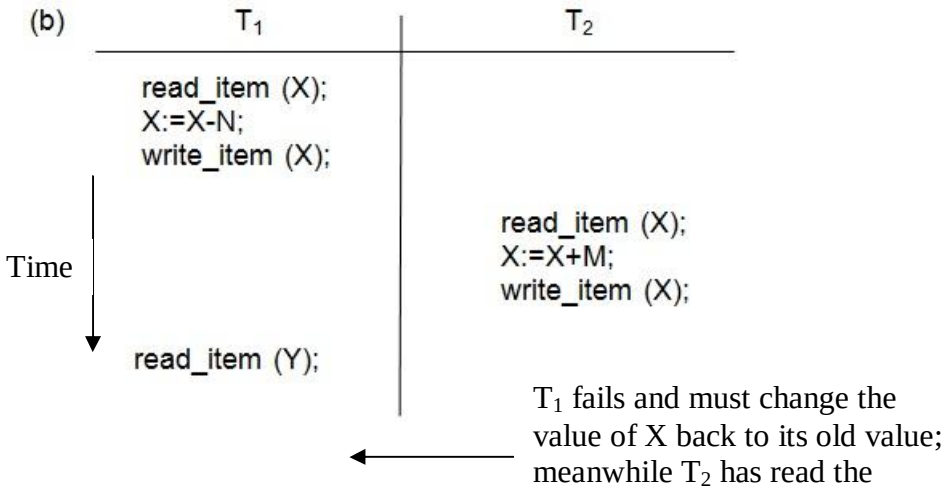
Fig 7.2: Two sample transactions. (a) Transaction T_1 (b) Transaction T_2

The types of problems that may be encountered with these two transactions if they are run concurrently are as follows.

The Lost Update Problem –

This problem occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of some database items incorrect. Suppose that transactions T_1 and T_2 are submitted at approximately the same time, and suppose that their operations are interleaved as shown in **Figure 7.3 (a)**; then the final value of item X is incorrect, because T_2 reads the value of X before T_1 changes it in the database, and hence the updated value resulting from T_1 is lost.





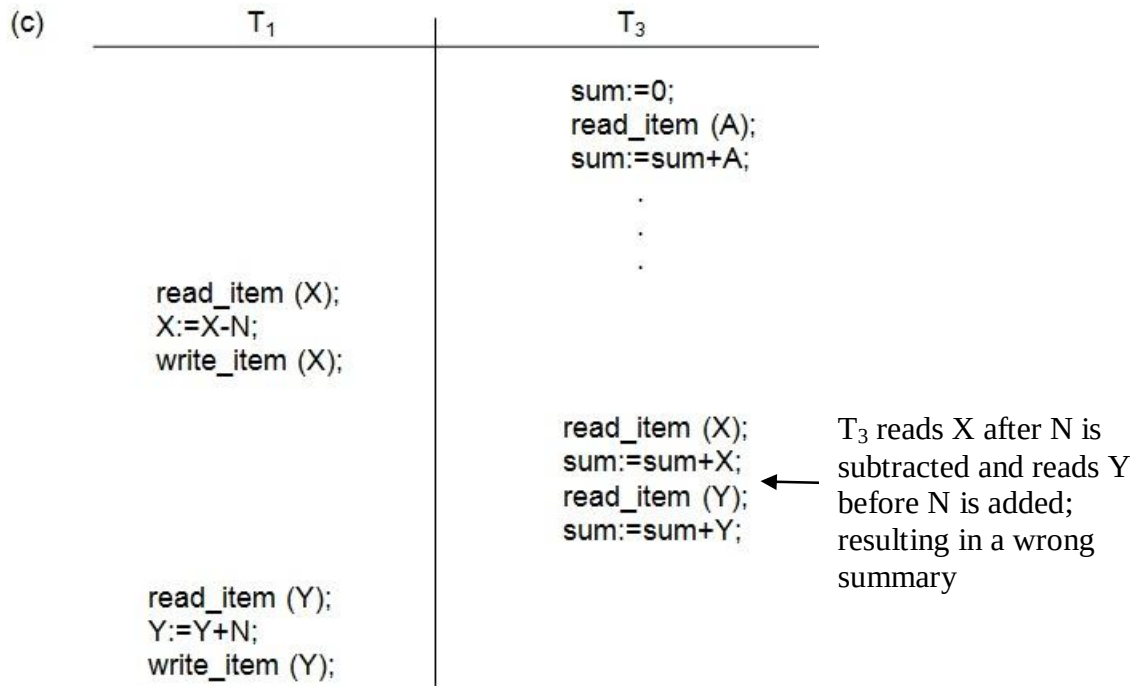


Fig 7.3: (a) The lost update problem (b) The temporary update problem (c) The incorrect summary problem

The Temporary Update (or Dirty Read) Problem –

This problem occurs when one transaction updates a database item and then the transaction fails for some reason. The updated item is accessed by another transaction before it is changed back to its original value. **Figure 7.3 (b)** shows an example where T_1 updates item X and then fails before completion, so the system must change X back to its original value. Before it can do so, however, transaction T_2 reads the "**temporary**" value of X, which will not be recorded permanently in the database because of the failure of T_1. The value of item X that is read by T_2 is called dirty data, because it has been created by a transaction that has not completed and committed yet; hence, this problem is also known as the dirty read problem.

The Incorrect Summary Problem –

If one transaction is calculating an aggregate summary function on a number of records while other transactions are updating some of these records, the aggregate function may calculate some values before they are updated and others after they are updated. For example, suppose that a transaction T_3 is calculating the total number of reservations on all the flights; meanwhile, transaction T_1 is executing. If the interleaving of operations shown in **Figure 7.3 (c)** occurs, the result of T_3 will be off by an amount N because T_3 reads the value of X after N seats have been subtracted from it but reads the value of Y before those N seats have been added to it.

Another problem that may occur is called unrepeatable read, where a transaction T reads an item twice and the item is changed by another transaction T' between the two reads. Hence, T receives different values for its two reads of the same item.

7.3.4 NEED OF RECOVERY

Whenever a transaction is submitted to a DBMS for execution, the system is responsible for making sure that either (1) all the operations in the transaction are completed successfully and their effect is recorded permanently in the database, or (2) the transaction has no effect whatsoever on the database or on any other transactions. The DBMS must not permit some operations of a transaction T to be applied to the database while other operations of T are not. This may happen if a transaction fails after executing some of its operations but before executing all of them.

Types of Failures –

Failures are generally classified as Transaction, system, and media failures. There are several possible reasons for a transaction to fail in the middle of execution:

- **A computer failure (system crash)** – A hardware, software, or network error occurs in the computer system during transaction execution. Hardware crashes are usually media failures-for example, main memory failure.

- **A transaction or system error –** Some operation in the transaction may cause it to fail, such as integer overflow or division by zero. Transaction failure may also occur because of erroneous parameter values or because of a logical programming error. In addition, the user may interrupt the transaction during its execution.

- **Local errors or exception conditions detected by the transaction –** During transaction execution, certain conditions may occur that necessitate cancellation of the transaction. For example, data for the transaction may not be found. An exception condition, such as insufficient account balance in a banking database, may cause a transaction, such as a fund withdrawal, to be canceled. This exception should be programmed in the transaction itself, and hence would not be considered a failure.

- **Concurrency control enforcement –** The concurrency control method may decide to abort the transaction, to be restarted later, because it violates serializability or because several transactions are in a state of deadlock.

- **Disk failure –** Some disk blocks may lose their data because of a read or write malfunction or because of a disk read/write head crash. This may happen during a read or a write operation of the transaction.

- **Physical problems and catastrophes –** This refers to an endless list of problems that includes power or air-conditioning failure, fire, theft, sabotage, overwriting disks or tapes by mistake, and mounting of a wrong tape by the operator.

CHECK YOUR PROGRESS

1. Fill in the blanks:

(a) Transaction Processing Systems are the systems with large _____ and hundreds of concurrent users executing database _____.

(b) Multiple users can access databases and use computer systems simultaneously because of the concept of _____.

(c) _____ reads a database item named X into a program variable.

(d) _____ writes the value of program variable X into the database item named X.

(e) A _____ includes read_item and write_item operations to _____ and _____ the database.

(f) The _____ of a transaction is the set of all items that the transaction reads, and the _____ is the set of all items that the transaction writes.

(g) The _____ problem occurs when one transaction _____ a database item and then the transaction fails for some reason

(h) During _____ a transaction T reads an item twice and the item is changed by another transaction T' between the two reads

7.4 TRANSACTION AND SYSTEM CONCEPTS

While discussing about Transaction Processing Systems there are some other relevant concepts that form the basis of effective and efficient transaction processing. Such issues include some discussion on the relevant operations like the different states that a transaction goes through while it is being executed during transaction processing. The system log, which keeps information needed for recovery, is the key player in case the need of a system recovery arises.

7.4.1 Transaction States and Additional Operations

A transaction is an atomic unit of work that is either completed in its entirety or not done at all. For recovery purposes, the system needs to keep track of when the transaction starts, terminates, and commits or aborts. Hence, the recovery manager keeps track of the following operations:

- **BEGIN_TRANSACTION**: This marks the beginning of transaction execution.
- **READ OR WRITE**: These specify read or write operations on the database items that are executed as part of a transaction.
- **END_TRANSACTION**: This specifies that READ and WRITE transaction operations have ended and marks the end of transaction execution. However, at this point it may be necessary to check whether the changes introduced by the transaction can be permanently applied to the database (committed) or whether the transaction has to be aborted because it violates serializability or for some other reason.
- **COMMIT_TRANSACTION**: This signals a successful end of the transaction so that any changes (updates) executed by the transaction can be safely committed to the database and will not be undone.
- **ROLLBACK (OR ABORT)**: This signals that the transaction has ended unsuccessfully, so that any changes or effects that the transaction may have applied to the database must be undone.

Figure 7.4 shows a state transition diagram that describes how a transaction moves through its execution states. A transaction goes into an **active state** immediately after it starts execution, where it can issue **READ** and **WRITE** operations. When the transaction ends, it moves to the **partially committed state**. At this point, some recovery protocols need to ensure that a system failure will not result in an inability to record the changes of the transaction permanently. Once this check is successful, the transaction is said to have reached its commit point and enters the committed state.

Once a transaction is committed, it has concluded its execution successfully and all its changes must be recorded permanently in the database.

However, a transaction can go to the **failed state** if one of the checks fails or if the transaction is aborted during its active state. The transaction may then have to be rolled back to undo the effect of its **WRITE** operations on the database. The **terminated state** corresponds to the transaction leaving the system.

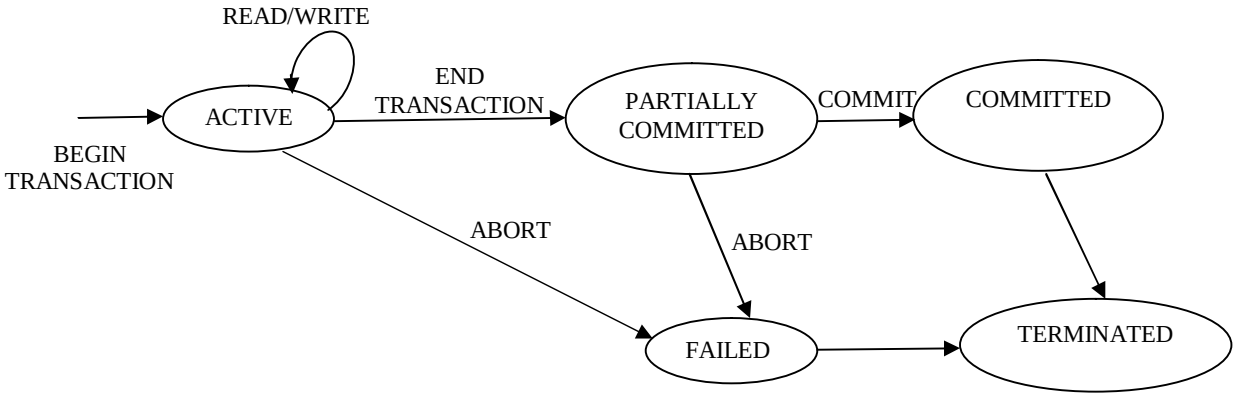


Fig 7.4: State transition diagram of states for transaction execution

7.4.2 THE SYSTEM LOG

The system maintains a log to keep track of all transaction operations that affect the values of database items in order to be able to recover from failures that affect transactions. This information may be needed to permit recovery from failures. We now list the types of entries – called log records – that are written to the log and the action each performs. In these entries, T refers to a unique transaction-id that is generated automatically by the system and is used to identify each transaction:

- [start-transaction, T]: Indicates that transaction T has started execution.
- [write_item, T, X, old_value, new_value]: Indicates that transaction T has changed the value of database item X from old_value to new_value.
- [read_item, T, X]: Indicates that transaction T has read the value of database item X.

- [commit, T]: Indicates that transaction T has completed successfully, and affirms that its effect can be committed (recorded permanently) to the database.
- [abort, T]: Indicates that transaction T has been aborted.

7.4.3 COMMIT POINT OF A TRANSACTION

A transaction T reaches its **commit point** when all its operations that access the database have been executed successfully and the effect of all the transaction operations on the database have been recorded in the log. Beyond the commit point, the transaction is said to be **committed**, and its effect is assumed to be permanently recorded in the database. The transaction then writes a commit record [commit, T] into the log. If a system failure occurs, we search back in the log for all transactions T that have written a [start_transaction, T] record into the log but have not written their [commit, T] record yet; these transactions may have to be rolled back to undo their effect on the database during the recovery process. Transactions that have written their commit record in the log must also have recorded all their WRITE operations in the log, so their effect on the database can be redone from the log records.

It is important to note that the log file must be kept on disk. Updating a disk file involves copying the appropriate block of the file from disk to a buffer in main memory, updating the buffer in main memory, and copying the buffer to disk. It is also common to keep one or more blocks of the log file in main memory buffers until they are filled with log entries and then to write them back to disk only once, rather than writing to disk every time a log entry is added. This saves the overhead of multiple disk writes of the same log file block. At the time of a system crash, only the log entries that have been written back to disk are considered in the recovery process because the contents of main memory may be lost. Hence, before a transaction reaches its commit point, any portion of the log that has not been written to the disk yet must now be written to the disk. This process is called **force-writing** the log file before committing a transaction.

7.5 DESIRABLE PROPERTIES OF TRANSACTION

Transactions should possess several properties. These are often called the **ACID** properties, and they should be enforced by the concurrency control and recovery methods of the DBMS. The following are the ACID properties:

1. **Atomicity**: A transaction is an atomic unit of processing; it is either performed in its entirety or not performed at all.

2. **Consistency preservation**: A transaction is consistency preserving if its complete execution takes the database from one consistent state to another.
3. **Isolation**: A transaction should appear as though it is being executed in isolation from other transactions. That is, the execution of a transaction should not be interfered with by any other transactions executing concurrently.
4. **Durability or permanency**: The changes applied to the database by a committed transaction must persist in the database. These changes must not be lost because of any failure.

The atomicity property requires that we execute a transaction to completion. It is the responsibility of the transaction recovery subsystem of a DBMS to ensure atomicity. If transaction fails to complete for some reason, such as a system crash in the midst of transaction execution, the recovery technique must undo any effect of the transaction on the database.

The preservation of consistency is generally considered to be the responsibility of the programmers who write the database programs or of the DBMS module that enforces integrity constraints. A database state is a collection of all the stored data items in the database at a given point in time. A consistent state of the database satisfies the constraints specified in the schema as well as any other constraints that should hold on the database. A database program should be written in a way that guarantees that, if the database is in a consistent state before executing the transaction, it will be in a consistent state after the complete execution of the transaction, assuming that no interference with other transaction occurs.

Isolation is enforced by the concurrency control system of the DBMS. If every transaction does not make its updates visible to other transactions until it is committed, one form of isolation is enforced that solves the temporary update problem and eliminates cascading rollbacks. There have been attempts to define the level of isolation of a transaction. A transaction is said to have Level 0 isolation if it does not overwrite the dirty reads of higher level transaction. Level 1 isolation has no lost of updates; and Level 2 isolation has no lost updates and no dirty reads. Finally, Level 3 isolation has in addition to degree 2 properties, repeatable reads.

Finally, durability property is the responsibility of the recovery subsystem of the DBMS.

CHECK YOUR PROGRESS

2. Fill in the blanks:

(a) A transaction is an _____ unit of work that is either completed in its entirety or not done at all

(b) _____ marks the beginning of transaction execution.

(c) _____ specifies that READ and WRITE transaction operations have ended and marks the end of transaction execution.

(d) _____ signals a successful end of the transaction.

(e) _____ signals that the transaction has ended unsuccessfully.

(f) A transaction is _____ if its complete execution takes the database from one consistent state to another.

(g) A _____ is a collection of all the stored data items in the database at a given point in time.

(h) A transaction is said to have _____ isolation if it does not overwrite the dirty reads of higher level transaction.

7.6 CHARACTERIZING SCHEDULES BASED ON RECOVERABILITY

When transactions are executing concurrently in an interleaved fashion, then the order of execution of operations from the various transactions is known as a **schedule** (or history).

Schedules (Histories) of Transactions

A schedule (or history) S of n transactions T_1, T_2, ... , T_n is an ordering of the operations of the transactions subject to the constraint that, for each transaction T_i that participates in S, the operations of T_i in S must appear in the same order in which they occur in T_i. For the purpose of recovery and concurrency control, we are mainly interested in the read_item and write_item operations of the transactions, as well as the commit and abort operations. A shorthand notation for describing a schedule uses the symbols r, w, c, and a for the operations read_item, write_item, commit, and abort, respectively, and appends as subscript the transaction id (transaction number) to each operation in the schedule. In this notation, the database item X that is read or written follows the r and w operations in parentheses. For example, the schedule of **Fig 7.3(a)**, which we shall call S_a, can be written as follows in this notation:

$$\mathbf{S_a: r_1(X); r_2(X); w_1(X); r_1(Y); w_2(X); w_1(Y);}$$

Two operations in a schedule are said to **conflict** if they satisfy all three of the following conditions: (1) they belong to different transactions; (2) they access the same item X; and (3) at least one of the operations is a write_item(X).

A schedule S of n transactions T_1, T_2,.....T_n, is said to be a complete schedule if the following conditions hold:

1. The operations in S are exactly those operations in T_1, T_2,..... T_n, including a commit or abort operation as the last operation for each transaction in the schedule.
2. For any pair of operations from the same transaction T_i, their order of appearance in S is the same as their order of appearance in T;
3. For any two conflicting operations, one of the two must occur before the other in the schedule.

For some schedules it is easy to recover from transaction failures, whereas for other schedules the recovery process can be quite involved. Hence, it is important to characterize the types of schedules for which recovery is possible, as well as those for which recovery is relatively simple. First, we would like to ensure that, once a transaction T is committed, it should never be necessary to roll back T. The schedules that theoretically meet this criterion are called **recoverable schedules** and those that do not are called **non recoverable**, and hence should not be permitted. A schedule S is recoverable if no transaction T in S commits until all

transactions T' that have written an item that T reads have committed.

Recoverable schedules require a complex recovery process, but if sufficient information is kept (in the log), a recovery algorithm can be devised. In a recoverable schedule, no committed transaction ever needs to be rolled back. However, it is possible for a phenomenon known as **cascading rollback** (or cascading abort) to occur, where an uncommitted transaction has to be rolled back because it read an item from a transaction that failed. Because cascading rollback can be quite time-consuming (since numerous transactions can be rolled back), it is important to characterize the schedules where this phenomenon is guaranteed not to occur. A schedule is said to be **cascadeless**, or to avoid cascading rollback, if every transaction in the schedule reads only items that were written by committed transactions.

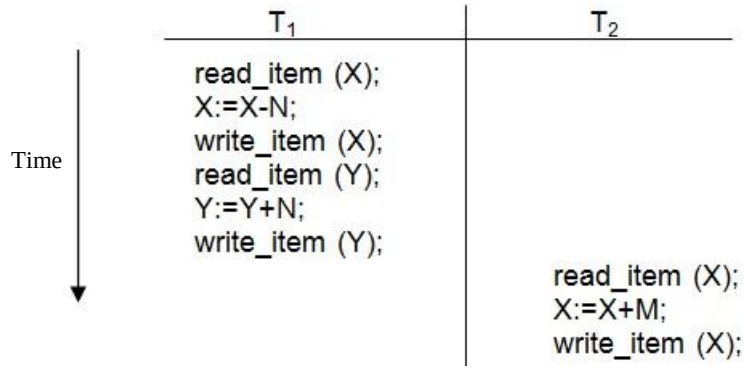
Finally, there is a third, more restrictive type of schedule, called a **strict schedule**, in which transactions can neither read nor write an item X until the last transaction that wrote X has committed (or aborted). Strict schedules simplify the recovery process. In a strict schedule, the process of undoing a write_item(X) operation of an aborted transaction is simply to restore the before image (old_value or BFIM) of data item X. This simple procedure always works correctly for strict schedules, but it may not work for recoverable or cascadeless schedules.

7.7 Characterizing Schedules Based on Serializability

In the case of characterizing schedules based on Serializability we characterize the type of schedules that are considered correct when concurrent transactions are being executed. Let us suppose that two users – two airline reservation clerks – submit to the DBMS transactions T_1 and T_2 of **Figure 7.5** at the same time. If no interleaving of operations is permitted, there are only two possible outcomes:

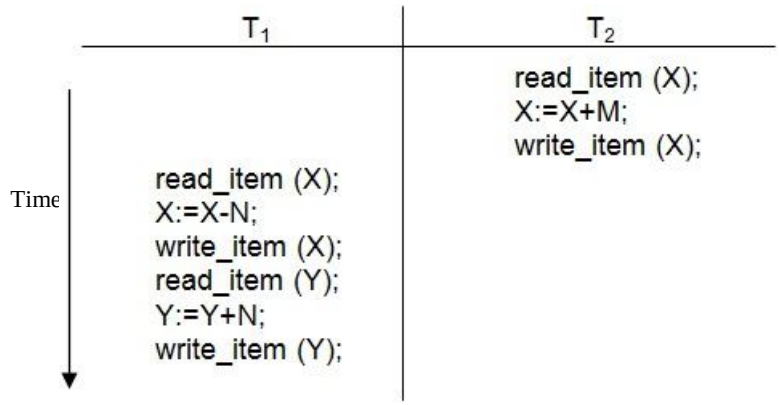
1. Execute all the operations of transaction T_1 followed by all the operations of transaction T_2.
2. Execute all the operations of transaction T_2 followed by all the operations of transaction T_1.

(a)



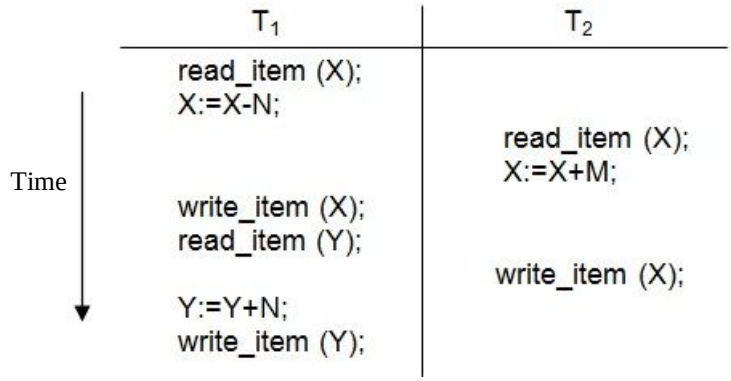
Schedule A

(b)



Schedule B

(c)



Schedule C

(d)

T_1	T_2
read_item (X); X:=X-N; write_item (X);	
	read_item (X); X:=X+M; write_item (X);
read_item (Y); Y:=Y+N; write_item (Y);	

Schedule D

Fig 7.5: Examples of serial and non-serial schedules involving transactions T_1 and T_2. (a) Serial schedule A: T_1 followed byT_2 (b) Serial schedule B: T_2 followed by T_1 (c) Two non-serial schedules C and D with interleaving of operations

Serial, non-serial, and conflict-serializable schedules –

Schedules A and B in **Fig 7.5 (a)** and **(b)** are called serial because the operations of each transaction are executed consecutively, without any interleaved operations from the other transaction. In a serial schedule, entire transactions are performed in serial order: T_1 and then T_2 in **Fig 7.5 (a)**, and T_2 and then T_1 in **Fig 7.5 (b)**. Schedules C and D in **Fig 7.5 (c)** are called non-serial because each sequence interleaves operations from the two transactions.

Formally, a schedule S is **serial** if, for every transaction T participating in the schedule, all the operations of T are executed consecutively in the schedule; otherwise, the schedule is called non-serial. Hence in a serial schedule, only one transaction at a time is active – the commit/abort of the active transaction initiates execution of the next transaction. No interleaving occurs in a serial schedule. One reasonable assumption we can make, if we consider the transactions to be independent, is that every serial schedule is considered correct. We can assume this because every transaction is assumed to be correct if executed on its own. Hence, it does not matter which transaction is executed first. As long as every transaction is executed from beginning to end without any interference from the operations of other transactions, we get a correct end result on the database. The problem with serial schedules is that they limit concurrency or interleaving of operations. In a serial schedule, if a transaction waits for an I/O operation to complete, we cannot switch the CPU processor to another transaction, thus wasting valuable CPU processing time. In addition, if some transaction T is quite long, the other transactions must wait for T to complete all its operations before commencing. Hence, serial schedules are generally considered unacceptable in practice.

A schedule S of n transactions is **serializable** if it is equivalent to some serial schedule of the same n-transactions. There are n! possible serial schedules of n transactions and many more possible non-serial schedules. We can form two disjoint groups of the non-serial schedules: those that are equivalent to one or more of the serial schedules, and hence are serializable; and those that are not equivalent to any serial schedule and hence are not serializable.

Two schedules are called **result equivalent** if they produce the same final state of the database. However, two different schedules may accidently produce the same final state. For example, in **Figure 7.6** schedules S_1 and S_2 will produce the same final database state if they execute on a database with an initial value of X=100; but for other initial values of X, the schedules are not result equivalent. For two schedules to be equivalent, the operations applied to each data item affected by the schedules should be applied to that item in both schedules in the same order. Two definitions of equivalence of schedules are generally used: conflict equivalence and view equivalence.

Two schedules are said to be **conflict equivalent** if the order of any two conflicting operations is the same in both schedules. Two operations in a schedule are said to conflict if they belong to different transactions, access the same database item and at least one of the two operations is a write_item operation. If two conflicting operations are applied in different orders in two schedules, the effect can be different on the database or on other transactions in the schedule, and hence the schedules are not conflict equivalent.

Using the notion of conflict equivalence, we define a schedule S to be **conflict serializable** if it is (conflict) equivalent to some serial schedule S. in such a case, we can reorder the non-conflicting operation in S until we form the equivalent serial schedule S.

S_1	S_2
read_item (X);	read_item (X);
X:=X+10;	X:=X*1.1;
write_item (X);	write_item (X);

Fig 7.6: Two schedules that are result equivalent for the initial value of X=100 but are not result equivalent in general

Two schedules S and S' are said to be view equivalent if the following three conditions hold:

1. The same set of transactions participates in S and S', and S and S' include the same operations of those transactions.
2. For any operation $r_1(X)$ of T_1 in S, if the value of X read by the operation has been written by an operation $w_1(X)$ of T_1, the

same condition must hold for the value of X read by operation $r_1(X)$ of T_1 in S'.

3. If the operation $w_k(Y)$ of T_k is the last operation to write item Y in S, then $w_k(Y)$ of T_k must also be the last operation to write item Y in S'.

The idea behind view equivalence is that, as long as each read operation of a transaction reads the result of the same write operation in both schedules, the write operations of each transaction must produce the same result. The read operations are hence said to see the same view in both schedules. Condition 3 ensures that the final write operation on each data item is the same in both schedules, so the database state should be the same at the end of both schedules. A schedule S is said to be **view serializable** if it is view equivalent to a serial schedule.

CHECK YOUR PROGRESS

3. Fill in the blanks:

(a) In _____ an uncommitted transaction has to be rolled back because it read an item from a transaction that failed.

(b) A schedule is said to be _____ if every transaction in the schedule reads only items that were written by committed transactions.

(c) Two schedules are called _____ if they produce the same final state of the database.

(d) Two schedules are said to be _____ if the order of any two conflicting operations is the same in both schedules.

(e) A schedule is _____ if it is (conflict) equivalent to some serial schedule.

(f) A schedule is said to be _____ if it is view equivalent to a serial schedule.

7.8 LET US SUM UP

- The concept of transaction provides a mechanism for describing logical units of database processing
- A transaction is an executing program that forms a logical unit of database processing
- The lost update problem occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of some database items incorrect
- Failures are generally classified as Transaction, system, and media failures
- The concurrency control method may decide to abort the transaction, to be restarted later
- For recovery purposes, the system needs to keep track of when the transaction starts, terminates, and commits or aborts
- The system maintains a log to keep track of all transaction operations that affect the values of database items in order to be able to recover from failures that affect transactions
- A transaction T reaches its commit point when all its operations that access the database have been executed successfully and the effect of all the transaction operations on the database have been recorded in the log
- A schedule S of n transactions is serializable if it is equivalent to some serial schedule of the same n-transactions

7.9 ANSWERS TO CHECK YOUR PROGRESS

1.
(a) databases, transactions.
(b) multiprogramming.
(c) read_item(X).
(d) write_item(X).
(e) transaction, access, update.
(f) read-set, write-set.
(g) temporary update, updates.
(h) unrepeatable reads.

2.

(a) atomic.
(b) BEGIN_TRANSACTION.
(c) END_TRANSACTION
(d) COMMIT_TRANSACTION.
(e) ROLLBACK (OR ABORT)
(f) consistency preserving.
(g) database state
(h) Level 0

3.

(a) cascading rollback
(b) cascadeless
(c) result equivalent
(d) conflict equivalent
(e) conflict serializable
(f) view serializable

7.10 FURTHER READINGS

- Prof. Sushant S. Sundikar: Introduction to Database Management System
- Elmasri, Navathe, Somayajulu, Gupta: Fundamentals of Database Systems

7.11 MODEL QUESTIONS

1. What is a database transaction? Describe the different operations that are performed during a transaction

2. What is the need of concurrency control? Discuss the problems that may occur when two transactions run concurrently

3. Discuss the different types of failures? What is meant by catastrophic failure?

4. Draw a state diagram, and discuss the typical states that a transaction goes through during execution.

5. What is a System log used for? What are the typical kinds of records in a system log?

6. Discuss the ACID properties of a database transaction

7. What is a schedule (history)? Define the concepts of recoverable, cascadeless and strict schedules.

8. Discuss how Serializability is used to enforce concurrency control in a database system.

9. Discuss serial, non-serial and conflict serializable schedules

*****

UNIT-8 CONCURRENCY CONTROL AND RECOVERY

UNIT STRUCTURE

8.1 LEARNING OBJECTIVES

After going through this unit, you will able to

- learn the meaning of concurrency
- learn the Concurrency Control Techniques
- describe the idea of Controlling Concurrency
- learn the types of timestamp ordering
- describe the concept of Granularity
- describe the database recovery and different recovery techniques

8.2 INTRODUCTION

In the previous chapter we learnt about transaction and serializability of transaction. This chapter discusses about the different concurrency control techniques and locking how locking helps in transaction isolation. Most of the techniques discussed ensures the serializabality by using different protocols. The chapter discusses about the techniques like two phase locking, timestamp ordering, optimistic protocols etc. The effect of granularity, in currency control is also discussed. The completes with the discussion about database recovery

8.3 TWO PHASE LOCKING TECHNIQUES FOR CONCURRENCY CONTROL

In case of shared data, locking is a mechanism commonly used to solve the problem of synchronized access. Conceptually, it is a variable associated with a data item. It is important here to note that each data item has a lock associated with it. Before a transaction T1, access a data item, the scheduler first examines the associated lock with the data item. If no transaction holds the lock on the data item, then the scheduler obtains the lock on behalf of T1. If another transaction T2, hold the lock, then T1, has to wait until T2 gives up the lock. That is, the scheduler will not give T1, the lock until T2, releases it. The scheduler thereby ensures that only one transaction can hold the lock at a time, so only one transaction can access the data item at a time. Locking can be used by a scheduler to ensure serializability of transaction. To present such a locking protocol, we need some notation.

Transactions access data items either for reading or for writing them. We therefore associate two types of locks with data items: read locks and write locks. Let us assume that rl[x] denote a read lock on data item x and wl[x] denote a write lock on x. rl[x] (or wl[x]) to indicate that transaction T, has obtained a read (or write) lock on x. We use $rl_i[x]$ (or $wl_i[x]$) to denote the operation by which T_i releases its read (or

write) lock on x. In this case, we say T_i unlocks x. It is the job of a two Phase Locking (2PL) scheduler to manage the locks by controlling when transactions obtain and release their locks. A transaction is said to follow the Two Phase Locking protocol, if all locking operations (either read or write lock) first perform the unlock operation in any transaction. The rules according to which a basic 2PL scheduler manages and uses its locks:

1. When it receives an operation $p_i[x]$ from the Transaction Manager, the scheduler tests if $pl_i[x]$ conflicts with some $ql_i[x]$ that is already set. If so, it delays $p_i[x]$, forcing T_i to wait until it can set the lock it needs. If not, then the scheduler sets $pl_i[x]$, and then sends $p_i[x]$ to the Database Manager.

2. Once the scheduler has set a lock for T_i, say $pl_i[x]$, it may not release that lock at least until after the Database Manager acknowledges that it has processed the lock's corresponding operation, pi[x].

3. Once the scheduler has released a lock for a transaction, it may not subsequently obtain any more locks for that transaction on any data item.

from the above rule it has been noticed that, according to rule (1), prevents two transactions from concurrently accessing a data item in conflicting modes. Thus, conflicting operations are scheduled in the same order in which the corresponding locks are obtained.

The rule (2) supports rule (1) by ensuring that the Database Manager processes operations on a data item in the order that the scheduler submits them. For example, suppose T_i obtains rl[x], which it releases before the Database Manager has confirmed that $r_i[x]$ has been processed. Then it is possible for T_j to obtain a conflicting lock on x, $wl_j[x]$, and send $w_j[x]$ to the Database Manager. Although the scheduler has sent the Database Manager $r_i[x]$ before $w_j[x]$. Without rule (2) there is no guarantee that the Database Manager will receive and process the operations in that order.

Rule (3), called the two phase rule, is the base for the name two

phase locking. Each transaction may be divided into two phases: a growing phase or expanding phase (first phase) during which it obtains locks, and a shrinking phase (second phase) during which it releases locks.

8.4 TYPES OF TWO PHASE LOCKING

There are basically two types of Two Phase Locking. They are :

Conservative 2PL

It is possible to construct a Two Phase Locking scheduler that never aborts transactions. This technique is known as **Conservative Two Phase Locking** or **Static 2X**. As, Two Phase Locking most of the time causes abortions because of deadlocks; Conservative Two Phase Locking avoids deadlocks by requiring each transaction to obtain all of its locks before any of its operations are submitted to the Database Manager. This is done by having each transaction to declare its readset and writeset in advanced. Specifically, each transaction T_i first tells the scheduler all the data items it will want to Read or Write, for example as part of its Start operation. The scheduler tries to set all of the locks needed by T_i. It can do this by ensuring that none of these locks conflicts with a lock held by any other transaction. If the scheduler succeeds in setting all of T_i's locks, then it submits T_i's operations to the Database Manager as soon as it receives them. After the Database Manager acknowledges the processing of T_i's last database operation, the scheduler may release all of T_i's locks. If, on the other hand, any of the locks requested in T_i's Start conflicts with locks presently held by other transactions, then the scheduler does not grant any of T_i's locks. Instead, it inserts T_i along with its lock requests into a waiting queue. Every time the scheduler releases the locks of a completed transaction, it examines the waiting queue to see if it can grant all of the lock requests of any waiting transactions. If so, it then sets all of the locks for each such transaction and continues processing as just described. In Conservative Two Phase Locking, if a transaction T_i is waiting for a lock held by T_j, then T_i is holding no locks. Therefore,

no other transaction T_k can be waiting for T_i. Since deadlock is the only reason that a Two Phase Locking scheduler ever rejects an operation and thereby causes the corresponding transaction to abort, Conservative Two Phase Locking never aborts a transaction. By delaying operations sooner than it has to, namely, when the transaction begins executing, the scheduler avoids abortions that might otherwise be needed for concurrency contro1 reasons.

Strict Two Phase Locking

Almost all implementations of Two Phase Locking use a variant called **Strict Two Phase Locking**. This differs from the Basic 2PL scheduler described in the previous section that it requires the scheduler to release all of a transaction's locks together, when the transaction terminates. More specifically, T_i's locks are released after the Database Manager acknowledges the processing of c_i or a_j, depending on whether T_i commits or aborts respectively. The reasons for adopting this policy is Firstly, consider when a Two Phase Locking scheduler can release some $ol_i[x]$. To do so the scheduler must know that: (1) T_i has set all of the locks it will ever need, and (2) T_i will not subsequently issue operations that refer to X. One point in time at which the scheduler can be sure of (1) and (2) is when T_i terminates, that is, when the scheduler receives the c_i or a_i operation. In fact, in the absence of any information from the Transaction Manager aside from the operations submitted, this is the earliest time at which the scheduler can be assured that (1) and (2) hold. A second reason for the scheduler to keep a transaction's locks until it ends, and specifically until after the Database Manager processes the transaction's Commit or Abort, is to guarantee a strict execution.

Distributed Two Phase Locking

Two phase locking can also be realized in a distributed Database Management System. A distributed Database Management System consists of a collection of communicating nodes, each of which is a centralized Database Management System. Each data item is stored at exactly one node. The scheduler manages the data items stored at its site. This means that the scheduler is responsible for controlling access to the items. A transaction submits its operations to a

Transaction Manager. The Transaction Manager then delivers each Read(x) or Write(x) operation of that transaction to the scheduler that manages x. When a scheduler decides to process the Read(x) or Write(x), it sends the operation to its local Database Manager, which can access x and return its value (for a Read) or update it (for a Write). The Commit or Abort operation is sent to all nodes where the transaction accessed data items. The schedulers at all nodes, taken together, constitute a distributed scheduler. The task of the distributed scheduler is to process the operations submitted by the Transaction Managers in a globally serializable and recoverable manner. It is possible to build a distributed scheduler based on Two Phase Locking. Each scheduler maintains the locks for the data items stored at its node and manages them according to the Two Phase Locking rules. In Two Phase Locking, a Read(x) or Write(x) is processed when the appropriate lock on x can be obtained, which only depends on what other locks on x are presently owned. Therefore, each local Two Phase Locking scheduler has all the information it needs to decide when to process an operation, without communicating with the other sites. It is problematic to decide when to release a lock. To enforce the two phase rule, a scheduler cannot release a transaction T_i's lock until it knows that T, will not submit any more operations to it or any other scheduler. Otherwise, one scheduler might release T_i's lock and sometime later another scheduler might set a lock for T_i, thereby violating the two phase rule.

It would appear that enforcing the two phase rule requires communication among the schedulers at different nodes. However, if schedulers use Strict Two Phase Locking, then they can avoid such communication. Because, the Transaction Manager that manages transaction T, sends T_i's Commit to all nodes where T_i accessed data items. By the time the Transaction Manager decides to send T_i's Commit to all those sites, it must have received acknowledgments to all of T_j's operations. Therefore, T_i has surely obtained all the locks it will ever need. Thus, if a scheduler releases T_i's locks after it has processed T_j's Commit (as it must under Strict Two Phase Locking), it knows that no scheduler will subsequently set any locks for T_i. To prove that the distributed Two Phase Locking

scheduler is correct we simply ensure that any history H it could have produced satisfies the properties of Two Phase Locking histories.

CHECK YOUR PROGRESS

1. Fill up the gapes

a) A lock is a _____ associated with a data item.

b) Changing the state of lock from one type to another is known as _____.

c) In _____ phase of locking, new locks on item can be granted.

d) in _____ phase, existing locks are released.

e) In conservative two phase locking technique, it never _____ transaction.

8.5 CONCURRENCY CONTROL BASED ON TIMESTAMP ORDERING

In timestamp ordering, the Transaction Manager assigns a unique timestamp, $ts(T_i)$, to each transaction T_i. It generates timestamps using some of the efficient techniques, in the context of timestamp-based deadlock prevention. The Transaction Manager attaches a transaction's timestamp to each operation issued by the transaction. It will therefore be convenient to speak of the timestamp of an operation $o_i[x]$, which is simply the timestamp of the transaction that issued the operation. A **Timestamp Ordering (TO)** scheduler orders conflicting operations according to their timestamps. More precisely, it applies the following rule, called the **TO rule**.

TO Rule: If $p_i[x]$ and $q_i[x]$ are conflicting operations, then the Database Manager processes $p_i[x]$ before $q_i[x]$ if and only if $ts(T_i) < ts(T_i)$. By enforcing the TO rule, it can be ensured that every pair of conflicting operations is executed in timestamp order. Thus, a TO

execution has the same effect as a serial execution in which the transactions appear in timestamp order.

Variations of TO :

8.5.1 Basic Timestamp Ordering

The Basic TO is a simple and aggressive implementation of the TO rule. It accepts operations from the Transaction Manager and immediately outputs them to the Database Manager in first-come-first-served order. To ensure that this order does not violate the TO rule, the scheduler rejects operations that it receives too late. An operation $p_i[x]$ is too late if it arrives after the scheduler has already output some conflicting operation $q_i[x]$ with $ts(T_i) > ts(T_j)$. If $p_i[x]$ is too late, then it cannot be scheduled without violating the TO rule. Since the scheduler has already output $q_j[x]$, it can only solve the problem by rejecting $p_i[x]$.

If $e_i[x]$ is rejected, then T_i must abort. When T_i is submitted again, it must be assigned a timestamp sufficiently large so that its operations are less likely to be rejected during its second execution.
To determine if an operation has arrived too late, the Basic TO scheduler maintains for every data item x the maximum timestamps of Reads and Writes on x that it has sent to the Database Manager, denoted by max-r-scheduled[x] and max-w-scheduled[x] respectively. When the scheduler receives $e_i[x]$, it compares $ts(T_i)$ to max-q-scheduled[x] for all operation types q that conflict with p. If $ts(T_i) <$ max-q-scheduled[x], then the scheduler rejects $e_i[x]$, since it has already scheduled a conflicting operation with a larger timestamp. Otherwise, it schedules $e_i[x]$ and, if $ts(T_i) >$ max-p-scheduled[x], it updates max-p-scheduled[x] to $ts(T_i)$.

The scheduler must communicate to the Database Manager to guarantee that operations are processed by the Database Manager in the order that the scheduler sent them. Even if the scheduler decides that $e_i[x]$ can be scheduled, it must not send it to the Database Manager until every conflicting $q_j[x]$ that it previously sent has been acknowledged by the Database Manager.

For proper maintenance of the communication with the Database Manager, the Basic TO scheduler also maintains, for each data item x, the number of Reads and Writes that have been sent to the Database Manager, but not yet acknowledged. These are denoted as r-in-transit[x] and w-in-transit[x] respectively. For each data item x the scheduler also maintains a queue, queue[x], of operations that can be scheduled in so far as the TO rule is concerned, but are waiting for acknowledgments from the Database Manager to previously sent conflicting operations.

8.5.2 Strict Timestamp Ordering

An improved version of Basic TO is the Strict TO, which is both Strict ensuring easy recoverability and serializable. Although the TO rule enforces serializability, it does not necessarily ensure recoverability. If w-in-transit[x] denotes the number of w[x] operations that the scheduler has sent to the Database Manager but that the Database Manager has not yet acknowledged. Since two conflicting operations cannot be “in transit” at any time and Writes on the same data item conflict, w-in-transit[x] at any time is either 0 or 1. The Strict TO scheduler is similar to Basic TO in every respect, except that it does not set w-in-transit[x] to 0 when it receives the Database Mamnager’s acknowledgment of a w[x]. Instead it waits until it has received acknowledgment of a_i or c_i. It then sets w-in-transit[x] to zero for every x for which it had sent $w_i[x]$ to the Database Manager. This delays all $r_j[x]$ and $w_j[x]$ operations with ts(T_j) > ts(T_i) until T_i has committed or aborted. This means that the execution output by the scheduler to the Database Manager is strict.

CHECK YOUR PROGRESS

2. State True or False

a) Timestamp is an unique identifier used to identify a transaction when it is aborted.

b) Arranging of transaction based on timestamp value is known as Timestamp ordering.

8.6 MULTI-VERSION CONCURRENCY CONTROL (MVCC)

In a multiversion concurrency control algorithm, each Write on a data item x produces a new copy or version of x, thereby keeping the original copy intact. The Database Manager that manages x keeps a list of copies or versions of x, which is the history of values that the Database Manager has assigned to x. For each Read(x), the scheduler not only decides when to send the Read to the Database Manager, but it also tells the Database Manager which one of the copy or version of x to read.

The main benefit of multiple versions for controlling concurrency is to help the scheduler avoid rejecting operations that arrive too late. For example, the scheduler normally rejects a Read because the value it was supposed to read has already been overwritten. With multiversions, such old values are never overwritten and are therefore always available for Reads. The Database Manager makes the different versions explicitly available to the scheduler. The disadvantage of MVCC is the utilization of storage space in turn wasting valuable memory space and for better processing, it is also important to do the archiving thereby incurring processing overhead.

It is important to note here that when a transaction is aborted, the scheduler does not create a new version. The new versions are only created for the transaction which is either active or committed. Thus, when the scheduler decides to assign a particular version of x to Read(x), the value returned is not one produced by an aborted transaction.

8.6.1 The schedulers for MVCC based Timestamp Ordering

According the TO algorithm, each transaction has a unique timestamp, denoted ts(T_i). Each operation carries the timestamp of its corresponding transaction. Each version is labeled by the timestamp of the transaction that wrote it. A Multiversion TO (M VTO) scheduler processes operations on first-come-first-served basis. The scheduler processes $r_i[x]$ by first translating it into $r_i[x_k]$, where x_k is the version of x with the largest timestamp less than or equal to ts(T_i), and then sending $r_j[x_k]$ to the Database Manager. It processes $w_i[X]$ by considering two cases.

If it has already processed a Read $r_j[x_k]$ such that ts(T_k) < ts(T_i) < ts(T_j), then it rejects $w_i[x]$. Otherwise, it translates $w_i[x]$ into $w_j[x_j]$ and sends it to the Database Manager. Finally, to ensure recoverability, the scheduler must delay the processing of c_i until it has processed C_j for all transactions T_j that wrote versions read by T_i.

8.6.2 Multiversion Two-Phase Locking

According to this concept there are three locking modes read, write instead of two namely read and write. So, the state of LOCK(X) for an item X can be read-lock, write-lock or certify. According to the locking concept, write_lock is considered as an exclusive lock, i.e. when a transaction holds a write_lock on an item, that item cannot be locked by any other transaction.

The idea behind multiversion Two Phase Locking is to allow other transactions T′ to read an item x while a single transaction T holds a write lock on x. This is accomplished by having two versions for each item x. One version must always have been written by some committed transaction, while the second version is created when a transaction T acquires a write lock on the item. So, other transaction can continue accessing the item from the committed version of x. Any transaction T can write the value of x′ when required, without affecting the value of the committed version of x. Once T is ready to commit, it must obtain a certify lock on all items that it currently holds write locks on before it can commit. To obtain a certify lock, a transaction may have to delay its commit, until all its write locked items are released by any reading transactions. Once certify lock is ob-

tained, the committed version of the data is updated with the new value and the certify lock is released.

Thus, in case of multiversion Two Phase Locking, read can proceed concurrently with a single write operation, which is not permitted in case of standard Two Phase Locking.

CHECK YOUR PROGRESS

3. Fill Up the gapes

a) In Multi Version Concurrency Control, old value of data items are kept, even after _____ of data item.
b) In Multi Version two phase locking, _____ types of locking is present.
c) In Multi Version Concurrency Control, a _____ version is created, whenever there is an update.

8.7 VALIDATION (OPTIMISTIC) CONCURRENCY CONTROL TECHNIQUES

In all the concurrency control techniques that we discussed so far, some kind of checking is done, before a database operation can be executed. These kind of checking adversely affect the performance of the database by slowing down the transaction execution process. For example in case of locking, a check is done to verify whether the item accessed by the transaction is locked or not. This gives extra overhead, thereby slowing down the transaction execution time.

In case of validation concurrency control technique, which is also known as Optimistic concurrency control technique, no checking is done while transaction is executing. . That is a transaction is started without locking the database item. For example, in case of update operation in a transaction, it is not directly applied to the database,

until the transaction reaches the end. During the execution phase of a transaction, all updates are applied to the local copy of the database. when transaction reaches the end of the execution, it goes through a phase known as **validation phase,** which checks whether any of the transaction update violate serializability. The information needed by the validation phase is kept in the system.

If serializability is not break or violated, the transaction is committed and the database is updated from the local copies, otherwise, the transaction is aborted and restarted later.

The different phases of validation check concurrency control protocol are:

1. **Read Phase :** A transaction read values of committed data items from the database. However, because transaction is committed, the updates or writes are applied only to the local copies of the data item kept in the transaction workspace.
2. **Validation Phase :** Checking is performed to ensure that update from a transaction will not violate the serializability.
3. **Write Phase :** Based on the status of the validation phase, if the validation phase is successful, the transaction updates are applied to the database and if it is unsuccessful, the updates are discarded and the transaction is restarted later.

In optimistic concurrency control technique, a transaction proceeds without any checking, thereby proceeds with a minimum overhead, until it reaches the validation phase. If the interference between the transaction is less, they are validated successfully, and result is updated to the database. On the other hand, when the interference among the transaction is high, it results in unsuccessful validation, and the results are discarded and restarted later. The optimistic technique assumes that, there is less interference among the transactions and hardly needs any checking during the transaction execution.

In optimistic algorithm, during the validation phase of the Transaction T_i, the algorithm checks that T_i does not interfere with any committed

transactions or with any transaction which is currently running in their validation phase. The validation phase of ant transaction T_i, checks that for any other transaction T_j that is either committed or it is in its validation phase holds one of the following conditions:

1. T_j completes its write phase, before T_i starts its read phase.
2. T_i starts its write phase only after T_j completes its write phase and read_set of T_i has no item in common with the write_set of T_j.
3. The read_set and write_set of T_i have no item is common with the write_set of T_j and T_i completes its read phase before T_i completes its read phase.

The above three conditions checked serially, and stars with the condition 1. If it is false, it checks the condition 2, and if it is also false, then only it checks the condition 3. If any one of the above three conditions held, it means there is no interference and T_i is validated successfully. On the other hand, it none of the above three conditions holds, the validation phase of T_i fails and is aborted and restarted later.

CHECK YOUR PROGRESS3

4. State True or False

a) Optimistic Concurrency Control is also known as validation 1technique.

b) In Optimistic Concurrency Control technique, a check is done while a transaction is executing.

c) In read phase, a transaction read values of committed data items from the database.

d) Validation phase ensures that serializability is not maintained.

e) In Optimistic Concurrency Control, transaction execution proceeds with a minimum overhead until the validation phase is reached.

8.8 GRANULARITY OF DATA ITEM

All the concurrency control technique categorize a database in the form of a set of data items. The different data items that may be considered are as follows :

a) It may as small as a field value of a database record.
b) It may be a database record.
c) A disk block.
d) A whole file.
e) A whole database.

The granularity of database is very important as far as the performance of concurrency and recovery concerned.

The size of the data item is called data item granularity. The granularity can be considered as - fine granularity, which refers to the smallest size of the data items and coarse granularity which refers to the largest size of the data items.

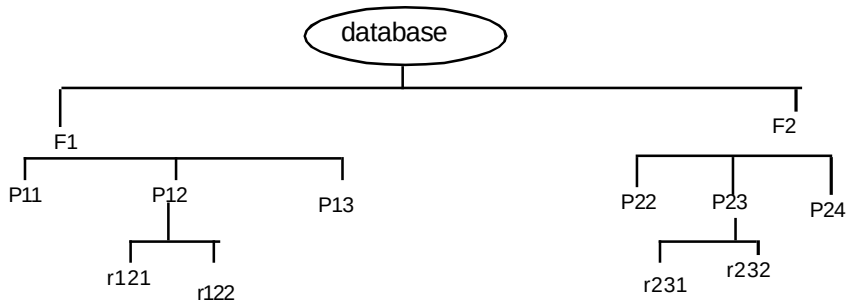
It is to be mentioned here that the degree of concurrency is dependent on the granularity of data item. When the size of the data item is large, it lowers the degree of concurrency, so the number of concurrent transaction is less. This is because, when the entire disk block is locked by a transaction, any other transaction trying to access items belongs to the same disk block is forced to wait. On the other hand if the database size is small, for example if it is a single record only, then many transactions can proceed parallelly, because it would be locking a different data item record.

On the other hand, when the size of the data item is small, more number of data item is present in the database. Since, every item is associated with a lock, there will be more number of lock and the lock manager need to manage a large number of locks which may be active at the same time. Since, more number of lock and unlock operations need to be performed, It increases the overall overhead

disadvantages of it.

on the database. At the same time, size of the lock table also increases.

The appropriate size of the data item is very difficult to decide and it depends on the type and number of transactions executing at one point of time. As the best size of the granularity depends on the transaction, so database system should support different level of granularity for a set of transactions. A hierarchy of granularity is depicted in the figure below:



Suppose a transaction T1 wants to update some of the records under the file F1. So, it will lock all the items down the hierarchy starting from F1. Now, suppose another transaction T2 want to read a record, which is also belongs to the file F1. The transaction manager of the Database System will check the compatibility of the requested lock, with the locks that are already held. While verifying the compatibility, if at any point of time, a conflict occurs, the request for the read(shared) lock is denied and the transaction T2 is denied and must wait.

But when T2 request for the read (shared) lock comes first, T2's request is granted. At this time, it is very difficult to check all the need for a lock conflict, and the time consumption is also very high. So, it becomes very much in efficient and in tern defeat the purpose of having multiple granularity locks.

To overcome the above problem and to implement multiple granularity level locking, another type of lock known as intention lock is implemented. The objective of this type of lock is to indicate along the path from the root to the desired node, what type of lock it would required. There are three types of intention locks namely,

1. Intention-Shared (IS)- which indicates that a shared lock will be requested on some descendent nodes.

2. Intention-Exclusive (IX)- which indicates that a shared lock will be requested on some descendent nodes.

3. Shared-Intention-Exclusive (SIX) – it indicates that the current node is locked in shared lock but an exclusive lock will be requested on some descendent nodes.

Apart from the different types of intention lock some appropriate locking protocol also need to be implemented for better performance.

CHECK YOUR PROGRESS

5. Fill up the gapes

a) Field value is the _____ granularity of data item.

b) Fine grain refers to the _____ data item size.

c) _____ grain refers to the large data item size.

d) Best data item size depends on the _____ type involved.

e) Intention lock is a type of lock applied to _____ granularity lock.

8.9 DATABASE RECOVERY CONCEPT

Recovery from transaction failure can be defined as the restoring of the database to the most recent consistent state just before the time of failure. To ensure this, the system must keep a record of all the changes that took place to the database due to the execution of various transactions. This information is maintained by the transaction log. In case of failure, with the help of this lock, the database can be recovered.

8.9.1 Category of recovery algorithm

There may be various possibilities due to which recovery becomes an important issue. In case of huge damage, which mainly causes due to the catastrophic failure, the database can be recovered by restoring the database kept as a backup or from the archive and reconstruct a more current state by reapplying or redoing the operations of committed transaction from the backup log up to the time of failure. On the other hand when the database is not physically damaged but has become inconsistent due to various inconsistencies by undoing some operations. There may be a need of redoing some of the operation in order to restore a consistent state of the database.

a) Differed Update / Immediate Update

To recover from the non-catastrophic failure, there are two main techniques applied. They are :

1. **Differed Update:** According to deferred update, the database is not physically updated to the disk until a database reaches its commit point. When a transaction reaches a commit point, then only the transaction update is recorded. Before reaching the commit point, all transaction updates are recorded to the local database. In case transaction fails before reaching the commit point, it is not required to make any changes to that database, so no undo is needed. There may be a necessity to redo some of the operations from the transaction logs. Because their effect may not have been recorded in the database. So, differed update is also sometime known as no-undo / redo algorithm.

2. **Immediate Update:** In this technique, the database may be update by some operations of the transaction before the transaction reaches its commit point. However these operations are also recorded in the log on the disk by force writing before they are applied to the database, thereby making the recovery still possible. In case a transaction fails before reaching the commit point, the effect of its

operation on the database must be undone i.e. the transaction must be rolled back. So, in case of immediate update, both undo and redo is needed. So, this algorithm is also known as UNDO/REDO algorithm. There may be another variation of the algorithm where all updates are recorded in the database before a transaction is committed requiring undo only, so it is known as undo/no-redo algorithm.

b) Caching of disk Blocks:

The caching of disk block typically is an operating system function. But as it is one of the important parameter as far as the efficiency of recovery procedure is concerned, it is handled by the DBMS by calling low level operating system routines. Typically a collection of in-memory buffer called the DBMS cache, is kept under the control of the DBMS for the purpose of holding these buffers. Associated with each buffer in the cache is a **dirty bit**, which can be included in the directory entry to indicate whether or not the buffer has been modified. When a page is first read from the database disk into a cache buffer, the cache directory is updated with the new disk address, the dirty bit is set to 0. As soon as the buffer is modified the dirty bit for the corresponding directory entry is set to 1. When the buffer content are replaced from the cache, the content must first be written back to the corresponding disk page only if its dirty bit is set to 1.

Another bit known as pin-unpin bit also used to indicate whether a page in the cache can be written back to the disk or not. Here, pinned (bit value 1) indicates page cannot be written back.

As far as flashing of modified buffer back to the disk is concerned, two main strategies are employed. They are:

i. In-place updating

ii. Shadowing

In case of in-place updating, it writes the buffer to the some original disk location, thus overwriting the old value of any changed data item on disk. Hence only a single copy of each database disk block is maintained.

In the case of shadowing on the other hand, it writes an updated buffer at a different disk location, so multiple versions of data items can be maintained. The old value of data item before updating is called the before image (BFIM) and the new value after update is called after image (AFIM). In shadowing both BFIM and AFIM are maintained so the necessity of maintaining log in eliminated.

c) Write-ahead logging

When in place updating is used, it is necessary to use a log for recovery. Here the recovery mechanism must ensure that the BFIM of the data item is recorded in the appropriate log entry and the log entry is flushed to disk before the BFIM is overwritten with the AFIM in the database on the disk. This process in known as write_ahead logging.

d) Steal/No-Steal approach

In No-Steal approach a cache page updated by a transaction cannot be written to disk before the transaction commits. The pin-unpin bit is used to realize the approach. The bit indicates if a page cannot be written back to the disk. On the other hand, in Steal approach, a cache page updated by a transaction can be

written to disk before the

transaction commits. Steal is use when the database cache manager needs a buffer frame for another transaction and the buffer manager replaces an existing page that had been updated but whose transaction has not committed.

e) Force/No-Force approach:

In Force approach all cache pages updated by a transaction are immediately written to disk when the transaction commits and otherwise it is called No-Force. A Typical Database systems use a Steal/No-Force strategy because of the following advantages:

– Steal approach avoids the need for a very large buffer space to store all updated pages in memory.

– No-Force approach provides the advantage of keeping an updated page of a committed transaction in the memory when another transaction needs to update it, thus eliminates the I/O cost of reading that page again from disk.

f) Check point :

Check point is another type of entry that is maintained in the log. The recovery manager of a DBMS must decide at what intervals to put a check point. The interval may be measured in terms of time say every after 'm' second or minute or in the number 't' of committed transactions since the last checkpoint. Here both 't' or 'm' are system parameters. All transactions committed before a checkpoint entry do not need to redo their write operations.

Checkpoint creating process consists of the following actions:

a. Suspend execution of transactions temporarily.
b. Force-write all main memory buffers that have been modified to disk.
c. Write a checkpoint record to the log, and force-write the log to disk.
d. Resume executing transactions.

The time needed to force-write buffers to disk may delay transactions execution and can be a big performance issue.

g) Transaction Rollback

There may be the necessity of roll back the transaction in some situation when a transaction fails after updating the database. In case of transaction rollback the write operations are undone, and it must be restored to their previous values (BFIS).

Another variation of rollback known as cascading rollback is also there where if transaction T1 is rolled back, any Transaction T2 that has read the value of some data item X written by T1 must also be rolled back and so on.

To improve the performance, Fuzzy checkpoint is used to reduce the delay by resuming the execution of the transactions after the checkpoint record is written to the log without having to wait for force-write step to be completed. The previous checkpoint is valid until the force-write step is completed. This in done by the system by maintaining a pointer to the valid checkpoint and changes it to the new checkpoint after the success of the force-write step.

CHECK YPUR PROGRESS

6. Fill up the gaps

a) The information about the changes applied to the database item by various transactions are kept by the _____.

b) Deferred update, updates the database only after the transaction is _____.

c) In case of immediate update, the database is updated _____ a transaction is committed.

d) AFIM stands for _____.

e) BFIM stands for _____.

7. State True or False

a) Dirty bit indicates whether the buffer is modified or not.

b) In shadowing, the updated buffer is written at one location.

c) In case of Force Approach, database updates are written to the disk immediately.

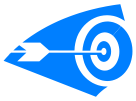
d) Creation of checkpoint is the responsibility of recovery manager.

e) Rollback refers to restoring of the database to their previous value.

8.10 LET US SUM UP

- Locking is a mechanism commonly used to solve the problem of synchronized access.

- Transactions access data items either for reading or for writing them. We therefore associate two types of locks with data items: read locks and write locks.
- There are basically two types of Two Phase Locking : conservative two phase locking and strict two phase locking.
- In timestamp ordering technique, the Transaction Manager assigns a unique timestamp, ts(T_i), to each transaction T_i .
- In Optimistic concurrency control technique, no checking is done while transaction is executing.



8.11 ANSWER TO CHECK YOUR PROGRESS

1. a) variable b) lock conversion c) Growing Phase d) shrinking phase e) aborts
2. a) F b) T
3. a) updation b) three c) new
4. a) T b) F c) T d) F e) T
5. a) smallest b) small c) Coarse d) transaction e) multilevel
6. a) system log b) committed c) before d) After Image e) Before Image
7. a) T b) F c) T d) T e) T

8.12 FURTHER READINGS

- Prof. Sushant S. Sundikar: Introduction to Database Management System
- Elmasri, Navathe, Somayajulu, Gupta: Fundamentals of Database Systems

8.13 MODEL QUESTIONS

a) What is a lock? Explain the function of lock in concurrency.

b) What is two phase locking protocol?

c) Explain how two phase locking protocol guaranty serializability.

d) What is a timestamp? explain.

e) What is certify lock? Explain the advantages and disadvantages of it.

f) What is intention lock? Describe its different types.

g) What do you mean by catastrophic failure?

h) Describe the function of a Dirty bit.

UNIT 9: SECURITY AND PRIVACY

UNIT STRUCTURE

9.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- describe database security
- definediscretionary access control.
- describe mandatory access control.
- describe role based access control
- describe encryption and public key infrastructures

9.2 INTRODUCTION

In previous chapter you have studied different concurrency technique such as two phase locking techniques; concurrency control based on timestamp ordering; multi-version concurrency control techniques; validation (optimistic) concurrency control techniquesetc. This unit discusses the techniques used for protecting the database against the

person who are not authorized to access either certain part of database or the whole database.

9.3 DATABASE SECURITY ISSUE

Database security is a very broad area that addresses many issues, including the following

- Legal and ethical issue regarding the right to access certain information. Sometime information may be deemed to private and cannot be accessed legally by unauthorized persons.
- Policy issue at the governmental, institutional, or corporate level as to what kind of information should not be made publicly available- for example credit rating and personal medical record.
- System related issues such as the system levels at which various security functions should be enforced. For example-where a security functions should be handled at the physical hardware level, the operating system level or the DBMS level.
- The need in some organizations to identify multiple security levels and to categorize the data users based on these classifications- for example, top secret, secret, confidential, and unclassified. The security policy of the organization with respect to permitting access to various classifications must be enforced.

Threats to Databases Threat to database result in the loss or degradation of some or all of the following security goals: integrity, availability, and confidentiality.

- **Loss of integrity**: Database integrity refers to the requirement that information be protected from improper modification. Modification of data includes creation, insertion, modification, changing the status of data, and deletion. Integrity is loss if unauthorized changes are made to data by either intentional or accidental acts. If the loss of the system or dataintegrity is not corrected, continued use of the

contaminated system or corrupted data could result in accuracy, fraud, or erroneous decision.

- **Loss of availability**: Database availability refers to making objects available to human user or a program to which they have a legitimate right.
- **Loss of confidentiality:** Database confidentiality refers to the protection of data from unauthorized disclosure. The impact of unauthorized disclosure of confidential information can range from violation of the Data Privacy Act to the jeopardization of national security.

Database Security and the DBA: The database administrator (DBA) is the central authority for managing a database system. The DBA's responsibilities include granting privileges to users who need to use the system and classifying users and data in accordance with the policy of the organization. The DBA has a DBAaccount in the DBMS, sometimes called a system or superuser account, which provides powerful capabilities. The DBA is responsible for the overall security of the database system.

Access Protection, User Accounts, and Database Audits

Whenever a person or group of persons need to access a database system, the individual or group must first apply for a user account. The DBA will then create a new **account number** and **password** for the user if there is a legitimate need to access the database. The user must **log in** to the DBMS by entering account number and password whenever database access is needed.

The database system must also keep track of all operations on the database that are applied by a certain user throughout each **login session**.

To keep a record of all updates applied to the database and of the particular user who applied each update, we can modify ***system log***, which includes an entry for each operation applied to the database that may be required for recovery from a transaction failure or system crash.

If any tampering with the database is suspected, a **database audit** is performed, which consists of reviewing the log to examine all accesses and operations applied to the database during a certain time period. A database log that is used mainly for security purposes is sometimes called an **audit trail**.

CHECK YOUR PROGRESS

1. Choose the correct answer

i) Modification of data includes

(a) Creation (b) Insertion

(c) Modification (d) All of the above

ii) DBA stands for

(a) Data Base Access. (b) Data Base Administrator.

(c) Data Bound Administrator. (d) Non of the above.

iii). A database log that is used mainly for security purposes is sometimes called

(a) System log. (b) Database Audit.

(c) Database trial. (d) Audit trial.

9.4 DISCRETIONARY ACCESS CONTROL BASED ON GRANTING AND REVOKING PRIVILEGES

The typical method of enforcing **discretionary access control** in a database system is based on the granting and revoking **privileges**.

There are two levels for assigning privileges

- **Theaccount level:** At this level, the DBA specifies the particular privileges that each account holds independently of the relations in the database.

- **The relation level:** At this level, the DBA can control the privilege to access each individual relation or view in the database

The privileges at the account level apply to the capabilities provided to the account itself and can include the CREATE SCHEMA or CREATE TABLE privilege, to create a schema or base relation; the CREATE VIEW privilege; the ALTER privilege, to apply schema changes such as adding or removing attributes from the relation; the DROP privilege, to delete relations or view; the MODIFY privilege, to insert, delete, or update tuples, and the SELECT privilege, to retrieve information from the database using a SELECT query. Notice that these account privileges apply to the account in general. If a certain account does not have the CREATE TABLE privilege, no relations can be created from that account.

The second level of privileges applies to the relational level, whether they are base relations or virtual relations.

To control the granting and revoking of relation privileges, each relation R in a database is assigned an owner account, which is typically the account that was used when the relation was created in the first place. The owner relation is given all the privileges on that relation.

In SQL the following types of privileges can be granted to each individual relation R:

- SELECT privilege on R: Gives the account retrieval privilege.
- MODIFY privileges on R: This gives the account the capability to modify the tuples of R.
- REFERENCE privilege on R: This gives the account the capability to reference relation R when specifying integrity constraints. This privilege can also restricted specific attributes of R.

The mechanism of views is an important discretionary authorization mechanism in its own right. For example if the owner A of relation R wants another account B to be able to retrieve only some field of R, then A can create a view V of R that includes only those attribute and then grant SELECT on V to B.

Revoking privilege- In some case it is desirable to grant privilege to a user temporarily. For example, the owner of relation may want to grant the SELECT privilege to a user for a specific task and revoke the privilege once the task is completed. Hence, a mechanism for revoking privileges is needed. In SQL REVOKE command is included for the purpose of cancelling privileges.

Propagation of privileges using the GRANT OPTION- Whenever the owner A of relation R grants privilege on R to another account B, the privilege can be given to B with or without the GRANT OPTION. If the GRANT OPTION is given, this means that B can also grant privilege on R to other accounts. Suppose that B is given the GRANT OPTION by A and that B grants the privilege on R to a third account C, also with GRANT OPTION. In this way, privilege on R can propagate to other account without the knowledge of the owner of R. If the o
owner account A now revoke the privilege granted to B, all the privilege that B propagate based on that privilege should automatically be revoked by the system.

An example

Suppose that the DBA creates four accounts --A1, A2, A3, and A4-- and wants only A1 to be able to create base relations; then the DBA must issue the following GRANT command in SQL:

GRANT CREATETAB TO A1;

In SQL2 the same effect can be accomplished by having the DBA issue a CREATE SCHEMA command as follows:

CREATE SCHAMA EXAMPLE AUTHORIZATION A1;

Now user account A1 can create tables under the schema called EXAMPLE.

Suppose that A1 creates the two base relations EMPLOYEE and DEPARTMENT; A1 is then owner of these two relations and hence *all the relation privileges* on each of them.

Suppose that A1 wants to grant A2 the privilege to insert and delete tuples in both of these relations, but A1 does not want A2 to be able to propagate these privileges to additional accounts:

GRANT INSERT, DELETE ON EMPLOYEE, DEPARTMENT TO A2;

EMPLOYEE

NAME	SSN	BDATE	ADDRESS	SEX	SALARY	DNO

DEPARTMENT

DNUMBER	DNAME	MGRSSN

Suppose that A1 wants to allow A3 to retrieve information from either of the two tables and also to be able to propagate the SELECT privilege to other accounts.
A1 can issue the command:

GRANT SELECT ON EMPLOYEE, DEPARTMENT TO A3 WITH GRANT OPTION;

A3 can grant the SELECT privilege on the EMPLOYEE relation to A4 by issuing:

GRANT SELECT ON EMPLOYEE TO A4;
(Notice that A4 cannot propagate the SELECT privilege because GRANT OPTION was not given to A4.)

Suppose that A1 decides to revoke the SELECT privilege on the EMPLOYEE relation from A3; A1 can issue:

REVOKE SELECT ON EMPLOYEE FROM A3;
(The DBMS must now automatically revoke the SELECT privilege on EMPLOYEE from A4, too, because A3 granted that privilege to A4 and A3 does not have the privilege any more.)
Suppose that A1 wants to give back to A3 a limited capability to SELECT from the EMPLOYEE relation and wants to allow A3 to be able to propagate the privilege. The limitation is to retrieve only the NAME, BDATE, and ADDRESS attributes and only for the tuples with DNO=5.
A1 then create the view:

```
CREATE VIEW A3EMPLOYEE AS
SELECT NAME, BDATE, ADDRESS
FROM EMPLOYEE
WHERE DNO = 5;
```

After the view is created, A1 can grant SELECT on the view A3EMPLOYEE to A3 as follows:

```
GRANT SELECT ON A3EMPLOYEE TO A3 WITH GRANT OPTION;
```

Finally, suppose that A1 wants to allow A4 to update only the SALARY attribute of EMPLOYEE;
A1 can issue:

```
GRANT UPDATE ON EMPLOYEE (SALARY) TO A4;
```

(The UPDATE or INSERT privilege can specify particular attributes that may be updated or inserted in a relation. Other privileges (SELECT, DELETE) are not attribute specific.)

CHECK YOUR PROGRESS

2. Choose the correct answer

i). To create a schema or base relation _____ privilege used.

(b) Create table (b) Drop.

(d) Select (d) Delete.

ii) To apply schema changes such as adding or removing attributes from the relation _____ privilege used.

(a) Alter. (b) Modify.

(c) Delete. (d) Non of the above.

iii) To insert, delete, or update tuples _____ privilege used.

(a) Alter. (b) Modify.

(c) Delete. (d) Insert.

iv) _____ command is used for the purpose of cancelling privileges

(a) Grant. (b) Revoke.

(c) Delete. (d) Non of the above.

9.5 MANDATORY ACCESS CONTROL

It is a security policy that classifies data and users based on security class. Typical security classes are top secret (TS), secret (S), confidential (C), and unclassified (U), where TS is higher level and U is lower level. Other security classifications exist, in which the security classes are organized in a lattice. The commonly used model for multilevel security, known as the Bell-LaPadula model, classifies each subject (user, account, programs) and object (relation, tuple, column, view, operation) in to one of the security classifications TS, S, C, or U. We will refer to the clearance (classification) of a subject S as **class(S)** and to the classification of an object O as **class (O)**. Two restrictions are enforced on data access based on the subject/object classifications

1. A subject S is not allowed read access to an object O unless class(S) ≥ class(O). This is known as the simple security property.
2. A subject S is not allowed to write an object O unless class(S) ≤ class(O). This is known as star property (or *-property).

Mandatory policy ensure a high degree of protection- in a way, they prevent any illegal flow of information. They are therefore suitable for military types of applications, which require a high degree of protection. However, mandatory policies have the drawback of being too rigid in that they require a strict

classification of subjects and objects into security levels, and therefore they are applicable to very few environments.

To incorporate multilevel security notions into the relational database model, it is common to consider attribute values and tuples as data objects. Hence, each attribute A is associated with a **classification attribute** C in the schema, and each attribute value in a tuple is associated with a corresponding security classification. In addition, in some models, a **tuple classification** attribute TC is added to the relation attributes to provide a classification for each tuple as a whole. Hence, a **multilevel relation** schema R with n attributes would be represented as

$R(A_1,C_1,A_2,C_2, \ldots, A_n,C_n,TC)$

where each $C_i$represents the classification attribute associated with attribute A_i.

The value of the TC attribute in each tuple t – which is the *highest* of all attribute classification values within t – provides a general classification for the tuple itself, whereas each C_i provides a finer security classification for each attribute value within the tuple.

The **apparent key** of a multilevel relation is the set of attributes that would have formed the primary key in a regular (single-level) relation.

A multilevel relation will appear to contain different data to subjects (users) with different clearance levels. In some cases, it is possible to store a single tuple in the relation at a higher classification level and produce the corresponding tuples at a lower-level classification through a process known as **filtering**.

In other cases, it is necessary to store two or more tuples at different classification levels with the same value for the *apparent key*. This leads to the concept of **polyinstantiation** where several tuples can have the same apparent key value but have different attribute values for users at different classification levels.

In general, the **entity integrity** rule for multilevel relations states that all attributes that are members of the apparent key must not be null and must have the *same* security classification within each individual tuple.

In addition, all other attribute values in the tuple must have a security classification greater than or equal to that of the apparent key. This constraint ensures that a user can see the key if the user is permitted to see any part of the tuple at all.

Other integrity rules, called **null integrity** and **interinstanceintegrity**, informally ensure that if a tuple value at some security level can be filtered (derived) from a higher-classified tuple, then it is sufficient to store the higher-classified tuple in the multilevel relation.

9.6 ROLE BASED ACCESS CONTROL

Role–based access control (RBAC) emerged rapidly in 1990s as a proven technology for managing and enforcing security in large scale enterprisewide systems. Its basic notion is that permissions are associated with roles, and users are assigned to appropriate roles. Roles can be created using the CREATE ROLE and DESTROY ROLE commands. The GRANT and REVOKE commands can be used to assigned and revoke privileges from role.

RBAC ensures that only authorize user can access to the certain data or resources. Users create session during which they may active a subset of roles to which they belong. Each session can be assigned to many roles, butit maps to only one user or a single subject.

Role hierarchy in RBAC is a natural way of organizing roles to reflect the organization’s lines of authority and responsibility.

Another important consideration in RBAC systems is the possible temporal constraints that may exist on roles, such as time and duration of role activations, and timed triggering of a role by an activation of another role.

Using an RBAC model is highly desirable goal for addressing the key security requirements of Web-based applications.

9.7 ENCRYPTION AND PUBLIC KEY INFRASTRUCTURES

Encryption is a means of maintaining secure data in an insecure environment.Encryption consists of applying an **encryption algorithm** to data using some prespecified**encryption key**. The resulting data has to be **decrypted** using a **decryption key** to recover the original data.

In 1976 Diffie and Hellman proposed a new kind of cryptosystem, which they called **public key encryption**. Public key algorithms are based on mathematical functions rather than operations on bit patterns.They also involve the use of two separate keys, in contrast to conventional encryption, which uses only one key.The use of two keys can have profound consequences in the areas of confidentiality, key distribution, and authentication.The two keys used for public key encryption are referred to as the **public key** and the **private key**. The private key is kept secret, but it is referred to as *private key* rather than a *secret key* (the key used in conventional encryption) to avoid confusion with conventional encryption.

A public key encryption scheme, or infrastructure, has six ingredients:

1. *Plaintext* : This is the data or readable message that is fed into the algorithm as input.
2. *Encryption algorithm* : Theencryption algorithm performs various transformations on the plaintext.
3. and 4.*Public and private keys*: These are pair of keys that have been selected so that if one is used for encryption, the other is used for

decryption. The execttransformations performed by the encryption algorithm depend on the public or private key that is provided as input.

5. *Ciphertext* : This is the scrambled message produced as output. It depends on the plaintext and the key. For a given message, two different keys will produce two different ciphertexts.

6. *Decryption algorithm*: This algorithm accepts the ciphertext and the matching key and produces the original plaintext.

Public key is made for public and private key is known only by owner.

A general-purpose public key cryptographic algorithm relies on one key for encryption and a different but related one for decryption. The essential steps are as follows:

1. Each user generates a pair of keys to be used for the encryption and decryption of messages.

2. Each user places one of the two keys in a public register or other accessible file. This is the public key. The companion key is kept private.

3. If a sender wishes to send a private message to a receiver, the sender encrypts the message using the receiver's public key.

4. When the receiver receives the message, he or she decrypts it using the receiver's private key. No other recipient can decrypt the message because only the receiver knows his or her private key.

The RSA Public Key Encryption algorithm, oneof the first public key schemes was introduced in 1978 by Ron Rivest, Adi Shamir, and Len Adleman at MIT and is named after them as the RSA scheme.The RSA encryption algorithm incorporates results from number theory, combined with the difficulty of determining the prime factors of a target.The RSA algorithm also operates with modular arithmetic – mod n.

Two keys, *d* and *e*, are used for decryption and encryption.An important property is that d and e can be interchanged.*n*is chosen as a large integer that is a product of two large distinct prime numbers, *a* and *b*.The encryption key *e* is a

randomly chosen number between *1* and *n* that is relativelyprime to *(a-1) x (b-1).* The plaintext block P is encrypted as $P^e mod\ n$.Becausethe exponentiation is performed *mod n,* factoring$P^e$to uncover the encrypted plaintext is difficult.However, the decryption key *d* is carefully chosen so that $(P^e)^d mod\ n = P$.Thedecryption key *d* can be computed from the condition that$d\ x\ e = 1\ mod\ ((a-1)x(b-1))$.Thus, the legitimate receiver who knows *d* simply computes$(P^e)^d\ mod\ n = P$ and recovers *P* without having to factor P^e.

CHECK YOUR PROGRESS

3. Choose the correct answer

i) Typical security classes are top secret (TS), secret (S), confidential (C), and_____.

(a) Simple (b) Unclassified.

(c) Classified (d) Non of the above.

ii) A subject S is not allowed to write an object O unless class(S) ≤ class(O). This is known as

(a) Star property. (b) Close property.

(c) Simple Access. (d) Non of the above.

iii) RBAC stand for

(a) Role–based access control. (b) Role–based action control.

(c) Role–based access concurrency. (d) Non of the above.

iv) _____is a means of maintaining secure data in an insecure environment.

(a) Encryption. (b) Decryption.

(c) Encoding. (d) Decoding.

9.8 LET US SUM UP

- Database security issues are legal and ethical issue, policy issue, system related issues.
- Threat to database result in the loss or degradation of integrity, availability, and confidentiality.
- The database administrator (DBA) is the central authority for managing a database system.
- If any tampering with the database is suspected, a **database audit** is performed.
- A database log that is used mainly for security purposes is sometimes called an **audit trail**.
- The typical method of enforcing **discretionary access control** in a database system is based on the granting and revoking **privileges**.
- In SQL REVOKE command is included for the purpose of cancelling privileges.
- Typical security classes are top secret (TS), secret (S), confidential (C), and unclassified (U).
- A subject S is not allowed read access to an object O unless class(S) ≥ class(O). This is known as the simple security property.
- A subject S is not allowed to write an object O unless class(S) ≤ class(O). This is known as star property (or *-property).
- The **apparent key** of a multilevel relation is the set of attributes that would have formed the primary key in a regular (single-level) relation.
- Role hierarchy in RBAC is a natural way of organizing roles to reflect the organization's lines of authority and responsibility.
- Encryption is a means of maintaining secure data in an insecure environment.
- *Public and private keys*: These are pair of keys that have been selected so that if one is used for encryption, the other is used for decryption.

9.9 ANSWERS TO CHECK YOUR PROGRESS

1. i)(d)	**ii)(b)**	**iii) (d)**	
2.i) (a)	**ii) (a)**	**iii) (b)**	**iv) (b)**
3.i)(b)	**ii)(a)**	**iii) (a)**	**iv)(a)**

9.10 FURTHER READINGS

1. R. Elmasri, S.B. Navathe, Fundamentals of Database System, Pearson

2. A. Leon, M. Leon, Fundamentals of Database Management System, Tata McGraw Hill.

9.11 MODEL QUESTIONS

1. What is database security?
2. What are the different database security issues?
3. What do you mean by Loss of Integrity?
4. What is meant by granting a privilege?
5. What is meant by revoking a privilege?
6. List the type of privilege available in SQL.
7. What is the difference between discretionary and mandatory access control?
8. Define the following terms: apparent key, polyinstantiation, filtering.
9. What are the relative merits of using DAC and MAC?
10. What is role-based access control? In what ways is it superior DAC and MAC?
11. What is the goal of encryption? What process is involved in encrypting data and then recovering it at the other end?
12. What is public key infrastructure? How does it provide security?
13. Give an example of an encryption algorithm and explain how it works.